Abstract of “Physically Plausible Human Pose and Control Estimation from Video” by Marek Vondrák, Ph.D., Brown University, May 2013.

We propose a new paradigm for vision-based human motion capture. This paradigm extends the traditional capture of poses by providing guarantees of physical plausibility for the motion reconstructions and mechanisms for adaptation of the estimated motions to new environments. We achieve these benefits by estimating control programs for simulated physics-based characters from (potentially monocular) images. The control programs encode motions implicitly, based on their “underlying physical principles” and reconstruct the motions through simulation. Feedback within the control allows application of the principles in modified environments, providing an ability to adapt the motion to external events and perturbations. We explore two control models: trajectory-control and state-space control. We first consider a trajectory-tracking controller model where the desired behavior of the character is encoded as a sequence of per-frame target poses tracked by the controller. We can recover this sequence incrementally and produce pose estimates that do not suffer from common artifacts such as excessive jitter, out-of-plane rotations or foot skate. However, the inference process is prone to overfitting, and because the feedback in control is limited and the decision making process rudimentary, the encoding prohibits effective motion adaptation. To address both these limitations, we then explore a more compact model that is less sensitive to the quality of observations and that learns and exploits the underlying structure of the motion for control. State-space controllers allow concise representation of motion dynamics through a sparse set of target poses and control parameters, in essence allowing a key-frame-like representation of the original motion. We represent state-space controllers using state machines that discriminatively characterize the desired behavior of the character in terms of motion phases (states) and physical events that cause the phases to switch (transitions, *e.g.*, a foot contact). Parameters of the controller parameterize the control programs that reproduce the individual phases in simulation. Because this control representation is sparse, we are able to integrate information locally from multiple (tens of) image frames in inference, inducing smoothness in the resulting motion, resolving some of the ambiguities that arise in monocular video-based capture and enabling inference with weak likelihoods. By learning the structure of the motion, we can automatically estimate controllers that reproduce the observed behaviors directly from monocular video. By using the structure and the feedback for control, we can provide motion adaptations. We demonstrate our approach by capturing sequences of walking, jumping, and gymnastics. We evaluate our methods quantitatively and qualitatively and illustrate that we can produce motion interpretations that go beyond state-of-the-art in pose tracking and are physically plausible.

Physically Plausible Human Pose and Control Estimation from Video

by
Marek Vondrák
Sc. M., Charles University, 2006
Sc. M., Brown University, 2010

A dissertation submitted in partial fulfillment of the
requirements for the Degree of Doctor of Philosophy
in the Department of Computer Science at Brown University

Providence, Rhode Island
May 2013

This dissertation by Marek Vondrák is accepted in its present form by
the Department of Computer Science as satisfying the dissertation requirement
for the degree of Doctor of Philosophy.

Date ________	________________
	Odest C. Jenkins, Director

Recommended to the Graduate Council

Date ________	________________
	Leonid Sigal, Reader Disney Research Pittsburgh
Date ________	________________
	Jessica Hodgins, Reader Carnegie Mellon University
Date ________	________________
	David Fleet, Reader University of Toronto
Date ________	________________
	James Hays, Reader

Approved by the Graduate Council

Date ________	________________
	Peter M. Weber Dean of the Graduate School

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Motion capture is a popular approach for creating natural-looking human characters. Traditionally, the data is captured using optical marker-based systems. However, such systems require instrumentation and calibration of the environment and the actor. One way to mitigate this limitation is to perform motion capture from video of an unmarked subject. This approach is particularly appealing for monocular video, which potentially allows motion capture from legacy footage. We address this problem by estimating controllers that are capable of simulating the observed human behavior.

We take as an input a sequence of images from one or more camera views that depict the motion of the person and a coarse initial 3D pose of the person in the first image. We then capture a 3D full-body character model from the images in the video and animate this model such that the motion that was observed is reproduced in the animation. We produce this motion using simulation and control, where control represents a "program" for internal actuation of the body and represents the motion implicitly, based on the "underlying principles of the motion". These principles reconstruct the motion but can also be directly applied to new characters in new environments in order to obtain adaptations to these situations. We focus on the estimation of human motion, in part, because of its potential for promising applications in robotics, gaming and computer graphics. For example, in gaming, currently, human motion perceived through the use of Microsoft Kinect sensor has been used to drive characters in virtual environments. Our goal is to provide a similar level of perception by only using images from a standard color camera and by exploiting physical constraints to regularize results.

We explore computer vision techniques, where visual information in the form of images is the only direct source of information available to the system. This modeling choice allows for the most generic form of motion capture, as it potentially does not impose any requirements on the scene and the appearance of the person, but it introduces new challenges. In our work, we address these challenges by using physical constraints, knowledge of the scene and implicit representations of motion through control. Specifically, we use these tools to constrain potential motion interpretations to only those that are physically plausible in the given environment, such that they can be realized by a human without falling. We demonstrate that by utilizing these constraints, we can generate motion estimates that do not exhibit non-physical artifacts such as foot skate and we can do so even when only information from a single camera view is available. We formally qualify our problem in the following problem statement.

1.1 Problem Statement

We focus on the problem of estimating a physically plausible three-dimensional motion of a single articulated human subject from a calibrated monocular or multiocular video. We are interested in recovering the 3D kinematic pose of the subject in every frame of the sequence, assuming the pose at the initial frame is known, such that

1. the resulting skeletal motion is representative of the motion observed in the images (subject is *tracked*, see part (a) in Figure 1.1) and
2. the recovered 3D pose trajectories do not violate physical laws imposed by the environment and the body model (recovered motion is *physically plausible*, see part (b) in Figure 1.1).

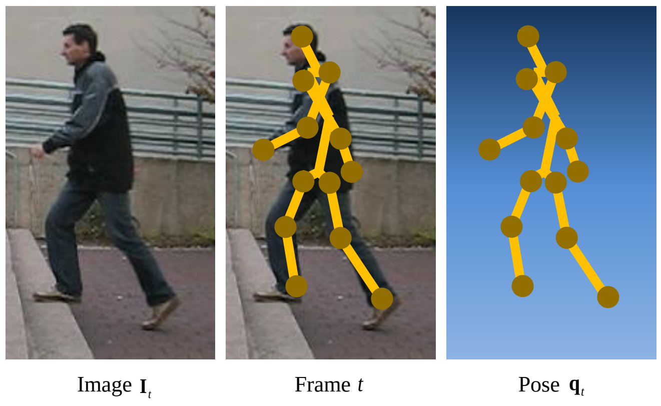
Our main focus is addressing physical plausibility in the recovered motions. Our approach consists of a physical model of the person and an inference mechanism. We assume the physical model can generate the motion of the person observed in the video by simulating a virtual character representation of the person from a given initial pose, and we seek parameters of this model such that the observed motion is reproduced in simulation (see Figure 1.2).

1.2 Motion Capture: State-of-the-art

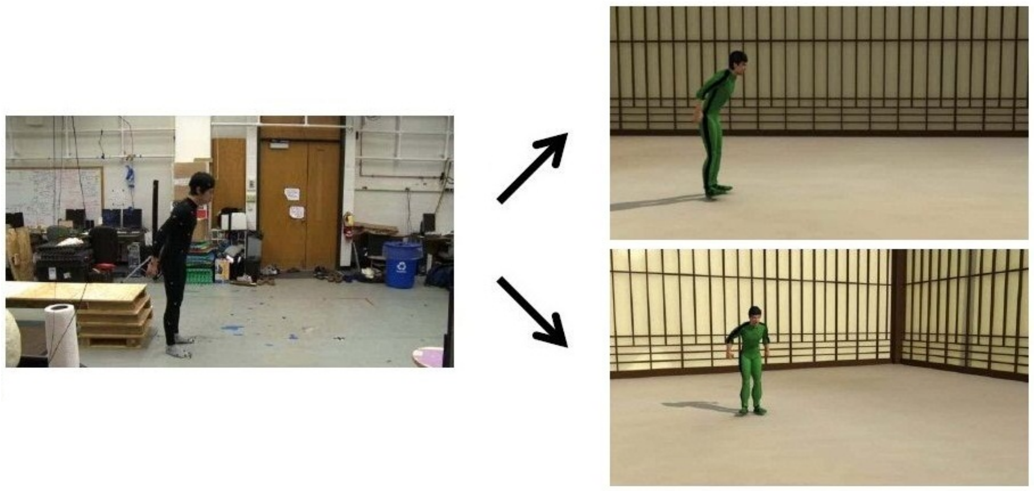
In order to motivate goals for our work, we provide a brief overview of the current state-of-the-art in motion capture. In doing so, we first discuss current applications for general motion capture as well as available methods that make these applications possible.

1.2.1 Applications

Current Applications. General estimation of human motion as a sequence of 3D poses over time (human motion capture) is important for many current and future applications in computer graphics, robotics, medicine and surveillance. Perhaps the most prominent use for human motion capture is production of realistic high-quality content for virtual reality environments, including video games and animated movies. Such content is most often produced through the use of a special motion capture equipment (*e.g.*, optical marker-based motion capture) that lets the computer "see" and accurately record the 3D skeletal motion of an actor in a motion capture laboratory or a studio. The captured motion is then applied to an animated character in the virtual environment to produce animations. This technique is particularly appealing because it (1) eliminates much of the labor associated with manually producing realistic animations and (2) the recorded performance



(a)



(b)

Figure 1.1: **Physically Plausible Motion Estimation.** We are interested in recovering the 3D kinematic pose of the person in every frame of the sequence such that (a) the resulting skeletal motion is representative of the motion observed from the view used for estimation, and (b) the recovered 3D poses trajectories do not violate physical laws and do not exhibit non-physical artifacts such as a lack of balance or foot skate, even when viewed from different angles.

captures the subtleties of the motion and ensures physical realism. When combined with methods for humanoid control in robotics, the recorded motion can be applied to humanoid robots in order to make them imitate the captured behaviors. In sports and medicine, the captured 3D pose sequences have been used to analyze the performance of professional athletes during practice or competition, to analyze body mechanics, or for studying diseases of the human musculoskeletal system.

Current Methods. The success of these applications heavily depends on the ability of the motion capture

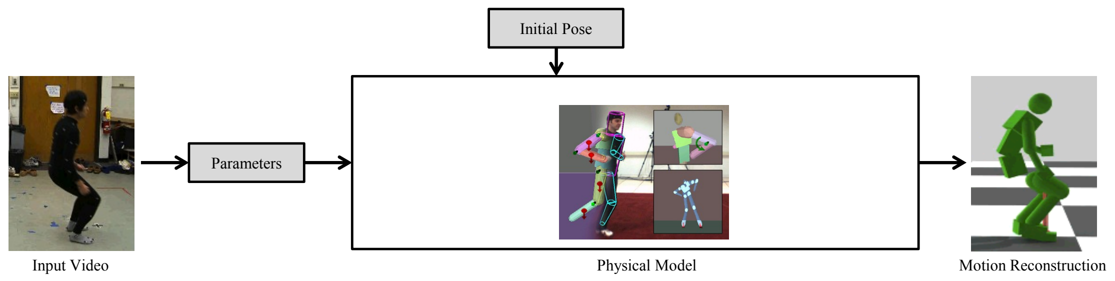


Figure 1.2: **Our Approach at a Glance.** We aim to estimate parameters of a physical model that generates the motion of the person observed in the input video. Given the input video (left) and an initial pose of the person (middle top), we seek parameters of the physical model (middle) such that the observed motion is reproduced by the model. The physical model animates a virtual character representative of the person from the initial pose using the model parameters, where the parameters characterize the specific behavior to be generated, and implicitly reconstruct the motion (right). The motion reconstruction is either directly produced by physical simulation of the character with the estimated model parameters in the model, or strongly biased to the results of such a simulation.

systems to record the observed human motion accurately and without non-physical artifacts. Meeting such demands is difficult in practice. The current methods attempt motion capture either from images or by relying on special equipment and manual labor. Vision-based techniques are challenged with perception ambiguities and noise, producing motion interpretations with feet sliding on the floor and not supporting the body properly. Commercial systems, on the other hand, make use of special equipment (*e.g.*, markers) to simplify the perception of the motion. Better perception simplifies motion estimation, improves the estimation accuracy and reduces violation of physical laws, but does not remove the violations completely. Our goal with our work is to address these shortcomings.

1.2.2 Vision-based Motion Capture

Multiple Views. Vision-based motion capture systems estimate motion of people by only analyzing pixels in images. This general approach is particularly appealing for applications because it can aid in capturing motions outside the motion capture studio and the approach does not require any use of special equipment, nor cooperation of subjects. Because images from plain cameras can be used as an input, one can use such systems to capture motion sequences in environments where the use of instrumentation (*e.g.*, markers) or other devices is not possible or would be difficult, as in jumping from a diving board. As such, vision-based motion capture is the least invasive and most general method for motion capture. It can also be applied to tracking other living beings besides humans, including wild animals in the nature.

With that said, the success of these systems is prone to perception ambiguities and the ability of the systems to model pixel-wise appearance of people in images effectively such that the pose of the person can be "seen" in the image. Modeling the changes in appearance between frames is difficult, because pixel values change drastically with the viewpoint and the pose of the person. Background clutter, ambiguities due to camera projection and occlusion due to clothing make the estimation even more difficult. Consequently, to reduce the ambiguities and provide more constraints for the estimation, vision-based motion capture has typically been done from multiple camera views [25, 46, 57, 110, 189, 172], although approaches for a single camera view have become available as well.

Single View. Estimation from a single camera view is more interesting, as it allows for more general applications. Single-view systems could potentially estimate motion from any existing video footage, such as a personal video taken by a cellphone camera or an old movie that can not be easily reproduced. For example, we may think of using such a system to reconstruct a motion of Charlie Chaplin in a movie clip downloaded from the Internet and use the 3D reconstruction to provide new experiences. For example, the estimated motion can be re-rendered from a different angle or a moving camera, to see how the scene would look like if it was originally shot this way. On the other hand, the lack of multiple views makes motion estimation ill-conditioned such that it is not always possible to tell which part of the body is closer to the camera. These ambiguities can not be resolved in inference without making use of prior knowledge and assumptions.

In order to address these challenges, strong motion priors [183, 22, 106, 158, 48, 177] have been employed to regularize estimation results. Despite many successes, the methods often produce impossible sequences that violate physical laws. More recently, it has been demonstrated that physics-based models and physical dynamics may provide an appropriate paradigm for addressing these shortcomings. With that said, the use of physical models in computer vision has been limited to only a few instances [118, 205, 26] and none of these methods was able to guarantee to produce motion interpretations that satisfy the laws exactly.

1.2.3 Commercial Motion Capture Solutions

Commercial motion capture systems rely on richer sources of information than monocular images in order to simplify the motion estimation task. Such simplifications facilitate better perception of the motion and include the use of multiple camera views, instrumentation of subjects and the environment with special sensors and other equipment, and having the person wear tight-fitting clothes. Despite the impressive performance, these systems still produce pose sequences that may not be physically plausible.

Markers. State-of-the-art optical marker-based motion capture systems (*e.g.*, Vicon) recover human motion by tracking retroreflective markers physically attached to the body of the person. Motion capture is realized by using an array of calibrated infrared cameras that observe the scene and perceive the markers, and a computer system that reconstructs 3D poses of the person from the 2D trajectories of the markers. Such marker-based systems are known to produce high quality pose estimates in real time and have enabled many current applications in computer graphics and movies. However, they are inherently limited to capturing cooperative subjects wearing motion capture suits in laboratory environments, and capturing sessions require thorough planning and setup.

Depth Maps. More recently, motion capture systems that use depth information as an alternative (*e.g.*, Microsoft Kinect) have been proposed. Depth-based motion capture systems use sensors capable of measuring pixel distances from the camera and are not restricted to work in controlled environments. Direct depth information allows the efficient segmentation of the person from the background and resolves pose depth ambiguities during pose inference. These motion capture approaches facilitate pose estimation from a single camera view, but the operational range of sensors is somewhat limited and prohibits tracking motions involving significant interactions of the person with the environment or outdoors.

1.3 Beyond Motion Capture: Thesis Goals

The previous research methods and commercial solutions for motion capture produce motion interpretations that may violate physical laws. Furthermore, the goal of those methods is only reconstruction of kinematic poses in the sequence. Our effort contrasts this goal and pursues a new *formulation for motion capture* that goes beyond pose estimation.

Our goal is to improve motion capture by providing a guarantee of *physical plausibility* for the motion reconstructions and mechanisms for motion *adaptation to new environments*. We aim to achieve this goal by using abstract motion representations that can be estimated directly from *monocular video observations*, although our formulation should also be applicable to other input modalities, such as markers. This formulation can improve applications from Section 1.2.1 by providing more physically plausible interpretations for the observed motions as well as can enable new applications beyond traditional motion capture (see Section 1.3.2). We further clarify and provide motivations for our goals below:

1. Monocular Observations.

   Our formulation should permit vision-based marker-less motion capture from a single camera view so that the capture can be realized in unrestricted environments (see Section 1.2.2).

2. Physical Plausibility.

   Our formulation should directly incorporate physical constraints in order to make estimation from the single view easier and to produce motion reconstructions that are physically plausible. Specifically, the poses in the reconstructed sequence should not violate physical laws and constraints imposed by the model of the body and the environment so that non-physical artifacts are eliminated from the motion interpretations (see Section 1.3.1).

3. Adaptability to New Environments.

   Our formulation should represent captured motions using higher-level abstractions such that the motions could be directly adapted to simulated characters in new situations. Towards that end, we propose to represent motions using "physical principles" that generated them in physical simulation (see Section 1.3.2).

1.3.1 Why Physical Constraints and Physical Plausibility?

Physical Models, Physical Constraints to Regularize Results. Accomplishing motion capture with a single camera and without the aid of instrumentation is difficult. The information about markers and/or pixel depths is not available and the motion of the person can only be perceived from indirect monocular measurements

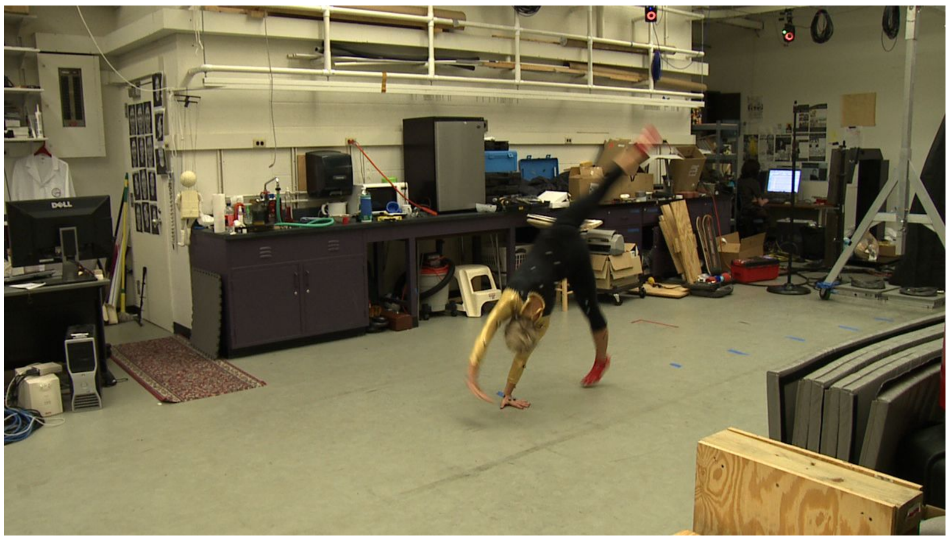


Figure 1.3: **Physical Constraints.** A motion of the person is naturally driven by physical laws and interactions of the person's body with environment. Our goal is to build a motion capture system that can incorporate these physical constraints directly into inference in order to provide more accurate and realistic motion interpretations.

(*e.g.*, silhouettes) which are inherently inaccurate and ambiguous. Our aim is to alleviate such ambiguities and to make motion estimation from a single camera view tractable by utilizing *physical models* of human motion, physical cues in the image observations and physical constraints defined by the character model and the environment. In doing so, we leverage recent advances in computer graphics, animation and robotics, where physical models have been shown to be effective in modeling physically plausible human dynamics through simulation.

Physical models represent human motion by modeling physical laws, constraints and underlying causes that make the body move and implicitly produce motions that are consistent with such laws. Synthesis of human motion through dynamical simulation in the physical model exploits the fact that human motion is naturally driven by the physical laws, gravity and interactions of the human body with the environment (see Figure 1.3). Simulation explicitly models these aspects in the synthesis process and produces realistic and smooth motions that respond to the mass and inertia of the person, friction, body contact, balance and physical disturbances, among others. Our goal is to demonstrate a motion capture system that can computationally account for these aspects during pose inference in order to regularize the estimation results, to determine which of the possible 3D motion interpretations, consistent with the image observations, are more reasonable, and favor them during the inference. In doing so, we use physics as the most generic prior for the motion of the person that is independent of the performed activity.

Physical Constraints to Produce Physically Plausible Reconstructions. Besides making the estimation easier, we demonstrate in our work that the physical models and physical constraints can accomplish the second core goal in our motion estimation problem formulation: consistency of motion interpretations with physical laws. The techniques that we propose can estimate not only poses representative of the images, but they can also ensure that the generated motions are coherent and physically plausible as a whole, such that they do not violate physical laws and physical constraints imposed on the body by the environment. In doing so, the produced motion interpretations implicitly honor the following constraints:

- *Kinematic constraints* that prescribe how body parts should be located with respect to the environment, or to other parts, so as to not cause penetration and to not violate joint angle limits. For example, feet can be automatically constrained to stay on or above the ground.
- *Dynamical constraints* that enable dynamically plausible interactions of the body with the environment. For example, the body can be automatically constrained to remain dynamically balanced such that the feet are in contact with the ground and support walking. When performing a handstand, the same constraints can request that the body is supported by the hands. These constraints are implicitly honored during simulation; if they were to be violated, the character would either fall or not locomote properly, resulting in a motion that does not match image observations.

Existing Methods, Physical Plausibility and Artifacts. Existing motion capture efforts focus only on the pose estimation part of the problem, developing search methods and models for human motion that allow for efficient pose inference alone and that ignore consistency with physical laws. Physical plausibility is addressed only indirectly, by focusing on improving estimation accuracy.

The existing methods often utilize *statistical models* of human motion, where physical dynamics of the human body and physical laws are encoded indirectly using statistical relations. These models are designed with the aim of supporting tractable pose inference and encouraging temporal coherence of the recovered poses, but they are unable to honor physical laws. As such, these models are effective for tracking specific classes of low-energy motions, such as walking on level terrain or jogging, where exact enforcement of physical laws is not critical and their effects can be approximated well using statistics. However, such models have difficulty tracking motions that are largely determined by interactions of the person with the environment, such as jumping off a ledge. In addition, because consistency with physical laws is not guaranteed by such models, physical realism and plausibility of the recovered motions as poses over time remains mostly unaddressed.

As a result, many existing methods for motion estimation produce pose sequences that can not be performed by a human without sliding, falling or having to fly. Such violations of physical laws in the reconstructions are demonstrated in the form of visually distinct and physically implausible *artifacts* that include foot skate, levitation, out-of-plane rotations, lack of balance and excessive jitter. We hypothesize these artifacts can be attributed to the poor approximation of the true dynamics of the human body and the environment in the employed models and the use of search methods for motion inference that are unable to honor physical laws.

When contrasted with physical models, statistical models provide a less fundamental representation of the motion that only encodes the effects of the causes for the motion, but not the causes themselves. We argue that by using physical models that represent the causes, together with appropriate search mechanisms, it is possible to improve the quality of tracking, resolve ambiguities that are present in observations and produce pose estimates that honor physical laws, thus address physical plausibility in the motion estimation problem.

Discussion. Our use of physical models is motivated by the ultimate desire to recover physically plausible 3D pose trajectories that could eventually match the quality of the estimates produced by the previous solutions while imposing simpler restrictions on the actors and environment during capture. By using physical models in the context of computer vision, we hope to arrive at a new class of models that are capable of addressing physical plausibility as well as enabling tracking of more complex dynamical motions. While we apply our methods to the problem of estimating 3D motion of a person from video, our approach is not limited to image observations. We can use similar physical constraints and models to smooth results from *e.g.*, marker-based motion capture. Furthermore, the benefits of using physical models are not restricted to the motion capture formulation that we propose in our work. Such models can also be used to produce *physically plausible motion predictions* for previous pose tracking methods, as discussed in Chapter 4. We claim such predictions alone improve the quality of recovered poses and help estimate motions that do not suffer from common artifacts such as excessive jitter, out-of-plane rotations or foot skate.

1.3.2 Why Motion Representation by Physical Principles?

Physical Principles. We aim to exploit physical models and model parameterizations that encode motions based on the "physical principles" that produce them in simulations. For example, the parameters of our model may capture how the observed motion segments into simpler actions, how these actions can be reproduced through simulation and how the actions can be sequenced into valid motions (see Figure 1.9). These motion representations provide a higher-level understanding of the motion and implicitly produce not only motion reconstructions for characters in the original environments but can also be applied to simulated characters in novel situations. This technique can be used to synthesize new similar motions in response to perturbations of the environment or morphology of the person. As such, the representations allow us to go beyond video inputs and kinematic pose capture and allow to ask "what if" questions.

Reconstruction in Original Environment. First, by applying the principles to a character in the original environment, the understanding of the physical process implicitly enables reproduction of the observed motion in simulation. Using our approach, and in contrast to the previous motion capture formulations, our method can generate motion reconstructions that obey physical laws set by the simulated physical model, as discussed in Section 1.3.1. Because the encoding of the desired motion by the principles is compact, it can be estimated from video effectively. Furthermore, the actual recovered poses do not have to be stored explicitly and can be always reconstructed by simulation.

New Environments. Second, the understanding of the underlying physical process also allows the environment in which the recorded activity takes place to be changed. By applying the principles to a character in a modified environment, new similar motions compliant with the new environment can be produced. For example, walking on level ground could be captured using our approach as a progression of four actions that the simulated character needs to imitate in order to walk (see Figure 1.9): the character needs to push off the ground with the right foot first, then it needs to swing the right foot forward, wait until the right foot hits the ground, then push off the ground with the left foot, wait until the left foot hits the ground and repeat. Using these concepts, we can characterize the entire process of walking in terms of the pattern for the foot-ground contact and the maneuvers that produce this pattern. This high-level description of walking is universal over walking styles and executions in different environments. Consequently, if the environment changes, and the character is supposed to walk down a slope instead, the high-level description of the motion can still be directly applied on a simulated character in this new environment and a new motion produced.

New Applications Beyond Current Motion Capture. For a large number of applications, estimation of raw motion as poses over time may not be the final goal and the captured sequences may need be further processed as a next step.

For example, with traditional kinematic motion capture, recovered 3D poses need be further edited before they can be applied on a character in a movie or in a video game. This editing may include matching the dynamics of the recorded actor to the dynamics of the new character, eliminating foot skate or adjusting the estimated motion to dynamic changes in the environment. We hypothesize some of the required adjustments may emerge automatically by reusing the estimated principles. These automatic adjustments would be particularly appealing for online applications, such as video games, where manual editing of poses is not feasible. For example, we envision the proposed form of motion capture could be used as an input device for a video game, where characters within the game would directly imitate motions of players recorded by the motion capture system. The recorded motions could be either applied directly on the characters in the game or, alternatively, they could be used to build an offline library of motion skills that the characters would exploit while navigating the game environment.

These aspects of motion editing currently fall outside the scope of computer vision and motion capture techniques. The overarching goal of this research is to estimate physical models of the character from video that are capable of performing the desired motion(s). We aim to learn the underlying physical principles that produced the motion in the video and encode the desired motion for the model using the principles. The proposed method and illustrated results are a step towards this ambitious goal that, we hope, will serve as a stepping stone for future research. We view this form of capture of the principles, in addition to the pose, as a next step in the evolution of motion capture.

1.4 Thesis Statement

The core objective of this thesis is to demonstrate that it is technically feasible to realize marker-less motion capture from a single camera view, as defined in our problem statement in Section 1.1 and with the properties motivated in Section 1.3. According to these motivations, we achieve this objective by trying to learn the underlying physical principles that produced the motion in the video. We illustrate that such an approach is enabled by the use of physical models of human motion and methods for humanoid control and we summarize this finding in the following thesis statement:

Physical models of human dynamics and methods for humanoid control can be used to estimate physically plausible human motions from image sequences, where physically plausible refers to motion reconstructions produced by physical simulations of a character model from an initial pose.

We explain the terms in the thesis statement next. As discussed in Section 1.1, to estimate a motion from image sequences means to estimate a full-body 3D pose of the person in every frame of the sequence such that the pose fits what is seen in the particular frame. The key aspect of our effort is that we want the estimated pose sequence to not only match the images, as is common for other motion capture systems, but we also want the sequence to be *physically plausible*. We discuss the concept of physical plausibility in the following section.

1.4.1 Physical Plausibility

We first explain what it means for a pose sequence (motion) to be physically plausible. We then discuss how physically plausible sequences can be synthesized using physical models and how it can be determined whether a given sequence is physically plausible. Finally, we discuss the consequences of our definition and implications on tractability of inference of physically plausible motions.

Overview of Physical Models, Physical Dynamics. We assume that the motion of the person can be generated by a physical model of human dynamics. Physical models produce the motion by approximating physical dynamics of the human body and actuation of the body. The physical dynamics component within the physical model produces the motion through the application and integration of forces, where the forces result from internal actuation of the body and general physical laws that account for the response to factors external to the body, including contact between the person and the environment, and gravity. We assume that the effects of the external factors are automatically incorporated in the modeled dynamics, such that it is possible to parameterize motions only in terms of the body actuation. We outline physical laws and models for the actuation of the body in Section 1.4.2 and Section 1.4.3, and discuss them later in depth in Chapter 3.

Physical Plausibility. Assuming a physical model is given, we have a definition for physical dynamics that evolves poses between frames and a definition for actuation forces that control the evolution. We then say a motion is *physically plausible* with respect to the dynamics, if the poses in the motion can be generated by the physical dynamics from an initial pose by applying actuation forces. Because we model physical dynamics on a computer, the modeled dynamics is discretized and approximated using a simulator. Consequently, in practical terms, physical plausibility means that the pose sequence can be synthesized by physical simulation of the character model within the physical model, where the behavior of the simulation is determined by the actuation forces. We restate this definition in equations below.

We abstract physical dynamics within a physical model as a state dynamics function $f : [\mathbf{x}_t, \mathbf{u}] \rightarrow \mathbf{x}_{t+1}$ that updates the state $\mathbf{x}_t$ of the physical model from the frame t to the next frame $t+1$ in response to controls $\mathbf{u}_t$. Here, $\mathbf{x}_t$ encodes the state of the physical model (world) at the frame t, where the encoding of the state includes a representation of the pose of the character, and $\mathbf{u}_t$ represents actuation forces for the character for the frame. The actuation forces can be either automatically generated by an actuation mechanism with the physical model, or otherwise. Because external factors are incorporated in the dynamics, we characterize the state dynamics as

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t). \tag{1.1}$$

Given the initial state $\mathbf{x}_1$ of the world, including the pose of the character, and the actuation forces $\mathbf{u}_{1:T-1}$ for all frames, where 1 is the first frame in the sequence and T is the last frame, we can apply Equation (1.1) recursively for the individual frames $t = 1, \ldots, T-1$ to compute the sequence of states $\mathbf{x}_{1:T}$ and the corresponding poses that result from the forces $\mathbf{u}_{1:T-1}$. On the other hand, given the poses, we can attempt to solve for the actuation forces that reproduce the sequence using the physical dynamics.

For a given model of physical dynamics expressed as a function $\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t)$, a pose sequence from $\mathbf{x}_{1:T}$ is considered physically plausible if there exists a sequence of actuation forces $\mathbf{u}_{1:T}$ that produces this pose sequence from a given initial state $\mathbf{x}_1$.

Note that actuation forces are specific to the physical dynamics. For the purposes of this work, we employ a humanoid character model from Section 1.4.3.3 where actuation forces are defined as joint torques, and refer to physical plausibility with respect to this model.

Discussion. The definition of physical plausibility is consistent with our earlier motivations on physical plausibility, which required poses in the sequence to follow general physical laws and physical constraints defined by the body and the environment. To explain this consistency, consider a sequence generated by the physical dynamics. Such a sequence implicitly obeys physical laws and constraints encoded by the dynamics. On the other hand, if a pose sequence follows physical laws and constraints that can be expressed in the dynamics, then there must exist actuation forces to act on the body that would reproduce that sequence in the model. Because physically plausible motions are, according to our definition, exactly those sequences that are raw results of physical simulations within the model of physical dynamics, the amount of jitter, foot slippage and ground plane penetration that they can contain depends only on the allowances of the simulator: the motion can only get as jittery as the internal actuation of the body permits, the feet can only slip as much as the ground contact properties allow and so on. Other artifacts are simply filtered out by the definition, because they can not be reproduced by the physical dynamics.

Our definition of physical plausibility is always specific to a particular model of physical dynamics. As such, reasoning about physical plausibility of motions can only be done with respect to a particular model. For example, when using a model that abstracts dynamics of a character with strong “muscles”, a given motion can be seen as plausible. However when the same motion is evaluated with respect to a model with weaker muscles, or a model with more detailed skeletal structure and a more faithful body actuation system, the motion may look overpowered and not plausible: the character would not be able to perform the motion the same way while staying balanced due to differences in the abstraction of the body and actuation. Note that motions synthesized by a given model of physical dynamics are by definition always plausible with respect to that model.

Note that physically plausible motion reconstructions do not need to provide the most accurate fit to the image observations. For many applications, it is more important to estimate motions that are smooth, human-like and that capture the essence of the observed behavior rather than the ones that fit the images the best, but also feature implausible artifacts. At the same time, physically plausible motions do not have to be physically realistic. *Physically realistic* motions are physically plausible motions produced by accurate models of human dynamics that use detailed skeleton models actuated by true muscle forces. The level of physical realism of the motions produced by a particular model depends on the accuracy of the model in approximating the true human body dynamics. In this work, we settle with a rigid body dynamics approximation of human dynamics commonly used in computer graphics and animation. We use models that are specifically designed to be approximations to the true body geometry and actuation of the body that is complex enough to model key physical interactions, yet simple enough to facilitate motion inference from weak image observations. Our models provide higher-fidelity modeling of human motion, but less realism than physically accurate musculoskeletal models from biomechanics that have thousands of degrees of freedom and require rich sensor data for simulation.

1.4.2 Physical Models of Human Dynamics

In order to *produce* physically plausible motions, we need a *physical model of human dynamics* that approximates true physical dynamics of the observed person and models the actuation of the body.

Overview. The goal of physical models of human dynamics is to model human motion by simulating underlying physical processes that cause the human body to move. We aim to exploit these processes for motion estimation and use physical models to reproduce the motion of the person in the video. We first describe general aspects that need be abstracted in any physical model of human motion.

Motion of the Human Body. A particular physical model approximates the human body itself as well as the physical processes that drive and produce the motion of the body. For the purposes of motion synthesis, we can think of the real human body as consisting of multiple interacting parts supported by a rigid skeleton. Motion of the body is driven by forces applied on the skeleton, resulting in change of the pose of the body. Forces are applied by the surrounding environment (*e.g.*, on contact) and muscles within the body, as a result of internal muscle activation. Muscles are activated either involuntarily, by reflexes, or voluntarily by a thought process and an active musculoskeletal control such that a desired motion, behavior or a high-level goal is accomplished.

Physical Model for Motion of the Human Body. When abstracting these concepts in the physical model, the following aspects thus need be considered and approximated: (1) shape and articulation of the human body, (2) actuation of the human body and (3) effects of the environment. These aspects are typically accounted for in the physical model by using parameterized components: a model of the geometry of the *environment* where the motion is to take the place, a model for a virtual *character* that approximates the articulation and shape of the person in the video, a model for *actuation* of the character that computes *actuation forces* to be applied on the skeleton of the character, and a *simulator* of physical dynamics that uses these models to produce the motion of the character. The character model, environment and actuation models provide constraints for the simulation while the simulator itself implements general physical laws, integrates forces acting on the body

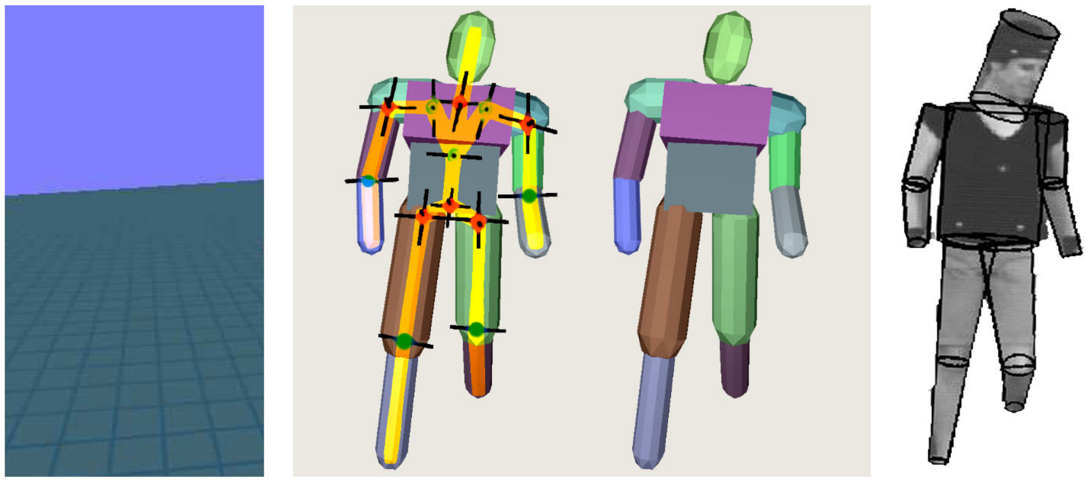


Figure 1.4: **Typical Environment and Character Model.** On the left, we illustrate a typical model of an environment consisting of only a ground plane. In the middle, we show articulation and shape of the humanoid character model from Chapter 4. The character model is actuated by actuation forces produced by motors at the joints in the articulation. Corresponding shape representation of the character in an image projection is visualized on the right.

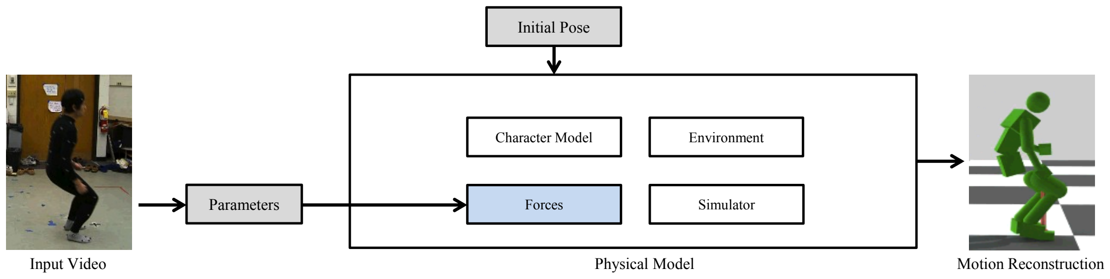


Figure 1.5: **Our Approach and Components within Physical Model.** At a high level, we represent the person as a simulated character actuated by forces. We optimize for the forces, parameterized by the model parameters, such that the motion generated by the simulation from the initial pose matches the video.

of the character and outputs a physically plausible motion that follows the constraints. See Figure 1.4 for an illustration of typical models of the environment and the character that we use in Chapter 4.

Motion Synthesis using the Physical Model. The physical model produces the motion of the person by applying *forces* to the skeleton of the simulated character (see Figure 1.5). The forces result from external disturbances in the environment (*e.g.*, gravity, wind), detected contacts between the body of the character and the environment (*e.g.*, non-penetration, friction), contacts between different parts of the body itself, body articulation at joints (*e.g.*, keeping body parts connected, damping of relative motion at joints and preventing

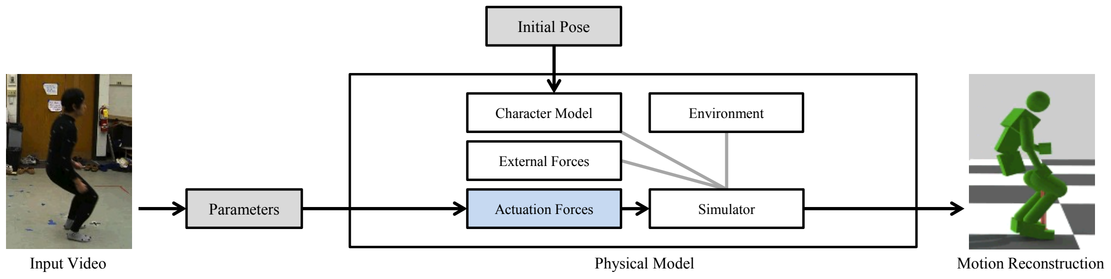


Figure 1.6: **Our Approach and Actuation Forces.** In more detail, we decompose forces to external forces due to environment and forces due to internal body actuation. Assuming we know the environment, external forces can be directly computed.

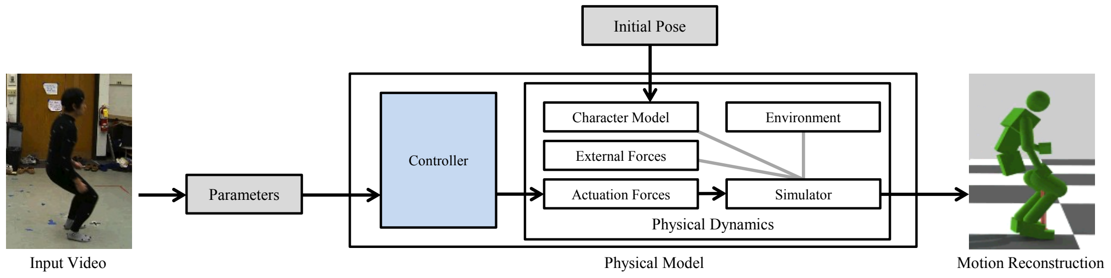


Figure 1.7: **Our Approach and Controllers.** We model internal body actuation using a parameterized controller and optimize for the parameters of the control model such that the generated motion matches the video.

hyperextension of limbs past their biomechanical limits) and internal body actuation.

Most of the abovementioned forces can be directly computed from physical laws, constraints and approximations of the human body implemented by the model, using the models of the character, environment and simulation. These general models encode general aspects of the motion such as how the foot should react to an impact with the environment or that the knee should not extend past its limit. The models alone, however, are not sufficient to constrain the motion to look like running, walking or any other voluntary activity. In order to produce motion of characters that appear life-like and move actively, the body has to be actuated through actuation forces (see Figure 1.6). We produce actuation forces using controllers and we discuss controllers in the following section (Section 1.4.3). Once actuation forces are determined, the entire process of generating the motion of the person using a physical model can be abstracted in terms of the physical dynamics function from Equation (1.1).

1.4.3 Methods for Humanoid Control

We provide actuation forces for the character using a controller (see Figure 1.8). Intuitively, a controller determines actuation forces by reacting to the changes of the state of the physical model such that a desired behavior of the character is achieved using the model (*e.g.*, the character walks or jogs). In doing so, the controller approximates underlying processes and physical principles that lead to the generation of the desired behavior within the physical model. We assume controllers are encoded using control parameters. Control parameters determine behavior of the controller and define the desired motion to be generated by the physical model. We employ controller models and motion parameterizations previously featured in computer graphics and humanoid robotics.

Formally, we define a controller as a function $g : \mathbf{x}_t \rightarrow \mathbf{u}_t$ that provides actuation forces $\mathbf{u}_t$ for the character in the state $\mathbf{x}_t$. We encode the particular mapping implemented by g using the controller parameters Θ such that

$$\mathbf{u}_t = g(\mathbf{x}_t; \Theta), \tag{1.2}$$

where g characterizes the used controller model and Θ describes what motion is to be produced using the control. We characterize the particular controller model that we use in this work in Equation (1.5).

Physical Principles for Control. Motivated by Section 1.3.2, we are interested in using controller models that encode control in terms of *physical principles*. The principles we want to capture in the controller representation should encode the general high-level structure of the motion such as the number of distinct phases that there are in the motion and the way the phases follow each other as well as the low-level structure characterizing how the body needs to be actuated to move from one phase to the next. A representation of the desired motion by physical principles needs to identify important physical events that take place during the course of the motion, learn the pattern for the events and encode how the body should react to these events. Specifically, the representation needs to encode how the body should be actuated after an event so it can arrive at the next event.

1.4.3.1 Controller Model

We realize control g as a parameterized two-level process. At the high level, a high-level *decision-making process* defines a *desired motion* for the character as a sequence of abstract *actions*. At the low level, a low-level *motion control process* computes actuation forces for the selected actions such that the desired behavior encoded by the actions is achieved (see Figure 1.8). Formally, we express this control model as

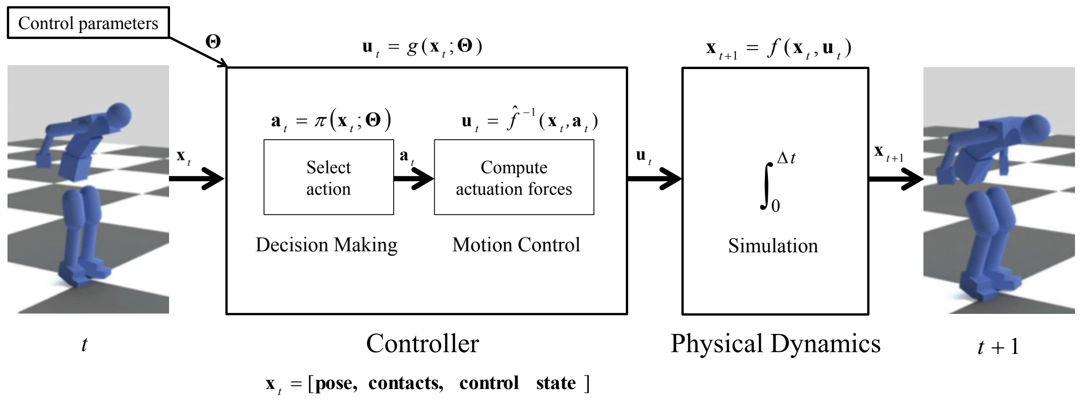
$$\mathbf{a}_t = \pi(\mathbf{x}_t; \Theta) \tag{1.3}$$

$$\mathbf{u}_t = \hat{f}^{-1}(\mathbf{x}_t, \mathbf{a}_t) \tag{1.4}$$

such that

$$g(\mathbf{x}_t; \Theta) := \hat{f}^{-1}(\mathbf{x}_t, \pi(\mathbf{x}_t; \Theta)), \tag{1.5}$$

where π implements the decision-making process, $\hat{f}^{-1}$ realizes the motion control process and $\mathbf{a}_t$ represents an action. See Figure 1.8 for an illustration.



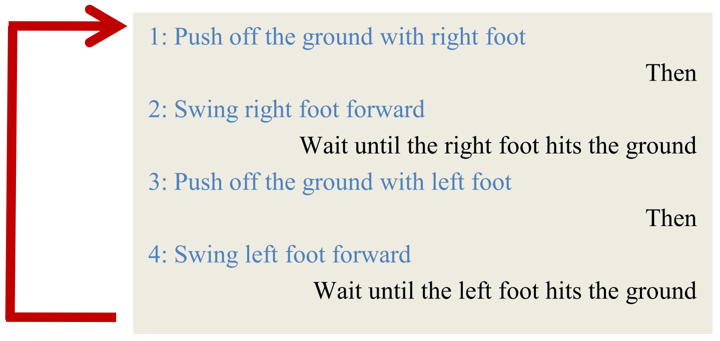
Control parameters
$\mathbf{\Theta}$
$\mathbf{u}_t = g(\mathbf{x}_t;\mathbf{\Theta})$
$\mathbf{a}_t = \pi(\mathbf{x}_t;\mathbf{\Theta})$
$\mathbf{u}_t = \hat{f}^{-1}(\mathbf{x}_t,\mathbf{a}_t)$
$\mathbf{x}_t$
Select action
$\mathbf{a}_t$
Compute actuation forces
Decision Making
Motion Control
$\mathbf{u}_t$
$\mathbf{x}_{t+1} = f(\mathbf{x}_t,\mathbf{u}_t)$
$\int_0^{\Delta t}$
Simulation
$\mathbf{x}_{t+1}$
t
Controller
Physical Dynamics
$t+1$
$\mathbf{x}_t = [\textbf{pose, contacts, control state}]$

Figure 1.8: **Controller.** Illustration of a concept of controller. A controller g computes actuation forces $\mathbf{u}_t$ for the character such that a desired motion, encoded by Θ, is generated by physical dynamics within the physical model. The controller monitors the current state $\mathbf{x}_t$ of the character and the world and computes actuation forces $\mathbf{u}_t$ such that the character is driven from the current pose towards the next desired pose by the model, as determined by Θ. We realize control as a sequence of actions $\mathbf{a}$ generated by a decision-making process π, where actions correspond to simple control tasks. We perform actions using a motion control process $\hat{f}^{-1}$ in the controller. The motion control process takes as an input the current action $\mathbf{a}_t$ and the state $\mathbf{x}_t$ of the model and outputs actuation forces $\mathbf{u}_t$ that are appropriate for the current state such that the action is performed using physical dynamics f. In selecting actions, we utilize contextual information that is specific to the decision-making process in the controller and that incorporates information about contact of the character with the environment. We encode this context in the state $\mathbf{x}_t$. Using the contact feedback, the controller can explicitly react to external events that happen in the modeled world and that introduce discontinuity in the generated motion, such as when the body support transfers from the left foot to the right foot. The ability of the controller to react to external events is important for the generalization of control to new environments.

Motion Control, Actions. We define actions as simpler control tasks for which actuation forces can be computed directly using the low-level motion control process $\hat{f}^{-1}$. Intuitively, actions can be associated with different phases of the motion such that each action encodes an actuation "program" that performs the motion in the corresponding phase. Given a particular action $\mathbf{a}$, the control process utilizes an approximation of the physical dynamics f in order to determine how the body should be actuated through actuation forces such that the action is performed.

We encode actions uniformly using vectors. Each action is described by a single desired pose that the control process attempts to achieve when the action is active and parameters of the actuation strategy that is used to do so. Parameters of the strategy represent various aspects of the body actuation, such as a speed with which the individual degrees of freedom in the pose are reached, as well as how the body should balance. We explain particular actuation strategies in Chapter 3, Chapter 4 and Chapter 5.

Decision-making Process, Finite State Machines. The action selection process is *reactive*. Rather than planning for an optimal sequence of actions offline, we select actions during the generation of the motion by reacting to changes of the state of the model. In doing so, we utilize state *feedback* that considers contact information of the body.


1: Push off the ground with right foot
Then
2: Swing right foot forward
Wait until the right foot hits the ground
3: Push off the ground with left foot
Then
4: Swing left foot forward
Wait until the left foot hits the ground

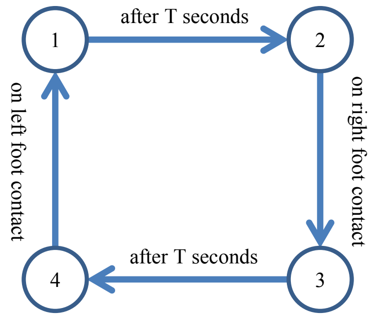


Figure 1.9: **Controller for Walking.** On the left, a high-level abstract description of walking is illustrated. In this abstraction, walking is characterized as a sequence of walking cycles. Each cycle is described as a progression of actions that the person needs to perform in order to complete the cycle and to keep walking. On the right, the same abstract description is rewritten in the form of a finite state machine, where machine states encode actions and transitions between the states describe how actions are switched. This particular state machine models walking as a sequence of four abstract actions that switch based on time or contact events.

We implement this process using the decision-making component π within the controller[1]. The decision-making component monitors which parts of the body are supported by the environment and responds to changes of this support (*e.g.*, change of foot contact, transfer of weight from one foot to the other, *etc.*) by switching the current action. We switch actions on contact changes because such changes introduce discontinuities in the motion, which suggests that an actuation strategy (action, motion phase) might need to switch too. For example, in order to locomote, the controller may be aware of which foot is currently supporting the torso, apply actuation forces to swing the non-supporting foot forward and switch between the feet in response to the transfer of the support (see Figure 1.9, left). Similarly, in order to jump, the controller may keep track of whether the character is airborne and switch to a "loading" action when the feet hit the ground such that the impact can be absorbed and the character will not fall (see Figure 1.13). We encode the desired sequencing of actions for the decision-making process using *finite state machines* (FSMs; see Figure 1.9, right), where states in the state machine represent actions and transitions between the states encode possible ways the actions can switch.

1.4.3.2 Motion Synthesis with Controller

Actuated Physical Model. We incorporate our controller in the physical model, so that the model becomes actuated (see Figure 1.7). Consequently, the resulting physical model consists of the controller g from Equation (1.2) that computes actuation forces for the character, and the physical dynamics f from Equation (1.1)

[1]Although the decision-making component in the controller is for notation simplicity expressed as a function π from a state to an action, the function has side effects. Specifically, the current state $\mathbf{x}_t$ of the model may change as a part of the evaluation of $\pi(\mathbf{x}_t)$, when the component reacts to contact feedback in the state.

that uses the forces to generate the poses of the character.

Assuming that we have an actuated physical model, an initial state $\mathbf{x}_1$ for the model and a parameterization of the desired motion for the character by Θ, we produce the motion from $\mathbf{x}_1$ by recursively applying g and f. We use Equation (1.2) to compute the actuation forces $\mathbf{u}_t$ for the individual frames $t = 1, \ldots, T-1$, and then use the actuation forces and Equation (1.1) to update the model states from $\mathbf{x}_t$ to $\mathbf{x}_{t+1}$,

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, g(\mathbf{x}_t; \Theta)). \tag{1.6}$$

The representation $\mathbf{x}_t$ of the state incorporates the pose of the character, contact information of the character, and the state of the controller. Because we always start the controller from the first state in the finite state machine, we only need to specify an initial pose of the character to bootstrap the process[2].

Feedback Simulation and Control Loop. We implement the motion synthesis process from Equation (1.6) using a feedback loop that incorporates character simulation and control (see Figure 1.10). Control within the loop selects actions and computes actuation forces for the character, simulation of physical dynamics updates the pose by applying the actuation forces. It also determines contact information for the pose in order to update the state of the controller and to switch actions within the controller.

Control and simulation run in lock step. Each iteration of the loop propagates the state of the character and the controller from one frame to the next. During this update (see Figure 1.10), the decision-making process π within the controller first selects an action $\mathbf{a}_t$ for the current frame t using the state machine and the contact information, using the contact information to switch the current state in the state machine. The result of the decision-making process is an action that is associated with the current state in the state machine and that is to be performed by the low-level motion control process. The low-level motion control process $\hat{f}^{-1}$ performs this action by computing actuation forces for the character such that the desired pose stored within the action is approached. Actuation forces are computed using inverse dynamics and applied to the character using the simulator for f within the physical model to produce the pose and contact information for the next frame. The resulting pose and contact information are returned back as a feedback input for simulation and control in the next iteration of the loop. We define contact information as an enumeration of all contact points of the character with the environment together with the contact forces that act at them.

A particular motion produced by the simulation and control loop is determined by the initial pose of the character, the parameters of the controller, and the physical dynamics. The controller parameters encode the state machine and actions within the machine. Using the loop, one set of controller parameters may encode a controller that produces walking while another set would define a controller for running. Note that in this particular case, both walking and running can be represented by the same state machine with the same number of states, except that actions within the machines are different and specific to either walking or running.

1.4.3.3 Humanoid Character Model

In order to make our methods computationally tractable, we use a *humanoid character* model similar to character models commonly used in computer graphics and humanoid robotics. This model approximates

[2]We estimate initial contact information from the geometry of the environment and the initial pose of the character using collision detection within the simulator.

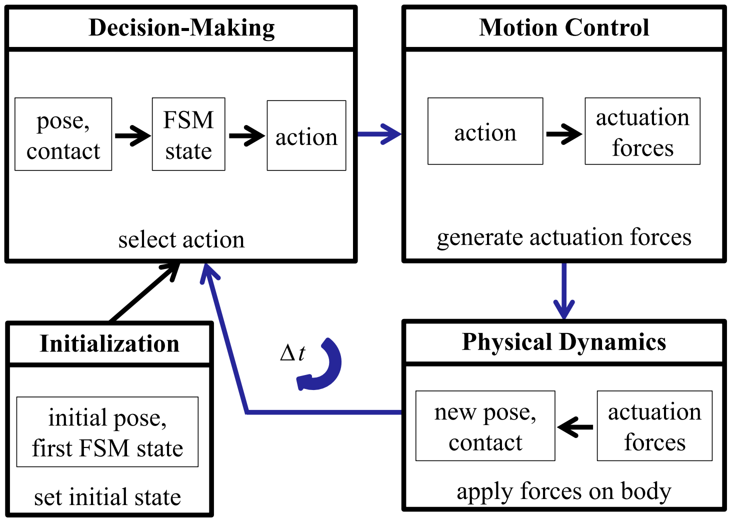


Figure 1.10: **Simulation and Control Loop.** Illustration of the generative model of character motion. The motion of the character is produced from an initial state by taking iterations in the loop. Each iteration advances the model state by physical time Δt from the current frame to the next by (1) selecting an action for the character to perform (decision-making), (2) generating actuation forces to perform this action (motion control) and (3) applying the actuation forces to the character model to update the pose and retrieve contact information for the pose (physical dynamics). Actions encode actuation "programs", parameterized by a desired pose that the character attempts to reach when the action is active and parameters used to set up the control laws to do so, taking scene geometry into account. Actions are selected by the decision-making process in the controller, considering the current pose of the character, contact information, current controller state and controller parameters. The updated pose and the corresponding contact information are looped back to the decision-making process to help select the action for the next frame.

human dynamics by using an articulated rigid body model for the body and low-level motion control by motors at joints to provide actuation forces for the body (see Figure 1.4).

The *rigid body model of the body* consists of a set of rigid body links connected by joints. Rigid body links represent different parts of the human body such as a head, torso or legs. The low-level motion control process within the controller powers the motors integrated with the joints and directly controls the motion of the connected body parts, by allowing them to rotate about actuated motor axes. Actuation forces are generated as joint torques using the motors. With these approximations of the human dynamics, simulation of the character in the physical model can be realized efficiently by off-the-shelf rigid body dynamics simulators while the low-level motion control using joint motors can be implemented by techniques based on *e.g.*, inverse dynamics. We discuss this particular physical model in depth in Chapter 3. Now that we have set our physical model, we will use $\mathbf{F}^{actuation}$ in place of $\mathbf{u}$ to refer to the actuation forces.

Because we assume that the model for the character and environment is given and fixed, physical dynamics of the character within the physical model is fully determined. Consequently, we can re-parameterize

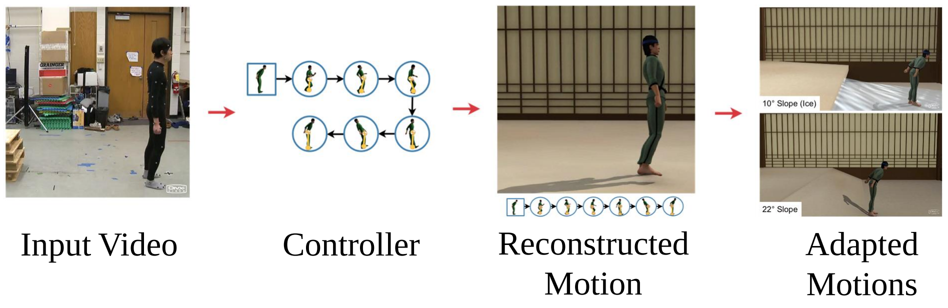


Figure 1.11: **Our Pipeline.** Our pipeline takes as an input a monocular video of a person performing a motion (left) and estimates a controller for the activity. The controller that we estimate implicitly captures the motion of the person through simulation; it generates the poses by computing actuation forces for the body and integrating them over time. Consequently, the produced motion meets our definition of physical plausibility from Section 1.4.1. Furthermore, because the forces result from reactions to various events, such as a change of foot contact (see Section 1.4.3.1), the estimated control mechanisms can be applied to simulated characters in new situations, as motivated in Section 1.3.2, such as when *e.g.*, these events happen at different time instants. For instance, we may modify the slope of the ground plane or modify frictional properties of the plane and see what the motion produced by the controller would look like in this new environment (right).

motions of the character using controllers (see Figure 1.7). We utilize this fact for addressing motion estimation in Section 1.5.

1.5 Approach Overview

We address the motion estimation problem defined in our problem statement by implementing an estimation pipeline from Figure 1.11 that adheres to the goals set in Section 1.3. This pipeline not only captures 3D poses of subjects, as is done with current motion capture systems, but it derives physics-based controllers for simulated characters that synthesize these poses in simulation. The representation of motions by parameterized controllers reduces motion estimation to the estimation of controllers:

Assuming the character model, its initial pose and the environment are known, we accomplish our goal and reconstruct the motion in the video by *matching parameters of the controller model* to the poses in the video such that the observed behavior is reproduced in simulation (see Figure 1.7).

Pipeline. The proposed pipeline takes as an input a video of a person performing an activity and estimates a controller for the observed behavior (see Figure 1.11). The estimation process does not rely on any auxiliary instrumentation, such as markers, and as such, our subjects can wear regular street clothes. In addition, the estimation can take place in unrestricted outdoor environments, as discussed in Section 1.2.2, and the controller provides motion generalizations for new environments (see Figure 1.11, right).

1.5.1 Motion Estimation

We realize motion estimation as an initialization followed by controller optimization. In order to implement these steps, we need a physics-based model for the simulated character and an optimization mechanism. The simulation works as a tool for the optimization and we use a standard model that we describe in Chapter 3 for completeness. For inference, we explore incremental optimizations by either *Particle Filter* or *Covariance Matrix Adaptation*.

Initialization. In *initialization*, we set up information that is needed for the modeling of the motion of the person. Towards that end, we manually build the model of the actuated character, the 3D model of the environment where the motion takes place and provide a rough estimate of the pose of the character in the first frame of the video. We also calibrate camera(s) that observe the scene so that we can hypothesize how the poses generated by the controller would look in images. We employ a generic character model from Chapter 3 that represents an average person.

Controller Optimization. We formulate controller estimation as an *optimization over actions in the controller*. We use an objective function for the optimization that attempts to make the synthesized motion produced by the controller appear as close to the input video as possible. The optimization consists of a joint (1) recovery of the state machine that segments the observed motion into actions separated by distinct physical events (*e.g.*, a change of foot contact) and (2) optimization of the encoding of the actions in the machine. The optimization process conceptually samples potential controllers to provide interpretation of the motion and uses the objective function to evaluate inconsistency of the controller hypotheses and the images.

Our objective function is formulated as a measure of the inconsistency between observed images and hypothesized synthetic images produced by the simulation with a proposed controller. We use a simple silhouette-based measure to evaluate the inconsistency, where we look at differences between observed silhouettes and hypothesized silhouettes generated by our model (see Figure 1.12). We assume the person is the only moving object in the environment and we apply background subtraction on the input images to recover the observed silhouette of the person for every frame in the sequence. We then measure the amount of overlap between the observed silhouettes and the hypothesized silhouettes, where hypothesized silhouettes are obtained by projecting the 3D geometry of the posed character from the simulation into the image using the camera projection. We use the control loop from Section 1.4.3.2 to produce poses for a particular control hypothesis.

Actions and Controllers. In implementing the proposed controller optimization approach, we have two options for modeling actions and the action selection process in the controller. For a start, we can either use controllers that utilize a trivial selection process that switches actions with every frame, or we can employ a more advanced model that attempts to switch actions intelligently, based on contact events and the underlying structure of the desired motion. We explore each of these two options for completeness, but entirely focus on the second more informed process, as motivated in Section 1.3.

When actions switch with every frame, actions generate a sequence of desired poses for the low-level motion control process that is *unstructured* and parameterized in a *dense* way, such that an explicit desired pose is provided for every frame in the sequence. The state machine for the selection of actions is implicit

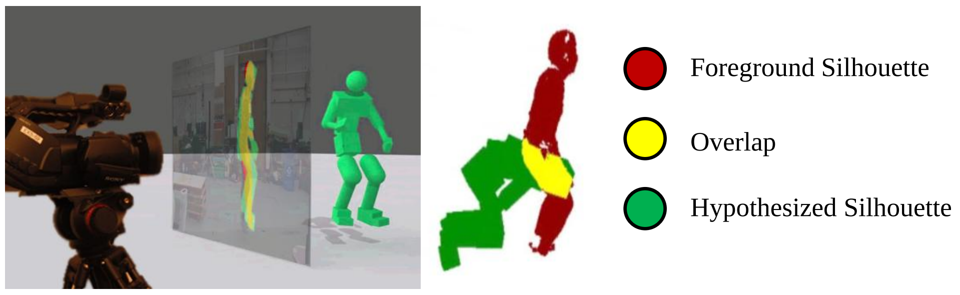


Figure 1.12: **Silhouette-based Measure of Motion Consistency.** We project hypothesized 3D poses of the model produced by the simulation through the camera to obtain hypothesized silhouettes (in green). We measure the consistency of the motion produced by the simulation as the amount of overlap (in yellow) between the hypothesized silhouettes (in green) and observed silhouettes (in red) obtained by background subtraction. We use the measure as an objective value for controller optimization.

in this case and consists of a chain of states for the frames in the sequence with the transitions happening regularly based on the time between two frames. The control problem is then equivalent to the tracking of the sequence of the desired poses using a *trajectory tracking controller* and the controller estimation reduces to the estimation of the dense desired pose trajectory for this controller. We study this modeling option in Chapter 4, where we recover such dense trajectories incrementally using a *Particle Filter*.

Alternatively, and according to our goals in Section 1.3, we can use actions that last for numbers of frames and that switch based on rules explicitly encoded by a state machine. With this option, desired poses in the sequence remain constant for extended periods of time and, as such, the parameterization of the desired pose trajectory for the low-level motion control is *structured* and *sparse*. Consequently, the representation of the controller using the state machine is *low-dimensional* and *compact*. By estimating the controller as a finite state machine, we force the estimation process to attempt to understand the underlying physical process that generates the observed behavior, learn the structure of the behavior and represent the entire motion in terms of this structure. In particular, the estimation process has to learn the phases of the motion, physical events that cause these phases to switch and the control strategies that reproduce these phases in simulation. We recover these compact controllers using an incremental stochastic optimization over an expanding window using the *Covariance Matrix Adaptation* (CMA) algorithm [67] in Chapter 5.

We briefly compare the two possible parameterizations of desired motion in Figure 1.13 and the corresponding controller estimation approaches in Table 1.1. At the highest level of abstraction, these two approaches differ in the parameterization of the desired pose trajectory and the level of understanding of the desired motion to be produced by the controller. At the low level, they differ in the optimization techniques they used to recover the desired pose trajectory and the motion, as well as the guarantees they provide on the produced motion reconstructions.

(a)

(b)

(c)

(d)

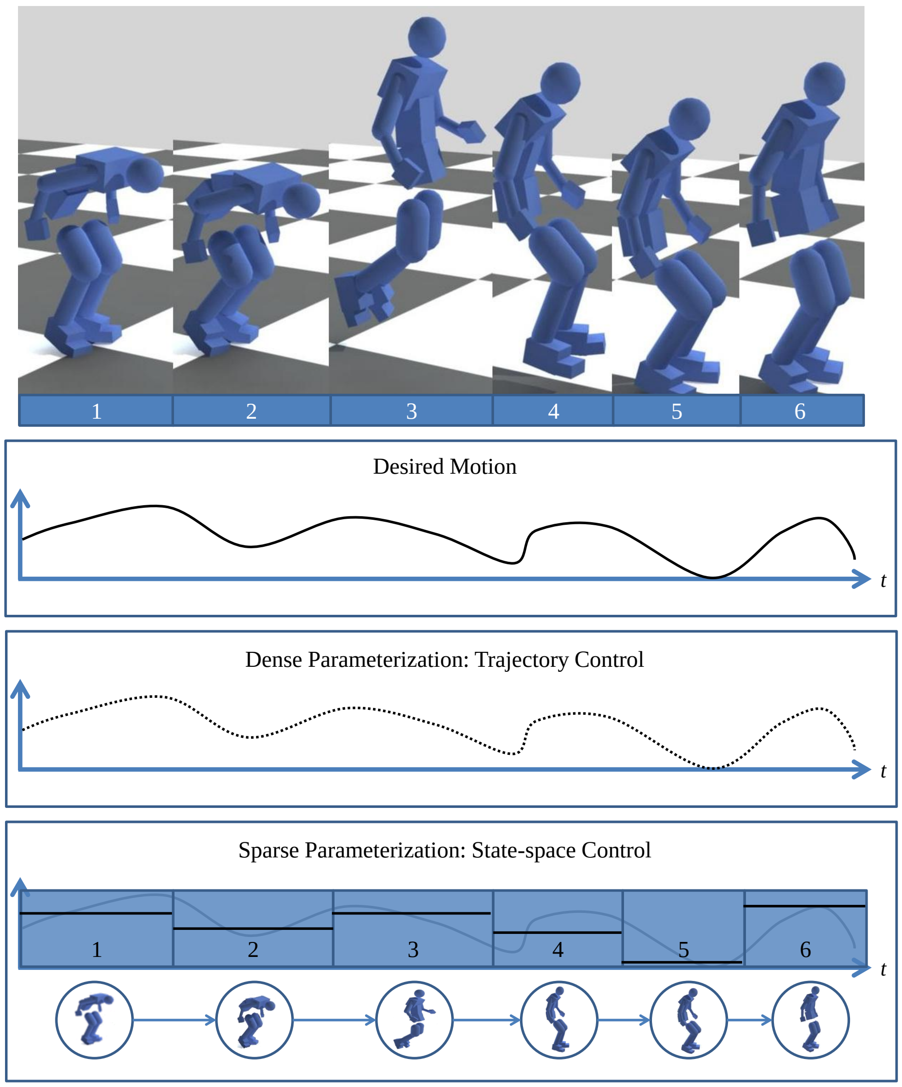

1
2
3
4
5
6
Desired Motion
t
Dense Parameterization: Trajectory Control
t
Sparse Parameterization: State-space Control
1
2
3
4
5
6
t

Figure 1.13: **Desired Motion at Different Levels of Abstraction.** We illustrate a jump at two different levels of abstraction. At a high level, a jump can be intuitively characterized as a sequence of six motion phases: compression (1), pushing off the ground (2), ascending (3), descending (4), loading (5) and unloading (6), visualized in (a). A particular jump instance can be characterized as a sequence of desired poses (b). In order to reproduce the jump by a controller, the low-level motion control process within the controller needs to compute actuation forces for the character such that the character will track the desired poses in the sequence. Without an understanding of the high-level structure of the motion, a trajectory tracking controller will store the desired pose for every frame in the sequence and attempt to track this sequence as closely as possible (see (c) and Chapter 4). Alternatively, the structure of the motion can be represented in the controller and desired poses can be specified for only specific key frames associated with the motion phases from (a) and replicated for the remaining frames in the same phase. Here, motion phases correspond to actions in a state-space controller (see (d) and Chapter 5).

	Chapter 4		Chapter 5
Decision-making Process	trajectory control		state-space control
Low-level Motion Control	constraint-base servoing		constraint-based servoing
Feedback	ground impact		contact and balance
Action Duration	one frame		many frames
Desired Pose Trajectory	dense		sparse
Desired Motion Prior	motion capture trajectories		pose-space PCA
Method	Section 4.2.4.1	Section 4.2.4.2	Section 5.5
Inference Over	desired poses	poses and desired poses	state machine
Motion Reconstruction	by simulation	biased towards simulation	by simulation
Reconstruction Accuracy	low	high	high
Adaptation to Environments	limited	none	through control feedback

Table 1.1: **Our Methods.** Illustration of the two controller models and controller estimation approaches that we discuss in our work.

1.5.2 Why Not Direct Force Estimation?

In order to address physical plausibility in motion capture, understanding of control mechanisms is not strictly necessary. Instead, one may attempt to directly estimate forces that act on the body from image observations and use them to re-synthesize the poses in the sequence. We argue this alternative approach is much harder to accomplish, is less general than the controller capture that we propose and loses the ability to generalize.

First, actuation forces result from an internal control process that actively actuates the body. Understanding of this process (causes for motion) not only helps regularize motion capture and allows to generalize, as discussed in Section 1.3.2, but also provides more fundamental representation of motion. Forces, on the other hand, are the result of this process, are specific to a particular motion execution and the environment subject to the time of capture and can not be generalized easily.

Second, generating and matching of force hypotheses to images results in a very difficult problem that requires direct modeling of forces in every frame of the sequence and the changes in forces between the frames. This modeling is difficult because too many variables have to be estimated for each frame from relatively little information: actuation forces are high-dimensional, because every joint angle in the body needs be potentially actuated, discontinuous on body contact change, and unstable, such that they change drastically between frames even when the body is almost still.

1.6 Challenges

We are addressing two difficult problems: vision-based motion capture and controller estimation. In doing so, we face challenges that can all be attributed to the fact that we attempt to (1) build a physics-based model of full-body 3D human motion that can produce all possible behaviors of interest in simulation and (2) we want to fit this model to a single view video.

In order to meet the objective of this thesis and realize motion capture through controller capture, we need to build a character model for the person that can be simulated and controlled efficiently and that uses compact high-level encodings of “actuation programs” for control. In building the model, we need to balance the following aspects: generality of the model, effectiveness of simulation and control, and abstraction of desired motion for control. In balancing these aspects, we also need to consider effects of visual perception that affect control inference.

Character Model. The model that we build needs to be general and realistic enough so it can generate the poses we wish to track. In doing so, the model needs to represent articulation of the human body with a sufficient level of detail and with a sufficient fidelity. In modeling the motion of the person, the model needs to account for compliance of body parts with the environment through contact, response to gravity and inertia and provide a means for body actuation.

While simpler low-dimensional body models are easier to build, simulate and control, they may be restricted to only specific classes of motions (*e.g.*, walking). For example, a model without arms will not be able to produce a hand stand and a model with abstracted legs without the parts for feet will have difficulty locomoting in a physically plausible way. More general full-body 3D models, on the other hand, abstract the human body with a sufficient level of detail, but require deliberate control and balancing strategies to produce desired self-balanced motions, are more difficult to build and more expensive to simulate. Control of these detailed models is challenging, because the models have many degrees of freedom to control and the body is subjected to the effects of gravity and contact with the environment. These effects are caused by factors external to the body, but they have to be anticipated and accounted for in control of the body. In order to keep the body balanced, the global position and orientation of the body, important for balance, has to be controlled, but it can only be affected indirectly, by controlling relative orientations of parts of the body. Typical models of the human body have 40 or more degrees of freedom and all of them have to be controlled (through low-level motion control) in order to achieve the desired coordination of the body.

Controller Model. In order to enable automatic estimation of controllers from video, we need to use a universal controller model that is not restricted to a particular activity or a motion style. This requirement makes control difficult, because it implies that (1) motion-specific strategies previously used in computer graphics and robotics can not be used for control, (2) the complete behavior of the controller has to be encoded using model parameters, and (3) this parameterization of the desired motion has to be rich enough so that all potential motions can be generated. For example, the same controller model must be able to generate walking, limping or gymnastics. Furthermore, the encodings of the motions have to be compact such that they can be estimated automatically from noisy image observations.

In designing a model for the controller, it is important to find a good tradeoff between the detailed understanding of the underlying processes that actuate the body, the complexity of the representation of the desired motion, and costs for simulation and controller estimation. For example, a controller that is given a desired pose of the character for every frame in the sequence may compute actuation forces using optimization, but it requires the poses to be already compliant with the environment and the dynamics of the character. Such a representation also results in a high-dimensional encoding of the desired motion that is difficult to estimate from video and limits generalization. On the other hand, structured higher-level representations of desired motion provide a more detailed understanding of the motion and an ability to generalize, but require complex inference. Our controllers fall within this second category.

Our controllers explicitly model the structure of the desired motion and exploit this structure for control, as explained in Section 1.5.1. Because the structure of the motion can not be assumed, we need to jointly optimize over the structure of the motion as well as the control mechanisms that reproduce the motion. The controller estimation results in a high-dimensional mixed optimization over discrete states in the state machine within the controller and continuous parameters of actions within the states. The optimization is difficult, because long motions consist of many phases and strategies (with many parameters) have to be found for all of them.

Estimation of Motion Structure, Contact. Estimation of motion structure involves segmentation of the observed motion into actions based on the coherence of motion phases and contact events that separate them. In segmenting the motion to actions, the segmentation needs to detect when contact events happen and reason about foot support. Contact and foot support is very difficult to measure but also critical for control, because the changes of it imply discontinuity in the dynamics. Reasoning about these properties is problematic because it requires hypothesizing about the pose and the actuation forces that currently act on the body. However, actuation forces and poses result from control. Furthermore, estimating contact and foot support alone is a difficult problem even if we already know an exact 3D pose reconstruction of the person for every frame in the sequence, because it still requires knowledge of the actuation forces. To address these challenges, we estimate poses, contact, foot support and control jointly and together with the estimation of the structure, and directly from the video observations.

Estimation of Control Mechanisms. Estimation of control mechanisms for actions consists of the search for continuous parameters within actions that give rise to the motion. This search is difficult because the strategies need to encode low-level motion control of the body and this encoding has to be versatile such that different motions can be produced. Consequently, the space of the strategies is high-dimensional and the search for the best strategy gets often stuck in local optima, resulting in the character failing to complete the motion (*e.g.*, the character falls) or exploiting a strategy that does not appear human-like. This search for strategies is further complicated by the fact that the strategies are global, affecting the entire motion, and cannot be estimated from only a few frames of the video. The estimated strategies need to properly respond to discontinuities in motion due to contact of the body with the environment, counteract effects of gravity such that the body stays balanced and produce actuation forces that stay within specified bounds. When setting force bounds for the character model, it is critical to find an effective compromise between loose bounds that may produce motions that look overpowered and super-human and more realistic bounds that, in our case, result in no solutions being found.

Visual Perception. Because we take monocular observations as an input, we need to deal with ambiguities due to (1) self occlusions, where body parts obstruct the view of other parts, (2) lack of explicit depth information, where relative ordering of body parts by their distance from the camera is unknown, (3) the change of appearance of the body as the person moves (*e.g.*, due to body shape deformation and clothing), and (4) observation noise, such that we are never able to perceive the state of the body accurately. Motion capture through controller estimation is particularly challenging in this setting, because all reasoning about the state, contact (as required by our formulation) and control of the character has to be done purely based on the appearance of the person in the images. For example, in the single view case, almost half of the body remains unobserved and the pose of the occluded parts can only be determined from the motion of the visible parts, history of past images and prior knowledge (experience).

In our work, we simplify the perception problem by assuming that the person always moves within the field of view of the camera and that the view of the body is never obstructed by other objects. We also use relatively simple models of the body appearance, shape and clothing, based on silhouettes. Despite the simplicity of our observations, we illustrate that by using physical simulation and control we are capable of recovering motions that are physically plausible and reproduce the observed behaviors. We leave the use of more effective appearance models for future work.

1.7 Contributions

In this work, we have developed methods for addressing physical plausibility in estimating human motion from video. We illustrate that physically plausible motions can be estimated directly from image sequences by using physical models of human dynamics and methods for humanoid control. We summarize the key contributions below:

- We introduce a definition for the physically plausible human motion estimation problem based on capturing and estimating the dynamics of the human body and control mechanisms observed from video,
- We propose methods for Particle Filtering with physically plausible motion predictions to improve the accuracy and physical plausibility of human motion estimation (Chapter 4),
- We propose methods to capture control state machines from human motion observed in video in order to recover a simulated character model that reproduces the motion (Chapter 5).

1.8 Thesis Outline

Chapter 1. *Introduction.* This chapter introduces the problem and motivates the formulations and techniques used in this work. Challenges, contributions and outline of the structure of this thesis are discussed as well.

Chapter 2. *Background.* This chapter reviews existing approaches for motion estimation in computer vision, techniques for generating physically plausible motion in computer graphics and methods for humanoid control in computer graphics/robotics.

Chapter 3. *Modeling Human Body in Dynamics.* This chapter introduces a physical model of the person that we use throughout this work. The chapter reviews concepts for modeling rigid body dynamics and illustrates how we use these concepts as a tool to build an articulated actuated model of the person for our methods.

Chapter 4. *Physics-based Particle Filtering.* This chapter describes our basic approach towards addressing physical plausibility in motion estimation. In this approach, we use a Particle Filter to directly construct the desired pose trajectory for the character model and use a trajectory tracking controller to perform and validate the proposed trajectories against image observations.

Chapter 5. *Controller Capture*. A desired pose trajectory can only be reliably estimated by a Particle Filter in presence of strong likelihoods. Rather than focusing on employing stronger observation models, we relax the assumptions of strong likelihoods and approximate the desired pose trajectory as a sequence of sparse actions selected by a state machine. We then optimize in a batch or incrementally over the structure of the machine and controller parameters in order to find a physically plausible interpretation of the entire motion.

Chapter 6. *Summary and Discussion*. This chapter summarizes the developed methods and proposes directions for future research.

Chapter 2

Background

Marker-less motion capture is a challenging problem, particularly when only monocular video is available. We estimate physically plausible three-dimensional motion of a person from monocular video by recovering a controller capable of generating the observed motion and replaying this motion in other environments and under physical perturbations. Our work relates to a number of distinct disciplines in computer science: pose and motion estimation in computer vision, synthesis of physically realistic human motion in computer graphics, methods for humanoid simulation and control in robotics, and controller learning from demonstration in artificial intelligence. In addressing vision-based motion capture, we use techniques from computer vision to analyze images and provide constraints for controller estimation. In turn, we use methods from computer graphics, robotics and stochastic optimization to estimate a controller that matches the constraints and produces a physically plausible motion interpretation of the observed motion using simulation.

The problems of motion estimation and humanoid control have been studied extensively in isolation in the computer vision, robotics, and computer graphics literature. Despite their research history, both remain open and challenging problems. Approaches that utilize physics-based constraints in estimation of human motion from video have been scarce to date (see Section 2.1.5.6) or required user intervention (see Section 2.3.4). Physics-based controllers for humanoid characters have been explored in computer graphics and robotics (see Section 2.3.5). The controllers often rely on motion capture data, *e.g.*, as a *target* trajectory for the desired controller behavior. However, motion capture data often needs to be modified or rectified in non-trivial ways beforehand to make it physically correct or a good match to a dynamic model of the character. In our formulation, we combine these two problems. Casting the video-based motion capture problem as one of direct estimation of underlying character control leads to the following two benefits: (1) the physics-based character controller can serve as an effective motion prior for estimating human motion with elements of physical realism from weak video-based observations, and (2) the recovered controller can be used to directly simulate dynamic and responsive virtual characters without any intermediate processing or rectification. To our knowledge, there is no previous work on estimating full-body controllers from video directly. In artificial intelligence, methods for learning controllers from demonstration have been developed, but they have only been applied to simpler low-dimensional control problems different from ours (see Section 2.2).

Our goal in this chapter is to provide a review of the state-of-the art in motion estimation through frame-to-frame pose tracking (see Section 2.1), realistic human motion synthesis in graphics (see Section 2.3) and controller learning in artificial intelligence (see Section 2.2). We start our review with the discussion of relevant approaches to motion estimation in computer vision. Readers are recommended surveys [121, 122, 139] by Moeslund *et al.*and Poppe *et al.*for a comprehensive review of motion estimation literature covering research from 1980 till 2006.

2.1 Motion and Pose Estimation

The problem of estimating articulated human motion from video has been an active topic of research in computer vision for over two last decades. Availability of high-performance computing and development of new algorithms has made significant advances towards solving the problem possible and has enabled current applications in surveillance, human-computer interaction and entertainment. Although substantial progress has been made, the current approaches to motion estimation still rely on many simplifying assumptions and have difficulty estimating motions that are smooth and consistent with physical laws.

The general problem of recovering the smooth 3D motion of a person in typical environments from monocular image observations remains largely unsolved. Most of the issues that have to be overcome are due to (1) the high dimensionality of the human pose, (2) subtleties of the motions with which humans move and (3) variability in imaging conditions, human appearance and clothing. As such, most previous approaches have concentrated on modeling aspects of the imaging process through which humans can be perceived and on developing search methods and motion models (based on statistical principles) that allow for efficient motion inference. However, in doing so, relatively little attention was paid to the quality of the recovered motions. As a result, many existing methods produce motions that violate constraints imposed on the body by the world or environment, producing visually distinct and physically implausible artifacts, including foot skate, out-of-plane rotations and jitter. While such results may be adequate for specific needs in *e.g.*, surveillance or activity recognition, many other applications can benefit from more accurate and physically plausible motion estimation. Our goal in this work is to pursue a formulation for human motion estimation that produces physically plausible motion interpretations.

A goal of any motion estimation method is to recover the motion of the person observed in the input images. This motion is characterized by a sequence of high-dimensional vectors that encode the poses that the person assumes in the individual image frames. While many pose sequences can produce a good fit to the images, only a small portion of all such sequences is physically plausible. Restricting (or biasing) the search for motions to physically plausible interpretations, that satisfy the laws of physics, seems to be one of the ways towards recovering smooth and naturally looking motions. Towards that end, our goal in this thesis is to build a motion estimation system that can take advantage of physical simulations of the model of the person in order to produce such plausible interpretations.

2.1.1 Methodology Overview

Traditional Pipeline. Motion estimation is typically realized as a cascade of operations consisting of model initialization, pose estimation in the initial frame and *frame-to-frame pose tracking*. For increased robustness against ambiguities in inference, motion estimation is most often formulated probabilistically.

The purpose of model *initialization* is to gather information required to bootstrap tracking. For example, an initial model of the person that encodes the pose, shape and appearance of the person at the first frame may be established. Next, the motion of the person is *tracked* between frames, which means that the initial model is updated through time in order to characterize poses in the remaining frames. Tracking consists of extracting meaningful information from the images that leads to the *estimation of poses* in the individual frames. This information is typically propagated temporally, so that the search for poses in the frames can be focused at relevant portions of the pose space and correspondences of estimated poses in consecutive frames can be established. In specific cases, when information from all image frames is available, the entire motion can be estimated as a whole and tracking and pose estimation processes are fused. Tracking can be followed by motion analysis and recognition, depending on the desired application.

Our Pipeline. Our formulation for motion estimation is substantially different from the traditional pipeline. Our pipeline is not based on frame-to-frame pose tracking, produces different outputs, uses different techniques to estimate the poses in the video, and provides a guarantee of physical plausibility for the motion reconstructions and mechanisms for motion adaptation to new environments. We characterize our goals and pipeline in full in Section 1.3, Section 1.5.1 and Section 2.1.5.10.

For the purposes of this review, we focus on kinematic pose tracking, implemented by the traditional pipeline, because this is the paradigm for motion estimation that has been explored in the literature thus far.

2.1.2 Body Model

In order to implement a pose tracking system, we need to define a model for the human body. At the very least, the body model needs to represent *kinematic properties* of the body including *joint articulation* and *pose*. Joint articulation encodes parts of the body and specifies how the parts connect to each other at joints. The pose of the body describes 3D or 2D locations of the parts and joints in the articulation and locates the parts in the image. In addition, *shape* and *appearance* of the body ("flesh") may need to be represented in the model. This additional information may be provided for two reasons. First, it can characterize the body with a greater fidelity which can be used for visualization purposes and can be beneficial for applications in *e.g.*, human-computer interaction and entertainment. Second, this detailed representation can facilitate more accurate pose inference, because the detailed shape and appearance information can simplify matching of the pose to the image. We next consider how the kinematic properties of the body can be represented in the body model.

2.1.2.1 Kinematic Models

Kinematic models encode kinematic properties within the body model and characterize the body pose. Most often, human bodies are modeled as three-dimensional or two-dimensional articulated structures with parts connected by joints.

Kinematic Tree. Representations by *kinematic trees* [115, 11, 46] model parts as nodes in trees (called "skeletons") and parameterize the configuration of the articulated body recursively, utilizing hierarchies defined by the trees. A body pose is encoded using the tree by the position and orientation of the root part in the reference space and orientations of the child parts relative to their parent parts in the tree. This representation can be defined for parts in both 2D and 3D. In 3D, the position and orientation of the root part is specified in the world space[1] and orientations of the remaining parts are defined as rotations in the spaces of the parents. General orientations in 3D can be expressed in many ways, using rotation matrices, Euler angles, axes-and-angles, unit quaternions [156] or exponential maps [59, 24]. Some joints, such as a knee or a elbow, permit only rotations about specific axes; rotations for those joints are then expressed as angles about those axes. In 2D, the position of the root part is defined directly in the image plane and rotations are modeled by 1D angles.

Parts. The recursive representation by a kinematic tree implicitly rules out many impossible poses. For example, parts can not be torn apart and elbows can not twist or bend backwards, if the model is equipped with joint angle limits. On the other hand, because such constraints on the pose are encoded implicitly in the recursive parameterization of the pose, parameter values are coupled and correlated both spatially and temporally in a complex way. All these aspects complicate pose inference and tracking.

As a popular alternative to the kinematic tree representation, body can be modeled as a collection of *independent parts*. In this representation, parts are parameterized independently such that the configuration of one part is given by its absolute position and orientation in the reference space, independently of the configuration of another. This representation is redundant and consistency of the pose (*e.g.*, connectivity of parts at joints) has to be enforced explicitly. However, because such an enforcement can take place later in the pose estimation process, pose recovery and tracking can be made easier. Specifically, part-based representations facilitate bottom-up pose reconstructions where parts are searched for in the images independently first and only assembled later to produce consistent pose interpretations. Part-based models can be used both in 3D and 2D.

In our simulations, we represent the body of the person as a collection of independent parts. We constrain the parts by articulation constraints in simulations such that the parts remain connected during pose updates.

2.1.2.2 Shape Models and Appearance Models

In order to detect the pose of the body in an image, it is often necessary to characterize the shape of the parts and their pixel-wise appearance in the image plane. This information may be used to predict how a hypothesized pose may look like in the image and to determine consistency with the image. It can also be used to look for parts of the body in the image directly.

[1]In special cases, only orientation of the root part with respect to the camera is encoded.

These imaging aspects are usually encoded using the models of *body shape* and *appearance*. These encodings can be either quantitative, such that the exact shapes and pixel values are stored, or qualitative, using statistics. For many applications, reconstruction of the pose is not the final goal and knowledge of the exact shape or appearance may be required. Towards that end, *quantitative models* maintain an explicit geometric representation of the shape and texture for the body model, such that it is possible to generate an image of the model from a given view. *Quantitative* models, on the other hand, do not record exact geometric and textural representations but provide qualitative information that characterizes the shape and appearance in one way or another. Qualitative representations of shape are often fused with qualitative representations of appearance such that only statistics over pixel values for the body need be stored. Shape and appearance models take different forms and can represent both 3D and 2D information.

3D Models. Shape of the human body in 3D has traditionally been modeled as a collection of *rigid parts*. In the context of computer vision, simple parametric representations by geometric primitive,s such as truncated cones [46, 162, 157], cylinders [160, 158], ellipsoids [24] or superquadrics [58], have been explored due to their simplicity and ease of use. More detailed rigid polygonal models [99] have been studied as well.

Despite the successes, it has been recently argued that rigid representations of shape do not facilitate accurate pose inference. Specifically, fine details in shape such as a stretching of skin about joints or bulging of muscles can not be expressed well using a rigid model, suggesting image evidence can not be explained fully using these models. Consequently, more complex and realistic *deformable models of the entire body* [69, 172, 57, 15, 14] that are capable of modeling subtleties of the human shape have been explored. These detailed models were originally used in the context of image synthesis in computer animation. In vision, they have been used to enable joint estimation of the 3D polygonal surface of the body and the pose. In its most simplistic form polygonal models were used as passive templates [15] that replaced rigid parts with a smooth polygonal surface that deformed with the motion of the underlying skeleton (skinning). In Starck *et al.*[172], the mesh model was actively deformed and vertices in the model moved through optimization in order to explain image evidence the best. The data-driven models of Balan *et al.*[14] and Guan *et al.*[66] make this optimization more tractable by ignoring clothing and optimizing in a low-dimensional statistical space of "naked" body shapes conditioned on the pose. In order to consider clothing and to explicitly model small scale deformations of the skin and garment, Gall *et al.*[57] optimizes for an offset from a deformed mesh produced by the skinning method of Kavan *et al.*[82]. This particular shape model is more general and is not limited to humans. Specifically, the model has been illustrated to be capable of fitting surface of animals as well.

In the context of 3D pose estimation and tracking, relatively simple appearance models of the body shape have been used. In most cases, shapes have been modeled as solids in the image plane that are optimized to cover a foreground blob obtained by foreground segmentation [14] or whose outlines are to coincide with edges in the image [46]. Sometimes, statistical features, computed over local image patches centered at detected keypoints in the image, are used to establish correspondences between keypoints in different views and frames [69, 172, 173] and use them as soft constraints for the reconstruction. With that said, more elaborate models of 3D surface appearance are usually not built or are only used for visualization purposes [69, 172, 173] and not to aid pose inference. A notable exception to this common practice are the works of Sidenbladh *et al.*[160, 158] and, more recently, Balan *et al.*[12] and Guan *et al.*[66]. Sidenbladh *et al.*proposes a generative texture-based model of part appearance where 2D pixel-wise texture of individual parts is maintained by the appearance model and mapped onto the body surface in order to verify consistency of future pose hypotheses with images. In Balan *et al.*and Guan *et al.*, surface appearance is modeled as a lit textured surface. In their case, the estimated texture records diffuse reflectivity of the points on the surface (albedo) and the final shaded appearance of the body is obtained by rendering, considering the estimated position of the light source. It was then demonstrated that such a representation of appearance facilitates inference of both the shape as well as the pose of the person.

2D Models. As a computationally more attractive alternative to 3D models, shape and appearance of the human body have also been modeled directly in 2D. Two-dimensional models are particularly popular because they work with images with clutter, in presence of unknown backgrounds, occlusion and under the effects of lighting artifacts, substantial pose changes and across many subjects, [6].

In 2D, body parts are most often modeled as rectangles [72, 107] or general planar patches [79] that can rotate [72, 107], scale [4] and/or skew [51, 163] in order to accommodate the 2D body pose. These shape models are typically integrated with appearance models to characterize pixel values within the parts. The combined models are then used to help localize the parts in the image. In the early work of Ju *et al.*[79], textures of planar parts were captured in the initial frame and optical flow was used to predict the locations of the parts at later frames based on the pixel correspondence. More recently, however, general *qualitative appearance models* [51, 211] became widely used. This adoption was mostly due to the fact that qualitative appearance models could use discriminative statistics that could capture how the parts stand out from the background as opposed to how they look like exactly, and the used statistics were robust to changes in pixel values in presence of subject, clothing and illumination changes. Furthermore, parameters of these models could be learned automatically from annotated training images [51, 6, 211] and the models could be automatically adapted [142] to observed subjects during pose estimation. Lastly, these representations allowed for efficient inference [163] of 2D pose.

When accurate recovery of 2D shape is required (*e.g.*, so that the person can be segmented from the background), the representation of parts by rectangles is too crude. In order to model body outline more faithfully, it has been recently proposed to explicitly represent the shape of the entire person using a *deformable shape model* [55, 65]. In *contour person* model of Freifeld *et al.*[55], shape of a “naked” person is modeled as a single closed contour using a 2D version of the statistical 3D shape model from Balan *et al.*[14]. Similarly to the 3D naked model [14], this 2D model is capable of capturing effects of local non-rigid deformation of the skin in response to the motion of the underlying 2D kinematic model as well as to model variability of body shape across population. This model was recently extended by Guan *et al.*[65] to explicitly model shape deformation due to clothing, allowing the shape model to fit image observations better. Contour shape models have so far been combined with simple appearance models: contours should appear as edges in the image and statistics of pixel intensities and colors inside and outside of the contour should differ [55], making the segmented shape stand our from the background.

In our work, we represent parts in the body using 3D rigid geometric primitives. We model the appearance of the parts as solid regions in the 2D image plane that we obtain by projecting the 3D geometry through the calibrated camera(s). We now move onto the discussion of what information we can extract from images in order to enable matching of poses to images.

2.1.3 Image Observation Models

Images from the input video sequence constitute the only source of information for the vision-based motion estimation system and provide cues through which the person can be observed. This information is generally referred to as observations, image evidence or *image features*. Image features can take a number of distinct forms that are abstracted by a *feature model*.

Feature models quantitatively or qualitatively describe regions in the image(s) so that the pose, shape, appearance and motion of the person can be estimated. These descriptions characterize images at different levels of abstraction. At the lowest-level, individual pixels in the image are described. These low-level descriptions, or *low-level features*, are extracted directly from the images by applying simple operations at the individual pixels and provide information on *e.g.*color, edges, gradients or silhouettes of objects in the scene. Higher-level descriptions, abstracted by *image descriptors*, combine information from lower-level features in order to qualitatively describe larger regions in the image. Ideally, image descriptions should be invariant to noise and differences in lighting and imaging conditions so that images can be described robustly.

The particular choice of the feature model is typically dictated by the employed inference method and the associated models used to represent the pose, shape and appearance of the person. We list some of the most popular feature models next. These models are usually combined in practice in order to build more powerful models that can exploit multiple sources of information.

2.1.3.1 Low-level Features

Silhouettes. Many current vision-based systems for pose estimation and tracking start the analysis of an input image with explicit *segmentation* of foreground objects from the background so that further analysis can be focused only to the foreground objects of interest. During segmentation, a binary image mask is constructed to identify pixels in the image “foreground”. Assuming the person is the only foreground object in the scene, as is often the case in controlled environments, the foreground segmentation mask quantitatively describes the *silhouette* of the observed person.

Silhouettes are quite popular for pose estimation and tracking in computer vision because (1) silhouettes can provide powerful cues on both the pose and shape of the body of the person [81, 46, 14, 66], (2) can be directly matched to hypothesized silhouettes synthesized by the shape model with respect to a proposed pose [81, 44, 46], (3) silhouettes can be extracted from images quite reliably and efficiently in controlled environments and (4) knowledge of the silhouette mask allows to exclude irrelevant portions of the image from further processing.

Silhouettes of foreground objects have traditionally been obtained by background subtraction, that is, by thresholding differences of pixel values in the input image from the pixels in a known background image. In chroma-keying [42], pixels in the background image are all supposed to have a single color distinct from the clothing of the person, usually blue or green, so that extraction of the silhouette of the person can be implemented by color thresholding. For less controlled environments, such as outdoors, per-pixel statistical models of background pixel values can be learned. Foreground silhouettes can then be extracted by thresholding "distances" of the pixel values of the input image from the distributions of the background pixels. In most cases, these distributions are either modeled as static Gaussians [204] or adaptive mixtures of Gaussians [174]. The mixture of Gaussians model [174] is particularly appealing because its multiple modes can model repetitive changes in the background (*e.g.*, motion of leaves in the wind, rain, a construction flasher, *etc.*) and the modeled distributions of background pixels can adapt to slow changes in the environment (*e.g.*, day and night, placement and removal of objects to and from the scene, *etc.*).

With that said, accurate recovery of silhouettes in general environments remains a challenge, especially when objects move rapidly, change their appearance with pose and in presence of clutter, occlusions and strong lighting and imaging artifacts (*e.g.*, specularities, shadows, image blur). While more complex segmentation models have been proposed, more recent 2D methods [142, 52, 6, 211] for motion and pose estimation do not rely on explicit pose segmentation and estimate poses directly from unsegmented images, or estimate the pose and the segmentation of the person from the background simultaneously [124, 65].

Edges. Edges are another popular means of describing images at the low level. Edges correspond to the points where pixel values or gradients change discontinuously and thus localize important structural aspects in the image, including boundaries of objects, changes in surface properties or variations in scene illumination that may be relevant for further analysis of the image. Unlike silhouettes, edge features describe images locally and capture fine details of the shape, both at its boundary as well as in the interior. As such, they can be used to disambiguate poses that yield similar silhouettes, support more exact matching of shape boundaries to the actual outlines observed in the image [58] or accurately segment the person from the background [65]. Similarly to silhouettes, edges can be directly matched to proposed segment boundaries [71, 58, 46] and can be extracted from images efficiently.

Most often, edges are localized by edge detectors that look for excessive magnitudes of first-order image gradients or zero-crossings of second-order image derivatives. Because image derivatives are used for edge detection, edge features are usually robust to local changes in contrast and lighting, making them appropriate for modeling appearance templates for segments in the body model [142] and using them as building blocks for higher-level descriptions of images. For pose estimation, edge features are usually used in conjunction with other features (*e.g.*silhouettes, gradients or color information) so that edges can be matched to the shape model robustly in complex and cluttered backgrounds, where only a small fraction of detected edges is relevant for the estimation of the pose of the body.

Color. Images are also often characterized in terms of their pixel-wise color properties. Models of pixel colors, as discussed above in the context of background subtraction, have been used to discriminate foreground pixels from the background. Similar principles can be used to model color appearance of segments in the body and to segment parts from other parts. However, because color appearance varies over people and time significantly *e.g.*, due to clothing and lighting conditions, building color models that would be general is difficult. Consequently, learned models of pixel color have mostly only been used for *skin detection* [78], utilizing the fact that skin color tends to remain distinctive even in presence of lighting variations. With that in mind, Lee *et al.*[95] uses skin detection to help localize position of the head and hand segments in 2D. Other methods [204, 85, 135] learn color appearance of parts online from the input image, by clustering pixels with similar colors to compact *blobs*. In Park *et al.*[135], extracted blobs are associated with parts in the body and tracked over time while gracefully handling self-occlusions and occlusions by other tracked people.

More recently, statistical color models have been combined with other feature models in order to build qualitative appearance models for the parts in the body. These models are then used to verify hypotheses about the locations and shapes of the parts proposed by a pose inference mechanism. The models characterize foreground regions for the proposed parts together with adjacent background regions and measure how the proposed foreground regions stand out from the background, rather than what the regions look like exactly. In Roberts *et al.*[146], this technique is implemented in the form of probabilistic shape template masks. These masks are superimposed on the hypothesized parts in the image and color appearance models of the shape foreground, defined by a given mask, and adjacent background are learned from the pixel values in the masked foreground and background regions. The quality of the segment hypothesis is then scored based on the difference of the foreground and background appearances, assuming the appearance of the foreground should be significantly different from the background for a valid hypothesis. A similar idea is exploited by other methods [141, 77, 65]. In Johnson *et al.*[77], candidate segment locations are first proposed based on edge/gradient cues [41] and the appearance of the foreground is learned from the pixels along the medial axis of the hypothesized shape. Ramanan *et al.*[142] and Ferrari *et al.*[52] employ an iterative "parsing" technique, where segments are first estimated using a weak edge-based model and a non-informative appearance model. The initial estimation results are then used to update the foreground and background color models for the parts and the parts are re-estimated using the updated models, possibly repeating the process. Ferrari *et al.*[52] propagates cues and appearance models across multiple frames in order to re-estimate appearance and poses in TV movie sequences.

Texture. As an alternative to the representation by colors, appearance of parts can be described by encoding textural structures of the surfaces of the shapes of the parts. These structures can be modeled quantitatively as bitmaps that directly record pixel values [79, 69, 172, 173, 160, 158] (or albedo [14, 65]) mapped onto the 2D or 3D surface of the shape, or qualitatively using statistical models. In pose tracking, the bitmap-based descriptions are used as templates that are matched to pixel values in images in order to localize parts. The templates can either be captured before tracking [99] or estimated together with the shape and pose [160, 158, 14] during tracking. Statistical models of texture, on the other hand, describe the surface structure in terms of qualitative properties (*e.g.*, histograms) such that it is possible to discriminate whether a given region has the same texture/material as another and separate (segment) the two regions from each other in the image. Similarly to the statistical models of color appearance, statistical models of textural appearance can be used to discriminate between foreground segments and background regions and can be learned efficiently from images [6].

Textural appearances are most often modeled statistically as a set of responses of oriented Gaussian derivative filters of different orders, orientations and scales [114, 51]. Following bag-of-words approach for statistical classification, Malik *et al.*and Leung *et al.*[114, 100], quantize and match the responses of filters to learned prototypical responses called *textons* and characterize the texture of the given surface patch as a histogram over textons. For discrimination between two textural appearances, χ^2 difference of the two histograms is used. Mori *et al.*[124] uses this textural appearance analysis for clustering pixels into small coherent regions called *superpixels* [145] and assembles the superpixels to form parts of the body.

Optical Flow. Optical flow records relative pixel motion between frames in an image sequence. This relative motion can then be used to segment moving objects from the background, solve for motion of the parts in the body model [79] or used to validate hypotheses about velocities with which the parts in the body move [27] while estimating the motion of the person. With that said, recovering optical flow is itself a hard open problem, especially when images are corrupted with noise, there are many objects in the scene that move in different ways and occlude each other and when the lighting conditions and appearance of objects change.

We now look into more global image features that operate on a higher level of abstraction. Contrary to the lower-level features that only analyze edges, colors and textures, these higher-level features provide a more abstracted understanding of the image and the regions within the image and allow to *e.g.*, match one image region to a similar instance of the same region in a different image/view.

2.1.3.2 Image Descriptors

Image descriptors are higher-level image features that characterize regions of interest within images at a higher level of abstraction by aggregating information from lower-level features such as edges, color and texture structure. These descriptions are qualitative and are designed so that they can be computed robustly from incomplete and noisy image observations and under image perturbations.

There have been many descriptor types with different properties studied in the literature. Despite the differences, the goal of all types of descriptors is to provide a concise high-level descriptions of image regions that are distinctive and robust to image perturbations. In addition to these properties, many descriptions are also invariant to specific geometric and photometric transformations of the images and objects within the scene, enabling applications in *e.g.*, object recognition, surface matching, and image indexing, that could be realized by operating on these abstract descriptions of the images. Here, we list some of the most common image descriptors used for pose and motion estimation and refer the reader to the survey of Mikolajczyk *et al.*[119] for a more detailed information on image descriptors.

Histogram of Oriented Gradients (HOG). Histogram of Oriented Gradients [41] decomposes an entire image patch to smaller rectangular cells of the same size and describes the cells by discrete distributions of oriented gradients (edge directions) detected for pixels in the cells. This description is compact, discriminative and robust to noise, background clutter and minor variations in the shape and appearance of the object depicted in the patch. Specifically, the description of cells by distributions implicitly records information about which edge orientations are dominant in which cells and thus encodes a generic template of the shape and its appearance. Because orientations are quantized, minor geometric and image transformations applied at the patch produce identical or similar descriptors, as long as dominant edges fall into the same cells and maintain their same quantized orientations. Lastly, edges tend to be preserved when images are corrupted by noise, making them appropriate for higher level descriptions of images and matching. For improved robustness to additional photometric effects, cells can be combined into overlapping blocks and contrast in the blocks locally normalized before edges are extracted. Although the descriptor was originally proposed to parameterize appearance of pedestrians, HOG descriptors have recently been shown to be effective for modeling shape and appearance of 2D parts in body models [77, 211] and used for pose estimation.

Shape Context. Shape Context descriptor [20] encodes a distribution of quantized vectors that connect the center of the image region with the points on the edges within the region. Unlike, the HOG descriptor, the Shape Context descriptor splits the image region to a log-polar grid and characterizes the entire region by a single histogram, encoding how the shape/edges vary with respect to the center point. Entries in the histogram are associated with cells in the grid and record how many points on the edges belong to a particular cell. Similarly to HOG, quantization makes the representation invariant to minor shape deformations and noise. Shape Context descriptors were originally introduced to match contours of two shapes [20]. This matching was realized by sampling center points on the contours and comparing descriptors computed from the center points in order to establish correspondences between those points. More recently, Agarwal *et al.*[2] employed a bag-of-words approach to construct a vocabulary of typical shape descriptors for person silhouettes and described human poses in terms of histograms over code words in the vocabulary. Mikolajczyk *et al.*[119] extended the Shape Context descriptor to record not only the distribution of the locations of the edges but also the orientations. This extended descriptor was used by Andriluka *et al.*[5, 6] in their 2D pose estimation approach for modeling the shape and appearance of limbs in the body model.

Scale-invariant Feature Transform (SIFT). Scale-invariant Feature Transform descriptors [111] describe oriented image patches centered at distinctive points of interest (key-points) in the image. The orientation and scale of a particular patch is automatically determined from the properties of the associated point of interest[2] and the patch is described with respect to this orientation and scale. Hence, the description is invariant to the global orientation and scale of the image. In computing the descriptor, the scaled and oriented image patch is split to a rectangular grid and described in terms of its cells. Each cell is characterized by a histogram of quantized gradient orientations weighted by the gradient magnitudes. These descriptions are distinctive and robust to changes in illumination, noise and minor changes in viewpoint. In the original work of Lowe [111], SIFT descriptors were applied at the problem of object recognition. SIFT descriptors were automatically extracted for a set of interest points detected in the image and their presence, spatial arrangement and similarity to the descriptors extracted from training images were used as cues for the recognition of the object in the image. In the context of pose and shape estimation, Starck *et al.*[173] used SIFT descriptors to establish correspondences between points on the body of the person across different camera views.

We next talk about how we can use image features to estimate the pose of a person from a single still image. In our work, we use silhouettes as the only source of information for the estimation.

2.1.4 Pose Inference

Pose estimation refers to the process of reasoning about cues extracted from image observations in order to recover the configuration of the body of the person observed in the image. This process can be either performed separately for an individual image or for a sequence of images as a part of frame-to-frame pose

[2]The orientation of the gradient at the interest point defines the orientation of the patch, scale at which the point was detected in the image defines the scale of the patch.

tracking, utilizing observations from many images. In this section, we review common inference methods for estimating human poses from a single monocular or multiocular image. We note that many actual methods for pose estimation employ generic inference mechanisms that are not specific to pose estimation. As such, these general mechanisms are adapted to this particular application through the use of the models of the body, shape and appearance. We have discussed these models in earlier sections and here we only review specific inference methods.

The problem of pose estimation is most often formulated as a probabilistic inference of a distribution of body configurations given observations extracted from the image, where the goal is to determine which poses are more likely to be correct interpretations of the image for the given observations and which are not. This probabilistic formulation naturally allows to incorporate uncertainty of the pose resulting from ambiguities and noise in the image observations. For example, when image observations in the form of silhouettes permit multiple correct interpretations for the pose, all such interpretations can have assigned high weights in the recovered pose distribution.

For the purposes of taxonomy, approaches to pose estimation can be classified as those that directly employ parameterized models of the body, shape and appearance (*generative model-based approaches*) in order to infer the pose, and those approaches that do not (*discriminative approaches*). Generative methods attempt to model the imaging process that produces the image and optimize for the parameters of this process such that the synthetic image matches the features extracted from the observed image as closely as possible. Discriminative methods, on the other hand, do not model the imaging process explicitly and directly focus on learning the probabilistic mapping from the observed features to the pose parameters.

2.1.4.1 Generative Methods

Analysis-by-synthesis. Generative methods aim to explicitly model the pixel formation process that produces an image of the person. Assuming pixel formation can be characterized using a parameterized model that encodes various aspects of the imaging process, including *e.g.*, pose of the person and shape and appearance of the body of the person, any observed image can be "explained" by a suitable set of model parameters that will reproduce this image. Pose of the person can then be estimated from an input image by maximizing an objective function (energy) that measures similarity between the observed image and the image synthesized by the model from a proposed set of model parameters, using the optimized parameters as an interpretation of the image and the pose of the body. The similarity measure is implemented by an appearance model. It scores the plausibility of the image interpretation with the model parameters in terms of how well the synthesized image matches the features extracted from the observed image. This approach to inference is in the literature known as *analysis-by-synthesis* or *generate-and-test*, reflecting on one of the possible ways how representative model parameters can be estimated, that is, by generating hypotheses about the model parameters and evaluating their objective values.

Optimization. Although conceptually simple, performing this optimization efficiently is difficult because the objective function typically involves many parameters and the dependence of the function value on the parameters is non-linear (*e.g.*, because of edges in the image). The parameters of the generative model at the very least include the parameters of the body model, shape model and appearance model and form a vector in a high-dimensional space. Even though some parameters can be known in advance and fixed, such as the parameters of the shape and the appearance models, parameterization of the pose itself often produces high-dimensional state vectors. In addition, although simpler lower-dimensional models of the human body can be used to decrease the dimensionality of the search space for the optimization, such models may not be expressive enough to explain the image evidence properly. Consequently, a good trade-off between the simplicity and expressiveness of the models has to be found. The objective function itself may be complex, discontinuous and can contain many local extrema that provide wrong interpretations of the image and leave other parts of the image unexplained (*e.g.*, two legs may occupy the same region in the image while the region for the other leg in the image is not matched to any part of the body). To make the function better suited for optimization, the optimization objective is often modified to penalize unlikely model parameters, effectively imposing a strong prior over the parameters. In such cases, one then looks for image interpretations that are both typical as well as explain the image evidence well. Despite these simplifications, characterizing the distribution of model parameters and finding the most probable parameters is still problematic, especially for models in 3D. Consequently, many optimization methods get trapped in local optima and rely on good initializations to bootstrap the optimization.

The choice of a particular optimization technique to implement inference in a generative method is typically dictated by the representation of the body pose. We thus proceed with further discussion of optimization techniques based on which pose models they exploit, considering the representations by kinematic trees and parts, as discussed in Section 2.1.2.1.

2.1.4.2 Generative Inference with Tree Models

Kinematic Tree Models. Recursive parameterizations of the human pose by kinematic trees produce high-dimensional vectors with coupled interrelated components[3] that contribute to the model parameter set. Optimizing these parameters and finding a global optimum is thus a very hard problem. Many 3D methods [58, 44, 168, 46, 14, 12, 66, 56] as well as 2D methods [72, 65] address this challenge by resorting to variants of *local search combined with various heuristics* that let them escape some of the local extrema.

Optimization. This search usually takes the form of *stochastic optimization* that outputs a distribution over model parameters given the image evidence, taking an initial estimate of this distribution as an input. Similarly to particle filters, these distributions are approximated by so-called particles that record hypotheses about model parameters with weights determined by their objective values. During optimization, new hypotheses are generated from the old hypotheses in order to explore the search space. Variants of simulated annealing [46, 56], importance sampling [14], partitioned sampling [15] or genetic algorithms [72] are some of the typical methods used for the space exploration. Deutscher *et al.*[46] follows the idea of annealing to escape local extrema. Towards that end, particles in the particle set are propagated through annealing layers, where each layer replaces particles in the set with new particles sampled in the neighborhoods of the old

[3]For example, the parameters of the location of a hand in the recursive parameterization depend on the parameters of the locations for the forearm, upperarm and torso.

particles. Weights of the particles used for the resampling are determined by the objective function whose shape gradually changes from being flat to more peaked, encouraging the search to explore different parts of the parameter space earlier in the annealing process and making the search more focused later. Bandouch *et al.*[15] employs a hierarchical approach to sampling that partitions search space to subspaces and explores the subspaces in stages. In the first stage, parameters describing the location of the torso are optimized. After a good fit is found for the torso, parameters for the other parts are optimized, proceeding from the torso towards the extremities.

Other methods [71, 58, 44, 168, 55] do not attempt to model the distribution over likely poses but *directly optimize* for the most likely pose that provides the best interpretation of the image. Gavrila *et al.*[58] uses the best-fit algorithm to search through the high-dimensional parameter space. Delamarre *et al.*[44] solves for a corrective motion that needs be applied at the current hypothesis in order to bring the contour of the synthesized pose into alignment with the contour of the observed person. Freifeld *et al.*and Guan *et al.*[55, 65] optimize the objective using a local gradient-free direct search simplex method.

The success of the kinematic tree based methods depends on providing good initial estimates for the model parameters that are close to the optimum. The initial estimates can either be obtained by using other inference techniques [162, 167, 6], manually [66] or by utilizing parameter estimates from past frames [46, 14], when pose estimation takes place for images in a sequence.

2.1.4.3 Generative Inference with Part-based Models

Part-based Models. While kinematic tree models parameterized body poses recursively and facilitated top-down inference (only hypotheses about the entire pose were made), part-based models parameterize body parts independently and enable bottom-up inference of the pose (hypotheses about different parts can be generated separately). In the part-based approach, search for the pose is decomposed to a number of simpler searches: parts in the body model are first detected in the image as independent entities based on their appearance, then they are assembled into consistent body configurations based on the spatial arrangements of the part detections. Part-based models have so far been most often only used for pose estimation in 2D [51, 163, 142, 52, 5, 6, 65, 211], but 3D methods [162, 94] have been implemented as well.

Pictorial Structures. This concept of bottom-up inference and of modeling deformable bodies as collections of rigid parts was first introduced by Fischler *et al.*[53]. In this *pictorial structures* model, parts were connected by manually tuned "springs" that encoded desired spatial arrangements of the parts and the approach was demonstrated to localize eyes, nose, mouth, head top and cheeks of the human face in images. This concept was later revisited by many others, this time for modeling the human body. In these modern implementations [51, 162], the bottom-up pose estimation has been formulated as a generic *stochastic inference in a Markov Random Field* where nodes correspond to the parts in the body model and edges represent potentials encoding spatial constraints between the parts. The estimated parameters for each part typically consist of the location, orientation and scale of the part in the image plane for 2D models and the position and orientation in world space for 3D models, assuming the shape of the part is known. A variety of algorithms were used to perform the inference in this model: belief propagation (phrased as dynamic programming)

[51], approximate belief propagation [162, 163, 6] or branch and bound [178].

Perhaps the most influential implementation of a pictorial structures model is that by Felzenszwalb *et al.*[51]. In this work, a computationally efficient framework based on dynamic programming is introduced. This implementation can infer globally optimal pose of the person when (1) constraints in the model are only formulated over adjacent parts in the body articulation and (2) the corresponding potentials due to constraints are Gaussian. However, both of these requirements are not practical. One of the issues that are desirable to avoid is that of double counting, when multiple parts (*e.g.*, the left and the right arm) are assembled to occupy the same region in the image. Such body configurations are legal in the original model and can only be avoided in a principled way by incorporating additional occlusion and non-penetration constraints over non-adjacent segments [163]. Doing so introduces loops in the graphical model representation and the dynamic programming inference can no longer be used. To address this issue, Sigal *et al.*[163] uses approximate belief propagation to handle models with loops. Tian *et al.*[178] introduces a branch and bound algorithm that can find globally optimal body configurations even in presence of loops. To make constraints over legal poses more informative, Andriluka *et al.*[5] shows that non-Gaussian priors over part locations can be expressed as Gaussian priors by using transformations. Recently, 2D pictorial structures paradigm was extended by Yang *et al.*[211] to model body as a composition of non-oriented semantic templates and search for the types and locations of the templates that would match the image evidence the best. Because template types can have associated semantic meanings, this model can capture not only spatial correlations between parts but also correlations between semantics of their types, such as that open eyes tend to appear together with a smiling mouth, *etc*.

Other Methods. Other formulations for part-based bottom-up inference have been proposed as well. Ren *et al.*[145] looks for part candidates in an edge map exploiting parallelism of part boundaries. Parts are assembled to a body configuration using Integer Quadratic Programming with constraints between body parts. Mori *et al.*[124] uses local segmentation to separate image to coherent superpixels that are assembled into parts. Parts are then aggregated into a body pose using a top-down combinatorial approach.

Discussion. Compared to kinematic tree models, the part-based search strategy makes the optimization more tractable as well as more robust to occlusions or lighting variations because (1) hypotheses about individual part locations can be generated and verified independently using local part detectors and (2) it can still be possible to reconstruct the pose even when some of the parts can not be detected reliably. In addition, because the part-based pose optimization methods do not require manual initialization, they can be used as a preprocessing step for other methods, *e.g.*, to bootstrap pose tracking or to initialize the kinematic tree based pose optimizations [65]. On the other hand, part-based methods are not suitable for motion estimation, because the independent search for parts makes direct incorporation of a global motion model (needed for ensuring temporal consistency of the pose reconstructions) difficult.

Our approach uses a physical model to generate hypotheses for the whole pose, but optimizes over the parameters of the physical model that drives the motion of the body to recover the pose, not the parameters of the pose. We now consider a different class of pose estimation methods that formulate pose estimation as a regression.

2.1.4.4 Discriminative Methods

In order to avoid explicit modeling of the imaging process using a generative model, discriminative methods directly learn mapping from observed image features (taken from either a single image [155] or images from multiple views [60]) to the pose. In doing so, images and poses are described by opaque (uninterpreted) fixed-length vectors that characterize the content of the image(s) and the correlations between poses and the vector descriptions are automatically discovered from training associations using machine learning techniques. The process of using a discriminative model consists of learning and inference. In the learning phase, typical images with labelled poses are gathered (training dataset) and processed by the machine learning technique of choice to learn the correlations. During inference, the correlations are used to recover the pose given an unseen input image. The key questions that need be answered when designing a discriminative model are thus (1) how images should be described so that the vector descriptions can discriminate between poses, (2) how the training dataset should be obtained and (3) what machine learning techniques should be used to model and learn the mapping from the image descriptions to the poses.

Bag-of-words, Probabilistic Regression. Discriminative models are often implemented through *probabilistic regression*, where image features are regressed onto 3D [155, 2, 167, 23] poses, using parametric [2, 167, 23, 132] or non-parametric [155] regression models. A prime example of the discriminative approach is the work of Agarwal *et al.*[2]. In this work, images are described following the bag-of-words methodology, where image descriptions record histograms over learned prototypical shape context descriptors [20] ("visual words") found in silhouettes of common people. Any silhouette is thus described only in terms of presence/absence of the prototypical shape context descriptors, ignoring the locations where they were detected in the image. The obtained shape context histograms are mapped to poses using non-linear unimodal regression by either a Support Vector Machine or a Relevance Vector Machine. In order to characterize multi-modal pose distributions that result from observation ambiguities, such as when a particular limb can not be seen or it can not be determined which side of the body is closer to the camera using the observation model, Sminchisescu *et al.*[167] and Bo *et al.*[23] use Bayesian Mixture of Experts to model the regression. Ning *et al.*[132] goes further and replaces shape context descriptors with appearance and context descriptors [132] that are invariant to background clutter and allow for pose estimation without explicit background subtraction. Probabilistic regression based on Gaussian Processes (GP) models for large data sets have been explored by Urtasun *et al.*[182].

Other Methods. As an alternative to the parametric models discussed above, Shakhnarovich *et al.*[155] proposes a non-parametric exemplar-based method that implements computationally efficient Nearest Neighbor regression and Kernel regression for large datasets using parameter sensitive hashing. By using an exemplar-based method, arbitrarily complex correlations between images and poses can be represented in the model.

Discussion. Discriminative methods serve as a popular alternative to the generative models that do not require any (or only very little) prior knowledge of the image, person or the imaging and camera conditions. Consequently, discriminative methods have been successfully used to initialize local search in generative models [12] and to initialize generative models for pose tracking in 3D, when the initial pose of the person, needed for tracking, is not known. Note that all aspects of the imaging process, including the camera projection and pixel changes due to the variability of the pose, shape and appearance of the person, are implicitly encoded in the learned probabilistic function, based on how these aspects were captured in the training images. This feature is both good and bad. On one hand, effects of all these aspects are encoded in the model automatically. On the other, the fact that they are represented implicitly implies that huge datasets are required for training. Obtaining such datasets and learning from those sets is difficult. First, the dataset needs to capture all possible poses, shapes and appearances of people in images[4]. Consequently, it is difficult to build such large datasets from real data, and many current methods, thus, rely on artificial sets produced by graphics software, resulting in learning the aspects of the synthetic imaging process of the particular graphics program as opposed to the real-world process. Second, learning the mapping from these large datasets that can generalize beyond the training data is both a difficult computational problem [23] as well as a hard machine learning problem.

2.1.5 Motion Inference

Motion estimation refers to the process of estimating a sequence of poses representative of the motion of the person observed in the input images. Similarly to pose estimation, features extracted from images are used to reconstruct the poses in the sequence. With that said, the estimation process is significantly different from an iterated pose estimation applied at the individual frames separately.

First, the goal of motion estimation is to recover a pose sequence that is temporally coherent. That is, the pose reconstructions in consecutive frames should be similar and consistent with respect to the model of the body and ambiguities in the visual perception should be resolved consistently across frames (*e.g.*, foreshortening of the body is not explained by changing the dimensions of the body and the left and right parts of the body do not flip between frames). Ideally, the estimated pose sequence should look smooth and natural, even when seen from a different viewpoint that was not used for the estimation. Second, cues extracted from different image frames should participate in the inference in order to exploit important temporal correlations between poses in the frames, resolve ambiguities and improve overall plausibility of the reconstruction. For example, while it may not be possible to estimate the pose robustly for a particular frame due to *e.g.*, occlusions or lack of distinctive strong features, candidate poses can possibly be disambiguated by incorporating features from other frames and/or using prior knowledge about the person and the motion. To address these goals, motion estimation conceptually extends previous models for pose estimation by adding a notion of time and by incorporating generative models of *state dynamics* to explicitly represent the evolution of model states in time.

Motion estimation typically consists of two distinct phases: *initialization*, where initial models of the kinematic structure of the body, shape and appearance are built (or specified by an operator) from the first few frames in the sequence, and *tracking*, where these models are updated through time using *a generative inference framework* to estimate the poses in the rest of the sequence. During tracking, a model of the motion of the person, incorporated in the state dynamics, is utilized in order to facilitate continuity of poses, guide the motion inference and resolve ambiguities.

[4]To reduce the size of the database and to avoid the learning of pose translation, training images are most often cropped such that they only contain the bounding box of the person. In inference, a detection window is slid across the image and poses are detected and recovered with respect to the position of the window. This strategy is equivalent to assuming that the camera implements orthographic projection.

In the rest of this section, we discuss some of the typical approaches to initializing tracking, models of body state dynamics to model motion of the person and methods for 3D motion inference using these models.

2.1.5.1 Initialization

The purpose of initialization is to establish the initial state of the model, to be later updated during tracking. The initialization comprises defining the initial kinematic structure of the body, shape and appearance of the body as well as the initial pose of the person so that tracking can be started. The exact initialization procedures are specific to the actual employed models of the body, shape and appearance and have traditionally been treated as research problems of their own, separate from motion estimation.

The initialization procedures roughly correspond to two conceptually decoupled steps consisting of (1) person-specific initializations that do not depend on the image sequence to be tracked and (2) initializations that depend on both the person and the image sequence (activity). The first step can be thought of as a pre-processing step that includes taking measurements of the body shape of the person in order to set up the shape model [46], perhaps followed by the capture of the appearance of the person [69, 172]. Kinematic structure of the body is often assumed to be given and fixed by the motion estimation algorithm. This pre-processing step typically needs only be performed once for a particular person and the scene. The second step in initialization consists of establishing the initial pose of the person in the first frame of the image sequence in order to allow tracking. Both the two initialization steps can be done either manually or by making use of some level of automation.

Earlier methods for 3D motion estimation employed relatively simple shape models consisting of parameterized rigid primitives associated with segments in the body model. Parameters of such shape models incorporated direct anthropometric measurements including lengths and widths of limbs and were most often either measured/tuned manually [46] or obtained automatically using optical motion capture [13]. Gavrila *et al.*[58] estimated shape parameters automatically from an image of the person in a predefined calibration pose. The idea of calibration poses, further developed into general calibration protocols, was later used to estimate parameters of more detailed body shapes such as meshes, [69, 15]. Similar manual procedures [44, 46], procedures utilizing optical motion capture [13] and vision-based calibration protocols [58] were used to provide initial poses for tracking. More recently, however, advances in discriminative pose inference from a single image made fully automatic 3D pose initializations possible. Such methods are robust to variability in person appearance and background clutter [6] and either provide direct 3D pose initializations [2, 132] or only 2D pose estimates [164, 6] that have to be mapped to 3D. Towards that end, Sigal *et al.*[164] learns a conditional discriminative mapping from a 2D pose to 3D poses and samples from the learned pose distribution to bootstrap 3D tracking. Newer methods combine discriminative pose inference with a generative inference. Balan *et al.*[12] uses importance sampling initialized by the discriminative pose estimation approach of Agarwal *et al.*[2] to jointly estimate the 3D pose and shape of the person in one step. Gall *et al.*[56] introduces a global generative pose optimization method based on simulated annealing that does not require any initialization. The annealing approach is applied at every frame in the sequence to provide rough estimates of the pose and the pose estimates are further refined by optimizing locally and jointly for the shape and the pose in order to fit the image best.

We initialize our model by manually providing a rough estimate of the initial pose and use a generic shape model for the body that represents an average person.

2.1.5.2 State Dynamics

Models of body state dynamics capture evolution of body states (poses, shapes and appearance) from an initial state over time. The purpose of state dynamics is to propose informative updates of the body model for pose inference in future frames as well as to facilitate frame to frame consistency of the model representation. Most often, shape and appearance of the body are parameterized relative to the pose [46, 14]. That is, *e.g.*, instead of recording the actual posed geometric shapes, the shapes are determined from the parameterization of the pose and the parameters of the shape model. This way, variations of the shape and appearance due to pose changes can be separated from the encoding of the pose and modeled separately, independently of the pose. Consequently, the representations of the shapes and appearance can remain fixed or nearly constant over time. In other cases where shape and appearance models can be learned from individual images separately [141], the representations may need not be tracked and propagated through time by the state dynamics model. Consequently, in either case, modeling body state dynamics really means modeling pose dynamics and, as such, pose dynamics is a major concern of our work.

State Dynamics. State dynamics models encode temporal relationships between body states in the frames in the sequence and implicitly approximate underlying physical processes that govern the motion of the person. The frame to frame state correspondences are typically characterized by conditional probabilistic distributions encoding what states are more likely to follow a particular past state, with respect to some discretization of time, as in

$$p(\mathbf{x}_t|\mathbf{x}_{t-1}). \tag{2.1}$$

Here, $\mathbf{x}_t$ encodes the body state at the discrete time instant t and $\mathbf{x}_{t-1}$ records the state at the corresponding preceding time instant $t-1$. In accordance with standard practices in computer vision, the time instants correspond to times of frames and t thus indexes frames in the sequence (as opposed to provide a physical time of the frame when it was taken by the camera). We may use the terms time t and frame t interchangeably.

Motion Model, Predictions. Most often, parameters of the shape and appearance models are fixed in the model and only the representation of the pose is allowed to change. Hence, typically, $\mathbf{x}_t = \mathbf{q}_t$, where $\mathbf{q}_t$ represents the *kinematic pose* of the body at the time t. A great deal of modeling state dynamics thus consists of representing the relationships between consecutive poses and we can think of the state dynamics model as of the *motion model* of the person.

Given this model structure, state dynamics models can be used for making *predictions* about the future motion of the person. Specifically, assuming an estimate of the initial state $\mathbf{x}_1$ at the frame 1 is given, sampling the conditional distribution recursively for the future frames produces a hypothesized state sequence $\mathbf{x}_1, \ldots, \mathbf{x}_t$ that can be directly validated against image evidence within the context of an analysis-by-synthesis framework in order to find the best interpretation of the sequence[5]. The ability of the state dynamics model to produce accurate predictions is thus important for the success of such an approach for motion estimation. In

[5]The same pose sampling techniques are also used for synthesizing motion in computer graphics (see Section 2.3).

general, predictions of the poses made by the model should be (1) close to the actual observed poses and (2) physically feasible so that the recovered motion does not violate physical laws imposed by the body model and the environment. Although the sampled sequences can be further refined by the inference mechanism in order to better fit image observations (*e.g.*, by applying local search on the poses), the refinement can cause the poses to violate physical laws. Consequently, approximating the true physical processes that govern the human motion at a sufficient level of fidelity in predictions is essential to physically plausible tracking.

Current Models. Most current approaches to tracking [160, 46, 22, 186] employ relatively simple models of body state dynamics. In these approaches, the true physical dynamics of the human body is either approximated in a very crude way, by only encouraging temporal coherence of model states [46], or indirectly, by learning statistical models of human motion from reference human motion capture data, as in [186].

2.1.5.3 Linear Motion Models

Linear Dynamical Systems. To facilitate continuity in the recovered poses, motion of the person has often been modeled as a *Linear Dynamical System* (LDS), where the state at the current frame was predicted by a linear transformation of the past state with additive Gaussian noise, using

$$p(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(A\mathbf{x}_{t-1}, \Sigma) \tag{2.2}$$

with the matrix A to model the transformation of the past state toward the current state, $\mathcal{N}$ representing the Gaussian noise function and Σ determining the amount of noise to be added to the transformed state. Variants of linear dynamical models with different state representations and manually tuned [46, 13] or learned [133] A matrices have been used in the literature. We review some of those variants next.

Smooth Prior. Earlier approaches to tracking by Sidenbladh *et al.*[160], Deutscher *et al.*[46] or Balan *et al.*[13] model the motion of the person by the non-informative *smooth* motion prior,

$$p(\mathbf{q}_t|\mathbf{q}_{t-1}) = \mathcal{N}(\mathbf{q}_{t-1}, \Sigma), \tag{2.3}$$

assuming the relative motion between consecutive frames is small and can be explained by sampling from Gaussians centered at the previous states. While effective for modeling slow low-energy motions such as walking, the slow motion assumption can be easily violated for other sequences.

Higher-order Models. Other variants attempt to account for higher-order dependencies between kinematic poses and permit larger changes in the pose between frames. Towards that end, body state representations that incorporate histories of poses have been proposed[6] and integrated with the motion model. *Constant velocity* motion priors [133, 157] of the form

$$p(\mathbf{q}_t|\mathbf{q}_{t-1}, \mathbf{q}_{t-2}) = \mathcal{N}(\mathbf{q}_{t-1} + \gamma(\mathbf{q}_{t-1} - \mathbf{q}_{t-2}), \Sigma), \tag{2.4}$$

where $(\mathbf{q}_{t-1} - \mathbf{q}_{t-2})$ is an estimate of the (scaled) body velocity and $0 \leq \gamma \leq 1$ is a damping constant, are an example of motion models that exploit this paradigm. North *et al.*[133] explored general higher-order

[6]For example, the body state $\mathbf{x}_t$ at the frame t then consists of the parameterizations of the kinematic poses $\mathbf{q}$ from the frames $t-K$ through t, $\mathbf{x}_t = [\mathbf{q}_{t-K}, \ldots, \mathbf{q}_t]$, where K is the length of the pose history to be accounted for.

histories of kinematic poses from the past K frames with the transformation matrices A learned from training motions. It was shown that the learned structure of the matrix A allowed to automatically capture the coupling in the motion of different joints as exhibited during training, thus making the predictions more informed.

Switching Linear Dynamical Systems. With that said, linear dynamical models can only model state progressions that are linear, but human motion, in general, is non-linear (*e.g.*, on change of foot contact). For this reason, linear dynamical models have been extended to approximate human pose dynamics as a progression of linear models that switch over time. These models, known as *Switching Linear Dynamical Systems* (SLDS), consist of a set of LDS models and an indexing variable that determines which linear model is currently in effect for making predictions. Pavlovic *et al.*[136] used SLDS models for tracking and synthesizing human walking. Despite the successes, it is not clear how the switching mechanism scales for more complex motions that require many LDS components or how the models can be constrained to only produce biomechanically possible pose predictions, such as that joint angles would stay within their respective biomechanical limits and parts of the body would not penetrate each other.

To address these concerns, it became customary to employ data-driven models that learn the non-linear pose dynamics from offline demonstrations of a given activity.

2.1.5.4 Data-driven Motion Models

Data-driven models aim to utilize examples from a training motion (obtained offline by *e.g.*, optical motion capture) in order to learn what kinematic poses are typical for the particular person and the activity and to learn how these poses can be sequenced into valid motions. These models are then used to formulate pose dynamics with respect to the set of the typical poses and to bias pose inference during tracking to the poses in this set. The set of typical poses can be represented either in the original high-dimensional space [155] or in one of its many explicit low-dimensional embeddings. These embeddings have a long history in computer vision [182, 33, 184, 7], graphics [196, 197] and robotics [76]. Here, we review the use of these models in computer vision.

Linear Embeddings. For example, Sidenbladh *et al.*[158] and Urtasun *et al.*[184] learn a *linear, activity-specific* kinematic model for the entire pose trajectory from a set of training sequences that capture the typical variations in the execution and style of the activity. The set of training sequences is processed by Principal Component Analysis (PCA) in order to (1) reveal the components that characterize the major stylistic variations in the pose and the pose sequencing and to (2) parameterize poses in motions that are similar to the training sequences in terms of their stylistic parameters (as linear weights of the PCA components) and progressions along the motion (frame index). Linear dynamical models from Equation (2.3) are used to model the evolution of the stylistic parameters and the frame index of the pose over time during tracking. Examples of successful tracking of walking [158, 184] and running [184] using these models have been demonstrated in the literature.

Non-linear Embeddings. With that said, typical poses often lie on a non-linear manifold in the high-dimensional pose space, as is the case even for walking [184, 183], and the structure of this manifold may be parameterized more efficiently using a non-linear model. For this reason, explicit low-dimensional *non-linear activity-specific embeddings* [33, 112, 166, 185, 183, 186] (or mixtures of low-dimensional linear-embeddings [105]) became popular. Low-dimensional embeddings are particularly attractive because they reduce the dimensionality of the search space and allow to formulate tracking and pose dynamics in these more manageable spaces. For example, pose dynamics that is non-linear in the original high-dimensional space may be well approximated by linear dynamics in the low-dimensional pose space. Models that are capable of such embeddings often take the form of *Gaussian Processes Latent Variable Models* (GPLVMs) [182, 185, 183, 186]. These models provide smooth probabilistic mapping from the low-dimensional poses, either on the pose manifold or in the vicinity of the manifold, to the corresponding high-dimensional poses and can model the uncertainty in this mapping. Therefore, they can facilitate efficient tracking in the low-dimensional space. Furthermore, they can represent poses outside the training set and cluster poses that are close in the high-dimensional space to the nearby poses in the low-dimensional space, such that it is possible to refine a pose by refining its low-dimensional representation.

Multiple Embeddings. The embeddings from above are, however, only effective when they are built from motion sequences that do not exhibit too much variability and when they all come from a single activity. To enable tracking of different activities and styles in the same activity, multiple manifolds have to be built and integrated with the model and transitions between the manifolds represented. Towards that end, Urtasun *et al.*[185] constrains the topology of the learned pose manifolds. Specifically, the individual dimensions of the recovered manifolds are confined to have particular semantic interpretations that represent different aspects of the motion of the person, such as the motion style or progression along the motion, and manifolds are switched based on this structure.

Discussion, Learning from Motion Capture. Note, in many cases, particular motion executions and styles are largely driven by the physics of the human body and result from the interactions of the body with the environment. For example, while walking up a slope, our body tends to tilt forward to counteract gravity and to produce a more efficient gait. When moving on an icy surface, we are more careful about keeping our body balanced at all times so we will not fall. These observations suggest that many interesting motions can be generated readily from the physical and biomechanical principles of the human body. With that said, data-driven methods do not learn these principles directly, because they are only allowed to observe the effects of the principles, as they are exhibited in the training motion capture data.

Motion capture data can be thought of as a snapshot of the dynamics that occurred at the time when the motion was captured and within a given environment where it was performed (*e.g.*, typically on a non-compliant level surface/ground). Models learned from such data are thus limited to a specific motion, or class of motions (*e.g.*, walking [183, 6], golf swing [186]), subject to the effects of the environment at the capture. Consequently, when the effects of the environment change (such as when the floor becomes tilted), the learned models have difficulty generalizing to these new situations. For example, if a data-driven model is trained on the data of a person walking on level ground, the model may generalize well to other people walking on the level ground with different styles or speeds [183, 6]. However, the model is not able to generalize to a person walking up the stairs, because the necessary changes in the upper and lower body dynamics (only needed for this particular activity) have not been observed during training and are due to physics of the human body that is not represented in the model. While, in theory, physical dynamics could possibly be learned from data that captures all variability across all possible motion executions, styles and environmental conditions, obtaining such amount of data is not realistic and in many cases not even possible (*e.g.*, diving). Even if the data can be captured, learning from such large data sets would pose a significant challenge. Consequently, representing the physical principles that produce the desired motion explicitly in the model may be more effective.

2.1.5.5 Geometric Motion Models

To model geometric aspects of the human motion, *geometric models* attempt to directly constrain pose kinematics with constraints on the locations of body parts imposed by the environment [148, 147]. For example, these models are able to encode constraints that the feet should not penetrate the ground plane [148] or that the feet or hands must be in some fixed relative configuration (as dictated by the environment) with respect to one another [147]. Such geometric models are not able, however, to allow dynamically plausible environmental interactions, *e.g.*, encode that the feet must be in contact with the ground in such a way as to support the resulting motion so that the body stays balanced, *etc*.

2.1.5.6 Physical Motion Models

One way to address the weaknesses of all previous models is to incorporate physical and geometric constraints directly in the motion model by using physical simulation and control, as we propose in Chapter 1. The main motivation for doing so is that simulation and control allow to model the underlying physical and biomechanical causes for the motion, as opposed to only the effects of these causes, as is the case with the previous models.

Physical Models in Graphics, Features. Physical simulation has a large body of existing work in animation [18, 70, 87, 126, 210, 215, 206, 40, 96] and robotics [32, 140, 171, 210, 209] and is now a commodity technology. Together with control mechanisms [70, 40, 209, 96, 213, 43], that provide actuation forces to actively actuate the body, simulation produces motions that are driven by physics and consistent with physical laws. The simulation allows to naturally account for various physical factors that affect human motion, such as a person's mass, gravity, interaction with the ground plane, friction and responses to self-collisions and physical disturbances. On the other hand, control allows to react to these aspects in order to produce motions that adapt to the effects of the aspects and that are desired. Modeling all these principles makes it possible for the model to generalize to different situations, by simulating the effects of the principles in these situations. In the context of offline physics-based motion synthesis, optimization methods [138, 50, 151, 116, 108, 202] have been used to generate physically realistic motions from constraints (*e.g.*, desired poses at specific time instants and foot placement constraints) and optimization objectives that abstract physical and biomechanical aspects of the human motion.

The generalization capabilities of physical models and the ability to generate plausible human-like motion have been so far demonstrated mostly in graphics and robotics, where physical models have been shown to be able to faithfully model environmental interactions, plausibly re-target the original kinematic motions to other environments [138], dynamic interactions with the environment [206] and temporal executions [116]. We conjecture that the use of similar models in motion estimation would allow generation of pose predictions that are (1) physically plausible, (2) effectively generalize to a wide range of motion executions, environmental conditions and perturbations (beyond the scope of data-driven methods) and that (3) do not rely on large databases of human motion capture data. With that said, the use of physics-based human models for motion prediction in vision has been limited to date.

Passive Dynamics Without Contact in Vision. Earliest work on integrating physical models with vision-based person tracking can be attributed to Metaxas *et al.*[118] and Wren *et al.*[205]. Metaxas *et al.*approximated physical dynamics of the human body by simulating an articulated model of the person under the effects of damping forces. Because gravity and contact of body parts with the environment (and other parts) were not modeled, the motion synthesized by the model was linear and resulted only from the velocity and inertial properties of the parts. Consequently, it was possible to integrate the model with Kalman filter for motion inference, but it is reasonable to assume that equally effective motion predictions and motion reconstructions could be produced by using much simpler linear constant velocity models. Wren *et al.*utilized a similar model of linear dynamics without control. Both these two methods were demonstrated to be able to track only simple upper body motions of stationary subjects in controlled environments without contact and required rich image observations to facilitate tracking, including segmented 3D markers [118] and stereo features [205].

Abstracted Low-Dimensional Models for Walking in Vision. More recently, Brubaker *et al.*[27, 26] explored planar approximations of simplified lower-body dynamics for predicting walking on the ground plane. Their 2D physical models, inspired by the powered walking model of Kuo *et al.*[93], modeled the human body as a collection of connected point masses that represented the body center and two straight [27] or possibly bent [26] legs with rolling feet. This particle system was subjected to (1) external gravity and contact forces to account for physically plausible interactions of the legs with the ground plane and (2) internal forces to control and power the model. A controller of the model made the simulated body locomote and push forward by applying (1) torsional spring forces between the legs and (2) "toe-off" impulses on the stance (back) foot when transferring weight between the feet. By varying parameters of the controller, that consisted of the parameters of the springs and the amount of the push on weight transfer, the controller was able to synthesize human-like gaits with different speeds and step lengths. This property was used for estimating walking motions from monocular video using a particle filter as an inference mechanism. Similar to us in spirit, the goal was to explain the motion of the person in terms of the parameters of the controller that would reconstruct the observed motion the best. However, because the physical model operated only on the 2D simplified model of the lower body of the person, pose predictions had to be lifted to 3D in order to enable full-body 3D tracking. Towards that end, an additional 3D kinematic model was introduced and tracked together with the 2D model. Motion inference was formulated as that of over the control parameters of the 2D model and that of the kinematic poses of the 3D model, constraining the motion of the 3D model to be consistent with the motion of the 2D model. The remaining degrees of freedom that were not represented by the 2D model (such as the global heading or roll) were let move freely, assuming smooth dynamics sampled by a random walk. Note that although the motion of the underlying 2D abstract model was physically plausible, the final reconstructed 3D motion was not a result of physical simulation. As such, the tracker was driven towards physically plausible 3D interpretations but was not able to guarantee them.

Our Model. In contrast to the above physical models, our physics-based approach is formulated directly in 3D, models full body motion, is not limited to walking, or any other activity, and can guarantee that the recovered 3D motions are results of raw simulations, as demonstrated in Chapter 5. Next, we discuss how motion models can be used for motion inference.

2.1.5.7 Generative Inference

In this section, we outline a standard approach for motion estimation using a generative model. The approach extends the pose estimation paradigm from Section 2.1.4.1 by using a generative model that characterizes the imaging process of the person for the entire frame sequence, as opposed to only a single frame. This imaging process is usually encoded in the model by characterizing (1) how body states in the frames can be generated from an initial state through the use of a model of state dynamics (see Section 2.1.5.2) and (2) how pixels in the corresponding images can be formed from the proposed body states using shape and appearance models (see Section 2.1.2.2). This way, the variability in appearance of the person due to pose changes is decomposed from the motion of the person.

Optimization Objective. Analogously to the pose estimation problem, the motion of the person can be estimated from the input image sequence by maximizing an objective function that measures similarity between the observed images and the images synthesized by the generative model from a proposed set of model parameters, using the optimized parameters as an interpretation of the image sequence and the motion of the person. Once optimized, motion reconstruction can be obtained *e.g.*, as the set of poses generated from the optimized parameters. As before, the similarity measure can be implemented by a generative appearance model of the body of the person. The measure scores the plausibility of the image interpretations with the model parameters in terms of how well the synthesized images in frames match the features extracted from the corresponding observed images and, due to the decomposition of appearance from the motion, can be evaluated on the frame-by-frame basis using the methods from Section 2.1.4.1.

Note that the generative methodology for motion estimation can be combined with discriminative appearance models and discriminative pose estimation for scoring the pose hypotheses from the motion optimization. In this case, the generative model is restricted to modeling only state dynamics of the body, the objective function is formulated with respect to a discriminative appearance model of the body of the person and the synthetic images are not generated. Discriminative pose estimation models are then used to produce 2D or 3D pose estimates from the still images for the individual frames in the sequence and the poses proposed by the motion optimizer are scored based on their consistency with these estimates. With that said, there are no purely discriminative methods for the motion estimation itself; the generative inference framework is still needed at least for the modeling of the motion of the person (in terms of state dynamics) and propagation of the pose estimates over time.

Optimization. The generative motion optimization can be implemented within the framework of analysis-by-synthesis, where the optimizer samples the underlying generative model to propose various interpretations of the image sequence, the interpretations are scored based on their consistency with the observed images (or individual pose estimates from the still frames) and the scores are used as a feedback to guide further optimization in order to find the best interpretation. This inference can also be implemented probabilistically, where one is interested in recovering a distribution of likely image/pose interpretations given observed image features. The main difficulties in performing this optimization are due to the number of parameters that have to be optimized and the non-linear dependencies of the objective value on the parameters. The parameters of the generative model for motion estimation need to encode at least the pose of the body for every frame in the sequence and this number grows with the length of the sequence. Consequently, full optimization over all parameters in the generative model soon becomes untractable.

Batch and Incremental Optimization. The optimization, as described above, assumes that it operates over the entire sequence. This assumption may not be practical for long sequences and solutions that are able to leverage information from only portions of the sequence, while only optimizing over smaller subsets of the parameters in the model, have been proposed. Depending on what image observations participate in the estimation of the pose for the current frame, methods for motion estimation can thus be classified to those that perform the inference online and those that estimate the poses in the sequence in a batch. *Online methods* combine the image evidence from the past frames and the current frame, without considering any future frames. As such, online inference can proceed incrementally, without having to wait until the full sequence is captured. *Batch methods*, on the other hand, operate on the entire image sequence, making use of the full image evidence from all the past, current and future frames, thus having a chance to produce more informed results. While appealing, the batch method can only be applied for short image sequences. First, batch optimization requires all image evidence to be kept in the memory and to participate in the inference, requiring to deal with huge amounts of data, which results in a significant computational burden. Second, in many cases, it is not possible to optimize over the entire sequence at a time, because the number of parameters needed to explain the sequence may be too high. In those situations, the optimization may proceed incrementally, by e.g., only optimizing for a single frame at a time. Consequently, most research effort has been concentrated on developing efficient online inference methods.

We next discuss previous optimization methods in motion estimation. These methods store an explicit pose representation of the person in the model state for every frame in the sequence and propagate this pose through time using a model of state dynamics. The particular type of the pose representation dictates how state dynamics can be formulated and affects the structure of the search space for optimization. Therefore, we review these methods based on the pose representations that they employ.

2.1.5.8 Generative Inference with Tree Models

Kinematic Tree Models. For the purposes of motion estimation, human poses in model states and motion models have most often been parameterized recursively using kinematic trees (see Section 2.1.2.1), facilitating top-down inference. These parameterizations have become common in part, because they implicitly honor body articulation constraints that request the parts to stay connected at joints. Consequently, when the poses are updated by a motion model, the constraints remain always maintained, regardless of the formulation of the model. Specifically, because the constraints do not have to be explicitly accounted for during inference, black-box inference methods that do not have the knowledge of the constraints can be used and together with simple smooth or statistical motion models.

Using these pose representations, motion optimizations have usually been implemented incrementally over individual frames. Each frame was interpreted using a local optimization over the parameters of the model state at the given frame, initialized by the state predictions from the previous state by the state dynamics model, either in 3D [58, 44, 168, 46, 14, 12, 66, 56] or 2D [72, 65]. We now describe this incremental optimization in greater detail.

The entire incremental optimization is typically performed online, with the local per-frame optimizations implemented using a stochastic search that exploits the sampling techniques similar to the techniques previously used for estimating poses from still images in Section 2.1.4.2. The main difference in this search is that the sampling process now considers time. That is, model state hypotheses for the current frame are sampled not only spatially (with respect to image coordinates in the current frame, as before) but also temporally, making use of the motion model, to model how the hypotheses may change over time.

Per-frame Optimization. Extending the still image case, the goal of the per-frame optimization for the current frame is to approximate the distribution over likely model parameters for that frame given the image evidence from the first frame up to and including the current frame. As before, this distribution is usually represented by weighted particles that record possible hypotheses about the model parameters with the weights determined by the values of the objective function evaluated for the hypotheses. Assuming the initial hypotheses for the first frame are already determined (or given), the entire incremental optimization proceeds in steps to update the state hypotheses from frame to frame. At each step, the motion model is first applied to the hypotheses from the previous frame in order to produce predictions about the new values of the model parameters for the current frame. The predictions may then be refined using another local optimization in order to fit the predicted parameters better to the actual image observations available at that frame and the incremental optimization proceeds to the next frame.

A prime example of this concept is a *particle filter*, first used for vision-based tracking by Isaard *et al.*[74]. Other popular examples include simulated annealing [46, 56] and partitioned sampling [15], differing from the basic particle filter in the sampling techniques they employ to explore the state space and in their abilities to escape local minima.

In general, these classes of methods are called top-down because the hypotheses for poses are generated for the entire body at once. Contrary, bottom-up methods discussed next generate hypotheses for parts of the body separately and try to combine them in order to arrive at a plausible interpretation of the entire body.

2.1.5.9 Generative Inference with Part-based Models

Part-based Models. Part-based models parameterize segments in the body using independent coordinates, attempt to locate the segments independently in the image and assemble the possible detections into consistent body configurations using optimization, facilitating bottom-up inference. Conceptually, the optimization proceeds similarly to the previous optimizations methods for kinematic trees, where the motion estimation is formulated as a (probabilistic) optimization of an objective function.

In this case, however, the optimization is now more challenging because the objective function involves many more parameters, due to redundancy in the pose representation, and the parameters in the state vectors for frames are constrained by constraints that have to be explicitly accounted for during the optimization. Specifically, the optimization mechanism needs to be aware of the articulation constraints due to joints in the body in order to produce consistent poses for frames that maintain these constraints. In motion estimation, these constraints need be maintained not only for a single frame but also over time. Maintaining these constraints exactly has been shown difficult and all previous methods that exploit the part-based approach implement constraints only approximately, in terms of penalty terms in the objective function that penalize the violations. Consequently, it is not guaranteed that valid poses will be found. As before, to handle the high dimensionality of the optimization problem, the methods employ online incremental optimizations to optimize the entire motion, where each frame in the image sequence, or a number of consecutive frames, is optimized independently using a local optimization. Here, we illustrate one example of such a method.

Loose-limbed Model. Sigal *et al.*[162] proposes a *loose-limbed model* that extends the part-based optimization for a still image to optimize pairs of consecutive frames in the sequence, and uses it to implement local optimizations in the incremental optimization of the entire motion. Towards that end, pose estimation from a still image is formulated as an inference in a graphical model using belief propagation and the graphical model is extended to model temporal dependencies between frames. In this graphical model, nodes correspond to states of body parts at individual time instants and edges encode geometric and temporal constraints between the parts. The model can be seen as a model for a single frame that was replicated such that each frame receives its own set of nodes and edges to characterize the state of the body at that frame. Nodes for corresponding body parts from consecutive frames are then connected by edges to model temporal relationships between the states of the parts, as dictated by the motion model. Motion inference then amounts to the general inference in this graphical model, which contains loops.

Discussion. Part-based models can also be integrated with generative inference to validate pose hypotheses produced by the previous top-down motion estimation methods. Towards that end, the objective function of the top-down method can be modified to measure the similarity between the proposed full-body poses and the results of the pose estimation from the single frame obtained using a part-based method from Section 2.1.4.3. Effectively, the full-body poses are thus matched against pose detections, as opposed to image features, replacing noisy image observations with more robust detections. A typical example of this approach is the work of Andriluka *et al.*[6].

2.1.5.10 Inference in Our Methods

Our methods, as outlined in Section 1.5, follow the generative top-down approach for motion estimation but differ from the previous top-down methods in a couple of critical aspects.

We exploit a compact representation of motion based on physical principles from Section 1.3.2 that reduces the dimensionality of the search space for optimization. Unlike the previous methods, we do not represent poses in the motion explicitly, as a sequence of independently parameterized and tracked poses in frames, but produce these poses implicitly, by simulating a physical model of the human body from an initial pose. Furthermore, we do not store the current pose in the model state and we do not formulate optimization in motion estimation as that over the kinematic pose. Instead, we represent parameters of the physical process that produces the motion in the model state and optimize parameters of this process. The physical model effectively re-parameterizes the motion of the person in terms of control mechanisms that produce this motion in simulation. This parameterization leverages the underlying physical structure of the motion and represents the motion as a composition of physical actions that the person needs to perform in order to complete the motion. This parameterization of motion, encoded in our generative model, is sparse and compact compared to the previous representations by pose sequences. Consequently, our model is easier to optimize and also facilitates physically plausible motion estimation as stated in our problem statement (see Section 1.1). Because we optimize over the control mechanisms for the entire motion, and not the per-frame poses, our method is not similar to any of the previous methods.

Our methods exploit both the batch and the incremental approach for the optimization of the entire motion, in terms of the control mechanisms that explain (portions) of the sequence. Unlike previous methods, our incremental approach is able to guarantee that the current interpretation produced by the incremental optimizer is physically plausible at any time in the process. Specifically, interpretations for new frames are optimized in the process such that they will not break physical laws and will follow consistently on the interpretations from the previous frames. When the incremental optimization completes, the entire motion is interpreted by a single sequence of poses directly produced by physical simulation with the estimated controller (see Section 1.4.3, Section 1.5 and Chapter 5).

2.2 Learning from Demonstration

Our research on human motion capture also relates to learning of controllers and policy functions for decision processes in computer graphics, robotics and artificial intelligence. According to our motion capture formulation through controller capture (see Section 1.5.1), we aim to use the video as a demonstration of the desired motion for a humanoid character and learn a controller that reproduces it. Consequently, we provide a brief overview of general techniques for learning from demonstration, as currently applied to problems in computer graphics and robotics, and discuss how the methods relate to our approach.

2.2.1 Overview

Learning from demonstration refers to a general methodology for deriving control "programs" for dynamic systems, where the programs to achieve desired system dynamics (behaviors) are learned from recordings of expert demonstrations of the dynamics. At the core of learning from demonstration is establishing of a control policy for the system that decides what actions should be applied on the system when in a particular state such that a desired demonstrated behavior is achieved. We assume actions are selected using a *controller* according to a *policy function* that maps states of the system to the actions. The goal of learning from demonstration is to learn a policy function for the controller by observing state-action pairs executed by an expert.

This methodology for deriving control is particularly appealing, because it allows users to focus on how the desired dynamics can be demonstrated effectively as opposed to how it can be achieved in control. As such, it potentially allows users to program complex dynamic systems without having to actually derive mathematical formulas for the control. Consequently, the methodology has many applications. In robotics, this methodology has been used to teach robots perform various skills, such as grasping objects [144], navigating environments [144, 88, 1, 214, 101] or playing robot soccer [64], among others. In computer graphics, learning from demonstration has been used to animate life-like humanoid characters in virtual environments [179, 97, 38, 102].

We next overview general methods for learning policy functions, focusing on applications in computer graphics. Policy functions can take many forms and can be learned in different ways. Most often, policy functions are learned through *classification* [152] or *regression* [153, 127, 61, 62], by using *reinforcement learning* [165, 179, 97, 38, 102] or *inverse reinforcement learning* [103, 101], or other methods [144, 1]. Policy functions can also be built manually, as often practitioned in computer graphics (see Section 2.3.5). We refer interested readers to the survey of Argall *et al.*[8] for a comprehensive review of methods for learning from demonstration in robotics.

2.2.2 Regression and Classification

When temporal ordering of demonstrated state-action pairs is not considered in the learning process, the policy function can be learned by either regression or classification. Regression and classification methods use demonstrated state-action pairs to directly learn a mapping function from states to actions. The intent of the learning algorithms is to approximate the policy of the expert such that the demonstrated states map to demonstrated actions and meaningful generalizations are produced for states that are not present in the demonstration. The particular learning methods primarily differ in whether the mapping functions produce continuous or discrete actions. The methods also differ based on whether they require full demonstration data during run-time or only an aggregate information extracted from the demonstration. In learning the mapping, the methods build the mapping function either incrementally as demonstration data arrives or from the entire demonstration in a batch.

Classification. In classification, the set of actions is discrete and the mapping function assigns input states with action labels. As an example, Saunders *et al.*[152] uses action labels to identify primitive motor commands for a robot, such as moving forward, backward or turning left or right, and implements the mapping function using k-Nearest Neighbors classification. In Chernova *et al.*[34], classification is applied to the problem of simulated driving on a highway, where the simulated driver chooses between staying in the current lane or shifting one lane to the left or right in order to avoid collisions with other cars.

Regression. In contrast, regression methods produce mapping functions that output continuously parameterized, and potentionally high-dimensional, actions. These outputs may be obtained by averaging actions from state-action pairs using *e.g.*, k-Nearest Neighbors regression [175, 21], by performing regression locally over demonstrated states close to the query states [153, 127], by combining outputs from multiple regressors [187, 61, 30, 64, 29] or by using other methods, such as Neural Networks [137]. Here, we illustrate a few uses of such methods for applications in robotics. Locally Weighted Regression [153] has been employed by Nakanishi *et al.*[127] to learn desired pose trajectory for biped robot locomotion. A sparse incremental variant of the regression, Locally Weighted Projection Regression (LWPR) [188], has been explored by Grollman *et al.*[61] for learning basic robot soccer skills. Later variants of this work [62] replaced LWPR regression with Sparse Online Gaussian Processes [39] in order to reduce memory consumption and improve computational performance for the learned tasks.

2.2.3 Learning State Machines

Multiple Actions. Regression methods are beneficial for modeling decisions for processes with continuous actions in high-dimensional spaces. However, because they assume that the demonstrated state-action pairs sample a policy function (as a mapping from states to actions), they are inherently limited to learning behaviors that are expressed as functions. As such, they expect that only one true action can happen in each potential state and assume that similar states lead to similar actions.

This restriction may prevent successful learning of the demonstrated behavior [63], because either the expert may be inconsistent in choosing their actions (*e.g.*, different actions can be demonstrated for the same or a similar state) or the information that the expert uses to discriminate the actions is hidden and not directly represented in the state vector. For example, when the expert teaches a robot to scan a game field to look for a ball using kinesthetic teaching (when the robot records its state as its pose is being physically manipulated by an expert [30, 29]), the recorded state information in the demonstration may permit both a "turn-head-left' and "turn-head-right" action when the ball is not seen. Although both actions are valid in the given situation, they may be interpreted by the regression algorithm as noisy instances of a "keep-head-still" action encoded as an average of the two other actions. Consequently, an undesired behavior would be learned [63].

Context for Regression. To address this difficulty within the regression framework, one may assume the expert always selects the same action when in particular situation and attempt to recover the hidden contextual information that allows to disambiguate actions. Using this decomposition, it is possible to learn policy functions for different contexts using regression and use the policy function for control that is currently applicable for the present context. For example, to disambiguate the "turn-head-left" action from the "turn-head-right" action in the example above, the robot may need to be aware of whether it is currently scanning the field from the right to the left or from the left to the right and apply different policy functions based on this context. Additionally, to allow autonomous operation, the robot needs to know how contexts switch over time such that *e.g.*, the context for scanning from the right to the left becomes effective after the scanning from the left to the right has completed (and vice versa).

Learning State Machines. Mathematically speaking, the problem of finding an appropriate decomposition for the observed behavior can be phrased as a learning of a *state machine* for the behavior, where states represent different contexts for regression and transitions encode when and how the contexts switch. More generally, the contexts can be interpreted as identifications of subtasks within the overall task and the learning of the state machine corresponds to the learning of the subtasks, policy functions for the individual subtasks and transitions between them [63]. Our motion capture through controller capture attempts to achieve a similar high-level objective, learning of a state machine based controller for the observed behavior. However, unlike the discussed methods, our approach is not motivated by the desire to resolve action conflicts in regression but to segment the observed motion into longer-term actions that last over many steps (tens of milliseconds), are specific to the motion and are separated by physical events. As such, our methods are both more constrained as well as use different techniques for learning, based on optimal control.

Existing methods for learning state machines from demonstration in robotics often address only specific parts of the state machine learning problem. When the number of states in the state machine and the policy functions for the states are already known, methods that can learn transitions between the machine states are available [21, 129]. If instead the number of states in the state machine and the transitions between the states are given, policy functions for the individual states can be derived [149, 176, 128]. In the most general case, when nothing is known about the task, methods that can estimate the number of machine states and policies for the states have been implemented [64, 29]. However, with the exception of the method of Butterfield *et al.*[29], they have been unable to learn transitions between the states. Given that our approach has a similar constraint (we do not know the number of states in the state machine beforehand), we further discuss this class of methods.

The goal of the method of Grollman *et al.*[64] is to determine how many subtasks there are in the demonstration and to associate state-action pairs from the demonstration with subtasks, such that policy functions can be learned for each subtask using regression. The method learns states in a state machine, but it can not predict when the current state should switch. Consequently, the method can not be directly used for control. Niekum *et al.*[131] addresses a similar high-level goal of segmenting the demonstrated task into subtasks. Unlike [64], however, each subtask is encoded as a dynamical system that starts in a particular configuration, evolves over time and ends in a final configuration. Given the ordering of the subtasks in the segmentation, the method is able to apply the learned dynamical systems on a robot in order to replay the demonstration and adapt it to perturbations. With that said, the method does not learn general policy functions for the subtasks and, similarly to [64], does not recover the principles that cause the subtasks to switch.

Butterfield *et al.*[29] recovers a full state machine with states for different subtasks, a model for switching subtasks and policy functions for the states, where states correspond to different regression contexts. The method uses Hierarchical Dirichlet Process Hidden Markov Model to encode the switching process and a regression based on Gaussian processes to model per-frame policies for the individual subtasks. Our method is different in assuming that states produce constant policy functions that select the same long-term actions, and in conditioning the transitions between the states based on physical events, not decisions made by a statistical model. We next discuss our methods for learning state machines and policy functions in more detail.

Our Methods. Typical regression methods operate at the granularity of a few milliseconds, such that decisions about actions are made very frequently. Contrary, for stability in physical simulations, we employ a hierarchical form of control with a decision making process that selects longer-term actions and a low-level motion control process that performs these actions using manually developed policies. We select actions using a state machine that encodes the underlying structure of the observed motion. We learn the number of states and transitions within the machine automatically and jointly with actions for states (encoded through control parameters) using incremental optimization. Estimated state transitions in the machine are conditioned on distinct physical events that happen in the simulated physical world, such as a change of foot support or body contact. This modeling option promotes the success of learning from noisy observations, as is our case, because it encourages recovery of longer-term stable actions. As such, encoding of the controller is sparse and the learning of parameters for actions is reinforced by observations from many frames. The conditioning of transitions on physical events also improves the ability of the learned controller to generalize beyond the single motion demonstration, when the learned events happen at different time instants. Our parameterization of desired motion for the character using a sparse set of key poses have been previously used in graphics [70, 213, 203] and robotics [3]. Effects of these sparse parameterizations on the ability of experts to demonstrate motions to robots in kinesthetic learning of robot arms have been studied in [3]. Unlike the previous work [64, 29] in learning state machines from demonstration, we do not recover a distribution over state machines but a single state machine that best fits the observations.

2.2.4 Markov Decision Processes

We now consider a different class of learning algorithms that explicitly utilize temporal ordering of state-action pairs in the demonstration, account for the dynamics of the system state in the process of learning of the policy function, and permit multiple valid actions to be paired with the same state in the demonstration. These algorithms are usually formulated within the context of Markov Decision Processes that we overview next.

Markov Decision Process. A Markov Decision Process models decisions in a dynamic system with a set of states, actions, transitions and rewards. At any time instant, the system resides in one of the states from the state set and transitions to a new state by taking an action from the action set, where decisions to take particular actions are motivated by rewards (scalar numbers). The problem within MDP is that of deriving a policy for selecting actions, as a mapping function from states to actions, such that the cumulative reward associated with the actions selected by the policy is maximized if the system runs for a long time.

In formulating real problems as MDPs, the goal is to encode all contextual information that may be relevant for the optimal selection of the action for the task at the current time (and in the future) in the state vector, such that the decision can be made locally, and only based on the current state. Assuming we can model the contextual information for the task we want to learn, the policy function can be learned from training data that demonstrates optimal decisions made by the expert in typical situations. Note that the optimal policy does not necessarily choose actions with highest immediate rewards and instead considers future consequences of the actions taken.

Example. Many practical problems of interest can be phrased as MDPs. Perhaps the closest field that uses these models, in a way that relates to our work, is computer graphics, where MDPs have been used to model decision processes for humanoid characters in virtual environments.

Consider a motion synthesis problem of stitching clips from human motion capture such that the modeled character walks to a target (we discuss this problem in more depth in Section 2.3.3). This problem can be formulated as an MDP using state-actions pairs, where (1) states encode potential contexts for the character (pose, motion, obstacles, target location), (2) actions represent possible clips that can be replayed and result in the change of the state, and (3) rewards associated with the pairs encode how beneficial it is to select a particular clip in a given state, such that the character moves towards the target. Given this abstraction and assuming an initial state, the problem of the motion synthesis is to optimize for a sequence of actions in the MDP such that the overall reward is maximized. While the optimization can be performed through offline search, for each initial state of the character and the target location, it is often not practical because the generated motion can not directly adapt to changes in the environment and the target location (without reoptimization). As an alternative, one may attempt to learn what optimal motion clips are typically replayed when the character is in a particular state [179] and use this information for the synthesis in place of the offline search. The goal of learning from demonstration is to learn policy functions that can make these informed decisions.

Policy functions for MDP problems are typically learned by either reinforcement learning or inverse reinforcement learning, depending on whether rewards for actions are known or not.

Reinforcement Learning. Reinforcement learning derives policy functions for MDP problems with known rewards for demonstrated state-action pairs. Because the learning essentially involves precomputing optimal action sequences for all possible state combinations, the process requires some discretization [38] or approximation [179] of the state-action space so that the optimal policy can be learned and stored in the computer. Finding an appropriate discretization is a key challenge for applications in graphics, because state parameterizations are usually continuous [179]. In robotics, it is often assumed the space of actions is already discrete such that actions encode basic motor commands for the actuation of a robot and, furthermore, the number of possible actions is relatively low [165], which simplifies learning.

Inverse Reinforcement Learning, Inverse Optimal Control. Specifying appropriate rewards for actions in a MDP can be non-trivial [8]. In fact, in many real-world cases, an expert may be able to demonstrate what actions should be taken in specific situations while actually being unable to quantify the rewards (say which action is better and by how much) and tell the motivations behind the particular decisions they have made. As such, the demonstrated experience may very well depend on the expert's intuition. Consequently, for practical problems of interest, demonstrations often do not reveal the intent/objective of the motion and this intent (encoded through rewards) has to be determined by the learning algorithm.

The problem of learning rewards from demonstration is known as *inverse reinforcement learning* [103, 101]. It is an instance of a more general inverse optimal control problem, where the goal is to learn the objective for control. The problem is very difficult and only a few advances have been made so far. Current methods for inverse optimal control model the unknown rewards for actions as either linear [144, 88, 214, 130, 101] or non-linear [103, 101] combinations of feature vectors extracted from the actions, where the weights are optimized by the learning algorithm. Weights for the features have been optimized in the literature either using methods based on large margin structured classification [144], using principle of maximum entropy to resolve ambiguities in the weights [214, 101] or genetic algorithms [130]. In robotics, inverse optimal control has been used to plan foot steps for a quadruped robot in challenging environments [144, 88], grasp objects by a robotic hand [144] and navigate car drivers [1, 214], among others. Until recently, however, methods for inverse optimal control required discrete state-action spaces. Motivated by the needs in graphics, Levine *et al.*[101] uses local approximations of reward functions to learn rewards for high-dimensional continuous actions and demonstrates it is possible to learn driving styles for cars in physics-based 2D driving simulations using their method.

2.2.5 Applications in Humanoid Motion Synthesis

Learning controllers for tasks from demonstration is a very appealing paradigm. It provides an automated way to determine general control policies without having to manually program the control systems. Furthermore, by having the systems learn the policies, the systems are forced to understand the underlying objectives of the demonstrated behavior(s). Consequently, the learned control mechanisms may provide meaningful generalizations when they are applied to systems in new situations that are substantially different from the demonstrations. These properties are beneficial for applications in *e.g.*, human motion synthesis in computer graphics, where one may represent demonstrated motions for the character using controllers. Such a representation of motions may be particularly helpful for human motion capture, as motivated in Section 1.3, because it may allow to naturally adapt the observed motions to new environments and perturbations.

With that said, the current general methods for learning controllers are often limited to discrete control spaces with a small number of dimensions, making them difficult to apply to problems in human motion synthesis. In order to keep the dimensionality of the space low, one typically has to carefully engineer the structure of the state-action space (and the set of features for actions when using inverse optimal control methods), as required by the particular task. Specifically, one has to decide what information is going to be stored in the state, what contextual information may be needed for decision making for the task and what actions can be performed by the character. Given these difficulties, the general learning methods have to date only been applied to selecting kinematic motion clips for characters from prerecorded motion databases [179, 97, 102] (see Section 2.3.3) and self-balanced step-actions for simulated physics-based characters from a motion library [38].

2.2.6 Relation to Our Motion Capture Approach

Our formulation of human motion capture through controller capture from video (see Section 1.5.1) relates to general learning from demonstration. At the core, the underlying problem is the same learning of controllers, except that observations, in our case, come from images. Despite this high-level similarity, we do not formulate controller learning within the learning from demonstration framework. Our approach to controller estimation utilizes optimal control principles previously explored in computer graphics, assumes a physics-based humanoid controller model from Section 2.3.5.2 and recovers the policy function using direct optimization. We describe research in physics-based controller design that relates to our work in Section 2.3.5.3. We characterize our approach by contrasting it to the general controller learning paradigms from above and provide motivations for pursuing our specific formulation below:

- State-action space:

  Current methods for learning controllers within the learning from demonstration framework require state-action spaces with relatively small number of dimensions, have difficulty handling continuous parameters in states and actions and require manual crafting of states, actions (and features for rewards in the case of inverse optimal control) in order to accommodate the particular task to be learned (see Section 2.2.5).

However, actions in our control model have many dimensions and all parameters are continuous. We also do not know which task we are going to observe and our motion capture approach needs to work across all tasks that we may want to capture. Consequently, we need to use a method with a control model that is not specific to any particular task. In our method, we use actions that are learned for the observed task (*i.e.*, they are not taken from a predefined motion library) and the learning proceeds incrementally as video frames arrive.

- Demonstrations:

  The general learning methods based on the learning from demonstration formulation typically require many demonstrations of the same behavior (with necessary variation) in order to learn a meaningful policy for the task. In contrast, our system has a chance too see only one such a demonstration and learns the controller from this only motion instance.

2.3 Motion Synthesis

We estimate controllers within an analysis-by-synthesis framework with the synthesis component realized using simulation and control. We implement the analysis component through optimization that we constrain by constraints extracted from the images. In implementing the synthesis component, we use methods for synthesizing physically realistic human motion from graphics and methods for humanoid control from robotics. We review these methods and related work here.

Motion Synthesis. In this section, we outline methods for human motion synthesis. We assume the desired motion for the synthesis can be abstracted using motion objectives that encode various constraints on the motion of the character and survey methods from graphics and robotics that produce the motion that meets these objectives.

The objectives for the synthesis can take many different forms and control various aspects of the motion. The objectives also characterize the desired motion at different levels of detail and abstraction. For example, in one case, the objectives may just specify a start and an end point for the character in the environment. In another case, the objectives may prescribe the poses that the character should approach at specific key time instants and when toes or feet should stay in contact with the ground. This form of encoding of desired motion is appealing, because it allows to characterize key aspects of the motion while leaving unimportant details unspecified and to be determined by the motion synthesis process automatically. The objectives can be defined either by a user or generated programmatically.

Given the objectives, the goal of motion synthesis is to produce a pose sequence such that (1) the generated *motion is plausible* with respect to what pose sequences are typical/natural for the character, as dictated by an underlying motion model of the character, and (2) the *motion meets the specified motion objectives*. At a high-level, we can produce motions using either data-driven kinematic methods or physics-based methods.

Data-driven Kinematic Methods. One of the typical ways to generate realistic animations in computer graphics that meet specified objectives is to edit, blend and stitch clips from a prerecorded database of human motion capture data. This data-driven process usually consists of two steps: (1) building a data structure to organize clips in the motion database and (2) using the data structure to search for an optimal sequence of clips, that meet the objectives and is plausible, and assembling the clips into the final motion. The data structure, built offline, represents the clips as well as records the possible ways how the clips can be stitched and blended into new motions. Organizations into graph-like structures abstract this concept naturally. These structures can be either searched for optimal motion sequences during runtime through planning (see Section 2.3.2) or such searches can be "precomputed" by learning controllers (see Section 2.3.3).

Physics-based Methods. Kinematic data-driven models from above can only synthesize motions that look human-like as long as they remain close to the sequences recorded in the motion database. In practice, it means that either the repertoire of the skills that the character can perform is limited (because not all possible skills and motion executions in all possible environments can be recorded) or physical plausibility of the synthesized motion is compromised, resulting in visual artifacts. Some of the common artifacts are sliding of feet on the ground and maintaining of poses that are not dynamically balanced. These artifacts are usually caused by the blending of motion clips that have different contact patterns and by applying the recorded motion to characters in non-compliant environments, such as when there is clutter on the ground that was not present in the scene at the time of the capture. While some of these most apparent problems can be mitigated by *e.g.*, fixing or correcting the position of the contact foot based on geometric constraints imposed on the body by the environment using inverse kinematics, there is no guarantee such resulting motions would obey physical laws. A more principled answer to these problems may be the use of physics-based methods that account for the physical laws directly through simulation or optimization.

Physics-based methods produce new animations of characters through physical simulation of the body of the character effected by forces or by optimizing for the forces. As such, they generate motions that explicitly follow Newtonian principles, that are not restricted to copying pre-recorded kinematic data and that properly comply with the environment. The generated motions are fully determined by the initial pose of the character, its initial velocity, the forces exerted on the character over the course of the simulation and the employed simulation/dynamics model. These models can be arbitrarily complex, ranging from simpler rigid body dynamics approximations of the human body to biomechanically correct musculo-skeletal models [126] that model the character as a rigid human skeleton actuated by real muscles. Although some of these models can be easier to simulate than others, given the forces, the real challenge comes in the computation of the internal actuation forces that will produce the desired motion of the character such that all specified motion goals are met. As such, the major burden in building physics-based animation systems is modeling actuation of the body. See Section 1.4.2, Section 1.4.3 and Section 1.5.2 for an additional overview of physics-based modeling of human motion and why modeling of actuation forces is difficult.

As demonstrated by previous research in computer graphics, it is feasible to determine actuation forces for rigid body based models. In those cases, actuation forces can be obtained either through offline search (see Section 2.3.4), by building reactive physics-based controllers for the motion or by combining motion planning with reactive control. We next survey particular methods for the motion synthesis.

2.3.1 Planning

Many methods for human motion synthesis in graphics can be phrased as a deliberative planning for actions of the character model such that the specified motion objective is met. The origins of these algorithms can be traced back to robotics, where they have been used to plan for the motion of articulated robots in environments with obstacles [86, 91, 83]. There, planning was usually phrased as a search for paths in the high-dimensional configuration space of the robot and the goal was to develop algorithms that could explore these spaces efficiently. Motivated by the problem of finding a path for the robot from the start location to the end, Kuffner *et al.*[91] explored the space by sampling random configurations for the robot within the proximity of obstacle-free configurations and recorded the discovered reachable configurations into trees. Kavraki *et al.*[83] used a similar idea to precompute a "roadmap" for the entire space and used the map for path planning. Khatib *et al.*[86], on the other hand, uses local planning based on potential fields to attract the robot away from nearby obstacles. Similar ideas have been reapplied later within the context of computer graphics.

In the context of realistic human motion synthesis in graphics, planning has been applied to configuration spaces of humanoid characters performing actions. Actions typically consist of either shorter motion clips (see Section 2.3.2) or actuation forces (see Section 2.3.4). Planning is complicated in this case, because aspect of body contact with environment have to be accounted for. For example, the planner must be aware of when feet are in contact with the ground and plan for the motion of the torso accordingly such that the resulting motion will not make the feet slide on the floor. The planning is often prone to local extrema in the optimization objective, such that the resulting motion does not meet all the objectives and violates some constraints.

2.3.2 Motion Graphs

In this part, we explore kinematic data-driven methods that generate motion by stitching motion capture clips. These methods work by first preprocessing training motion capture data in order to identify usable motion clips for stitching, organize the clips into data structures such that the clips can be found efficiently and then use the data structure to plan for the motion of the character.

Perhaps one the first uses of graphs to organize kinematic motion, motivated by the needs in the video game industry, was demonstrated by Miziguchi *et al.*[120]. In order to realize real time animation of interactive characters in the game, Miziguchi *et al.*captures motion for the character in all possible situations that can happen in the game. The captured motions are then manually segmented and organized into constituent actions that the character can perform and this organization is represented by a graph. In this graph, nodes correspond to the actions and edges record possible transitions between them. Transitions are equipped with interpolation information that needs to be applied on the previous and the new motion segment in order to produce a seamless blend and allow for a naturally looking transition. While effective, the repertoire of possible transitions is restricted to exactly those defined by an animator when building the structure and the transitions can only happen at the designated time instants. Consequently, the character may be slow to respond to changes in the desired motion (goals) and the range of motions that can be synthesized is limited.

Motion Graph. To address these problems, Kovar *et al.*[90] builds a structure called *motion graph* from motion capture data automatically that permits any two motion segments (portions of motion clips from the database) to be assembled into a longer motion as long as the frames near the seam look similar. Because the opportunities for seams are searched for automatically, the resulting structure is more rich and allows to switch between clips more often, resulting in the ability to generate motions with greater variability. Unlike the previous data structure [120], motion graphs organize the motion in a slightly different way. Nodes in the motion graph correspond to the seams and directed edges represent the segments that can be pasted. Consequently, paths in the motion graph define possible segment concatenations. New motions can be synthesized by optimizing for a path in the graph that satisfies given motion objectives, by *e.g.*, a branch and bound algorithm to deal with the complexity of the graph.

Variants. Computationally more efficient variants [9, 10, 89], variants employing parametric compact representations of motion clips [68] and extensions capable of generating motions beyond the motion database [150] have been introduced later. Arikan *et al.*[9] builds a hierarchy of graphs to search at different levels of detail using a randomized search algorithm to reduce computational costs. A dynamic programming based search has been employed in [10]. To produce compact graph representations that are faster to search, parametric motion graphs [68] have been developed. In this parametric representation, nodes parameterize families of logically similar motion clips and edges encode possible transitions between parameterized clips in the families. Other extensions have focused on improving the ability of the method to synthesize motions that were not seen in the motion database. Producing new motions for characters is difficult because the character needs to properly locomote/maintain contact with the ground such that the motion will appear sufficiently believable. Ad-hoc blending of different motion clips usually results in motions that suffer from artifacts, such as sliding of the feet on the ground and motion of the torso that is not properly supported by the legs. Safonova *et al.*[150] addresses this problem by only interpolating motion segments that have similar foot contact patterns. Other methods [179] interpolate motion only between foot contacts in order to prevent slippage.

We now consider the second approach for motion generation, where the logic of selecting an optimal path in the motion graph is replaced with interactive control of the character. The purpose of control is to make the decisions about what motion clips should be played next interactive, as the motion progresses.

2.3.3 Kinematic Controllers

Based on the learning techniques from Section 2.2, the process of searching for an optimal path in the motion graph can be learned. Treuille *et al.*[179] and others [97, 98] learn *kinematic controllers* for common navigation tasks for characters in game environments. Learning corresponds to precomputing optimal ways to achieve specific goals for characters in particular situations and is typically performed offline by *e.g.*, reinforcement learning [179, 97]. The result of the learning process is a policy function for a kinematic controller for the character that selects new motion segments online and in response to the current state and goals of the character. Specifically, because the segments in the sequence are not pre-planned, the synthesis process can respond to immediate changes in the motion objectives (*e.g.*, direction of walking for an interactive character) and external disturbances during the synthesis.

A key challenge in building policy functions is (1) determining the repertoir of motion segments for the character, (2) finding contextual properties that are relevant for making decisions in control, and (3) encoding these properties together in a low-dimensional space so that an appropriate policy can be learned. Given the number of possible motion segments, motion goals and states that the character can be in, learning the policy function and storing it in memory is difficult. To address the challenge, the state space for control has to be crafted in a specific way in order to accommodate the task, enable low-dimensional encoding and facilitate learning.

For example, in order to learn a policy for a character that can navigate an environment with obstacles with a main goal to reach a target location, the encoding of the state vector may need to incorporate at least the pose of the character, the direction towards the target, and a vector towards the closest obstacle [179]. Using these encodings, the decision process may have enough contextual information to select optimal motion segments based on the situation that the character is currently in. The learned policy will try to select motion segments in order to achieve the global goal (moving the character towards the target) as well as to react to local immediate goals (avoiding obstacles). The training data may consist of recordings of an expert navigating the character through common environments in this case [179].

Kinematic Controllers. Treuille *et al.*[179] learns controllers for interactive characters in video games that can walk, run, turn and avoid stationary or moving obstacles while responding to user inputs in real time. Lee *et al.*[97] uses a similar approach but can accommodate a wider variety of behaviors. Towards that end, a more efficient encoding of the policy function is needed such that it can still be kept in memory and yet provide a detailed enough approximation that allows for making optimal decisions.

To make characters more responsive to changes in motion goals (*e.g.*, desired heading, disturbances), Lee *et al.*[98] organizes the motion database to a structure called the *motion field*. This structure does not organize motion segments but body configurations of the character and uses changes in body configurations between frames as actions for control. Given a query configuration of the character, the structure allows to enumerate all neighboring configurations in the database. Each found configuration can then be used as an action for control that brings the character from the query configuration to this new configuration. Using this representation, the goal is to learn a controller that switches between neighboring configurations of the body, as opposed to motion segments, making the controller more responsive and the character free from following edges in the motion graph. The controller directly chooses a new configuration for the character for the next frame and these decisions are made with every frame. This form of control allows the controller to produce a motion from any initial configuration of the body and the produced motion is not limited to pasting segments from the database. Furthermore, body configurations can be interactively altered by physical simulation or inverse kinematics to model external disturbances. At the same time, the information from the motion database can be used to make the character "recover" from such disturbances, by naturally attracting the configuration of the body towards a recorded motion using kinematic control. As before, the policy function of the controller is learned using reinforcement learning but it needs to be substantially compressed in order to allow control at this detailed frame-configuration level.

Kinematic Controllers and Planning. Levine *et al.*[102] combines controllers [179] for specific motion skills such as jumping or obstacle avoidance with online motion planning in space and time. This planning allows to implement navigation for characters in complex dynamic environments. A space-time planner is used to determine which controllers must be executed and in which order so that the character reaches a given goal destination. As the character performs the planned motion, it can properly react to the changes in the environment and, when possible, make use of such changes to its advantage in order to follow a more cost-effective plan. For example, when certain changes in environment can be anticipated, such as a regular movement of platforms, cars or trains in a video game, the planner is able to decrease the travel cost for the character by having it jump on the platforms and the cars. The proposed planner, based on the A^* algorithm, can find paths in space and time even if no valid path from the initial configuration to the target configuration exists in any static snapshot of the environment.

We next consider methods for motion synthesis that utilize physical laws and physical simulation. These methods computationally account for physical aspects of the motion, as opposed to modeling them indirectly by relying on motion capture data. We first consider a physics-based method that searches for forces to be applied on the character model such that specified motion objectives are met.

2.3.4 Space-time Optimization

Space-time optimization method proposed by Witkin *et al.*[203] generates human motion by assuming a physics-based model for the character and optimizing for forces to act on the model such that the desired motion is produced.

The goal of space-time optimization is to search for a motion that (1) complies with the equations of motion of the given physical model, so that the motion looks plausible, (2) satisfies animation constraints (motion goals), so that the desired motion is produced, and (3) minimizes a certain overall objective, such as an amount of "muscle" exertion/energy spent performing the motion, so that the motion is fully determined. Given the overall objective, the method discretizes the equations of motion over frames and simultaneously optimizes for the states of the character at the individual frames as well as the forces that need to be applied on the character between the frames in order to attain those states, subject to physical constraints and animation constraints.

Here, physical constraints request that the changes of the character states between frames are consistent with the discretized equations of motion and the estimated forces. Animation constraints correspond to the animation goals specified by an operator and may include a desired initial pose, a desired end pose and desired/target poses at other specific time instants. These constraints can be sparse, such that they are specified for only distinct time instants. In particular, one does not need to specify desired poses for the character for every frame in the sequence, which distinguishes the method from direct inverse dynamics. Because the optimization is performed both in space and time, the optimization objective can determine the motion between constraints automatically.

Basic Optimization. Given the constraints, the original space-time optimization [203] and its more recent variants [138, 50, 151, 108, 116, 201] aim to automatically find a way to perform the desired motion such that the constraints are satisfied and the overall optimization objective is met. However, the optimization is expensive and can only be applied to offline characters. Because automatic handling of contact requires non-linear constraints and produces non-linear motions, that can not be handled by the optimizers, the method also requires manual specification of contact configurations for the feet of the character in frames in order to succeed [108, 31, 201, 202]. Manual specification of contact constraints fixes the contact configurations for the body and effectively turns linear-complementarity contact constraints (see Section 3.2.2.4) to equalities that can be understood by the current optimizers.

VideoMocap. Wei and Chai [201] introduces a VideoMocap system, where the space-time optimization method [203] is applied to obtain realistic interpretations of motions of subjects observed in annotated videos. Because video observations (or pose reconstructions from single frames) alone do not provide reliable constraints for the space-time optimization (*e.g.*, because the pose reconstructions from different frames may conflict when used as targets for the optimization), the proposed method relies on manual annotation of the frames in the video. Towards that end, an operator needs to specify target poses for distinct frames in the video as well as contact constraints for the limbs. The method uses automatic 2D pose tracking in the image plane to determine additional pose constraints for the inbetween frames such that the reconstructed motion will be forced to imitate the style of the motion observed in the video. In contrast, our motion estimation methods do not rely on manual intervention in specifying the target poses for the character and we also do not rely on manual specification of contact or foot placement constraints, as here [201] or in other methods [202, 108] based on space-time optimization.

Optimization with Style. While direct space-time optimization allows to describe many motions using only sparse sets of constraints, many stylistic variations can not be described in this simple way. Specifically, stylistic variations in motion executions for low-energy activities, such as walking, are difficult to generate purely from physical constraints and simple optimization objectives that *e.g.*, only minimize the amount of the actuation force. For example, “sneaky” walking is more a result of the person’s own preferences than of the principles of optimal human walking, biomechanics of the human body and physical dynamics. Consequently, in order to constrain the search for motions with these simple objectives to the desired style, one needs to specify desired poses for many frames, making the approach less practical.

To address this challenge, Liu *et al.*[108] takes a biomechanically inspired approach and assumes that any walking style results from differences in the underlying actuation model of the body and is optimal with respect to that model. It is assumed that the actuation model is parameterized and a given style can be produced using standard space-time optimization that minimizes the amount of the actuation force by choosing the right parameters of the actuation model. The parameters of the actuation model encode personal preferences over which “muscles” to use to actuate the body, as well as joint stiffnesses and other biomechanical properties of the body. The actuation parameters for a given style can be learned from a short reference motion sample automatically. Once learned, the actuation model can be used during space-time optimization to produce new motions with the demonstrated style.

Wei *et al.*[202] combines data-driven methods for modeling human motion with space-time optimization. Instead of changing the actuation model to only produce motions with a given fixed style, as was the case with the previous approach, the proposed method [202] uses a generic model that learns what force trajectories are typical for the motion styles observed in the given motion database and attempts to produce motions that mimic the observed styles. Towards that end, the energy minimization objective within the space-time optimization framework is replaced with the one that favors actuation forces that are both typical and that produce motions that are closer to the sequences in the database. Typical forces are modeled using a data structure called the *force field*. This structure consists of a Gaussian Process based model that learns from the motion database what actuation forces are typically applied at the character in a particular state. New motions are synthesized from the force field using space-time optimization with a new objective that maximizes consistency with the specified animation constraints, applies forces close to those predicted by the force field and penalizes motions that violate the laws of physics. Because the optimization is strongly biased to find physically valid motions, consistent with the optimized forces, the optimizer can produce naturally looking motions even when the environment changes from the one used for training or when the character is perturbed by external disturbances.

Discussion. Despite these successes, space-time optimization has downsides. First, space-time optimization only gives forces valid with respect to the particular discretization of the equations of motions and an optimization instance with a given set of animation constraints. The method can not guarantee that the optimized motion can actually be reproduced by a physical simulation with the obtained forces and there is no direct way to adapt/generalize these forces for changes in the environment or perturbations. Furthermore, when the set of constraints changes after the motion has been optimized, the entire motion has to be reoptimized. Next, none of the methods based on space-time optimization to date can automatically optimize for foot placement. Consequently, desired changes in the contact configuration of the character (when contact forces should and should not act on the feet, with respect to the environment) have to be manually specified as constraints for the optimization, disqualifying the method for automatic applications in computer vision. Lastly, the optimization can only operate in a batch mode and, as such, can only optimize sequences with a limited length. In contrast, our methods do not require operator intervention in specifying contacts and can run online.

2.3.5 Physics-based Controllers

As an alternative to space-time optimization, one can produce actuation forces for the character using a *physics-based character controller* and apply the forces on the character in simulation.

Physics-based controllers are designed to imitate particular motion skills by computing the actuation forces that reproduce the skills in simulation. Actuation forces are computed using a control policy function that reacts on the current state of the character and the environment and outputs the forces that should be applied on the body of the character in this particular state in order to meet animation goals and achieve the desired behavior. As such, the policy function actively controls the simulation and the motion of the character. Because actuation forces are applied online during the simulation, and in response to the changes of the body state and the environment that happen in this simulation, the control can directly react to changes in the environment and the animation constraints as well as respond to external perturbations. Analogously to kinematic control, physics-based controllers replace offline search for actuation forces with the online application of the control policy function and simulation. Consequently, given a motion skill to be reproduced, the key problem in motion synthesis using physics-based control is learning/building an appropriate policy function for the skill.

Building a control policy function is difficult. Such a function depends on many input variables, including dynamic state of the body, environment, corresponding body contact configuration and animation goals. The actuation forces that it outputs are used to actuate individual parts of the body and must be computed in such a coordinated way so that (1) a desired motion of the entire body is achieved and the motion skill is reproduced and (2) the body is kept balanced. Consequently, the relation between the control inputs and force outputs, that are both high-dimensional and continuous, is complex. As such, the function is usually not built manually nor learned directly. Instead, various algorithmic procedures and control strategies based on constraint solving or optimization have been employed in the literature to implement the computation within the function.

Passive Dynamics Controllers. For special cases of motions, surprisingly simple control models based on passive dynamics have been built in robotics. McGeer [117] illustrates that simplified humanoid walking can be realized by a passive dynamic walker model that can walk stably on a slope solely under the effects of gravity without any actuation or control. Kuo *et al.*[93] extends the model by a simple actuation mechanism that powers the legs using a spring and applies impulses to toes when leg support transfers in order to produce an energy efficient gait on the level ground. Collis *et al.*[37] demonstrates that it is possible to build detailed humanoid robots that can walk and turn without the need for precise control. These robots still exploit passive dynamics principles, similar to humans, in order to generate motions that are energy efficient and appear human-like, but substitute small active power sources for gravity so they can walk on the level ground. The robots can walk successfully by only adding small amounts of power at ankles or hips.

Powered Controllers. For general full-body motions, more elaborate control that powers the body may be needed to produce the desired motion. In these cases, control is typically realized as a two-level procedure. At the high level, the controller synthesizes target poses for the character, considering the desired motion, balance and environment constraints. At the low level, low-level motion control process within the controller generates actuation forces to approach the target poses (see Section 1.4.3.1). The current strategies differ in the ways how the *desired motion* (animation goals) is specified for the controller and how this motion is adjusted to maintain balance of the character. Next, we review control strategies based on how the desired motion is defined.

2.3.5.1 Unstructured Desired Motion

One way to define the desired motion is to employ a data-driven approach and characterize the motion by an unstructured sequence of desired 3D poses that the character should follow. The sequence prescribes the target pose for the character for every frame in the simulated sequence and the goal of the controller is to track these poses as closely as possible while maintaining balance [170, 209, 40, 96, 125, 109]. The target poses can come directly from reference motion capture data, in which case the controller attempts to "replay" the captured motion in a way that will comply with the simulated character and the environment, or they can be synthesized from the motion capture data *e.g.*, using kinematic techniques from Section 2.3.2 and Section 2.3.3. In the latter case, the controller will "filter" the kinematic motion such that the resulting motion produced by simulation will be consistent with physical laws and the constraints imposed on the character by the environment. We refer to this class of controllers as *data-driven tracking controllers*.

Because the reference motion to be tracked by the controller usually does not match the dynamics of the simulated character model exactly (*i.e.*, the simulated body approximates the captured human body to only a certain extent), a form of balance has to be incorporated to recover from the discrepancies. Towards that end, data-driven controllers are usually decomposed to two components: a balancing controller and a tracking controller.

Overview. The purpose of the balancing controller is to plan for the motion of the character through contact changes such that the character will stay balanced while tracking the reference motion. This planning is made difficult by the discontinuities in velocities and changes in motion that happen when contacts emerge or vanish. In fact, just making the character stand still upright on a spot may pose a significant challenge, especially when the character has too many degrees of freedom to control and/or the character is subjected to strong external disturbances (*e.g.*, pushes). Consequently, the motion plan has to be continually re-evaluated and updated to the current situation such that appropriate corrective actions are taken by the character. To reduce some of these challenges, for practical reasons, the motion planning is usually implemented on a simplified lower-dimensional model of human dynamics, for which efficient balancing strategies can be found. Given the planned motion of the simplified model, the tracking controller optimizes for the actuation forces to be applied on the full-body model in order to track the target poses while keeping the dynamics of the body consistent[7] with the low-dimensional motion plan of the simplified body. Actuation forces for the full body are usually obtained through quadratic programming [40, 209].

Balance. Lower-dimensional inverted pendulum models [40, 125, 181] with dynamics modeled by a linear quadratic regulator [40] or non-linear quadratic regulator [125] are typical model choices for approximating humanoid balance. For walking, da Silva [40] linearizes non-linear walking dynamics into linear segments that switch on foot contact changes. Each segment in the motion corresponds to a distinct phase in the walking cycle where the body is supported by a single foot. Balance for each such segment is then approximated using a low-dimensional inverted pendulum model and the motion of the full body is constrained to be consistent with this model. Mordatch *et al.*[123] models lower-dimensional dynamics using a spring-loaded inverted pendulum model to build controllers that can traverse challenging terrains and react to strong external disturbances. Macchietto *et al.*[113] maintains character balance by simultaneously tracking desired values for the angular and linear momentum of the full-body model as well as the specified target poses. Lee *et al.*[96] dynamically modulates reference pose trajectory for the lower body in order to compensate for the possible loss of balance. Here, heuristic rules based on the position and velocity of the center of mass of the character, similar to [213], have been used to adjust the desired joint angles of the legs. Other, more conservative, methods for humanoid balance have been applied mostly in the context of robotics. There, *zero-moment point strategy* [195] is typically used to restrict the motion of the robot such that its center of mass is always kept above the center of pressure or within the support polygon.

Discussion. The key benefit of data-driven tracking controllers [40, 209, 113] for humanoid control is the ability to reproduce particular style in the generated motion. In particular, because the target poses are specified for every frame in the sequence, the produced motion is not only physically plausible, but also the

[7]For example, one may want to constrain the center of pressure (COP) of the full-body model to coincide with the COP of the simplified model, as in [209].

subtleties and style of the produced motion that come from the reference motion capture data are directly reflected in the motion. With pure data-driven methods [209], however, it is often difficult to handle large deviations from the desired motion, through *e.g.*, perturbations or unanticipated impacts with the environment, and changes in the environment geometry. A notable exception is the approach of [109] which allows physics-based reconstruction of a variety of motions including: walks, cartwheels, and kicking, by randomized sampling of controls within user-specified bounds. As a result, the original motion capture trajectories are effectively converted into open-loop control.

We now review another class of controllers that assume the desired motion has a specific underlying structure (*i.e.*, actions and transitions) and exploit this structure in the model.

2.3.5.2 Structured Desired Motion

Reactive Control. *Model-based methods* generate motion of the character by making use of models of motion specific to given skills or activities. These models assume a structure for the desired motion and utilize this structure for control. A typical example of this approach is modeling of stable gait patterns for walking in robotics [73, 80, 93] or computer graphics [213, 123, 38], where cyclic walking is modeled as a sequence of distinct motion phases that switch by reacting to the change of foot support.

Sparse Parameterization. Model-based approaches characterize the desired motion in terms of its structure, that may encode the phases in the motion, and sparse target values for the controller to track. Unlike the previous data-driven methods, the parameterization of the desired motion in model-based methods is sparse, such that the target values are specified for only specific time instants (key frames) in the motion, allowing greater flexibility for the controller to move the character between the targets. The values to be tracked can include target poses [70, 213], target position for the center of mass [43] or target values for the angular momentum [43]. We are particularly interested in controller models that model the structure of the desired motion using finite state machines, as described in Section 1.4.3.1. These models are generally known under the name *state-space controllers*, suggesting the control is implemented by switching states in the machine. The states can switch either based on manually defined transitions [70, 213] that are taken in response to simulation events, based on events and an optimized motion plan [43], or using a learned policy function [38].

State-space Control. Hodgins *et al.*[70] demonstrated state-space controllers to be effective in modeling variety of motions, including walking, running, vaulting and diving. Methods for augmenting such models with balance control [213, 43] have also been introduced, making them less reliant on environment and terrain. These controllers have also been shown to be composeable [35, 38] and even capable of encoding stylistic variations [199] through use of different parameters.

Balance in these methods is usually achieved by dynamically modifying the target values using heuristics based on simulation feedback [213] or by directly optimizing for the actuation forces that meet both tracking and balancing goals, as in [43]. Yin *et al.*[213] implements a form of implicit balancing, where the controller attempts to maintain a desired world space orientation of the root segment and upper legs, as dictated by the current state in the machine. The desired pose is further heuristically adjusted according to a balance feedback from the simulation in order to improve the ability of the character to balance. De Lasa [43] implements a different balancing strategy, where desired joint angle values, positions of end effectors, the value of the angular momentum and the position of the center of mass are tracked by the controller while keeping the body balanced. Actuation forces for control are obtained from motion constraints using a prioritized optimization that favors satisfaction of balance constraints over the pose tracking constraints. Because the desired behavior of the character is encoded by constraints, the same control mechanisms can be applied to characters in new situations, characters with modified body models and/or mass distributions as well as modified geometry.

Our approach to motion control falls within the category of reactive state-space controllers. The main benefit of using this class of models is the compactness of their representations, robustness to external disturbances and ability to generalize to new situations.

Controllers and Step Planning. State-space controllers are often combined with online planning [123, 38] in order to build more robust task-specific control policies for responsive characters that can navigate challenging terrains based on user commands and in response to perturbations. Mordatch *et al.*[123] plans for the gait, footsteps and transitions of a lower-dimensional body model that responds to the commands. The produced plan is tracked by a full-body controller and is continually refined based on the simulated full-body dynamics. Coros *et al.*[38] precomputes a high-level control policy for the character using reinforcement learning to select actions (steps) online that let the character move efficiently towards the goal. In robotics, planning methods based on zero-moment point strategy [195, 80] are used to deliberately plan placement of feet so as to preserve dynamical stability. Yagi *et al.*[208] varies step length and clearance of the swing leg and switches between forward-stepping, side-stepping and stepping on and off an obstacle as obstacles are discovered in order to navigate cluttered environments. Kuffner *et al.*[92] plans for statically-stable footstep locations for a humanoid robot in a known littered environment via dynamic programming such that the number and complexity of the step motions of the robot is minimized.

2.3.5.3 Controller Design

Building or adjusting existing controllers to produce motions with a particular desired style that *e.g.*, matches a given reference motion has been shown problematic.

Previously, controllers for physics-based humanoid characters have only been built manually, following the state-space control paradigm, by relying on tedious manual tuning of parameters, as well as manual specification of the controller structure. A few recent methods addressed automatic parameter estimation through optimization [198, 199, 212], but only for short sequences with manually defined optimization objectives to specify the desired motion for the character and close enough initializations for the optimization. Using this approach, a controller for walking was optimized in Wang *et al.*[198] by specifying the structure for the controller and an optimization objective that characterized walking. The objective was constructed by looking for various aspects of human motion that are specific to walking, such as regularity of the gait, stability of the upper body, energy expenditure, *etc.*, and encoding them in the objective function such that motions that violate these principles are penalized in the optimization. While this approach worked for walking, it is not clear what the underlying biomechanical principles would be for many other motions. Consequently, it is not feasible to manually design objective functions that produce given desired motions and with a particular style. Furthermore, all of the previous methods for building controllers assumed the structure of the motion was already given and coded manually for a given behavior.

Our Methods. In our formulations, we use images to demonstrate what the desired motion for the character should look like. We use a general objective function for optimization that is not specific to the motion and we automatically optimize both the structure as well as parameters of the reactive state-space controller to reproduce the motion that was observed (see Chapter 5). In doing so, we do not require manual initialization of the optimizer and our method is able to recover controllers for long sequences.

Chapter 3

Physical Model Of Human Body

Motivated by recent successes of physics-based modeling of virtual characters in computer graphics, we model the body of the person as a collection of rigid bodies connected by joints that move in the surrounding environment. The rigid bodies in the collection model parts of the body, such as the head, chest, torso or hips, and the joints connect the parts together in order to approximate articulation of the human body. Each body in the rigid body system has associated a shape and mass so it can respond to forces and interact with other bodies on contact. We model the environment as a collection of fixed rigid bodies that have an infinite mass and can not move.

Unlike kinematic models, the only way to influence the motion of the body model is by applying forces on it. To facilitate further exposition and illustrate how we are going to model the motion of the body, we discuss next what kinds of forces we need to account for, what options we have for modeling these forces and what approach we take in our implementation to compute the forces.

Forces. The motion of our body model is determined by forces that act on its parts. The forces result from external factors, such as the gravity, effects of the environment (contact) and interaction of body parts with other parts (joint articulation, joint angle limits, contact), as well as internal body actuation. Body actuation powers the model by approximating effects of true muscles and keeps the body balanced. It computes actuation forces for the parts of the body in order to deliberately control the motion of the body and produce an intended behavior. In more detail, we model the following types of forces:

1. *Gravity*

   Gravity force $\mathbf{F}^{gravity}$ models attraction of the model towards the ground plane and accelerates the parts of the body downward.

2. *Contact*

   Contact force $\mathbf{F}^{contact}$ models non-penetration and friction between body parts and the parts and the environment.

   The force acts at contact points detected by the underlying collision detection library based on the current placement of the bodies in the world and their geometric shapes. Here, contact points can be imagined as the points where any two bodies touch and where the contact force may need to act in order to support the bodies and prevent their penetration. For example, in order to account for the contact between the foot and the ground, a force may need to act on the toe to prevent the foot from falling through the ground.

Contact force is computed according to a contact model. In the practice, the force is produced either from constraints, that prevent bodies in contacts from moving against each other and resist sliding, or by using a penalty method that penalizes penetrations and slippage, but does not prevent it from happening. We use the constraint-based formulation to implement contact.

3. *Articulation*

   Articulation force $\mathbf{F}^{articulation}$ implements joints and body articulation and anchors body parts together.

   This force has to be modeled in systems where the motion of the entire body is modeled as a collection of independently moving parts. Using this parameterization, in so called *maximal coordinates*, there is nothing in the underlying equations of motion that stops the simulator from popping the parts apart, except the articulation forces that anchor the parts together. In order to implement the desired coordination of the motion properly, articulation forces need be produced from constraints on the relative motion of the parts.

   Alternatively, one can model the motion of the parts in the body relative to the motion of their parent parts, parameterizing the state of the body recursively using a reduced set of *generalized coordinates*, and simulate the entire articulated body as a single entity. This other approach has the benefit that the articulation constraints are implicitly accounted for in the equations of motion. However, it does not offer as much flexibility, because the equations are specific to the particular branching structure and have to be reformulated for each new kind of structure. In addition, the generalized coordinates approach prohibits simulation of certain effects, such as tearing of an arm on a strong pull. We follow the maximal coordinates approach for simulation.

4. *Joint angle limits*

   Joint angle limits force $\mathbf{F}^{limits}$ approximates biomechanical constraints on the valid pose of the body and aim to prevent the parts in the body to extend over respective angle limits at the joints. For example, the force can model the fact that the knee should only bend forward.

   Joint angle limits forces are typically produced either from constraints that prevent extension of parts past the angle limits, using penalty methods or are simply ignored. Penalty methods do not prevent violations of the limits but attempt to recover from the violations by applying restoration forces. We implement joint angle limits using constraints for robustness.

5. *Actuation*

   Actuation force $\mathbf{F}^{actuation}$ actuates parts in the body so that the body performs an intended coordinated motion.

The forces are responsible for the "powering" of the joints in the body so that the parts of the body are positioned as intended and in such a way that the parts support the entire body and prevent it from falling. For example to make an arm contract at an elbow to a specified angle, as dictated by a high-level decision-making process in a controller, a spring-like force may need be applied on the forearm and the upper arm to achieve the effect. The required actuation forces are produced from the overall control goal typically by either penalty methods, constraints over the velocities and positions of the parts or optimization. We employ constraints for the actuation of the body.

Although the force types implement different aspects of the motion of the body, they can be modeled by similar principles. Specifically, each type attempts to achieve a specific objective that can be phrased as keeping the rigid body system in a particular state. It can be shown that all these aspects can be phrased as constraints over relative motion of the bodies in the system and the desired behavior of the system can be achieved by computing the forces that maintain the constraints. Note that all force types, except for actuation forces, result from physical laws and universal constraints that do not depend on the application. As such, they can be directly provided by the underlying simulator. Actuation forces, on the other hand, result from the control of the body and are specific to the application.

To model the behavior of the system by constraints, there are two questions to be answered: (1) how the interesting behaviors (*e.g.*, non-penetration, friction, body actuation) we want to achieve can be expressed as constraints and (2) how the constraints can be translated to forces to realize the objective of the constraint. We give detailed answers to both these questions in the remainder of this section. For now, let us provide a high-level informal overview of possible options for modeling constraints and methods for implementing the constraints.

Constraints. Constraints encode aspects of the desired behavior of the rigid body system. They are abstracted by equality or inequality equations over positions and velocities of bodies and request selected properties of rigid bodies to be either kept fixed at a given value or above/below the value. Motivated by our initial examples, we may want to request non-penetration of the foot with the ground by specifying that the distance between the toe and the ground plane needs be positive or zero. At the same time, we may want to contract the arm such that the angle between the forearm and the upper arm equals $\pi/2$, as described by another constraint. By encoding the entire desired behavior as a collection of constraints, we can characterize each aspect of the behavior more effectively.

Constraint Methods. Constraint methods directly solve for the constraint forces that need be applied on the bodies so that the desired collaborative behavior described by the constraints is achieved exactly and all constraints hold. The methods are designed such that the system never deviates from the desired state defined by the constraints and produces stable systems that can be integrated robustly with large integration steps.

Penalty Methods. Penalty methods attempt to approximate constraints by letting the system deviate from the desired state and applying penalty (restoration) forces to restore the deviations. As opposed to solving for the exact forces, penalty methods rely on a feedback loop, where current violations determine the restoration forces. The penalty forces for a given constraint are computed locally and independently of other constraints. Consequently, penalty forces due to different constraints compete and easily fail to achieve the collaborative

Method	Contact	Articulation	Actuation
Wang *et al.*[198]	Constraints	Constraints	Penalty Forces
Brubaker *et al.*[26]	Penalty Forces	N/A	Penalty Forces
Kokkevis *et al.*[198]	Constraints	Generalized Coordinates	Constraints
Ours	Constraints	Constraints	Constraints

Table 3.1: **Body Modeling Options.** Illustration of a few previous methods for modeling the body of the person and the design choices to implement various aspects of the behavior of the body.

behavior. To ensure the system stays stable, extremely small integration steps have to be taken, such as of 10^{-4} seconds. Nevertheless, penalty forces are easy to implement and cheap to compute.

Previous methods used both the constraint methods and penalty methods to implement various behavior aspects of the rigid body system. In our work, we model all forces using constraints and solve for the constraint forces using a constraint solver. This way, we can handle all behavior aspects uniformly within the constraint solving framework as well as achieve a robust and stable performance. See Table 3.1 for the illustration of the modeling options.

Chapter Outline. We next discuss in this chapter how we are going to model the motion of a single rigid body in the rigid body system. This rigid body can either be a part of the model of the body of the person or any other rigid body in the simulated world, such as the ground, a falling brick or a projectile. In later sections, we introduce constraints to glue rigid bodies into an articulated structure that we then use to represent the body of the person. We will also use constraints to control the motion of the structure and to model interactions of the structure with the environment.

In explaining these concepts, we provide equations that fully characterize the motion of the model of the person in simulation and let us perform the simulation. Our goal in this chapter is to provide enough detail on the simulation of the model so that (1) understanding of the following chapters that use this simulation as a tool is improved and (2) the entire motion capture system can be reproduced. With that said, we note that it is not necessary to build the simulation system from scratch according to our explanations. In fact, the formalization of the rigid body dynamics that we employ in our work and present here is standard and is directly implemented by available physics simulators, both commercial and open-source. We provide these explanations for completeness.

Underlying Formalisms. We explore the simulation paradigm where states of rigid bodies are parameterized in *maximal coordinates* and *constraint forces* enforce the constraints. The use of maximal coordinates brings us the necessary flexibility that we need in modeling the motion of the body: an ability to add or remove constraints dynamically as they emerge (*e.g.*, contact points), without the need for reparameterizing body states and reformulating equations of motions in terms of these new parameters. Furthermore, the maximal coordinates approach encourages a modular design of the simulator and the constraints because no constraints are hard-coded in the equations of motions. The theory behind simulation, the dynamics model, as well as the implementation ideas draw from papers [17, 18], tutorials [19], and the dissertation thesis [16] of David Baraff. We adopt lo-hi linear complementarity problems formulations from the Open Dynamics Library (ODE) [169] by Russell Smith. Derivations of the concepts can be found in [17, 18, 19, 49, 191, 194, 192].

Technical Details Overview. Our model of the person is integrated directly with the simulation engine Crisis [190] that represents system dynamics. In doing so, we use constraints to update body pose over time and to achieve articulation and actuate the body. Crisis employs simulation, constraint and contact model formulations that are identical to those in ODE [169]. Our model differs from ODE in a way that it uses an explicit integration and supports both acceleration and velocity constraints.

We formulate constraints as linear equations over acceleration and velocities of body pairs that should be maintained subject to constraint force bounds, and use the Lagrange multiplier method to express constraint forces and impulses as a solution to a lo-hi linear complementarity problem (LCP) built from the constraint specifications. We use a generic constraint model capable of modeling both bilateral as well as unilateral constraints, soft constraints and constraints with finite force bounds. We use a constraint solver to solve for the constraint force (or constraint impulse) for the individual constraints such that the resulting forces for all constraints are all within the bounds and all complementarity conditions from the LCP formulation hold. We use the method of [18] and a variant of the Dantzig pivoting solver from [169] to obtain valid constraint forces for constraints with fixed force bounds. To handle friction constraints with friction force bounds depending on contact surface normal forces, we alternate between fixing and re-estimating the bounds for the LCP. To minimize a risk of over-constraining the LCP, all the constraints that we use to build the model of the person are soft to a certain extent. We also use constraints with finite force bounds (*e.g.*, for body actuation) which can always be satisfied by applying forces with maximum possible magnitudes.

We use Newton-Euler equations of motion to describe the motion of the simulated rigid body system under effects of external forces (*e.g.*, gravity), actuation forces and other constraint forces produced by the constraint solver. For the experiments in this work, we integrate the underlying equations of motion using an Euler method with a sufficiently small step size (*e.g.*, $1/120$ s).

3.1 Rigid Body

In this section, we develop concepts that let us define and model the motion of a single rigid body in an otherwise empty world. In doing so, we follow the approach presented in [19]. We intuitively imagine the body as a solid structure that has a fixed non-deformable shape and that can translate and rotate in response to forces that act on it.

We characterize the body in terms of its mass properties that describe the distribution of the mass of the body over its volume, the position and orientation of the body in the world space and the velocity with which the body moves in the world. The motion of the body results from forces that are applied on the body. The relation between the forces and the motion is indirect. The forces first effect the velocity of the body and the velocity of the body makes the body change its position and orientation in the world.

We next introduce the parameterization of the position and orientation of the body, discuss the concept of velocity and force of the body and show how the force affects the position and orientation of the body. Using these relations, we will produce the motion of the body by computing forces and simulating the effects of the forces.

3.1.1 Concepts

Body Space. Because the rigid body can not deform, its shape and volume can be defined within a local coordinate system associated with the body called the *body space*. The body space establishes a local coordinate frame of reference for the body.

Mass Properties. Mass properties are statistics computed from the distribution of the mass over the body. The statistics encode how the body responds to forces and we use this information to simulate the motion of the body. The responses to forces are independent of the shape of the body and the information about the shape is only needed for modeling contact between multiple bodies, as discussed in Section 3.4.

We define *mass properties* of the body as moments of the *density function* $\rho : \mathbb{R}^3 \to \mathbb{R}^+$ that characterizes the distribution of the mass of the body over the points $\mathbf{r}_b$ in the body space:

$$\begin{aligned}
\text{total mass } m &= \int \rho(\mathbf{r}^b)\,\mathrm{d}\mathbf{r}^b \qquad (3.1)\\
\text{center of mass in body space } \mathbf{r}^b_{cm} &= \frac{\int \mathbf{r}^b \cdot \rho(\mathbf{r}^b)\,\mathrm{d}\mathbf{r}^b}{m}\\
\text{moment of inertia about the body x axis } \mathbf{I}^b_{xx} &= \int ((\mathbf{r}^b_y)^2 + (\mathbf{r}^b_z)^2) \cdot \rho(\mathbf{r}^b)\,\mathrm{d}\mathbf{r}^b\\
\text{moment of inertia about the body y axis } \mathbf{I}^b_{yy} &= \int ((\mathbf{r}^b_x)^2 + (\mathbf{r}^b_z)^2) \cdot \rho(\mathbf{r}^b)\,\mathrm{d}\mathbf{r}^b\\
\text{moment of inertia about the body z axis } \mathbf{I}^b_{zz} &= \int ((\mathbf{r}^b_x)^2 + (\mathbf{r}^b_y)^2) \cdot \rho(\mathbf{r}^b)\,\mathrm{d}\mathbf{r}^b\\
\text{product of inertia about the body x, y axes } \mathbf{I}^b_{xy} &= \int \mathbf{r}^b_x \cdot \mathbf{r}^b_y \cdot \rho(\mathbf{r}^b)\,\mathrm{d}\mathbf{r}^b\\
\text{product of inertia about the body x, z axes } \mathbf{I}^b_{xz} &= \int \mathbf{r}^b_x \cdot \mathbf{r}^b_z \cdot \rho(\mathbf{r}^b)\,\mathrm{d}\mathbf{r}^b\\
\text{product of inertia about the body y, z axes } \mathbf{I}^b_{yz} &= \int \mathbf{r}^b_y \cdot \mathbf{r}^b_z \cdot \rho(\mathbf{r}^b)\,\mathrm{d}\mathbf{r}^b,
\end{aligned}$$

where the moments of inertia $\mathbf{I}^b_{xx}, \mathbf{I}^b_{yy}, \mathbf{I}^b_{zz}$ and the products of inertia $\mathbf{I}^b_{xy}, \mathbf{I}^b_{xz}, \mathbf{I}^b_{yz}$ are computed in the body space and are expressed *relative to the origin* 0 of the body space. We further record these statistics into the *body space inertia matrix* $\mathbf{I}_{body}$,

$$\mathbf{I}_{body} = \begin{pmatrix} \mathbf{I}^b_{xx} & -\mathbf{I}^b_{xy} & -\mathbf{I}^b_{xz} \\ -\mathbf{I}^b_{xy} & \mathbf{I}^b_{yy} & -\mathbf{I}^b_{yz} \\ -\mathbf{I}^b_{xz} & -\mathbf{I}^b_{yz} & \mathbf{I}^b_{zz} \end{pmatrix} \qquad (3.2)$$

for notation convenience. We encode the mass properties of the rigid body using a pair that consists of the total mass from Equation (3.1) and the body space inertia matrix from Equation (3.2),

$$\left(m, \mathbf{I}_{body}\right). \qquad (3.3)$$

We assume these properties do not change over time.

Position and Orientation. In order to simulate the body, we need to parameterize the location of the body in the world and place it in the world based on this parameterization. To place the rigid body in the world, the body space has to be mapped to the world space. Specifically, (1) axes of the body space have to be first rotated to achieve the desired orientation $\mathbf{R}$ of the body in the world space and then (2) the center of mass of the body has to be translated to achieve the desired position $\mathbf{x}$ in the world space.

To facilitate the mapping, we assume that the center of mass of the rigid body lies at the origin of the body space, $\mathbf{r}^b_{cm} = 0$. We then let

$$\mathbf{r} = \mathbf{R} \cdot \mathbf{r}^b + \mathbf{x} \tag{3.4}$$

for a point $\mathbf{r}^b$ in the body space and the corresponding mapped point $\mathbf{r}$ in the world space, where $\mathbf{R}$ is a 3×3 rotation matrix and $\mathbf{x}$ is a translation. Here, $\mathbf{R}$ encodes the *orientation of the body in the world space* and characterizes how the local axes of the body map to the world space. The $\mathbf{x}$ vector defines the *position of the body in the world space* and this position coincides with the center of mass of the body.

Velocity. Having placed the body in the world, we are interested in characterizing how the position and orientation of the body changes over time so we can model these quantities and produce the motion of the body in simulation. We decompose the instantaneous motion of the body over infinitesimally short time periods to the translational (*linear*) motion of the center of mass of the body and the rotational (*angular*) motion of the volume of the body and characterize each of these two motions separately using velocities.

The *linear velocity* $\mathbf{v}$ characterizes the instantaneous linear motion and describes the direction and speed with which the body translates. The linear velocity of the body is defined as the time derivative of the position $\mathbf{x}$ of the body such that $\mathbf{v} = \dot{\mathbf{x}}$.

The *angular velocity* ω characterizes the instantaneous rotational motion as a rotation about a time varying axis that passes through the center of mass of the body. The angular velocity ω is defined as world-space vector whose direction describes the instantaneous rotation axis and whose magnitude in $\mathrm{rad} \cdot \mathrm{s}^{-1}$ defines the instantaneous rotation speed.

Relation between Position and Velocity. The linear and angular velocity together characterize the instantaneous velocity of any point or a vector attached to the body. Specifically, if $\mathbf{r} = \mathbf{p} - \mathbf{x}$ is the vector between a point $\mathbf{p}$ on the body and the center of mass of the body $\mathbf{x}$ then

$$\dot{\mathbf{r}} = \omega \times \mathbf{r} \tag{3.5}$$

$$\dot{\mathbf{p}} = \mathbf{v} + \omega \times \mathbf{r} \tag{3.6}$$

according to [19]. Note that the same equation holds for a vector $\mathbf{r}$ attached to the body, such that it rotates with the body. Applying this equation on the local axes and the center of mass of the body, we obtain a relationship between the position of the body and its velocity,

$$\dot{\mathbf{x}} = \mathbf{v} \tag{3.7}$$

$$\dot{\mathbf{R}} = \omega^* \cdot \mathbf{R}, \tag{3.8}$$

where ω^* is a "cross-product matrix" such that $\omega^* \cdot \mathbf{r} = \omega \times \mathbf{r}$.

Momentum. We further express the linear and angular velocity of the body in the form of linear and angular momentum of the body, as defined below, so that we can conveniently characterize the relationship between forces and velocities. The reason for using momentums for expressing this relationship is that the velocity of the body can change because of inertial effects even when no force acts on the body, whereas the momentums remain constant in that case.

We define the *linear momentum* $\mathbf{P}$ and the *angular momentum* $\mathbf{L}$ of the body as

$$\mathbf{P} = m \cdot \mathbf{v} \tag{3.9}$$

$$\mathbf{L} = \mathbf{I} \cdot \omega, \tag{3.10}$$

$$\tag{3.11}$$

where

$$\mathbf{I} = \mathbf{R} \cdot \mathbf{I}_{body} \cdot \mathbf{R}^T \tag{3.12}$$

is the *world space inertia matrix* of the body, [19]. The momentum characterizes the kinetic motion of the body using the velocity and the mass properties of the body, as expressed in world space through m, and $\mathbf{I}$.

Mass Matrix. We encode the mass properties of the body expressed with respect to the world frame using the *mass matrix* $\mathbf{M}$,

$$\mathbf{M} = \begin{pmatrix} m \cdot \mathbf{E} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{pmatrix}, \tag{3.13}$$

where $\mathbf{E}$ is a 3×3 identity matrix. Note that the mass matrix depends on the current orientation of the body and, as such, is not necessarily constant.

Forces. Forces abstract causes for the motion of the body. They are modeled as vectors $\mathbf{F}(\mathbf{p})$ that act at specific locations $\mathbf{p}$ of the body and encourage the body to move along the directions they act. Forces affect both the instantaneous linear and angular motion of the body, possibly causing the body to spin, with the actual effects depending on the points of applications and the distribution of the mass in the body.

We model effects of forces by *force-torque* pairs. Given a force $\mathbf{F}(\mathbf{p})$ that acts on the body at the world space location $\mathbf{p}$, the force produces the force-torque pair $[\mathbf{F}(\mathbf{p}), \tau(\mathbf{p}, \mathbf{F}(\mathbf{p}))]$, where

$$\tau(\mathbf{p}, \mathbf{F}(\mathbf{p})) = (\mathbf{p} - \mathbf{x}) \times \mathbf{F}(\mathbf{p}) \tag{3.14}$$

is the *torque* about the center of mass of the body due to the force $\mathbf{F}(\mathbf{p})$. Intuitively, $\mathbf{F}(\mathbf{p})$ attempts to push the body in the direction of the force, while $\tau(\mathbf{p}, \mathbf{F}(\mathbf{p}))$ attempts to make the body spin along the direction of the torque vector with a speed proportional to the magnitude of the torque.

For example, the *gravity force* $\mathbf{F}^{gravity}$ that acts on the body to push it down toward the ground can be modeled by the pair

$$[m \cdot g, \mathbf{0}], \tag{3.15}$$

where g is the gravity constant indicating how strongly the force acts.

Effects of forces are additive. That is, to capture the overall effect of all force-torque pairs $[\mathbf{F}_i, \tau_i]$ due to all forces acting on the body, it is sufficient to maintain only the aggregate force-torque pair $[\mathbf{F}^{total}, \tau^{total}]$, where $\mathbf{F}^{total} = \sum_i \mathbf{F}_i$ is the *total external force* and $\tau^{total} = \sum_i \tau_i$ is the *total external torque* about the center of mass of the body.

Relation between Forces and Momentum. Forces applied on the body cause the motion of the body to change. This change is characterized in terms of the aggregate force-torque pair $[\mathbf{F}^{total}, \tau^{total}]$ and the momentum of the body as

$$\dot{\mathbf{P}} = \mathbf{F}^{total} \tag{3.16}$$

$$\dot{\mathbf{L}} = \tau^{total}. \tag{3.17}$$

The aggregate force-torque pair encodes how the total force works to push and spin the body and how this work affects the instantaneous linear and angular motion of the body, encoded by the momentum [49, 19]. Specifically, the equation says that the linear motion of the rigid body is equivalent to the motion of a particle with the total mass m affected by the total force $\mathbf{F}^{total}$.

Acceleration. We define the *linear acceleration* and *angular acceleration* of the body as the time derivatives of the linear velocity $\mathbf{v}$ and the angular velocity ω of the body,

$$\mathbf{a} = \dot{\mathbf{v}} \tag{3.18}$$

$$\alpha = \dot{\omega}. \tag{3.19}$$

It can be shown that at any time, the acceleration of the body due to the aggregate force-torque pair applied on the body is given by

$$\mathbf{a} = m^{-1} \cdot \mathbf{F}^{total} \tag{3.20}$$

$$\alpha = \mathbf{I}^{-1} \cdot (\tau^{total} + \tau^{coriolis}), \tag{3.21}$$

where $\tau^{coriolis} = (\mathbf{I} \times \omega) \times \omega$ is an implicit internal inertial (coriolis) torque due to body rotation, τ^{total} is the total external torque applied on the body and $\mathbf{F}^{total}$ is the total external force, [19].

From these equations, we see that the effects of forces and torques are not "immediate". These quantities control the acceleration of the body and the accelerations have to be integrated over time in order to modify the velocity. As such, forces change the velocity of the body indirectly, by only changing its time derivative.

Impulsive Forces and Relation to Momentum. In specific cases, it is important to change the velocity of the body directly and instantly, right at the current time (*e.g.*, to resolve an impact between bodies and prevent penetration). Towards that end, we introduce impulsive forces that change the linear and angular momentum of the body, as opposed to the time derivatives of the momentums (*i.e.*, what regular forces do).

We define *impulsive force* $\mathbf{F}^{imp}$ as a force with the "units of momentum". If $\mathbf{P}$ and $\mathbf{L}$ are the linear and angular momentum of the body and $\mathbf{F}^{imp}$ is applied to the body at the world space position $\mathbf{p}$, then the linear momentum $\mathbf{P}$ changes by the value $\Delta\mathbf{P}$ and the angular momentum $\mathbf{L}$ changes by the value $\Delta\mathbf{L}$,

$$\Delta\mathbf{P} = \mathbf{F}^{imp} \tag{3.22}$$

$$\Delta\mathbf{L} = \tau^{imp}, \tag{3.23}$$

where $\tau^{imp} = (\mathbf{p} - \mathbf{x}) \times \mathbf{F}^{imp}$ is the *impulsive torque* about the center of mass of the body due to the impulsive force $\mathbf{F}^{imp}$, [19]. These equations are analogous to Equation (3.16) and Equation (3.17), except that they relate impulsive forces to the changes of the momentums.

3.1.2 Equations of Motion: Second Order Dynamics

We use the rigid body concepts from above to formulate equations of motion for the rigid body. These equations, written in the form of *first order ordinary differential equations* (ODEs), fully determine the motion of the body in Newtonian dynamics under the effects of forces. Numerical integration of these equations produces the motion of the body. The equations can be integrated using standard numerical ODE solvers. We outline the integration process later.

We explore several formulations of the equations of motion. Each formulation defines a representation for the state $\mathbf{y}$ of the rigid body and an equation to describe how the time derivative of the state $\frac{\partial \mathbf{y}}{\partial t}$ changes over time in response to the forces applied on the body. The representation of the state encodes information about the current position and orientation of the body as well as velocity. We always use Equation (3.7) and Equation (3.8) to model the changes of the position and orientation of the body and only study different formulations for the modeling of the velocity of the body.

Here, we discuss second-order formulations of Newtonian dynamics, where the equations of motion relate the position of the body to its second-order time derivative (acceleration) through forces.

Momentum Form. Using the linear and angular momentum of the body to represent the velocity, we can parameterize the state $\mathbf{y}$ of the rigid body by the vector $\mathbf{y} = (\mathbf{x}, \mathbf{R}, \mathbf{P}, \mathbf{L})$. This parameterization produces the *equation of motion for the rigid body in the momentum form*, [19],

$$\frac{\partial}{\partial t}\begin{pmatrix}\mathbf{x}\\ \mathbf{R}\\ \mathbf{P}\\ \mathbf{L}\end{pmatrix} = \begin{pmatrix}\mathbf{v}\\ \omega^* \cdot \mathbf{R}\\ \mathbf{F}^{total}\\ \tau^{total}\end{pmatrix}, \tag{3.24}$$

where $\mathbf{v}$ and ω are auxiliary quantities derived from the state vector $\mathbf{y}$, $\mathbf{v} = m^{-1} \cdot \mathbf{P}$, $\omega = \mathbf{I}^{-1} \cdot \mathbf{L}$, $\mathbf{I} = \mathbf{R} \cdot \mathbf{I}_{body} \cdot \mathbf{R}^T$ and $\mathbf{I}^{-1} = \mathbf{R} \cdot \mathbf{I}^{-1}_{body} \cdot \mathbf{R}^T$.

Velocity Form. Alternatively, representing the velocity of the body using the linear and angular velocity vectors, the body state $\mathbf{y}$ can be parameterized by the vector $\mathbf{y} = (\mathbf{x}, \mathbf{R}, \mathbf{v}, \omega)$. This more common parameterization produces the *equation of motion for the rigid body in the velocity form*,

$$\frac{\partial}{\partial t}\begin{pmatrix}\mathbf{x}\\ \mathbf{R}\\ \mathbf{v}\\ \omega\end{pmatrix} = \begin{pmatrix}\mathbf{v}\\ \omega^* \cdot \mathbf{R}\\ m^{-1} \cdot \mathbf{F}^{total}\\ \mathbf{I}^{-1} \cdot ((\mathbf{I} \times \omega) \times \omega + \tau^{total})\end{pmatrix} \tag{3.25}$$

Mass Matrix Form. We further elaborate on the velocity form of the equation of motion and define the notion of *generalized quantities* for the rigid body. Using this new notation, we can treat the linear and angular motion of the body uniformly and we can express the dynamics of the rigid body using the mass matrix of the rigid body, producing an equation that resembles the Newton's second law for a particle.

We call any block vector consisting of a block due to a linear quantity and a block due to the corresponding angular quantity a *generalized quantity*. This way, we obtain the *generalized velocity* $\mathbf{v}_{gen} = (\mathbf{v}, \omega)$, the *generalized acceleration* $\mathbf{a}_{gen} = (\mathbf{a}, \alpha)$, the *generalized total external force* $\mathbf{F}_{gen}^{total} = (\mathbf{F}^{total}, \tau^{total})$, the *generalized coriolis force* $\mathbf{F}_{gen}^{coriolis} = (\mathbf{0}, \tau^{coriolis})$ and the *generalized position* $\mathbf{x}_{gen} = (\mathbf{x}, \mathbf{R})$.

Using Equation (3.20) and Equation (3.21), we can rewrite the relation between forces and accelerations using the generalized form as $\mathbf{M} \cdot \mathbf{a}_{gen} = \mathbf{F}_{gen}^{total} + \mathbf{F}_{gen}^{coriolis}$. Let us assume that the generalized coriolis force $\mathbf{F}_{gen}^{coriolis}$ is always implicitly incorporated into the total generalized external force $\mathbf{F}_{gen}^{total}$ and, to improve readability, let's remove the "$_{gen}$" subscripts and omit the "generalized" adjective whenever it is clear the generalized notation is used. Using these assumptions, we can write

$$\mathbf{M} \cdot \mathbf{a} = \mathbf{F}^{total}, \tag{3.26}$$

which yields a general relation between the total force $\mathbf{F}^{total}$ and the total acceleration $\mathbf{a}$ of the body due to the force. Because this relation is linear, the equation also holds for any force $\mathbf{F}$ that acts on the body and the corresponding *acceleration* $\mathbf{a} = \mathbf{M}^{-1} \cdot \mathbf{F}$ that the body gains in *response to the application of the force* $\mathbf{F}$[1]. The relation resembles the Newton's Second Law for a particle and, consequently, rigid bodies can be imagined as special particles with time-varying masses $\mathbf{M}$ that move in $\mathbb{R}^6$.

Using the generalized notation, we can parameterize the body state $\mathbf{y}$ using the vector $\mathbf{y} = (\mathbf{x}, \mathbf{v})$ and obtain the *equation of motion for the rigid body in the mass matrix form*,

$$\frac{\partial}{\partial t} \begin{pmatrix} \mathbf{x} \\ \mathbf{v} \end{pmatrix} = \begin{pmatrix} \mathbf{x}' \\ \mathbf{M}^{-1} \cdot \mathbf{F}^{total} \end{pmatrix}, \tag{3.27}$$

where $\mathbf{x}' = (\mathbf{v}, \omega^* \cdot \mathbf{R})$.

3.1.3 Equations of Motion: First Order Dynamics

Regular forces effect the position and orientation of the body by second-order (Newtonian) dynamics in which velocities change through the integration of forces while positions change through the integration of velocities. With impulsive forces, the effects of forces on positions and orientations are determined by first-order (impulsive) dynamics, where velocities change directly through the application of impulsive forces and positions change through the integration of velocities. Here, we describe equations of motion for impulsive dynamics that relates the position of the rigid body to its first-order time derivative through application of impulsive forces. We reuse and build upon the generalized notation from above.

Similarly to second-order dynamics, we couple linear and corresponding angular quantities to generalized quantities. This way, we obtain the *generalized impulsive force (impulse)* $\mathbf{F}_{imp} = (\mathbf{F}^{imp}, \tau^{imp})$ and the *generalized momentum* $\mathbf{F}_{imp}^{total} = (\mathbf{P}, \mathbf{L})$, that serves the purpose of an analog of $\mathbf{F}^{total}$ in second-order dynamics.

Impulsive Dynamics. Then, if $\mathbf{M}$ is the mass matrix of the rigid body and $\mathbf{v}$ is the generalized velocity of the body, the definitions of the linear and angular momentum in Equation (3.9) and Equation (3.10) directly provide that $\mathbf{M} \cdot \mathbf{v} = \mathbf{F}_{imp}^{total}$. Furthermore, the definitions of momentum updates in Equation (3.22) and Equation (3.23) state that the change $\Delta\mathbf{v}$ of the generalized velocity $\mathbf{v}$ due to the application of the generalized

[1]If $\mathbf{F}$ refers to the total external force exerted on the body, the coriolis force $\mathbf{F}^{coriolis}$ is assumed to be included in $\mathbf{F}$.

impulse $\mathbf{F}_{imp}$ equals $\Delta\mathbf{v} = \mathbf{M}^{-1} \cdot \mathbf{F}_{imp}$. Therefore, the first-order dynamics that relates velocities $\mathbf{v}$ to impulses $\mathbf{F}_{imp}$ is given by

$$\mathbf{M} \cdot \mathbf{v} = \mathbf{F}_{imp} \tag{3.28}$$

and the generalized momentum $\mathbf{F}_{imp}^{total}$ can be seen as a generalized *total external impulse* that acts on the body and that consists of the only term, the inertial term $(\mathbf{P}, \mathbf{L})$. Note that this formulation of first-order dynamics using generalized notation and the mass matrix directly matches the case of second-order dynamics in Equation (3.26), which relates accelerations $\mathbf{a}$ to forces $\mathbf{F}$ by $\mathbf{M} \cdot \mathbf{a} = \mathbf{F}$.

First-order Equation of Motion. The state $\mathbf{y}$ of a rigid body in a purely first-order dynamics system (with no acceleration) can be parameterized using the regular notation by a vector $\mathbf{y} = (\mathbf{x}, \mathbf{R})$. This parameterization produces the *first-order equation of motion for the rigid body*,

$$\frac{\partial}{\partial t} \begin{pmatrix} \mathbf{x} \\ \mathbf{R} \end{pmatrix} = \begin{pmatrix} \mathbf{v} \\ \omega^* \cdot \mathbf{R} \end{pmatrix}, \tag{3.29}$$

where the motion of the body is only produced by applying impulses and the body stops instantly if no impulse acts on the body[2].

We never model first-order rigid body dynamics directly. Instead, we combine first-order dynamics with second-order dynamics. Specifically, we use the second-order (Newtonian) equations of motion from Equation (3.24) to model the motion of the body with accelerations and use the first-order dynamics rules from Equation (3.28) to update the velocity of the body when an impulse needs be applied.

3.1.4 Simulation

We produce the motion of the rigid body by numerically integrating the second-order equations of motion of the body from Equation (3.24), using Equation (3.28) to apply impulses on the body. The motion of the body can be generated from the equations of motion using an arbitrary ODE solver.

Integration. In our implementation [191, 192], we use an explicit integration method. The integration process can be imagined as an iterative application of the integration function $f : \mathbf{y}_t \to \mathbf{y}_{t+1}$ on the initial state $\mathbf{y}_1$ of the body, where each $f(\mathbf{y}_t)$ updates the state of the body from a frame t to the frame $t+1$. During the update at the time t, the function proposes a number of hypothetical states $\hat{\mathbf{Y}}_t = \{\hat{\mathbf{y}}_{t'} \mid t \leq t' < t+1\}$ for some sampling of the time interval between the frames, evaluates the forces that act on the body in the proposed states at the sampled time instants and uses this information to compute $\mathbf{y}_{t+1}$. If an impulse is applied on the body during the state update, the update rules from Equation (3.28) produce a new initial state $\mathbf{y}$ for the integration function and the integration is restarted from this new state.

Euler Method. For example, if the integration is implemented using an explicit Euler method with one step per frame, then $\hat{\mathbf{Y}}_t = \{\mathbf{y}_t\}$ and the integration function $f : \mathbf{y}_t \to \mathbf{y}_{t+1}$ updates the state by

$$\mathbf{y}_{t+1} = \mathbf{y}_t + \Delta t \cdot \frac{\partial \mathbf{y}_t}{\partial t}, \tag{3.30}$$

[2]In contrast, in second order dynamics, the body stops accelerating if no force is applied but it keeps moving with a velocity determined by its momentum.

where Δt is the time difference between two frames and the difference equals the integration step size and $\frac{\partial \mathbf{y}_t}{\partial t}$ is evaluated from Equation (3.24) using the values of $\mathbf{F}^{total}$ and τ^{total} that act on the body in the state $\mathbf{y}_t$. That is, the new state is produced as

$$\begin{aligned}
\mathbf{x}_{t+1} &= \mathbf{x}_t + \Delta t \cdot \mathbf{v}_t \\
\mathbf{R}_{t+1} &= \mathbf{R}_t + \Delta t \cdot \omega_t^* \cdot \mathbf{R}_t \\
\mathbf{P}_{t+1} &= \mathbf{P}_t + \Delta t \cdot \mathbf{F}_t^{total} \\
\mathbf{L}_{t+1} &= \mathbf{L}_t + \Delta t \cdot \tau_t^{total}
\end{aligned}$$

In our implementation [191, 192], we implement a variety of higher-order integration methods that sample the time interval in order to approximate higher order time derivatives of the state and achieve higher accuracy. Our methods can also substep between frames and control the size of the integration step dynamically for improved stability and performance.

3.2 Constraints

The technique for modeling the motion of a single rigid body directly generalizes for the case of multiple bodies in a rigid body system. In order to produce the motion of multiple bodies, we only need to update states of all bodies in the system at the same time. In doing so, however, potential interactions between bodies have to be accounted for and modeled through forces. For example, in order to model a brick in the free fall, the only force that acts on the brick is the gravity force. When the brick approaches the ground, however, interaction between the ground and the brick has to be modeled. In particular, a force need to be applied on the brick to prevent it from going through the ground. In order to model the body of the person, we can use independent rigid bodies to model parts of the body, but we need to glue the bodies together at joints in order to model the articulation of the body, *etc*. As such, the key challenge in physical simulation of the rigid body system is modeling of forces so that the bodies in the system would move as desired.

Rather than trying to generate these forces directly, we characterize the desired motion of the system using *motion constraints* over accelerations, velocities and positions of *pairs of rigid bodies* and use a *constraint solver* to solve for the forces that produce the desired behavior described by the constraints. We use the Lagrange multiplier method to express constraint forces as a solution to a linear complementarity problem (LCP) built from the constraint specifications. We use a generic constraint model capable of modeling both bilateral as well as unilateral constraints, soft constraints and constraints with finite force bounds. We use constraints to (1) build articulated bodies consisting of rigid parts connected by joints in Section 3.3, (2) limit angles at the joints in the body to prevent infeasible body configurations in Section 3.5, (3) model contact between parts of the body and the environment in Section 3.4 and (4) power joints in the body in Section 3.6.

Because we do not integrate constraints with the equations of motions, which gives us the flexibility to change these constraints dynamically, the motion of the constrained rigid body system is produced by exactly the same original equations and the same numerical solvers as in the unconstrained case that we described earlier.

Constraint Forces and Accelerations. We implement constraints by computing *constraint forces*. Constraint forces act on the rigid bodies and implicitly enforce the desired behavior of the system by controlling the acceleration of the bodies in the system. We employ the approach of *Lagrange multipliers* to formulate the forces. This approach allows us to handle all constraints in the same uniform way and to combine constraints automatically and dynamically. Using this approach, we can characterize the desired behavior of the rigid body system in terms of individual constraint objectives, where each objective controls one aspect of the motion using separate, yet uniform, formulations (*e.g.*, body articulation, non-penetration, friction, actuation of the body, *etc.*). In modeling the motion of the person, all forces that we apply on the model of the person result either from constraints or universal physical laws, such as the gravity.

In general, we are often interested in defining constraints over positions and orientations of the bodies (*e.g.*, to model body articulation and non-penetration). However, as stated previously, we implement constraints by computing constraint forces, and forces are directly related to accelerations of the bodies, not the positions. Consequently, to compute constraint forces for other constraint types, we need to reformulate the constraints as constraints over accelerations.

Outline. In further exposition, we outline the general approach for maintaining constraints using Lagrange multipliers. We then formally introduce constraints over accelerations of bodies in the system and illustrate how acceleration constraints can be handled directly. We then introduce constraints over velocities and show how velocity constraints can be handled incrementally by converting them to constraints over accelerations and computing corresponding constraints forces. We also show how velocity constraints can be handled directly on the velocity level by computing constraint impulses. Lastly, we define constraints over positions and show how the position constraints can be handled incrementally by maintaining corresponding velocity or acceleration constraints.

3.2.1 Example: Point-To-Point Equality Constraint

We start our exposition of constraints and the explanation of the general constraint method with a motivational example that is of practical importance for the building of our physics-based model of the body of the person. The purpose of this example is to illustrate how one specific constraint type, the *point-to-point* constraint, can be implemented as well as to introduce general concepts for the constraint method that we utilize in later sections.

Imagine we are given two rigid bodies, denoted X and Y, that we want to glue together so that the bodies stay connected in motion. For example, one body may represent an upper arm of the body of the person, while the other one may represent the forearm and we may want to connect these parts at the elbow. We can realize the desired coupling between the motion of the two bodies by a constraint

$$\mathbf{p}_X = \mathbf{p}_Y, \tag{3.31}$$

where $\mathbf{p}_X$ is the point on the body X and $\mathbf{p}_Y$ is the point on the body Y and the two points are constrained to occupy the same world space location such that the bodies would not tear apart. For example, the $\mathbf{p}_X$ and $\mathbf{p}_Y$ points may define the anchor points for the upper arm and the forearm.

Position-level Constraint. In order to enable uniform handling of similar constraints, we abstract the constraint from Equation (3.31) in the form of a vector constraint function $\mathbf{C}_p : [\mathbf{x}_X, \mathbf{x}_Y] \rightarrow \mathbb{R}^m$ and the constraint equation,

$$\mathbf{C}_p(\mathbf{x}_X, \mathbf{x}_Y) = \mathbf{0} \in \mathbb{R}^m, \tag{3.32}$$

where the constraint function $\mathbf{C}_p$ is defined in terms of the generalized positions $\mathbf{x}_X$, $\mathbf{x}_Y$ of the two bodies for which the constraint is to be maintained and the function measures the violation of the constraint with respect to the position states of the bodies. The constraint function implicitly parameterizes the set of valid body state pairs $(\mathbf{x}_X, \mathbf{x}_Y)$ when the constraint holds exactly and this set corresponds to the position manifold $\mathbf{C}_p(\mathbf{x}_X, \mathbf{x}_Y) = \mathbf{0}$. The value of m defines the dimensionality of the constraint. In our example, $m = 3$ and $\mathbf{C}_p(\mathbf{x}_X, \mathbf{x}_Y) = \mathbf{p}_Y - \mathbf{p}_X$.

Maintaining Constraint Incrementally. Our goal is to maintain the constraint from Equation (3.32) incrementally, by (1) starting from an initial state where the constraint holds already and (2) computing appropriate constraint forces to affect the future motion of the constrained bodies such that the constraint will be kept maintained in the future as well. Specifically to our example, this strategy means that the simulation should start from a state where the bodies are connected and the constraint forces computed such that the bodies remain connected as the time progresses.

In maintaining the constraint incrementally, we may assume that the constraint holds at the current time such that $(\mathbf{x}_X, \mathbf{x}_Y)$ is on the position manifold. To keep the constraint maintained in the future, the constraint force applied on the bodies at this time needs to make sure that $(\mathbf{x}_X, \mathbf{x}_Y)$ stays on the position manifold during the integration step. In order to achieve this goal, the constraint force needs to cancel the components of the accelerations of the bodies due to applied external forces that make the bodies accelerate away from the position manifold.

Because the original constraint is defined over positions of bodies and forces are related to accelerations, the position-level constraint has to be reformulated as a constraint over accelerations of the bodies. Towards that end, the original constraint formulation from Equation (3.32) is differentiated twice with respect to time, producing the corresponding velocity-level and acceleration-level formulations of the constraint. Using the acceleration-level formulation of the constraint, we compute the constraint forces. We illustrate this process in the remainder of this section.

Velocity-level Formulation. We obtain the velocity-level formulation of the original constraint by taking the time derivative of Equation (3.32),

$$\begin{aligned}
\dot{\mathbf{C}}_p &= \frac{\partial \mathbf{p}_Y - \mathbf{p}_X}{\partial t} = \frac{\partial (\mathbf{x}_Y + \mathbf{r}_Y) - (\mathbf{x}_X + \mathbf{r}_X)}{\partial t} \\
&= (\dot{\mathbf{x}}_Y + \omega_Y \times \mathbf{r}_Y) - (\dot{\mathbf{x}}_X + \omega_X \times \mathbf{r}_X) \\
&= \dot{\mathbf{x}}_Y - \mathbf{r}_Y \times \omega_Y - \dot{\mathbf{x}}_X + \mathbf{r}_X \times \omega_X \\
&= \dot{\mathbf{x}}_Y - \mathbf{r}_Y^* \cdot \omega_Y - \dot{\mathbf{x}}_X + \mathbf{r}_X^* \cdot \omega_X \\
&= \begin{pmatrix} \mathbf{E} & -\mathbf{r}_Y^* \end{pmatrix} \cdot \begin{pmatrix} \dot{\mathbf{x}}_Y \\ \omega_Y \end{pmatrix} + \begin{pmatrix} -\mathbf{E} & \mathbf{r}_X^* \end{pmatrix} \cdot \begin{pmatrix} \dot{\mathbf{x}}_X \\ \omega_X \end{pmatrix} \\
&= \begin{pmatrix} -\mathbf{E} & \mathbf{r}_X^* & \mathbf{E} & -\mathbf{r}_Y^* \end{pmatrix} \cdot \begin{pmatrix} \dot{\mathbf{x}}_X \\ \omega_X \\ \dot{\mathbf{x}}_Y \\ \omega_Y \end{pmatrix} \\
&= \begin{pmatrix} \mathbf{J}_X & \mathbf{J}_Y \end{pmatrix} \cdot \begin{pmatrix} \mathbf{v}_X \\ \mathbf{v}_Y \end{pmatrix} \\
&= \mathbf{0},
\end{aligned}$$

where $\mathbf{r}_X$ is attached to the body X and $\mathbf{r}_Y$ is attached to Y such that $\mathbf{p}_X = \mathbf{x}_X + \mathbf{r}_X$ and $\mathbf{p}_Y = \mathbf{x}_Y + \mathbf{r}_Y$ and $\mathbf{J}_X$ and $\mathbf{J}_Y$ are 3×6 matrices called the *Jacobian matrices* due to the position constraint function $\mathbf{C}_p$ and the first and the second body.

This way, we are able to reformulate the original constraint, specified in terms of generalized positions, to a constraint specified in terms of *generalized velocities*. This velocity-level formulation of the constraint can be generally written using the velocity-level constraint function $\mathbf{C}_v : [\mathbf{v}_X, \mathbf{v}_Y] \to \mathbb{R}^m$ as

$$\mathbf{C}_v(\mathbf{v}_X, \mathbf{v}_Y) := \mathbf{J}_X \cdot \mathbf{v}_X + \mathbf{J}_Y \cdot \mathbf{v}_Y = \mathbf{0}, \tag{3.33}$$

where $\mathbf{J}_X = \begin{pmatrix} -\mathbf{E} & \mathbf{r}_X^* \end{pmatrix}$ and $\mathbf{J}_Y = \begin{pmatrix} \mathbf{E} & -\mathbf{r}_Y^* \end{pmatrix}$ for our example case.

Acceleration-level Formulation. We obtain the acceleration-level formulation of the original constraint by taking the time derivative of Equation (3.33),

$$\dot{\mathbf{C}}_v = \frac{\partial \mathbf{J}_X \cdot \mathbf{v}_X + \mathbf{J}_Y \cdot \mathbf{v}_Y}{\partial t} = \mathbf{J}_X \cdot \mathbf{a}_X + \mathbf{J}_Y \cdot \mathbf{a}_Y + \dot{\mathbf{J}}_X \cdot \mathbf{v}_X + \dot{\mathbf{J}}_Y \cdot \mathbf{v}_Y = \mathbf{0}$$

This way, we are able to reformulate the original constraint, specified in terms of generalized positions, to a constraint specified in terms of *generalized accelerations*. This acceleration-level formulation of the constraint can be generally written using the acceleration-level constraint function $\mathbf{C}_a : [\mathbf{a}_X, \mathbf{a}_Y] \to \mathbb{R}^m$ as

$$\mathbf{C}_a(\mathbf{a}_X, \mathbf{a}_Y) := \mathbf{J}_X \cdot \mathbf{a}_X + \mathbf{J}_Y \cdot \mathbf{a}_Y - \mathbf{c} = \mathbf{0}, \tag{3.34}$$

where $\mathbf{J}_X$ and $\mathbf{J}_Y$ are the Jacobian matrices as defined above, $\dot{\mathbf{J}}_X$ and $\dot{\mathbf{J}}_Y$ are the time derivatives of the matrices and $\mathbf{c} = -\dot{\mathbf{J}}_X \cdot \mathbf{v}_X - \dot{\mathbf{J}}_Y \cdot \mathbf{v}_Y$. Note, because the constraint is formulated directly in terms of generalized accelerations of the bodies and we know the relation between forces and accelerations, the equation provides a constraint on the forces that can act on the two bodies.

To complete the formulation of Equation (3.34) for our example case, we need to determine the value of $\mathbf{c}$. It is usually easier to compute $\mathbf{c}$ directly from $\dot{\mathbf{C}}_v$, rather than through the explicit derivation of the time derivatives of the Jacobian matrices,

$$\begin{aligned}
\dot{\mathbf{C}}_v &= \frac{\partial -\dot{\mathbf{x}}_X - \omega_X \times \mathbf{r}_X}{\partial t} + \frac{\partial \dot{\mathbf{x}}_Y + \omega_Y \times \mathbf{r}_Y}{\partial t} \\
&= (-\ddot{\mathbf{x}}_X - \dot{\omega}_X \times \mathbf{r}_X - \omega_X \times (\omega_X \times \mathbf{r}_X)) + (\ddot{\mathbf{x}}_Y + \dot{\omega}_Y \times \mathbf{r}_Y + \omega_Y \times (\omega_Y \times \mathbf{r}_Y)) \\
&= (-\ddot{\mathbf{x}}_X + \mathbf{r}_X^* \cdot \dot{\omega}_X - \omega_X \times (\omega_X \times \mathbf{r}_X)) + (\ddot{\mathbf{x}}_Y - \mathbf{r}_Y^* \cdot \dot{\omega}_Y + \omega_Y \times (\omega_Y \times \mathbf{r}_Y)) \\
&= \begin{pmatrix} -\mathbf{E} & \mathbf{r}_X^* & \mathbf{E} & -\mathbf{r}_Y^* \end{pmatrix} \cdot \begin{pmatrix} \ddot{\mathbf{x}}_X \\ \dot{\omega}_X \\ \ddot{\mathbf{x}}_Y \\ \dot{\omega}_Y \end{pmatrix} - \omega_X \times (\omega_X \times \mathbf{r}_X) + \omega_Y \times (\omega_Y \times \mathbf{r}_Y) \\
&= \begin{pmatrix} \mathbf{J}_X & \mathbf{J}_Y \end{pmatrix} \cdot \begin{pmatrix} \mathbf{a}_X \\ \mathbf{a}_Y \end{pmatrix} - \mathbf{c},
\end{aligned}$$

where $\mathbf{c} = \omega_X \times (\omega_X \times \mathbf{r}_X) - \omega_Y \times (\omega_Y \times \mathbf{r}_Y)$.

Given the original position-level formulation of the constraint in the form of $\mathbf{C}_p = \mathbf{0}$, we obtained the velocity-level formulation $\dot{\mathbf{C}}_p = \mathbf{0}$ and the acceleration-level formulation $\ddot{\mathbf{C}}_p = \mathbf{0}$ of the constraint. If $\mathbf{C}_p(\mathbf{x}_X(t), \mathbf{x}_Y(t)) = \mathbf{0}$ and $\dot{\mathbf{C}}_p(\mathbf{v}_X(t), \mathbf{v}_Y(t)) = \mathbf{0}$ at the current time t, we can implement the original position-level constraint incrementally by solving for the constraint forces to be applied on X and Y such that $\ddot{\mathbf{C}}_p(\mathbf{v}_X(t), \mathbf{v}_Y(t)) = \mathbf{0}$.

Constraint Force Basis. Our goal is to construct the constraint forces $\left(\mathbf{F}^{constraint}\right)_X$ and $\left(\mathbf{F}^{constraint}\right)_Y$ to be applied on the bodies in order to produce the accelerations $\mathbf{a}_X$, $\mathbf{a}_Y$ that are consistent with the constraint in Equation (3.34). Following the *Lagrange multiplier method*, [18], we define the constraint force for the constraint from Equation (3.34) as a linear combination of the rows of the Jacobian matrices,

$$\left(\mathbf{F}^{constraint}\right)_X = \mathbf{J}_X^T \cdot \lambda \tag{3.35}$$

$$\left(\mathbf{F}^{constraint}\right)_Y = \mathbf{J}_Y^T \cdot \lambda, \tag{3.36}$$

where the Jacobian matrices are *known a priori* and come from the definition of the constraint, and the coefficients (*Lagrange multipliers*) λ in the combination are *unknown* and have to be determined. As such, the constraint force is parameterized by λ. Furthermore, it is possible to predict, how the acceleration of the bodies changes had the constraint force been applied and we use this information to solve for the constraint force exactly.

Given the total external forces $\mathbf{F}_X^{total}$ and $\mathbf{F}_Y^{total}$ acting on the two bodies, the external forces produce the accelerations $\mathbf{a}_X = \mathbf{M}^{-1}X \cdot \mathbf{F}_X^{total}$ and $\mathbf{a}_Y = \mathbf{M}^{-1}X \cdot \mathbf{F}_Y^{total}$ of the bodies according to Equation (3.26). When the constraint force is added, the acceleration of the bodies changes and the resulting acceleration can be expressed as a function of the multipliers λ,

$$\mathbf{a}_X(\lambda) = \mathbf{M}^{-1}X \cdot (\mathbf{F}_X^{total} + \mathbf{J}_X^T \cdot \lambda) \tag{3.37}$$

$$\mathbf{a}_Y(\lambda) = \mathbf{M}^{-1}Y \cdot (\mathbf{F}_Y^{total} + \mathbf{J}_Y^T \cdot \lambda). \tag{3.38}$$

Solving for Multipliers. We solve for λ such that the resulting acceleration satisfies $\mathbf{J}_X \cdot \mathbf{a}_X(\lambda) + \mathbf{J}_Y \cdot \mathbf{a}_Y(\lambda) - \mathbf{c} = \mathbf{0}$ and the constraint in Equation (3.34) is maintained. Towards that end, we first stack individual vectors and matrices into global vectors and matrices that characterize the entire rigid body system, obtaining $\mathbf{a} = (\mathbf{a}_X, \mathbf{a}_Y)$, $\mathbf{J} = \begin{pmatrix} \mathbf{J}_X & \mathbf{J}_Y \end{pmatrix}$, $\mathbf{F}^{total} = (\mathbf{F}_X^{total}, \mathbf{F}_Y^{total})$, $\mathbf{F}^{constraint} = \mathbf{J}^T \cdot \lambda = ((\mathbf{F}^{constraint})_X, (\mathbf{F}^{constraint})_Y)$, $\mathbf{M} = \begin{pmatrix} \mathbf{M}_X & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_Y \end{pmatrix}$ and $\mathbf{J} \cdot \mathbf{a} = \mathbf{c}$.

Using this notation, we see that the acceleration $\mathbf{a}(\lambda)$ of the rigid body system after the total external force $\mathbf{F}^{total}$ and the constraint force $\mathbf{F}^{constraint}$ are applied on the system equals $\mathbf{a}(\lambda) = \mathbf{M}^{-1} \cdot (\mathbf{F}^{total} + \mathbf{F}^{constraint}) = \mathbf{M}^{-1} \cdot (\mathbf{F}^{total} + \mathbf{J}^T \cdot \lambda) = \mathbf{M}^{-1} \cdot \mathbf{F}^{total} + \mathbf{M}^{-1} \cdot \mathbf{J}^T \cdot \lambda$. This acceleration has to satisfy the constraint $\mathbf{J} \cdot \mathbf{a}(\lambda) = \mathbf{c}$ and so $\mathbf{J} \cdot \mathbf{M}^{-1} \cdot \mathbf{F}^{total} + \mathbf{J} \cdot \mathbf{M}^{-1} \cdot \mathbf{J}^T \cdot \lambda = \mathbf{c}$. We can rearrange the last equation as a system of linear equations in the form of

$$\mathbf{A} \cdot \lambda + \mathbf{b} = \mathbf{0}, \tag{3.39}$$

where $\mathbf{A} = \mathbf{J} \cdot \mathbf{M}^{-1} \cdot \mathbf{J}^T$ is a 3×3 matrix, $\mathbf{b} = \mathbf{J} \cdot \mathbf{M}^{-1} \cdot \mathbf{F}^{total} - \mathbf{c}$ is a 3×1 vector and $\lambda \in \mathbb{R}^3$ are the Lagrange multipliers to be solved for. Once λ are computed, the constraint force $\mathbf{F}^{constraint} = \mathbf{J}^T \cdot \lambda = ((\mathbf{F}^{constraint})_X, (\mathbf{F}^{constraint})_Y)$ is applied as a regular force on the bodies.

Hypersurface Interpretation. Maintaining the position-level constraint in Equation (3.32) corresponds to keeping the $(\mathbf{x}_X, \mathbf{x}_Y)$ point on the position manifold. We achieve this goal incrementally, by "projecting" the current acceleration of the system due to external forces to the set of valid accelerations, consistent with the acceleration-level formulation of the constraint in Equation (3.34).

Each row of the m rows in the acceleration-level formulation, $\mathbf{J}_X \cdot \mathbf{a}_X + \mathbf{J}_Y \cdot \mathbf{a}_Y = \mathbf{c} \in \mathbb{R}^m$, defines a hypersurface for valid accelerations in the space of all accelerations $(\mathbf{a}_X, \mathbf{a}_Y)$. Components within the linear combination $\mathbf{J}^T \cdot \lambda$ that implements the constraint force are associated with these hypersurfaces and realize projections to each of them.

Specifically, the $\mathbf{J}_k^T \cdot \lambda_k$, $1 \le k \le m$ component within the linear combination acts to project the acceleration $(\mathbf{a}_X, \mathbf{a}_Y)$ to the hypersurface due to the k-th constraint row, by affecting the acceleration along the normal of the hypersurface such that[3] $(\mathbf{J}_X)_k \cdot \mathbf{a}_X(\lambda) + (\mathbf{J}_Y)_k \cdot \mathbf{a}_Y(\lambda) - \mathbf{c}_k = 0$. The idea is that the constraint force component should only cancel the components in accelerations that move the $(\mathbf{x}_X, \mathbf{x}_Y)$ point away from the position manifold, along or against the normal direction, but let the point move freely along the manifold. Because the normal of the k-th hypersurface equals the k-th row of $\begin{pmatrix} \mathbf{J}_X & \mathbf{J}_Y \end{pmatrix}$, the constraint force component for the k-th row needs to apply the $\lambda_k \cdot (\mathbf{J}_X^T)_k$ force on the body X and $\lambda_k \cdot (\mathbf{J}_Y^T)_k$ on the body Y for some value of λ_k.

The linear system in Equation (3.39) represents interactions between the components $\mathbf{J}_k^T \cdot \lambda_k$ of the constraint force in changing the acceleration $\mathbf{a}(\lambda)$ of the rigid body system and the corresponding values $\mathbf{J} \cdot \mathbf{a}(\lambda) - \mathbf{c}_{k'}$ of the constraint function for the constraint rows $1 \le k' \le m$. Solving for λ in this system is equivalent to projecting the acceleration $\mathbf{a}$ to the intersection of all hypersurfaces such that $\mathbf{J} \cdot \mathbf{a}(\lambda) - \mathbf{c} = \mathbf{0}$.

Discussion. Although it is possible to build linear systems from Equation (3.39) for different constraints (and combinations of constraints) manually, doing so is not practical. In the following sections, we illustrate how

[3]Note that the value of the constraint function $\mathbf{J}_k \cdot \mathbf{a}(\lambda) - \mathbf{c}_k$ for the k-th row depends on the value of all multipliers λ, not only λ_k.

such systems can be build automatically for a variety of constraints and constraint types and how the systems can be solved using a constraint solver to produce a constraint force that satisfies all such constraints.

3.2.2 Acceleration Constraints

We now generalize the approach for handling a single constraint over two rigid bodies from the previous section for c constraints, n rigid bodies and other constraint types that are not described by equations with equalities.

We will focus directly on acceleration-level formulations of constraints, where accelerations of body pairs are constrained using linear functions. We use these constraints as building blocks for the construction of the physics-based model of the articulated body of the person. The constraints can take different forms and can be defined by equalities (see Section 3.2.2.3), inequalities (see Section 3.2.2.4) or equalities with bounded constraint forces (see Section 3.2.2.5). Each constraint type forces the value of the constraint function to a specific value, a range of values or combination of both subject to constraint force bounds.

3.2.2.1 Notation

We first define general notation for constraints, bodies and constraint forces that we are going to use in this section. We define this notation with respect to acceleration constraints defined by equality equations that force the value of the constraint function to a fixed value. We then provide formal definitions for other constraint types that we need to model in our system and show how constraint forces for all the constraint types can be solved for in terms of the multipliers λ, the matrix $\mathbf{A}$ and the vector $\mathbf{b}$. We obtain the value of λ as a solution to a *linear complementarity problem* (LCP) defined by the $\mathbf{A}$, $\mathbf{b}$ and bounds on the constraint force, as defined by the constraints.

Basic Notation. We will use the index j to index rigid bodies in the system, $j = 1, \ldots, n$, the index i to index constraints, $i = 1, \ldots, c$, and the index k to index constraint rows in constraint equations, $k = 1, \ldots, m_i$, where m_i is the number of constraint rows in the definition of the constraint i.

The vectors $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$, $\mathbf{v} = (\mathbf{v}_1, \ldots, \mathbf{v}_n)$ and $\mathbf{a} = (\mathbf{a}_1, \ldots, \mathbf{a}_n)$ will refer to the generalized position, velocity and acceleration of the rigid body system, $\mathbf{F}^{total} = \left(\mathbf{F}_1^{total}, \ldots, \mathbf{F}_n^{total}\right)$ will refer to the total external force exerted on the system and $\mathbf{F}^{constraint} = \left(\left(\mathbf{F}^{constraint}\right)_1, \ldots, \left(\mathbf{F}^{constraint}\right)_n\right)$ will refer to the total constraint force applied on the system due to all constraints.

Let $\mathbf{M}_j$ be the mass matrices of the individual bodies in the system. We define the *mass matrix of the rigid body system* $\mathbf{M}$ as the square block diagonal matrix $\mathbf{M}$ with the individual matrices $\mathbf{M}_j$ on the diagonal,

$$\mathbf{M} = \begin{pmatrix} \mathbf{M}_1 & \mathbf{0} & \ldots & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_2 & \ldots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \ldots & \mathbf{M}_n \end{pmatrix}.$$

Because $\mathbf{M}_j \cdot \mathbf{a}_j = \mathbf{F}_j^{total}$, we can express the acceleration $\mathbf{a}$ of the system due to the application of $\mathbf{F}^{total}$ as $\mathbf{a} = \mathbf{M}^{-1} \cdot \mathbf{F}^{total}$.

Constraint. We next describe the notation for a single constraint i. In defining the notation, we can imagine that the constraint describes an acceleration-level equality constraint from Equation (3.34) that we generalize for two selected bodies X_i and Y_i as

$$\mathbf{C}_a^i(\mathbf{a}_{X_i}, \mathbf{a}_{Y_i}) := \mathbf{J}_{i,X_i} \cdot \mathbf{a}_{X_i} + \mathbf{J}_{i,Y_i} \cdot \mathbf{a}_{Y_i} - \mathbf{c}_i = \mathbf{0}, \tag{3.40}$$

where $\mathbf{C}_a^i(\mathbf{a}_{X_i}, \mathbf{a}_{Y_i}) := \mathbf{J}_{i,X_i} \cdot \mathbf{a}_{X_i} + \mathbf{J}_{i,Y_i} \cdot \mathbf{a}_{Y_i} - \mathbf{c}_i$ is the constraint function for the constraint. The constraint may request *e.g.*, the two bodies to stay connected at a common anchor point, as in the point-to-point example from before.

Note that, although this particular constraint is defined by an equality equation in this case, requesting the linear combination of the accelerations of two bodies to equal a specific value, our system can also work with inequalities and equalities with bounded constraint forces that we introduce later in Section 3.2.2.4 and Section 3.2.2.5. While these other constraint types use different definitions for the constraint equation, they use the same constraint functions $\mathbf{C}_a^i$ and share the same notation that we explain here. We define the constraint notation with respect to the case of the equality constraint from Equation (3.40).

Single Constraint Notation. We assume the constraint i, defined *e.g.*, as in Equation (3.40), effects two bodies with the indices X_i and Y_i and is described by an equation that has m_i constraint rows. The constraint is characterized by a $m_i \times 6n$ matrix $\mathbf{J}_i$ of rank m_i called the *Jacobian matrix* of the constraint and the constraint equation vector $\mathbf{c}_i$ of the length m_i. We call X_i the first body constrained by the constraint and Y_i the second body. We refer to the value of $\mathbf{C}_a^i(\mathbf{a}_{X_i}, \mathbf{a}_{Y_i}) = \mathbf{J}_{i,X_i} \cdot \mathbf{a}_{X_i} + \mathbf{J}_{i,Y_i} \cdot \mathbf{a}_{Y_i} - \mathbf{c}_i$ as the value of the constraint function for the constraint.

The value of m_i defines the *dimensionality of the constraint* and determines how many m_i degrees of freedom (DOFs) are removed from the system by the constraint. For example, the point-to-point constraint we discussed earlier has $m_i = 3$, because the joint constrains the movement of the two bodies in three orthogonal directions at the anchor point. Note that although this constraint removes only 3 degrees of freedom, it affects both linear and angular properties of the two bodies. A hinge joint constraint removes two additional degrees of freedom, allowing only rotational motion of the bodies about a given axis at the anchor point, resulting in a constraint with $m_i = 5$, *etc*.

The matrix $\mathbf{J}_i$ consists of n blocks of size $m_i \times 6$. Blocks in the matrix are indexed by the bodies in the rigid body system such that each block is due to a particular body. Because the constraint is pair-wise by definition and can only effect the bodies X_i and Y_i, $\mathbf{J}_i$ has only two non-zero blocks, one due to the first constrained body X_i and one due to the second constrained body Y_i, referred to by $\mathbf{J}_{i,X_i}$ and $\mathbf{J}_{i,Y_i}$.

According to the Lagrange multiplier approach, the constraint i is implemented by applying a *constraint force* $\mathbf{F}_i^{constraint} = \mathbf{J}_i^T \cdot \lambda_i = \left(\left(\mathbf{F}_i^{constraint}\right)_1, \ldots, \left(\mathbf{F}_i^{constraint}\right)_n\right)$ to the rigid body system, where the constraint force is parameterized by the values of m_i multipliers encoded in λ_i and the generalized force $\left(\mathbf{F}_i^{constraint}\right)_j$ is applied on the j-th body. Each row $k = 1, \ldots, m_i$ of $\mathbf{J}_i$ removes one DOF from the system and contributes to the constraint force $\mathbf{F}_i^{constraint}$ by exerting the force $(\lambda_i)_k \cdot (\mathbf{J}_i)_k^T$ on the system. Due to the way $\mathbf{J}_i$ is defined, $\left(\mathbf{F}_i^{constraint}\right)_{X_i} = \mathbf{J}_{i,X_i}^T \cdot \lambda_i$ and $\left(\mathbf{F}_i^{constraint}\right)_{Y_i} = \mathbf{J}_{i,Y_i}^T \cdot \lambda_i$ are the only non-zero blocks of $\mathbf{F}_i^{constraint}$, where $\left(\mathbf{F}_i^{constraint}\right)_{X_i}$ is the constraint force applied to the first body X_i and $\left(\mathbf{F}_i^{constraint}\right)_{Y_i}$ is the constraint force applied to the second body Y_i.

Multiple Constraints Notation. To capture the effect of all constraints, defined *e.g.*, as in Equation (3.40), we stack the individual $m_i \times 6n$ Jacobian matrices $\mathbf{J}_i$ by rows to a single $m \times 6n$ Jacobian matrix $\mathbf{J}$, where $m = \sum_i m_i$ is the total number of DOFs removed from the system (constraint rows). After stacking, the matrix $\mathbf{J}$ is then a block matrix with $c \times n$ blocks indexed by constraints and bodies, with the only non-zero blocks being $\mathbf{J}_{i,X_i}$ and $\mathbf{J}_{i,Y_i}$ for $i = 1,\ldots,c$. Then, the total constraint force $\mathbf{F}^{constraint}$ exerted on the system equals $\mathbf{F}^{constraint} = \sum_i \mathbf{F}_i^{constraint} = \mathbf{J}^T \cdot \lambda$, where $\lambda = (\lambda_1,\ldots,\lambda_c)$ is a $m \times 1$ vector of Lagrange multipliers due to all constraints. Because individual constraints should not conflict each other, $\mathbf{J}$ is assumed to have full rank.

As before, we define $\mathbf{A} = \mathbf{J} \cdot \mathbf{M}^{-1} \cdot \mathbf{J}^T$ and $\mathbf{b} = \mathbf{J} \cdot \mathbf{M}^{-1} \cdot \mathbf{F}^{total} - \mathbf{c}$, where $\mathbf{c} = (\mathbf{c}_1,\ldots,\mathbf{c}_c)$. The matrix $\mathbf{A}$ is a $m \times m$ matrix that encodes interactions between components of the constraint force due to constraints and constraint rows in changing the values of the constraint functions for the constraints. It can be imaged as if it consists of $c \times c$ blocks due to individual constraint pairs such that the value of the (i_1,i_2)-th block of size $m_{i_1} \times m_{i_2}$ due to the i_1-th constraint and the i_2-th constraint is given by $\mathbf{A}_{i_1,i_2} = \sum_j \mathbf{J}_{i_1,j} \cdot \mathbf{M}_j^{-1} \cdot (\mathbf{J}_{i_2,j})^T$. This block models how the value of the constraint function $\mathbf{C}_a^{i_1}$ for the constraint i_1 changes in response to the scaling of the components of the constraint force due to the constraint i_2. We will use $\mathbf{A}_i$ to refer to the i-th block row of $\mathbf{A}$ due to the constraint i. Note that $\mathbf{A}$ is positive definite, because the matrices $\mathbf{M}_j$, $\mathbf{M}_j^{-1}$ are positive definite and $\mathbf{J}$ is assumed to have full rank. This fact is important for feasibility of solutions when solving for λ.

The vector $\mathbf{b}$ is a vector of the length m and can be imagined as consisting of c blocks due to individual constraints. We use $\mathbf{b}_i$ to refer to the i-th block $\mathbf{b}$ of the length m_i due to the constraint i.

Value of Constraint Function. When the total external force $\mathbf{F}^{total}$ is applied on the bodies in the system, the force produces the acceleration $\mathbf{a} = \mathbf{M}^{-1} \cdot \mathbf{F}^{total}$ and this acceleration has to be corrected by adding a constraint force $\mathbf{J}^T \cdot \lambda$ so that the constraints will hold. When the constraint force is added, the acceleration of the system changes and, because the constraint force is parameterized by λ, the resulting acceleration can be expressed as a function of λ,

$$\mathbf{a}(\lambda) = \mathbf{M}^{-1} \cdot (\mathbf{F}^{total} + \mathbf{J}^T \cdot \lambda). \tag{3.41}$$

Consequently, we can express the values of constraint functions for constraints in terms of λ.

Specifically, the value of the constraint function for the constraint i is determined by

$$\begin{aligned}
\mathbf{C}_a^i(\lambda) &:= \mathbf{J}_{i,X_i} \cdot \mathbf{a}_{X_i}(\lambda) + \mathbf{J}_{i,Y_i} \cdot \mathbf{a}_{Y_i}(\lambda) - \mathbf{c}_i \\
&= \mathbf{J}_i \cdot \mathbf{a}(\lambda) - \mathbf{c}_i \\
&= (\mathbf{J} \cdot \mathbf{a}(\lambda) - \mathbf{c})_i \\
&= \Big(\mathbf{J} \cdot \mathbf{M}^{-1} \cdot (\mathbf{F}^{total} + \mathbf{J}^T \cdot \lambda) - \mathbf{c}\Big)_i \\
&= \Big(\mathbf{J} \cdot \mathbf{M}^{-1} \cdot \mathbf{F}^{total} + \mathbf{J} \cdot \mathbf{M}^{-1} \cdot \mathbf{J}^T \cdot \lambda - \mathbf{c}\Big)_i \\
&= \Big((\mathbf{J} \cdot \mathbf{M}^{-1} \cdot \mathbf{J}^T) \cdot \lambda + \Big(\mathbf{J} \cdot \mathbf{M}^{-1} \cdot \mathbf{F}^{total} - \mathbf{c}\Big)\Big)_i \\
&= (\mathbf{A} \cdot \lambda + \mathbf{b})_i \\
&= \mathbf{A}_i \cdot \lambda + \mathbf{b}_i,
\end{aligned} \tag{3.42}$$

because $\mathbf{J}_i$ has the only non-zero blocks $\mathbf{J}_{i,X_i}$ and $\mathbf{J}_{i,Y_i}$. This relation between λ and the value of constraint functions is central for the process of computing λ and determining the constraint force. In particular, the relation encodes how a particular constraint i gets violated by applying a hypothetical constraint force with given values of the multipliers λ. Using this relation, we can solve for λ such that all constraints will be maintained exactly.

3.2.2.2 Overview and Outline

Given *acceleration constraints* in the system, defined *e.g.*, as in Equation (3.40), our goal is to illustrate how we can compute the constraint force, parameterized by λ, such that all constraints are maintained. We follow the Lagrange multiplier approach and *reformulate constraints, originally specified over accelerations of bodies in the rigid body system, as conditions over the multipliers λ that produce the constraint force. These conditions have a specific form and yield a problem in the literature known as the* LCP. We formulate the LCP in terms of $\mathbf{A}$ and $\mathbf{b}$, such that the solution to the LCP produces the multipliers λ that satisfy the acceleration constraints.

In formulating the LCP, we proceed first by providing formal definitions for the types of acceleration constraints that we need to model. We show for each acceleration constraint i with a given type the conditions on the values of λ, defined in terms of $\mathbf{A}$ and $\mathbf{b}$, that lead to the computation of the constraint force for the constraint. According to our earlier discussion, each constraint i will generate a constraint force of the same form $\mathbf{F}_i^{constraint} = \mathbf{J}_i^T \cdot \lambda_i$. However, different types of acceleration constraints will lead to different conditions on the values of λ, thus constraining how the constraint force can act. In deriving the actual conditions for λ for each constraint type and each constraint in the LCP, we will show that each constraint i can be characterized in terms of:

LCP1 *Linear conditions* on the values of λ that relate the value of the constraint function $\mathbf{C}_a^i(\lambda)$ to the values of *all* multipliers λ, using $\mathbf{A}$ and $\mathbf{b}$.

The purpose of these conditions is to force a value of λ such that the constraint i is maintained.

We express these conditions based on Equation (3.42), which characterizes how a particular constraint i gets violated by applying a constraint force $\mathbf{J}^T \cdot \lambda$ due to all constraints on the rigid body system. The goal of the linear conditions for the constraint i is to force a value of λ such that the resulting constraint force produces a value of the constraint function $\mathbf{C}_a^i(\lambda)$ that is requested by the constraint i.

We will show that acceleration constraints defined by *equalities* can be fully characterized using only the linear conditions on λ.

LCP2 *Bounds on the legal values of the multipliers λ_i* for the constraint force due to the constraint i.

The purpose of these conditions is to force a value of λ that is within given bounds so that the generated constraint force due to the constraint is bounded and/or only acts in a specific direction.

These conditions are specific to a particular constraint type and make the constraint force due to the constraint i act only in a specific direction and/or with a bounded magnitude. For example, the constraint force may be requested to only "push" the constrained bodies but not "pull". We discuss these conditions when we talk about inequality constraints and bounded equality constraints.

We will show that acceleration constraints defined by *inequalities* and *equalities with bounded constraint forces* require bounds on the values of the multipliers λ_i.

Bounding of the constraint multipliers has the effect that the constraint force for a given constraint can *e.g.*, only act in a specific direction, but not in the opposite direction (see Section 3.2.2.4) or that the amount of the force that the constraint can generate can not exceed a given limit (see Section 3.2.2.5). For example, to implement repulsification of two bodies using a constraint (to *e.g.*, prevent penetration), the corresponding constraint force needs to act to push the bodies away from each other, but it is not allowed to pull them back together. As such, the force can only act in one direction, allowing the bodies to move freely in the opposite direction.

LCP3 *Complementarity conditions* between the value of the constraint function $\mathbf{C}_a^i(\lambda)$ and the multipliers λ_i due to the constraint i.

The purpose of these conditions is to force a value of λ that minimizes the amount of the generated force for the constraint i.

These conditions force a value for λ_i that makes the constraint force act as lazily as possible. Specifically, the conditions prevent the constraint force due to the constraint i from acting if the constraint is already satisfied by an external force or a constraint force due to another constraint. We define complementarity conditions when we introduce inequality and bounded equality constraints.

Finally, we use the derived conditions to build the LCP representation of all acceleration constraints and show how the LCP can be solved to obtain λ and the constraint force.

3.2.2.3 Equality Constraints

We now introduce equality (bilateral) constraints. We use equality constraints to *e.g.*, implement joints that anchor parts of the model of the person together.

Definition. We define an *acceleration-level equality constraint i* that acts on two bodies X_i and Y_i by a constraint equation

$$\mathbf{J}_{i,X_i} \cdot \mathbf{a}_{X_i}(\lambda) + \mathbf{J}_{i,Y_i} \cdot \mathbf{a}_{Y_i}(\lambda) - \mathbf{c}_i = \mathbf{0}, \tag{3.43}$$

where $\mathbf{J}_{i,X_i}$ and $\mathbf{J}_{i,Y_i}$ are two $m_i \times 6$ matrices, $\mathbf{c}_i$ is a $m_i \times 1$ vector and m_i is the dimensionality of the constraint.

The $\mathbf{J}_{i,X_i}$ and $\mathbf{J}_{i,Y_i}$ matrices are called the Jacobian blocks due to the first and the second body and are supposed to have full rank. When the acceleration-level constraint implement a position-level constraint $\mathbf{C}_p(\mathbf{x}_{X_i}, \mathbf{x}_{Y_i}) = \mathbf{0}$ or a velocity-level constraint $\mathbf{C}_v(\mathbf{v}_{X_i}, \mathbf{v}_{Y_i}) = \mathbf{0}$, then $\mathbf{J}_{i,X_i} = \frac{\partial \mathbf{C}_p}{\partial \mathbf{x}_{X_i}}$, $\mathbf{J}_{i,Y_i} = \frac{\partial \mathbf{C}_p}{\partial \mathbf{x}_{Y_i}}$ or $\mathbf{J}_{i,X_i} = \frac{\partial \mathbf{C}_v}{\partial \mathbf{v}_{X_i}}$, $\mathbf{J}_{i,Y_i} = \frac{\partial \mathbf{C}_v}{\partial \mathbf{v}_{Y_i}}$.

The equality constraint constrains the accelerations $\mathbf{a}_{X_i}(\lambda)$ and $\mathbf{a}_{Y_i}(\lambda)$ of the two bodies such that the value of the constraint function $\mathbf{C}_a^i(\lambda) = \mathbf{J}_{i,X_i} \cdot \mathbf{a}_{X_i}(\lambda) + \mathbf{J}_{i,Y_i} \cdot \mathbf{a}_{Y_i}(\lambda) - \mathbf{c}_i$ is exactly $\mathbf{0}$. The constraint is called an equality constraint, because it is described by a linear equality.

LCP Conditions on Multipliers. The equality constraint only requires the satisfaction of $\mathbf{C}_a^i(\lambda) = \mathbf{0}$. In doing so, it does not impose any bounds on the multipliers λ_i due to the constraint and it does not impose any complementarity conditions on the value of the constraint function and the multipliers. Consequently, the LCP conditions for the equality constraint i are

$$\mathbf{C}_a^i(\lambda) = \mathbf{0}. \tag{3.44 [LCP1]}$$

According to Equation (3.42), these conditions can be rewritten using the matrix $\mathbf{A}$ and the vector $\mathbf{b}$ as

$$\mathbf{A}_i \cdot \lambda + \mathbf{b}_i = \mathbf{0}, \tag{3.45 [LCP1]}$$

which is a system of linear equations with the unknown λ.

3.2.2.4 Inequality Constraints

We now define inequality (unilateral) constraints that replace the $=$ sign in the constraint equation with the $\geq$ or $\leq$ sign. We use inequality constraints to *e.g.*, implement non-penetration between body parts and the parts and the environment. We also use inequality constraints to implement joint angle limits at joints in the body model of the person so that *e.g.*, forearm would not bend backwards.

Definition. We define an *acceleration-level inequality constraint i* that acts on two bodies X_i and Y_i by the constraint equation

$$\mathbf{J}_{i,X_i} \cdot \mathbf{a}_{X_i}(\lambda) + \mathbf{J}_{i,Y_i} \cdot \mathbf{a}_{Y_i}(\lambda) - \mathbf{c}_i \geq \mathbf{0}, \tag{3.46}$$

where $\mathbf{J}_{i,X_i}$ and $\mathbf{J}_{i,Y_i}$ are $m_i \times 6$ Jacobian matrices due to the constrained bodies, $\mathbf{c}_i$ is a $m_i \times 1$ vector and m_i is the dimensionality of the constraint. The inequality constraint constrains the accelerations $\mathbf{a}_{X_i}(\lambda)$ and $\mathbf{a}_{Y_i}(\lambda)$ of the two bodies such that the value of the constraint function $\mathbf{C}_a^i(\lambda) = \mathbf{J}_{i,X_i} \cdot \mathbf{a}_{X_i}(\lambda) + \mathbf{J}_{i,Y_i} \cdot \mathbf{a}_{Y_i}(\lambda) - \mathbf{c}_i$ is greater or equal to $\mathbf{0}$, componentwise.

Derivation. We now derive conditions on the multipliers λ due to the inequality constraint i.

LCP1 The constraint equation directly requires $\mathbf{C}_a^i(\lambda) \geq \mathbf{0}$, which we will use as one of our conditions on λ, hence

$$\mathbf{C}_a^i(\lambda) \geq \mathbf{0}. \tag{3.47}$$

LCP2 Furthermore, we need to restrict legal directions for the components of the constraint force, so that the force will not "pull" the bodies to the state where $\mathbf{C}_a^i(\lambda) = \mathbf{0}$. This restriction will result in conditions on the bounds of the multipliers λ_i.

According to our definition of the constraint force $\mathbf{F}_i^{constraint} = \mathbf{J}_i^T \cdot \lambda_i$ due to the constraint i, the constraint force acts to cancel the components of accelerations that make the bodies violate the constraint. In the case of an equality constraint, discussed in Section 3.2.1, the constraint force acted to project the accelerations of bodies due to external forces on the intersection of the hypersurfaces due to the constraint rows, by cancelling any accelerations along the directions of the normals of the hypersurfaces. In the case of an inequality (greater-or-equal) constraint, however, accelerations to the front of hypersurfaces, along the positive directions of the normals, are legal and must not be canceled. Consequently, the constraint force can only act along the positive directions of normals, which implies that the multipliers have to be bounded as

$$\lambda_i \geq \mathbf{0}. \tag{3.48}$$

LCP3 Lastly, the generated constraint force should do as little work as possible such that it only acts along a given hypersurface $k = 1, \ldots, m_i$ if and only if $\left(\mathbf{C}_a^i\right)_k(\lambda) = 0$. This requirement results in *complementarity conditions* on the values of $\left(\mathbf{C}_a^i\right)_k(\lambda)$ and $(\lambda_i)_k$ for each k.

The purpose of these conditions is to make the constraint force work as lazy as possible such that no energy is added to the system.

Recall that the $(\mathbf{J}_i)_k \cdot (\lambda_i)_k$ component within the linear combination $\mathbf{F}_i^{constraint} = \mathbf{J}_i^T \cdot \lambda_i$ implements the constraint force for the k-th row of the constraint i and controls the acceleration along the k-th hypersurface. We now define the complementarity conditions for the constraint row k.

If $\left(\mathbf{C}_a^i\right)_k(\lambda) > 0$ such that the bodies accelerate to the front of the hypersurface k under the constraint force $\mathbf{J}^T \cdot \lambda$, then there is no need to push the bodies in that direction even more. Hence, in order to minimize the amount of work done, the constraint force due to that hypersurface must vanish, that is, $(\lambda_i)_k = 0$,

$$\left(\mathbf{C}_a^i\right)_k(\lambda) > 0 \Rightarrow (\lambda_i)_k = 0. \tag{3.49}$$

Similarly, if the constraint force $(\mathbf{J}_i)_k \cdot (\lambda_i)_k$ is acting along the normal of the hypersurface k, $(\lambda_i)_k > 0$, then it should only work as little as possible to make the bodies stay on the hypersurface, that is, $\left(\mathbf{C}_a^i\right)_k(\lambda) = 0$,

$$(\lambda_i)_k > 0 \Rightarrow \left(\mathbf{C}_a^i\right)_k(\lambda) = 0. \tag{3.50}$$

The complementarity conditions can be rewritten in the form of single a condition,

$$\left(\mathbf{C}_a^i\right)_k(\lambda) \cdot (\lambda_i)_k = 0. \tag{3.51}$$

To interpret the complementarity of $\left(\mathbf{C}_a^i\right)_k(\lambda)$ and $(\lambda_i)_k$ in an intuitive way, imagine that the inequality constraint implements non-penetration between the ground and a ball at rest such that $\left(\mathbf{C}_a^i\right)_k(\lambda)$ measures the upward acceleration of the ball and $(\lambda_i)_k$ defines the amount of the non-penetration force applied on the ball to prevent it from going down through the ground. Now, if the ball already accelerates away from the ground with a positive acceleration, the non-penetration force need not act and so it is forced to not act. On the other hand, if the force acts, it should only act to cancel the downward acceleration, but it should not bounce the ball away from the ground with a positive acceleration.

LCP Conditions on Multipliers. The inequality constraint requires the satisfaction of $\mathbf{C}_a^i(\lambda) \geq \mathbf{0}$. In doing so, it imposes bounds on the multipliers λ_i due to the constraint, $\lambda_i \geq \mathbf{0}$, and it requires complementarity between the values of the constraint function and the multipliers. Consequently, the LCP conditions for the inequality constraint i are

$$\mathbf{C}_a^i(\lambda) \geq \mathbf{0} \tag{3.52 [LCP1]}$$

$$\lambda_i \geq \mathbf{0} \tag{3.53 [LCP2]}$$

$$\mathbf{C}_a^i \cdot \lambda_i = 0, \tag{3.54 [LCP3]}$$

where $\lambda_i \cdot \mathbf{C}_a^i = \sum_{k=1}^{m_i} (\lambda_i)_k \cdot \left(\mathbf{C}_a^i\right)_k = 0$ in facts means $(\lambda_i)_k \cdot \left(\mathbf{C}_a^i\right)_k = 0$ for $k = 1, \ldots, m_i$, because both the products have to be non-negative.

According to Equation (3.42), these conditions can be rewritten using the matrix $\mathbf{A}$ and the vector $\mathbf{b}$ as

$$\mathbf{A}_i \cdot \lambda + \mathbf{b}_i \geq \mathbf{0} \tag{3.55 [LCP1]}$$

$$\lambda_i \geq \mathbf{0} \tag{3.56 [LCP2]}$$

$$(\mathbf{A}_i \cdot \lambda + \mathbf{b}_i) \cdot \lambda_i = 0. \tag{3.57 [LCP3]}$$

Less-Or-Equal Inequalities. Notice that we defined the inequality constraint as a "greater-or-equal" constraint. We note that the "less-or-equal" variant that replaces the $\geq$ sign with the $\leq$ sign in the constraint equation can be defined and handled analogously. The "less-or-equal" variant can also be converted to the "greater-or-equal" case by multiplying the constraint equation by -1.

3.2.2.5 Bounded Equality Constraints

As the next step, we define a constraint that behaves like an equality constraint that is maintained subject to constraint force bounds. The constraint works as hard as possible to satisfy the constraint equality, but it is allowed to be violated if a force bound is hit. We specify bounds for the constraint force as bounds on the values of the multipliers due to the constraint rows in the equality. The bounds determine whether the generated constraint force due to the constraint can "pull" and/or "push" and how strongly when working to maintain the equality.

We use bounded equality constraints to build "motors" with limited power that implement actuation of the model of the person and to model friction between parts of the body and the parts and the environment. These constraints employ a best-effort strategy to achieve the constraint objective (*e.g.*, to contract an arm or to stop contacting bodies from sliding), but are allowed to fail gracefully if the objective can not be satisfied because of a power limit (*e.g.*, joint torque bound, friction cone), given by the bounds.

Motivation. Our goal is to define a constraint i that acts like an equality constraint $\mathbf{J}_i \cdot \mathbf{a}(\lambda) - \mathbf{c}_i = \mathbf{0}$ but that breaks if the magnitude of the generated constraint force $\mathbf{F}_i^{constraint} = \mathbf{J}_i^T \cdot \lambda_i$ exceeds a limit.

If the constraint is one-dimensional, such that $m_i = 1$, the magnitude of the constraint force is given by $||\mathbf{J}_i^T|| \cdot |(\lambda_i)_1|$. Because $||\mathbf{J}_i^T||$ is fixed, given the constraint definition, limiting of the magnitude can be realized by enforcing lower and upper bounds on the legal values of the multiplier $(\lambda_i)_1$. Consequently, we assume the constraint force is bounded by specifying the limits on the multiplier.

In the general case of multi-dimensional constraints, we approximate the L_2 norm in $||\mathbf{F}_i^{constraint}||$ with L_∞ norm, for computational tractability, and implement the bounding of the constraint force by limiting the multipliers. Towards that end, we assume that each multiplier has its own bounds, independent of the values of other multipliers, such that bounding of different multipliers can be realized separately. We assume that the bounds for the bounded equality constraint i are specified by the vectors λ_i^{lo} and λ_i^{hi} of the length m_i such that the bounding of the multipliers λ_i due to the constraint can be realized componentwise by requesting

$$\left(\lambda_i^{lo}\right)_k \leq (\lambda_i)_k \leq \left(\lambda_i^{hi}\right)_k, \tag{3.58}$$

where $k = 1, \ldots, m_i$.

Note that the bounding of the multipliers and the graceful failure of the constraint can not be emulated by treating the constraint as an original (unbounded) equality constraint, solving for the exact constraint force due to λ and clamping the multipliers λ_i into the valid range by applying Equation (3.58). The clamping may change the values of possibly all constraint functions $\mathbf{C}_a^{i'}(\lambda)$, thus, breaking all other constraints $i' \neq i$ in the system. Instead, the clamping has to be integrated with the LCP formulation such that the solution process that computes λ is aware of the effects of Equation (3.58). Towards that end, we define a new bounded equality constraint type.

Similarly to equality and inequality constraints, we will define this new constraint in terms of the constraint force $\mathbf{F}_i^{constraint} = \mathbf{J}_i^T \cdot \lambda_i$ and LCP conditions on the corresponding multipliers λ_i to handle the clamping. The multipliers λ_i have to be not only clamped in the LCP formulation but they also have to be tied to the value of the constraint function $\mathbf{C}_a^i(\lambda)$ by complementarity conditions to minimize the amount of the generated force.

Definition. We define an *acceleration-level bounded equality constraint* i as a constraint that acts on two bodies X_i and Y_i, has a dimensionality m_i and is specified by two $m_i \times 6$ matrices $\mathbf{J}_{i,X_i}$ and $\mathbf{J}_{i,Y_i}$, a vector $\mathbf{c}_i$ of the length m_i and bounds $\lambda_i^{lo} \leq \mathbf{0}$ and $\lambda_i^{hi} \geq \mathbf{0}$ on the values of the multipliers λ_i due to the constraint. The constraint equation requests

$$\mathbf{J}_{i,X_i} \cdot \mathbf{a}_{X_i}(\lambda) + \mathbf{J}_{i,Y_i} \cdot \mathbf{a}_{Y_i}(\lambda) - \mathbf{c}_i = \mathbf{0} \tag{3.59}$$

subject to constraint force multipliers bounds

$$\lambda_i^{lo} \leq \lambda_i \leq \lambda_i^{hi}, \tag{3.60}$$

such that Equation (3.59) can be violated if a multiplier bound is hit.

Specifically, the constraint works to keep the value of the constraint function $\mathbf{C}_a^i(\lambda) = \mathbf{J}_{i,X_i} \cdot \mathbf{a}_{X_i}(\lambda) + \mathbf{J}_{i,Y_i} \cdot \mathbf{a}_{Y_i}(\lambda) - \mathbf{c}_i$ at $\mathbf{0}$, componentwise, but it is allowed to produce $\left(\mathbf{C}_a^i\right)_k(\lambda) > 0$ or $\left(\mathbf{C}_a^i\right)_k(\lambda) < 0$ for a constraint row k as long as the multiplier $(\lambda_i)_k$ of the constraint force $(\mathbf{J}_i)_k^T \cdot (\lambda_i)_k$ due to the constraint row hits either the lower bound $\left(\lambda_i^{lo}\right)_k$ or the upper bound $\left(\lambda_i^{hi}\right)_k$.

Derivation. We now derive conditions on the multipliers λ due to the bounded equality constraint i. Because individual multipliers are bounded independently in our definition of the bounded equality constraint, we derive the conditions for the multipliers for each constraint row k separately. If the multiplier bounds do not permit generation of any force, $\left(\lambda_i^{lo}\right)_k = \left(\lambda_i^{hi}\right)_k = 0$, we ignore the constraint row.

We follow up on the hypersurface interpretation of the equality constraint $\mathbf{C}_a^i(\lambda) = \mathbf{0}$.

LCP1 According to the definition, the constraint is supposed to work as hard as possible to maintain $\left(\mathbf{C}_a^i\right)_k(\lambda) = 0$ and the equation can only break if the constraint force $(\mathbf{J}_i)_k^T \cdot (\lambda_i)_k$ is at a bound, hence

$$\left(\lambda_i^{lo}\right)_k < (\lambda_i)_k < \left(\lambda_i^{hi}\right)_k \Rightarrow \left(\mathbf{C}_a^i\right)_k(\lambda) = 0. \tag{3.61}$$

LCP2 The constraint definition also directly requires the multiplier $(\lambda_i)_k$ to be bounded, hence

$$\left(\lambda_i^{lo}\right)_k \leq (\lambda_i)_k \leq \left(\lambda_i^{hi}\right)_k. \tag{3.62}$$

LCP3 When a constraint force bound is hit, the constraint equation is allowed to break such that $\left(\mathbf{C}_a^i\right)_k(\lambda) \geq 0$ or $\left(\mathbf{C}_a^i\right)_k(\lambda) \leq 0$.

The breakage results from the fact that the acceleration along the normal of the hypersurface due to the constraint row k has to be cancelled by the constraint force $(\mathbf{J}_i)_k^T \cdot (\lambda_i)_k$ in order to maintain $\left(\mathbf{C}_a^i\right)_k(\lambda) = 0$ exactly. When the multiplier $(\lambda_i)_k$ is forced to stay within the bounds, $\left(\lambda_i^{lo}\right)_k \leq (\lambda_i)_k \leq \left(\lambda_i^{hi}\right)_k$, the generated constraint force may not be strong enough to cancel the acceleration, resulting in $\left(\mathbf{C}_a^i\right)_k(\lambda) \geq 0$ or $\left(\mathbf{C}_a^i\right)_k(\lambda) \leq 0$, depending on which bound is hit.

Specifically, if the lower bound is hit, $(\lambda_i)_k = \left(\lambda_i^{lo}\right)_k > 0$, the constraint force was trying to "pull" $\mathbf{a}(\lambda)$ from the front of the hypersurface onto the hypersurface, because $\left(\lambda_i^{lo}\right)_k > 0$. Since the bound was hit, the resulting λ may yield the acceleration $\mathbf{a}(\lambda)$ that is still in the front of the hypersurface, hence $\left(\mathbf{C}_a^i\right)_k(\lambda) \geq 0$,

$$(\lambda_i)_k = \left(\lambda_i^{lo}\right)_k \Rightarrow \left(\mathbf{C}_a^i\right)_k(\lambda) \geq 0. \tag{3.63}$$

Similarly, if the upper bound is hit, $0 < \left(\lambda_i^{hi}\right)_k = (\lambda_i)_k$, the constraint force was trying to "push" $\mathbf{a}(\lambda)$ from the back of the hypersurface onto the hypersurface, because $0 < \left(\lambda_i^{hi}\right)_k$. Since the bound was hit, the resulting λ may yield the acceleration that is still in the back of the hypersurface and so $\left(\mathbf{C}_a^i\right)_k(\lambda) \leq 0$,

$$(\lambda_i)_k = \left(\lambda_i^{hi}\right)_k \Rightarrow \left(\mathbf{C}_a^i\right)_k(\lambda) \leq 0. \tag{3.64}$$

LCP Conditions on Multipliers. The bounded equality constraint imposes bounds on the multipliers λ_i due to the constraint, requires satisfaction of $\mathbf{C}_a^i(\lambda) = \mathbf{0}$ subject to constraint force bounds and ties the value of the constraint function $\mathbf{C}_a^i(\lambda)$ to the value of the multipliers λ_i using complementarity constraints. The LCP conditions for the bounded equality constraint are

$$\left(\lambda_i^{lo}\right)_k \leq (\lambda_i)_k \leq \left(\lambda_i^{hi}\right)_k \tag{3.65 [LCP2]}$$

$$(\lambda_i)_k = \left(\lambda_i^{lo}\right)_k \Rightarrow \left(\mathbf{C}_a^i\right)_k(\lambda) \geq 0 \tag{3.66 [LCP3]}$$

$$(\lambda_i)_k = \left(\lambda_i^{hi}\right)_k \Rightarrow \left(\mathbf{C}_a^i\right)_k(\lambda) \leq 0 \tag{3.67 [LCP3]}$$

$$\left(\lambda_i^{lo}\right)_k < (\lambda_i)_k < \left(\lambda_i^{hi}\right)_k \Rightarrow \left(\mathbf{C}_a^i\right)_k(\lambda) = 0, \tag{3.68 [LCP1+ LCP3]}$$

where $k = 1, \ldots, m_i$.

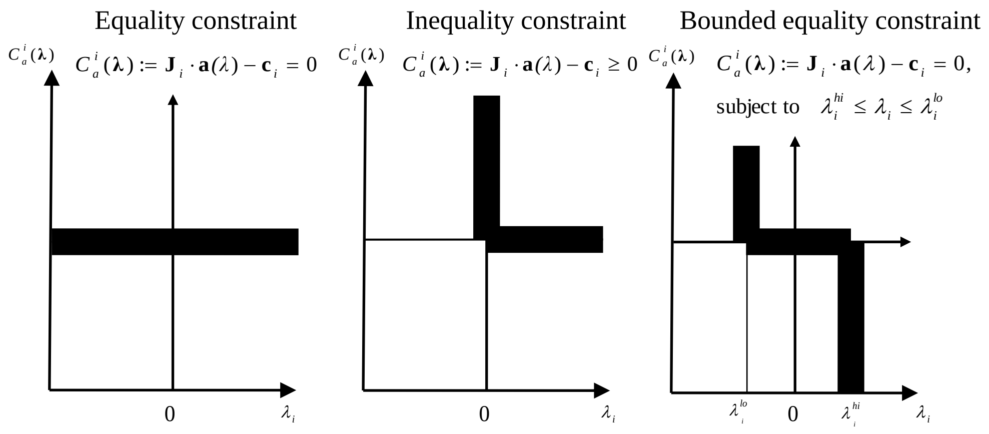

Equality constraint
Inequality constraint
Bounded equality constraint
$C_a^i(\lambda)$
$C_a^i(\lambda) := J_i \cdot a(\lambda) - c_i = 0$
$C_a^i(\lambda) := J_i \cdot a(\lambda) - c_i \geq 0$
$C_a^i(\lambda) := J_i \cdot a(\lambda) - c_i = 0,$
subject to $\lambda_i^{hi} \leq \lambda_i \leq \lambda_i^{lo}$
0
λ_i
λ_i^{lo}
λ_i^{hi}

Figure 3.1: **LCP Conditions.** Visualization of complementarity conditions on the values of λ_i and $\mathbf{C}_a^i(\lambda)$ due to different kinds of one dimensional constraints i, where $\mathbf{C}_a^i(\lambda) = \mathbf{J}_{i,X_i} \cdot \mathbf{a}_{X_i}(\lambda) + \mathbf{J}_{i,Y_i} \cdot \mathbf{a}_{Y_i}(\lambda) - \mathbf{c}_i = \mathbf{A}_i \cdot \lambda + \mathbf{b}_i$. Thick lines indicate permissible values for the $(\lambda_i, \mathbf{C}_a^i(\lambda))$ pairs. As can be seen, the equality constraint (left) requests $\mathbf{C}_a^i(\lambda)$ to be zero and allows λ_i to take an arbitrary value. The inequality constraint (middle) requests both $\mathbf{C}_a^i(\lambda)$ and λ_i to be non-negative and complementary to each other such that $\mathbf{C}_a^i(\lambda) \cdot \lambda_i = 0$. The bounded equality constraint (right) generalizes the two previous cases by introducing explicit bounds $\lambda_i^{lo} \leq \mathbf{0}$ and $\lambda_i^{hi} \geq \mathbf{0}$ on the values of λ_i. Note that, we illustrate the situation for one-dimensional constraints with one constraint row. For constraints with multiple dimensions, each row produces similar LCP conditions as illustrated in the diagram.

These conditions can be rewritten using the matrix $\mathbf{A}$ and the vector $\mathbf{b}$ as

$$\left(\lambda_i^{lo}\right)_k \leq (\lambda_i)_k \leq \left(\lambda_i^{hi}\right)_k \tag{3.69 [LCP2]}$$

$$(\lambda_i)_k = \left(\lambda_i^{lo}\right)_k \Rightarrow \mathbf{A}_i \cdot \lambda + \mathbf{b}_i \geq 0 \tag{3.70 [LCP3]}$$

$$(\lambda_i)_k = \left(\lambda_i^{hi}\right)_k \Rightarrow \mathbf{A}_i \cdot \lambda + \mathbf{b}_i \leq 0 \tag{3.71 [LCP3]}$$

$$\left(\lambda_i^{lo}\right)_k < (\lambda_i)_k < \left(\lambda_i^{hi}\right)_k \Rightarrow \mathbf{A}_i \cdot \lambda + \mathbf{b}_i = 0 \tag{3.72 [LCP1+ LCP3]}$$

for $k = 1, \ldots, m_i$. See Figure 3.1, right.

Special Cases. The bounded equality constraint type generalizes both inequality and unbounded equality constraints (see Figure 3.1):

- If we set $\lambda_i^{lo} = \mathbf{0}$ and $\lambda_i^{hi} = +\infty$, then the bounded equality constraint i turns to an equivalent greater-or-equal constraint i with the same Jacobian blocks and the vector $\mathbf{c}_i$.
- By setting $\lambda_i^{lo} = -\infty$ and $\lambda_i^{hi} = \mathbf{0}$, the bounded equality constraint i turns to an equivalent less-or-equal constraint.
- Finally, $\lambda_i^{lo} = -\infty$ and $\lambda_i^{hi} = +\infty$ produced an unbounded equality constraint.

Consequently, we only need to implement bounded equality constraints.

3.2.2.6 Soft Constraints

The constraints we have defined so far were all considered "hard" such that the requested values of the constraint functions were to be maintained exactly. In many cases, such a behavior is not desirable and we may want to specify constraints that are designed to break.

In this section, we introduce a special kind of "soft" constraints, defined according to [169], that request slack in the value of the constraint function such that the original constraint equation is not maintained exactly. These constraints request the slack to be proportional to the constraint force that implements the constraint, allowing bigger slack for constraints that have to work harder at the current time. When soft constraints are combined with constraint stabilization (*e.g.*, see Section 3.2.4.1), interesting spring-like behaviors can be realized, where soft constraints allow the original hard constraint to break and the stabilization mechanism works to recover from the breakage, by reducing the current constraint error. We use these two mechanisms to implement body actuation in Section 3.6.

Note that this definition of soft constraints may be different from "soft constraints" used in computer graphics. Specifically, our soft constraints request an exact slack in the value of the constraint function and this *slack is enforced exactly*. In contrast to other soft constraint formulations, in particular, we do not use optimization to minimize the slack, but achieve the desired slack. As such, our soft constraints are a variant of the original hard constraints that do not request any slack. Similarly to hard constraints, soft constraints prescribe control objectives for the simulation that are all achieved exactly by solving a LCP. We implement soft constraints as a different form of hard constraints and look for an exact solution, via a LCP, that satisfies all of them. Consequently, the effects of soft constraints are predictable and guaranteed.

We introduce soft constraints for the case of unbounded equality constraints, but the exactly same approach can be taken to implement soft inequality and soft bounded equality constraints.

Definition. We define a *soft acceleration-level equality constraint i* that acts on two bodies X_i and Y_i by a constraint equation

$$\mathbf{J}_{i,X_i} \cdot \mathbf{a}_{X_i}(\lambda) + \mathbf{J}_{i,Y_i} \cdot \mathbf{a}_{Y_i}(\lambda) - \mathbf{c}_i - \mathbf{s}_i \cdot \mathbf{E} \cdot \lambda_i = \mathbf{0}, \tag{3.73}$$

where $\mathbf{J}_{i,X_i}$ and $\mathbf{J}_{i,Y_i}$ are two $m_i \times 6$ matrices, $\mathbf{c}_i$ is a $m_i \times 1$ vector, $\mathbf{s}_i$ is a $m_i \times 1$ vector of softness constants, m_i is the dimensionality of the constraint and λ_i is a vector of Lagrange multipliers due to the constraint. The individual softnesses constants $(\mathbf{s}_i)_k$ are positive numbers and indicate how much soft the particular constraint row k is. A zero softness indicates that the constraint row should be hard and work exactly as before.

When compared to the corresponding hard constraint in Equation (3.43), the constraint equation for the soft constraint defines the slack $\mathbf{s}_i \cdot \mathbf{E} \cdot \lambda_i$ that is to be achieved with respect to the hard constraint. This slack is proportional to the softness constants and the multipliers λ_i that measure how hard the constraint force needs to work to maintain the constraint. The soft formulation attempts to trade off the constraint slack (error) for the amount of the generated force, by directly requesting the slack at the k-th row to be $(\mathbf{s}_i)_k \cdot (\lambda_i)_k$, allowing more slack for constraint rows that have to work harder.

The soft constraint equation in Equation (3.73) serves as a replacement for Equation (3.43). The soft equality constraint constrains the accelerations $\mathbf{a}_{X_i}(\lambda)$ and $\mathbf{a}_{Y_i}(\lambda)$ of the two bodies such that the value of the *soft constraint function* $\mathbf{C}^i_{a,s}(\lambda) = \mathbf{J}_{i,X_i} \cdot \mathbf{a}_{X_i}(\lambda) + \mathbf{J}_{i,Y_i} \cdot \mathbf{a}_{Y_i}(\lambda) - \mathbf{c}_i - \mathbf{s}_i \cdot \mathbf{E} \cdot \lambda_i$ is exactly $\mathbf{0}$. In doing so, it constrains the accelerations such that corresponding hard constraint function $\mathbf{C}_a^i(\lambda)$ attains the error $\mathbf{s}_i \cdot \mathbf{E} \cdot \lambda_i$.

Derivation. Our goal is to reformulate the soft constraint as a set of hard conditions on the multipliers λ for the LCP. As it turns out, the incorporation of softness constants only changes the definition of the matrix $\mathbf{A}$. As such, the LCP conditions over λ can be expressed using this new matrix $\mathbf{A}$, that we refer to as $\mathbf{A^s}$, in the same way as before.

To show these results, we need to evaluate how the value of the soft constraint function $\mathbf{C}_{a,s}^i(\lambda)$ changes as a function of λ and use it everywhere in place of the original hard constraint function $\mathbf{C}_a^i(\lambda)$. Let us assume that all constraints i have defined softness constants $\mathbf{s}_i$ that we stack into $\mathbf{s} = (\mathbf{s}_1, \ldots, \mathbf{s}_c)$. Let us also define the softness matrix $\mathbf{S}$ as a square matrix with the values of $\mathbf{s}$ on the diagonal, $\mathbf{S} = \mathbf{s} \cdot \mathbf{E}$. Then, by following the derivation from Equation (3.42), we see that

$$\begin{aligned}
\mathbf{C}_{a,s}^i(\lambda) &:= \mathbf{J}_i \cdot \mathbf{a}(\lambda) - \mathbf{c}_i - \mathbf{s}_i \cdot \mathbf{E} \cdot \lambda_i \\
&= (\mathbf{J} \cdot \mathbf{a}(\lambda) - \mathbf{c} - \mathbf{S} \cdot \lambda)_i \\
&= \Big(\mathbf{J} \cdot \mathbf{M}^{-1} \cdot \Big(\mathbf{F}^{total} + \mathbf{J}^T \cdot \lambda\Big) - \mathbf{c} - \mathbf{S} \cdot \lambda\Big)_i \\
&= \Big((\mathbf{J} \cdot \mathbf{M}^{-1} \cdot \mathbf{J}^T + \mathbf{S}) \cdot \lambda + \Big(\mathbf{J} \cdot \mathbf{M}^{-1} \mathbf{F}^{total} - \mathbf{c}\Big)\Big)_i \\
&= ((\mathbf{A} + \mathbf{S}) \cdot \lambda + \mathbf{b})_i \\
&= (\mathbf{A^s} \cdot \lambda + \mathbf{b})_i \\
&= \mathbf{A^s}_i \cdot \lambda + \mathbf{b}_i \qquad (3.74)
\end{aligned}$$

where

$$\mathbf{A^s} = \mathbf{J} \cdot \mathbf{M}^{-1} \cdot \mathbf{J}^T + \mathbf{S} = \mathbf{A} + \mathbf{S} \qquad (3.75)$$

$$\mathbf{b} = \mathbf{J} \cdot \mathbf{M}^{-1} \cdot \mathbf{F}^{total} - \mathbf{c}. \qquad (3.76)$$

LCP Conditions on Multipliers. Because the objective of the soft constraint is to keep $\mathbf{C}_{a,s}^i(\lambda)$ at $\mathbf{0}$, not $\mathbf{C}_a^i(\lambda)$, we can use Equation (3.74) in place of Equation (3.42) and follow the original derivation from Section 3.2.2.3 to obtain the LCP conditions for λ for the soft constraint. The conditions that we get this way are formulated in terms of the new matrix $\mathbf{A^s}$ but are otherwise identical to the original conditions in Equation (3.45 [LCP1]).

Equalities, Inequalities, Bounded Equalities. We define and implement soft variants of equality constraints, inequality constraints and bounded equality constraints by (1) assigning softness constants $(\mathbf{s}_i)_k$ with the rows k in the original formulations of the constraints i and (2) using the matrix $\mathbf{A^s}$ from Equation (3.75), that incorporates the effects of the softnesses, for the LCP. We use the same LCP conditions over multipliers as in the hard constraints case.

3.2.2.7 Reduction to LCP

Given the definitions of (soft) constraints i over accelerations of bodies that are currently in effect, our goal is to reformulate these constraints in the form of LCP such that the solution to the LCP produces the constraint force, parameterized by λ.

We use LCP conditions over λ to capture the effect of the acceleration constraints. Once reformulated as a LCP, we solve this LCP to obtain λ. The solution to the LCP is a vector λ that determines the constraint force $\mathbf{J}^T \cdot \lambda$ to be applied on the rigid body system. The value of λ is computed by th LCP solver such that the resulting acceleration $\mathbf{a}(\lambda)$ of the rigid body system is consistent with all acceleration constraints.

The LCP that we construct has a special structure, known as the *lo-hi linear complementarity problem*, [169]. We formulate the LCP as described below. The LCP is characterized by:

- The unknown multipliers λ that will determine the constraint force and that we need to solve for.
- Known matrix $\mathbf{A^s}$ and the vector $\mathbf{b}$ from Equation (3.75) and Equation (3.76).

  Content of these matrices and vectors is computed from the definitions of $\mathbf{A^s}$ and $\mathbf{b}$ and the specification of the constraints, using the values of $\mathbf{J}_{i,X_i}$, $\mathbf{J}_{i,Y_i}$, $\mathbf{c}_i$, $\mathbf{s}_i$ and the current external force $\mathbf{F}^{total}$. See Section 3.2.2.1 and Section 3.2.2.6 for information on the block structure of $\mathbf{A}$, $\mathbf{A^s}$ and $\mathbf{b}$ and how the blocks can be filled.

- Known LCP conditions over the values of the multipliers λ_i and the values $\mathbf{A^s}_i \cdot \lambda + \mathbf{b}_i$ of the soft constraint functions $\mathbf{C}^i_{a,s}(\lambda)$, as given by constraint specifications.

  The LCP conditions are specific to each constraint *i* and the type of the constraint. They consist of the conditions [LCP1], [LCP2] and [LCP3] and are given by the definition of each constraint. Note, $\mathbf{A^s}$ is used in place of $\mathbf{A}$ to account for the softnesses of constraints:

  1. The unbounded equality constraint uses the LCP conditions from Equation (3.45 [LCP1]),

  $$\mathbf{A^s}_i \cdot \lambda + \mathbf{b}_i = \mathbf{0}. \tag{3.77}$$

  2. The inequality constraint uses the LCP conditions from Equation (3.55 [LCP1]), Equation (3.56 [LCP2]), Equation (3.57 [LCP3]),

  $$\mathbf{A^s}_i \cdot \lambda + \mathbf{b}_i \geq \mathbf{0} \tag{3.78}$$
  $$\lambda_i \geq \mathbf{0} \tag{3.79}$$
  $$(\mathbf{A^s}_i \cdot \lambda + \mathbf{b}_i) \cdot \lambda_i = 0. \tag{3.80}$$

  3. The bounded equality constraint uses the LCP conditions from Equation (3.69 [LCP2]), Equation (3.70 [LCP3]), Equation (3.71 [LCP3]), Equation (3.72 [LCP1+ LCP3]),

  $$\left(\lambda_i^{lo}\right)_k \leq (\lambda_i)_k \leq \left(\lambda_i^{hi}\right)_k \tag{3.81}$$
  $$(\lambda_i)_k = \left(\lambda_i^{lo}\right)_k \Rightarrow \mathbf{A^s}_i \cdot \lambda + \mathbf{b}_i \geq 0 \tag{3.82}$$
  $$(\lambda_i)_k = \left(\lambda_i^{hi}\right)_k \Rightarrow \mathbf{A^s}_i \cdot \lambda + \mathbf{b}_i \leq 0 \tag{3.83}$$
  $$\left(\lambda_i^{lo}\right)_k < (\lambda_i)_k < \left(\lambda_i^{hi}\right)_k \Rightarrow \mathbf{A^s}_i \cdot \lambda + \mathbf{b}_i = 0. \tag{3.84}$$

Second Order Inverse Dynamics. We abstract the process of building the LCP and solving the LCP using a *constraint solver* component [191] within the system. The constraint solver component works as a black box that implements *inverse dynamics* within the rigid body system. The component takes as an input the external force that acts on the system and the formulations of constraints that are to be maintained. It automatically builds the LCP to capture the effects of constraints, solves the LCP using a *LCP solver* [191, 169, 192] and outputs the constraint force that arises in response to the external force and the constraints and that satisfies the constraints.

When solving for the constraint force, all constraints are handled uniformly and no special knowledge of constraints, besides the descriptions in terms of *e.g.*, Equation (3.73), is required by the constraint solver. Furthermore, the set of constraints can change dynamically and new constraints (and behaviors) can be automatically implemented by simply providing descriptions for the constraints. We exploit these features in later sections when we implement body articulation, limits at joint angles, actuation of the body and contact with the environment, when we implement all these aspects of the motion of the model of the person by constraints.

We internally use a modified formulation of the LCP in our constraint solver implementation [191] based on [18]. This formulation lets us maintain equality constraints in an articulated structure to implement articulation of the structure in linear time, thus rivaling performance of simulations that use reduced coordinates to express the motion of the structure.

3.2.3 Velocity Constraints

The constraints we have defined in Section 3.2.2 were all formulated over accelerations of bodies. The constraints were maintained by solving for the constraint force that adjusts the accelerations such that the constraints are maintained, utilizing second-order dynamics rules from Equation (3.26).

In specific cases, it is important to characterize a desired behavior of the system directly in terms of velocities of the bodies. For example, when a foot hits the ground plane, we need to change the velocity of the foot instantly so that the foot would not go down through the ground. Towards that end, we introduce constraints that are directly formulated over velocities of the bodies in the system and solve for an impulsive constraint force that adjusts the velocities of the bodies directly such that the velocity constraints will hold, utilizing first-order dynamics rules from Equation (3.28).

We define and handle velocity constraints analogously to the acceleration case, except that we replace Equation (3.26) with Equation (3.28) and use velocities and impulsive forces instead of accelerations and forces. In doing so, we utilize the notation similarity between the second-order and the first-order dynamics that we established in Section 3.1.3.

We next describe the notation that we use to express constraints over velocities of bodies in the system. We then introduce formulations of the constraints and directly show how impulsive constraint forces can be computed in terms of λ to satisfy the constraints. The impulsive constraint force will be computed analogously to the acceleration case via a LCP, defined in terms of the velocity-level variants $\mathbf{A}^{\mathbf{v},\mathbf{s}}$ and $\mathbf{b}^{\mathbf{v}}$ of $\mathbf{A}^{\mathbf{s}}$, $\mathbf{b}$ and impulsive constraint force bounds λ^{lo} and λ^{hi}. The LCP is built identically to the acceleration case, except that acceleration-level quantities, relevant to second-order dynamics, are replaced with the corresponding velocity-level quantities in the formulation. When implementing a constraint solver on the computer, an identical code can be used to implement velocity-level constraints, by using an abstraction wrapper that accesses either the velocity-level or acceleration-level quantities.

Notation. We reuse the generalized notation from Section 3.2.2.1 and bring it to the velocity level. Towards that end, we define $\mathbf{F}_{imp}^{total} = \left(\left(\mathbf{F}_{imp}^{total}\right)_1, \ldots, \left(\mathbf{F}_{imp}^{total}\right)_n\right)$ to refer to the total external impulse (momentum) applied on the rigid body system and $\mathbf{F}_{imp}^{constraint} = \left(\left(\mathbf{F}_{imp}^{constraint}\right)_1, \ldots, \left(\mathbf{F}_{imp}^{constraint}\right)_n\right)$ to refer to the total constraint impulse applied on the system due to all velocity constraints.

We will work directly with velocity-level soft constraint functions $\mathbf{C}_{v,s}^{i}(\lambda)$,

$$\mathbf{C}_{v,s}^{i}(\lambda) := \mathbf{J}_{i,X_i} \cdot \mathbf{v}_{X_i}(\lambda) + \mathbf{J}_{i,Y_i} \cdot \mathbf{v}_{Y_i}(\lambda) - \mathbf{c}_i^{imp} - \mathbf{s}_i \cdot \mathbf{E} \cdot \lambda_i, \tag{3.85}$$

where $\mathbf{J}_{i,X_i}, \mathbf{J}_{i,Y_i}$ are $m_i \times 6$ Jacobian matrices defined as before, $\mathbf{c}_i^{imp}$ is a $m_i \times 1$ vector, $\mathbf{s}_i$ is a $m_i \times 1$ vector with softness constants, λ_i is the $m_i \times 1$ vector of Lagrange multipliers that parameterizes the impulsive constraint force $\left(\mathbf{F}_{imp}^{constraint}\right)_i = \mathbf{J}_i^T \cdot \lambda$ due to the constraint, $\mathbf{v}(\lambda)$ is the velocity of the rigid body system after the entire impulsive constraint force $\mathbf{J}^T \cdot \lambda$ due to all velocity constraints is applied on the system and m_i is the dimensionality of the constraint.

Using first-order dynamics from Section 3.1.3, the velocity $\mathbf{v}(\lambda)$ is given by

$$\mathbf{v}(\lambda) = \mathbf{M}^{-1} \cdot \left(\mathbf{F}_{imp}^{total} + \mathbf{J}^T \cdot \lambda\right). \tag{3.86}$$

Consequently, following the derivation of Equation (3.74) the value of the velocity-level soft constraint function $\mathbf{C}_{v,s}^{i}(\lambda)$ equals

$$\mathbf{C}_{v,s}^{i}(\lambda) = \left(\mathbf{A}^{\mathbf{v,s}}\right)_i \cdot \lambda + \left(\mathbf{b}^{\mathbf{v}}\right)_i, \tag{3.87}$$

where

$$\mathbf{A}^{\mathbf{v,s}} = \mathbf{J} \cdot \mathbf{M}^{-1} \cdot \mathbf{J}^T + \mathbf{S} = \mathbf{A} + \mathbf{S}, \tag{3.88}$$

$$\mathbf{b}^{\mathbf{v}} = \mathbf{J} \cdot \mathbf{M}^{-1} \cdot \mathbf{F}_{imp}^{total} - \mathbf{c}^{imp} \tag{3.89}$$

and $\mathbf{c}^{imp} = \left(\mathbf{c}_1^{imp}, \ldots, \mathbf{c}_c^{imp}\right)$. As such, $\mathbf{A}^{\mathbf{v,s}}$ is identical to the acceleration case and $\mathbf{b}^{\mathbf{v}}$ differs from the acceleration case by using $\mathbf{F}_{imp}^{total}$ instead of $\mathbf{F}^{total}$ and $\mathbf{c}^{imp}$ instead of $\mathbf{c}$.

Definitions of Constraints. We define a *velocity-level constraint i* in terms of the soft constraint function $\mathbf{C}_{v,s}^{i}$ from Equation (3.85) that is described by $\mathbf{J}_{i,X_i}$, $\mathbf{J}_{i,Y_i}$, $\mathbf{c}_i^{imp}$ and $\mathbf{s}_i$. The constraint exerts a constraint impulse $\left(\mathbf{F}_{imp}^{constraint}\right)_i = \mathbf{J}_i^T \cdot \lambda_i$ parameterized by the multipliers λ_i and implements either

- an *unbounded equality constraint*, when the value of the function is requested to be exactly $\mathbf{0}$, such that

$$\mathbf{C}_{v,s}^{i}(\lambda) := \mathbf{J}_{i,X_i} \cdot \mathbf{v}_{X_i}(\lambda) + \mathbf{J}_{i,Y_i} \cdot \mathbf{v}_{Y_i}(\lambda) - \mathbf{c}_i^{imp} - \mathbf{s}_i \cdot \mathbf{E} \cdot \lambda_i = \mathbf{0}, \tag{3.90}$$

- a *inequality constraint*, when the value of the function is requested to be greater-or-equal to $\mathbf{0}$, such that

$$\mathbf{C}_{v,s}^{i}(\lambda) := \mathbf{J}_{i,X_i} \cdot \mathbf{v}_{X_i}(\lambda) + \mathbf{J}_{i,Y_i} \cdot \mathbf{v}_{Y_i}(\lambda) - \mathbf{c}_i^{imp} - \mathbf{s}_i \cdot \mathbf{E} \cdot \lambda_i \geq \mathbf{0}, \tag{3.91}$$

or

- a *bounded equality constraint*, when the value of the value of the function is requested to be kept at $\mathbf{0}$, such that
$$\mathbf{C}^i_{v,s}(\lambda) := \mathbf{J}_{i,X_i} \cdot \mathbf{v}_{X_i}(\lambda) + \mathbf{J}_{i,Y_i} \cdot \mathbf{v}_{Y_i}(\lambda) - \mathbf{c}^{imp}_i - \mathbf{s}_i \cdot \mathbf{E} \cdot \lambda_i = \mathbf{0}, \tag{3.92}$$
subject to bounds on the multiplier values, $\lambda^{lo}_i \leq \lambda_i \leq \lambda^{hi}_i$, where $\lambda^{lo}_i \leq \mathbf{0}$ and $\lambda^{hi}_i \geq \mathbf{0}$.

LCP Conditions on Multipliers. Velocity-level constraints produce LCP conditions on the multipliers λ that are specific to the type of the constraint and are identical to the cases of corresponding acceleration constraints. The conditions can be formally derived by following the original derivations for acceleration constraints while replacing

- acceleration $\mathbf{a}$ with velocity $\mathbf{v}$,
- total external force $\mathbf{F}^{total}$ with total external impulse $\mathbf{F}^{total}_{imp}$,
- constraint force $\mathbf{F}^{constraint}$ with impulsive constraint force $\mathbf{F}^{constraint}_{imp}$,
- vectors $\mathbf{c}$ with $\mathbf{c}^{imp}$ and
- soft acceleration-level constraint functions $\mathbf{C}^i_{a,s}$ with soft velocity-level constraint functions $\mathbf{C}^i_{v,s}$.

The resulting LCP conditions that are obtained this way are formulated with respect to the matrix $\mathbf{A}^{\mathbf{v},\mathbf{s}}$ defined in Equation (3.88) and the vector $\mathbf{b}^{\mathbf{v}}$ defined in Equation (3.89).

First Order Inverse Dynamics and Multiplier Bounds. Velocity-level constraints are solved using a constraint solver that implements first-order inverse dynamics. The solver takes the total external impulse $\mathbf{F}^{total}_{imp}$ and the descriptions of velocity-level constraints, formulates the LCP over λ in terms of $\mathbf{A}^{\mathbf{v},\mathbf{s}}$ and $\mathbf{b}^{\mathbf{v}}$ and the multiplier bounds λ^{lo}, λ^{hi}, as described in this section, solves the LCP and outputs the impulsive constraint force $\mathbf{F}^{constraint}_{imp} = \mathbf{J}^T \cdot \lambda$ that satisfies the velocity constraints.

Note that the multiplier bounds λ^{lo}, λ^{hi} limit the components of the impulsive constraint force, not the constraint force, in this case. As such, they have "units of momentum" and determine how much momentum of the rigid body system can change along a given impulsive force direction at one application of the impulse. These units are different from Newtons, used by the multiplier bounds in the acceleration case. If the equations of motion are integrated using an Euler method with the step size Δt (see Section 3.1.4), then the impulsive bounds λ^{lo}_v, λ^{hi}_v, can be related to the acceleration bounds λ^{lo}_a, λ^{hi}_a as

$$\lambda^{lo}_a = \frac{\lambda^{lo}_v}{\Delta t} \tag{3.93}$$
$$\lambda^{hi}_a = \frac{\lambda^{hi}_v}{\Delta t}, \tag{3.94}$$

because the effects of an impulse $\mathbf{F}_{imp}$ can be simulated by the force $\mathbf{F} = \frac{\mathbf{F}_{imp}}{\Delta t}$. We use these equations to implicitly adjust the bounds when switching between velocity-level and acceleration-level formulations of constraints.

Maintaining Velocity Constraints by Constraining Accelerations. Note that if *all* velocity-level constraints already hold, such that no impulses need be produced to maintain them, we can constrain accelerations to be consistent with the velocity formulations. By constraining the accelerations in this case, (1) we can ensure that the velocity constraints will hold in the future as well, after the accelerations have been integrated by the simulator, and (2) velocities of the bodies in the system will not have to be corrected by impulses.

Our goal is to limit the use of impulses as much as possible. First, we want to simulate Newtonian second-order dynamics that is driven by forces and not impulses, which implement only first-order dynamics. Second, each time an impulse is applied, the state of the rigid body system is changed disruptly and integration (simulation) has to be restarted from this new state (see Section 3.1.4). Thus, whenever possible, we attempt to maintain velocity constraints incrementally, by only applying forces and strictly adhering to Newtonian second-order dynamics.

The acceleration level formulations of velocity constraints are obtained by taking the time derivatives of the velocity-level constraint functions from Equation (3.85) and using them as the acceleration-level constraint functions for the corresponding types of acceleration constraints[4].

3.2.4 Position Constraints

We next introduce constraints formulated over positions and orientations of rigid bodies. Such constraints are important because they can express many practical aspects of desired motion of the rigid body system in a natural way. For example, to build an articulated model of the person, position constraints can be used to attach parts of the body together at joints in the body, limit the motion of the parts relative to each other to prevent biomechanically impossible poses of the body (*e.g.*, bending of the knee backwards), ensure non-penetration of the parts and to actuate the body.

With that said, position constraints can not be implemented by directly adjusting positions and orientations of bodies. In systems driven by physical laws, positions and orientations of bodies change only indirectly, in response to forces and impulses that are applied on them, as encoded by the equations of motion. Consequently, in order to implement position constraints, we need to reformulate such constraints as constraints over velocities and/or accelerations and maintain the original constraints incrementally, by ensuring the velocities and accelerations remain consistent with the original position formulations. In doing so, we follow and extend the approach previously outlined in Section 3.2.1.

We work with both equality constraints over body pairs, that fix a specific configuration of bodies, such as a desired angle at a joint or connection of bodies at a joint, and inequality constraints. Inequality constraints are only effective when a specific configuration of bodies is reached (*e.g.*, a joint angle is hit). We next define the constraints, outline the general course of action in maintaining the constraints and show how each constraint type can be implemented.

Definitions. We formulate position-level constraints using position-level constraint functions. We reuse the generalized notation introduced in Section 3.2.1 and Section 3.2.2.

[4]When we reformulate a velocity constraint as an acceleration constraint, we implicitly adjust the force bounds according to Equation (3.93) and Equation (3.94).

We define the *position-level constraint function* for the constraint i as a function $\mathbf{C}_p^i : [\mathbf{x}_{X_i}, \mathbf{x}_{Y_i}] \to \mathbb{R}_i^m$ that is differentiable with respect to time. The function acts over the bodies X_i and Y_i and has the dimensionality m_i. It defines a position manifold $\left\{(\mathbf{x}_{X_i}, \mathbf{x}_{Y_i}) \mid \mathbf{C}_p^i(\mathbf{x}_{X_i}, \mathbf{x}_{Y_i}) = \mathbf{0}\right\}$ for the points $(\mathbf{x}_{X_i}, \mathbf{x}_{Y_i})$.

A *position-level equality constraint i* constrains the generalized positions of the bodies X_i and Y_i to lie on the position manifold, such that

$$\mathbf{C}_p^i(\mathbf{x}_{X_i}, \mathbf{x}_{Y_i}) = \mathbf{0}. \tag{3.95}$$

A *position-level inequality constraint i* constrains the generalized positions of the bodies to lie on or in the front of the position manifold, such that

$$\mathbf{C}_p^i(\mathbf{x}_{X_i}, \mathbf{x}_{Y_i}) \geq \mathbf{0}. \tag{3.96}$$

Similarly to the case of acceleration and velocity constraint functions, the value $\mathbf{C}_p^i(\mathbf{x}_{X_i}, \mathbf{x}_{Y_i})$ of the constraint function for the constraint i can be interpreted as a measurement of the position error for the bodies at the position $\mathbf{x}_{X_i}$ and $\mathbf{x}_{Y_i}$. For simplicity, we assume that $m_i = 1$ for inequality position constraints.

Maintaining Constraints Incrementally. We maintain position constraints incrementally, by using their velocity-level and acceleration-level formulations. These formulations are obtained by taking the first-order and second-order time derivatives of the position-level constraint equations.

The position-level constraints are maintained by (1) starting from a state that honors the position-level formulations and (2) using the velocity-level formulations as real constraints for simulation. According to the discussion in Section 3.2.3, we always implement velocity constraints as constraints over accelerations when permitted (that is, when all velocity constraints are already satisfied). We describe the handling of each type of position constraint in greater detail next.

3.2.4.1 Equality Constraints with Stabilization

We use position-level equality constraints to model articulation of the model of the person and to actuate the model. We implement a particular position-level equality constraint i $\mathbf{C}_p^i(\mathbf{x}_{X_i}, \mathbf{x}_{Y_i}) = \mathbf{0}$ from Equation (3.95) by exploiting its velocity-level and acceleration-level formulations.

Velocity and Acceleration Formulations. We obtain the velocity-level formulation of the constraint from Equation (3.95) by differentiating the position-level constraint equation with respect to time. We get the velocity-level constraint equation

$$\mathbf{C}_v^i(\mathbf{v}_{X_i}, \mathbf{v}_{Y_i}) = \mathbf{0}, \tag{3.97}$$

where $\mathbf{C}_v^i(\mathbf{v}_{X_i}, \mathbf{v}_{Y_i}) = \dot{\mathbf{C}}_p^i(\mathbf{x}_{X_i}, \mathbf{x}_{Y_i}) = \mathbf{J}_{i,X_i} \cdot \mathbf{v}_{X_i} + \mathbf{J}_{i,Y_i} \cdot \mathbf{v}_{Y_i}$, which is a special case of the general velocity-level equality constraint from Equation (3.90).

We obtain the acceleration-level formulation of the constraint from Equation (3.95) by differentiating the position-level constraint equation with respect to time twice. We get the acceleration-level constraint equation

$$\mathbf{C}_a^i(\mathbf{a}_{X_i}, \mathbf{a}_{Y_i}) = \mathbf{0}, \tag{3.98}$$

where $\mathbf{C}_a^i(\mathbf{a}_{X_i}, \mathbf{a}_{Y_i}) = \dot{\mathbf{C}}_v^i(\mathbf{v}_{X_i}, \mathbf{v}_{Y_i}) = \ddot{\mathbf{C}}_p^i(\mathbf{x}_{X_i}, \mathbf{x}_{Y_i}) = \mathbf{J}_{i,X_i} \cdot \mathbf{a}_{X_i} + \mathbf{J}_{i,Y_i} \cdot \mathbf{a}_{Y_i} - \mathbf{c}_i$ and $\mathbf{c}_i = -\dot{\mathbf{J}}_{X_i,\cdot} \mathbf{v}_{X_i} - \dot{\mathbf{J}}_{Y_i,\cdot} \mathbf{v}_{Y_i}$, which is a special case of the general acceleration-level equality constraint from Equation (3.43).

Maintaining Position Constraint Incrementally. Given the velocity-level and acceleration-level formulations of the original constraint, we may attempt to maintain Equation (3.95) *incrementally* either

- on the acceleration level, by starting from a state where $\mathbf{C}_p^i = \mathbf{0}$ and $\dot{\mathbf{C}}_p^i = \mathbf{0}$ and applying constraint forces such that $\ddot{\mathbf{C}}_p^i = \mathbf{0}$, or
- on the velocity level, by starting from a state where $\mathbf{C}_p^i = \mathbf{0}$ and applying constraint impulses such that $\dot{\mathbf{C}}_p^i = \mathbf{0}$, eventually reducing to the first case.

This proposed approach can, however, only work when the velocity and acceleration formulations of the constraint are maintained exactly in simulation. In the first case, constraint forces are applied under the assumption that $\mathbf{C}_p^i = \dot{\mathbf{C}}_p^i = \mathbf{0}$ at any time, while in the second case constraint impulses are applied under the assumption of $\mathbf{C}_p^i = \mathbf{0}$. Neither of these assumptions need to hold in the practice. For example, the numerical solver that integrates the equations of motion may incur an integration error during simulation, constraint forces and impulses may be computed with an insufficient precision or we may even want to start from a state when the position constraint does not hold exactly. When these aspects occur, the incremental approach can easily fail at maintaining the original position constraint. To remedy the problem, we need to be able to reduce the violations of these assumptions as they happen. Towards that end, we stabilize constraints.

Constraint Stabilization on Velocity Level. Let us assume that we implement the position level constraint i by imposing a velocity constraint. If the constraint is currently *broken*, such that $\mathbf{C}_p^i \neq \mathbf{0}$, we want to generate a constraint impulse so that the constraint error $\mathbf{C}_p^i$ is decreased and the original position constraint is *stabilized*. A simple stabilization technique can be implemented for any position constraint by following a procedure suggested in [36]. In this procedure, one uses

$$\dot{\mathbf{C}}_p^i = -\mathbf{C}_p^i \cdot \alpha, \tag{3.99}$$

as a velocity constraint to implement the original position constraint instead of $\dot{\mathbf{C}}_p^i = \mathbf{0}$ from Equation (3.97), where $\alpha \geq 0$ is a constraint stabilization constant, dependent on the integration step size.

The constant determines what fraction of the constraint error should be reduced in one integration step. As such, the stabilization constant controls the speed with which the constraint error is fully removed (the constraint is stabilized) and it serves a role similar to gain constants in penalty methods. Note that, however, unlike the penalty methods, the constraint stabilization procedure discussed here guarantees that the constraint error is reduced as the time progresses and in presence of other constraints. Specifically, if we interpret $\mathbf{C}_p^i(t)$ as a measurement of the constraint error at the time t and if $t_{current}$ is the current time and Δt the integration step size, we can express the error decay between the time $t_{current}$ and $t_{current} + \Delta t$ using Taylor expansion at $t_{current}$ and Equation (3.99) as

$$\begin{aligned}\mathbf{C}_p^i(t_{current} + \Delta t) &\approx \mathbf{C}_p^i(t_{current}) + \Delta t \cdot \dot{\mathbf{C}}_p^i(t_{current}) \\ &= \mathbf{C}_p^i(t_{current}) - \Delta t \cdot \mathbf{C}_p^i(t_{current}) \cdot \alpha \\ &= \mathbf{C}_p^i(t_{current}) \cdot (1 - \Delta t \cdot \alpha).\end{aligned}$$

Consequently,

$$\mathbf{C}_p^i(t') \propto \mathbf{C}_p^i(t_{current}) \cdot \exp(-\alpha \cdot t') \tag{3.100}$$

for $t' \geq t_{current}$ and the *decay of the position constraint error is exponential* with the decay speed controlled by the constant α.

Constraint Stabilization on Acceleration Level. We can use the same stabilization technique to stabilize a broken velocity constraint so that the velocity constraint can be implemented by an acceleration constraint. This way, we can implement the original position constraint i by imposing an acceleration constraint. In this case, we need to reduce both the position error $\mathbf{C}_p^i$ as well as the velocity error $\dot{\mathbf{C}}_p^i$.

We can reduce these errors by using

$$\ddot{\mathbf{C}}_p^i = -\mathbf{C}_p^i \cdot \alpha - \dot{\mathbf{C}}_p^i \cdot \beta \tag{3.101}$$

as an acceleration constraint to implement the original position constraint instead of $\ddot{\mathbf{C}}_p^i = \mathbf{0}$ from Equation (3.98), where $\alpha \geq 0$ is a position constraint stabilization constant and $\beta \geq 0$ is a velocity constraint stabilization constant, both being dependent on the integration step size. We use $\beta = 2 \cdot \alpha$. Because $\dot{\mathbf{C}}_p^i = \mathbf{J}_i \cdot \mathbf{v}$, we can implement the stabilization by

$$\ddot{\mathbf{C}}_p^i = -\mathbf{C}_p^i \cdot \alpha - \mathbf{J}_i \cdot \mathbf{v} \cdot \beta. \tag{3.102}$$

Maintaining Position Constraint Incrementally with Stabilization. Using the stabilization techniques from above, we can implement the original position equality constraint i incrementally by requesting either

- the velocity-level equality constraint i

$$\mathbf{J}_{i,X_i} \cdot \mathbf{v}_{X_i}(\lambda) + \mathbf{J}_{i,Y_i} \cdot \mathbf{v}_{Y_i}(\lambda) - \mathbf{c}_i^{imp} = \mathbf{0}, \tag{3.103}$$

  where $\mathbf{c}_i^{imp} = -\mathbf{C}_p^i \cdot \alpha$, or

- the acceleration-level equality constraint i

$$\mathbf{J}_{i,X_i} \cdot \mathbf{a}_{X_i}(\lambda) + \mathbf{J}_{i,Y_i} \cdot \mathbf{a}_{Y_i}(\lambda) - \mathbf{c}_i = \mathbf{0}, \tag{3.104}$$

  where $\mathbf{c}_i = -\mathbf{C}_p^i \cdot \alpha - \mathbf{J}_{i,X_i} \cdot \mathbf{v}_{X_i} \cdot \beta - \mathbf{J}_{i,Y_i} \cdot \mathbf{v}_{Y_i} \cdot \beta - \dot{\mathbf{J}}_{X_i,\cdot} \mathbf{v}_{X_i} - \dot{\mathbf{J}}_{Y_i,\cdot} \mathbf{v}_{Y_i}$.

These constraints can be accompanied with constraint force/impulse bounds in order to turn them to bounded equality constraints such that powering limits can be accounted for. This way, the position constraint will only be maintained as long as the constraint has enough power. Furthermore, the constraints can be made soft by incorporating softness constants as introduced in Section 3.2.2.6. Using these two features, the original position constraint is encouraged to break and the stabilization mechanism works to reduce the breakage.

By using soft constraints and/or constraints with bounds that allow the position constraint to break, one can build spring-like systems that oscillate around the desired state and that do not reach the desired state exactly in one integration step. We use these techniques for the actuation of the model of the person in Section 3.6, where the current pose of the body is stabilized using soft constraints with powering bounds towards a desired pose.

3.2.4.2 Inequality Constraints with Stabilization

We use position-level inequality constraints to model joint angle limits in the model of the person and to enforce non-penetration between parts of the body and the parts and the environment. We maintain a particular one-dimensional position-level inequality constraint i, $C_p^i(\mathbf{x}_{X_i}, \mathbf{x}_{Y_i}) \geq 0$, from Equation (3.95) incrementally, by considering the velocity-level and acceleration-level formulations $\dot{C}_p^i(\mathbf{v}_{X_i}, \mathbf{v}_{Y_i}) \geq 0$ and $\ddot{C}_p^i(\mathbf{a}_{X_i}, \mathbf{a}_{Y_i}) \geq 0$.

We implement the constraint using an incremental approach with stabilization based on Section 3.2.4.1. Handling of the inequality constraint differs from the previous equality constraint case in one fundamental aspect though. First, the equality constraint works to keep positions of bodies on the constraint position manifold at all times. The approach from Section 3.2.4.1 does so by imposing the same kind of velocity or acceleration constraint at every integration step. On the other hand, the inequality constraint attempts to keep the positions of the bodies in the front of the position manifold or on the manifold, and no velocity or acceleration formulations of the constraints are considered by the dynamics until the bodies hit or already lie on the manifold. Consequently, the velocity and acceleration formulations of the constraint are only effective at specific time instants and we may need to use impulsive dynamics to implement the constraint. For example, when we use the inequality constraint to keep a ball in the free fall on or above the ground, no velocity or accelerations constraints are effective until the ball hits the ground. At the time of impact, however, the velocity of the ball has to be reversed instantly (via a constraint impulse) to avoid a violation of the position constraint.

The position inequality constraint i is satisfied by its velocity or acceleration formulations depending on the current value of the position constraint function $C_p^i(t_{current})$. We consider the following cases, with respect to a tolerance ε:

1. $|C_p^i(t_{current})| < \varepsilon$:

   When $C_p^i(t_{current}) = 0$, the positions of bodies lie on the constraint position manifold (*e.g.*, the ball just hit the ground or rests on the ground). To ensure the positions do not move past the manifold, we request the $\dot{C}_p^i(t_{current}) \geq 0$.

   We implement this request as a velocity constraint with position stabilization using the technique from Equation (3.103),

$$\mathbf{J}_{i,X_i} \cdot \mathbf{v}_{X_i}(\lambda) + \mathbf{J}_{i,Y_i} \cdot \mathbf{v}_{Y_i}(\lambda) - c_i^{imp} \geq 0, \tag{3.105}$$

   where $c_i^{imp} = -C_p^i \cdot \alpha$, α is a position stabilization constant and $\mathbf{J}_{i,X_i}$ and $\mathbf{J}_{i,Y_i}$ come from the formulation of $\dot{C}_p^i$.

   According to the discussion from Section 3.2.3, this velocity constraint can be implemented on the acceleration level when all velocity constraints already hold (*e.g.*, the ball rests on the ground, but it does not strike it with a non-zero speed). In that situation, we use the following acceleration constraint with position and velocity stabilization terms from Equation (3.104),

$$\mathbf{J}_{i,X_i} \cdot \mathbf{a}_{X_i}(\lambda) + \mathbf{J}_{i,Y_i} \cdot \mathbf{a}_{Y_i}(\lambda) - c_i \geq 0, \tag{3.106}$$

   where $c_i = -C_p^i \cdot \alpha - \mathbf{J}_{i,X_i} \cdot \mathbf{v}_{X_i} \cdot \beta - \mathbf{J}_{i,Y_i} \cdot \mathbf{v}_{Y_i} \cdot \beta - \dot{\mathbf{J}}_{X_i,} \cdot \mathbf{v}_{X_i} - \dot{\mathbf{J}}_{Y_i,} \cdot \mathbf{v}_{Y_i}$ and α is a position stabilization constant and β is a velocity stabilization constant.

As before, we can use soft velocity or acceleration constraints and constraints with multiplier bounds to implement the original position constraint.

2. $C_p^i(t_{current}) \geq \varepsilon$:

   When $C_p^i(t_{current}) > 0$, the positions of bodies are in the front of the position manifold (*e.g.*, the ball is in the air and does not touch the ground) and, consequently, no constraints will be defined for dynamics.

3. $C_p^i(t_{current}) \leq -\varepsilon$:

   When $C_p^i(t_{current}) < 0$, the current state violates the position constraint (*e.g.*, the ball is below the ground). We roll the simulation state back to the previous state in this case and use a binary search algorithm [19] to find the latest valid state at the time $t < t_{current}$, where $|C_p^i(t)| < \varepsilon$. We then resolve this new situation using (1).

In the remainder of this chapter, we illustrate how we use position constraints to implement articulation of the model of the person, angle limits at joints, contact and perform low-level actuation (motion control) of the body.

3.3 Articulated Body

In this section, we illustrate how we build an articulated physics-based model of the person from its rigid parts. We connect the parts using joints that we implement as position constraints from Section 3.2.4. The constraints restrict relative motion of the parts to prevent tearing and to ensure that only relevant rotations about specific joint axes are possible. We discuss the articulated body model in greater detail next, focusing on the formulations for the constraints that implement the joints in the body.

Articulated Body. We model the articulated body of the person as a composition of rigid parts connected by joints of specific types. Each part in the model is represented as a rigid body from Section 3.1 and is described by (1) the mass properties from Equation (3.3), (2) a rigid shape that defines the geometry of the part and (3) anchor points for the locations of joints where the part connects to other parts.

Shapes and Mass Properties. We model the shape of a part as an abstracted assembly of rigid geometric primitives such as boxes, capsules and spheres, and we use this geometric information to model the interaction of the part with other parts and the part with the environment in Section 3.4. Mass properties encode how the part responds to forces and we derive these properties automatically from the rigid shape information using the definition of the properties in Section 3.1.1, assuming a known uniform density for the part [45].

In motion estimation, we use the shape representation of parts to predict how a particular pose of the model is going to look like in an image, when projected through the camera. Note, the shape representation used for the projection (*image shape*) may be more detailed than the rigid representation used for the simulation (*simulation shape*). For example, the image shape model can attempt to capture *e.g.*, the deformation of the skin/cloth based on the pose of the body or model the variation of the thickness of limbs along the bones in the body.

Joint	Parent Joint	Joint Type	Number of DOFs	Geometry Type
Root	Ground	Root	6	Cube
Spine	Root	Saddle	2	Cube
Neck	Spine	Ball-and-socket	3	Capsule
Right Clavicle	Spine	Saddle	2	Capsule
Right Shoulder	Right Clavicle	Ball-and-socket	3	Capsule
Right Elbow	Right Shoulder	Hinge	1	Capsule
Left Clavicle	Spine	Saddle	2	Capsule
Left Shoulder	Left Clavicle	Ball-and-socket	3	Capsule
Left Elbow	Left Shoulder	Hinge	1	Capsule
Right Hip	Root	Ball-and-socket	3	Capsule
Right Knee	Right Hip	Hinge	1	Capsule
Left Hip	Root	Ball-and-socket	3	Capsule
Left Knee	Left Hip	Hinge	1	Capsule

Table 3.2: **Structure of Body Model in Chapter 4.** Our simpler model has 13 body parts associated with joints and the total number of 31 degrees of freedom.

Joint Types. We implement *ball-and-socket* joints that connect two body parts together at a common anchor point but do not restrict the relative motion of the parts otherwise, *hinge* joints that only permit rotation about a common hinge axis at the joint and *saddle* joints that allow rotation about two axes at the joint. We refer to the constraints imposed by the joints as the joint articulation constraints.

3.3.1 Our Models

We use generic body models that represent an "average" male person. In designing these models, we are inspired by current character models in computer graphics. These models use sufficient number of parts to capture the articulation of the human body in various poses faithfully and define the part geometry and anchor points such that the locations of joints in the models coincide with the locations of major joints in the human body. We use two body models, a *simpler model* with 13 rigid parts (see Figure 4.4, Figure 5.3 and Chapter 4) and a more *detailed model* with 18 parts (see Table 3.2, Table 3.3 and Chapter 5), resulting in pose spaces with $N_{dofs} = 31$ or $N_{dofs} = 43$ degrees of freedom (DOFs). For better compliance of the model with the environment, the detailed body model uses parts to represent feet and hands in the model.

We define joint types, anchor points and shape and mass properties for each part in either model manually, based on biomechanical principles, proportions of the average body, the average weight and general population statistics [45]. We organize parts in the models into kinematic tree hierarchies such that children in the hierarchies connect to their parents by joints. The root parts are "connected" to the world space origin on the ground by virtual 6-DOF global "joints" that track how the root parts translate and rotate in the world. We illustrate these hierarchies for the two models in Table 3.2 and Table 3.3.

Joint	Parent Joint	Joint Type	Number of DOFs	Geometry Type
Root	Ground	Root	6	Cube
Spine	Root	Ball-and-socket	3	Cube
Clavicle	Spine	Ball-and-socket	3	Cube
Neck	Clavicle	Ball-and-socket	3	Cube
Right Shoulder	Clavicle	Ball-and-socket	3	Capsule
Right Elbow	Right Shoulder	Hinge	1	Capsule
Right Wrist	Right Elbow	Ball-and-socket	3	Cube
Left Shoulder	Clavicle	Ball-and-socket	3	Capsule
Left Elbow	Left Shoulder	Hinge	1	Capsule
Left Wrist	Left Elbow	Ball-and-socket	3	Cube
Right Hip	Root	Saddle	2	Capsule
Right Knee	Right Hip	Hinge	1	Capsule
Right Ankle	Right Knee	Ball-and-socket	3	Cube
Right Ball	Right Ankle	Hinge	1	Cube
Left Hip	Root	Saddle	2	Capsule
Left Knee	Left Hip	Hinge	1	Capsule
Left Ankle	Left Knee	Ball-and-socket	3	Cube
Left Ball	Left Ankle	Hinge	1	Cube

Table 3.3: **Structure of Body Model in Chapter 5.** Our more detailed model has 18 body parts associated with joints and the total number of 43 degrees of freedom.

3.3.2 Articulation Constraints

In this section, we show formulations for the types of articulation constraints that we use in our body models. We define constraint equations for ball-and-socket joints, saddle joints and hinge joints. We implement articulation constraints as position-level equality constraints from Section 3.2.4.1. In defining the constraints, we use the notation from Section 3.2.4 and assume we define a constraint with the index i that connects the part X_i (first) to the parent part Y_i (second).

3.3.2.1 Ball-and-socket Joint

Definition. A ball-and-socket joint i is defined by the anchor points $\mathbf{r}^b_{X_i}$ and $\mathbf{r}^b_{Y_i}$ that are fixed in the body spaces of the parts X_i and Y_i such that they move with the parts. Together, the point locations are constrained to occupy the same position in the world space. The joint removes three degrees of freedom from the system, but does not restrict the rotation about the anchor point in any other way. The connected parts are allowed to rotate with respect to each other in three orthogonal directions at the anchor location.

Formulation. The constraint works exactly like our point-to-point example constraint in Section 3.2.1. If $\mathbf{x}_{X_i}$ and $\mathbf{x}_{Y_i}$ are the positions of the parts X_i and Y_i and $\mathbf{R}_{X_i}$ and $\mathbf{R}_{Y_i}$ are the orientations of the parts, then $\mathbf{p}_{X_i} = \mathbf{x}_{X_i} + \mathbf{r}_{X_i}$ and $\mathbf{p}_{Y_i} = \mathbf{x}_{Y_i} + \mathbf{r}_{Y_i}$ are the world space positions of the anchors at any time instant, where $\mathbf{r}_{X_i} = \mathbf{R}_{X_i} \cdot \mathbf{r}^b_{X_i}$ and $\mathbf{r}_{Y_i} = \mathbf{R}_{Y_i} \cdot \mathbf{r}^b_{Y_i}$. The positions are constrained using a position-level constraint such that

$$\mathbf{C}^i_p := \mathbf{p}_{Y_i} - \mathbf{p}_{X_i} = \mathbf{0}. \tag{3.107}$$

Implementation. Following the derivations from Section 3.2.1, the constraint produces

$$\mathbf{C}_p^i = \mathbf{p}_{Y_i} - \mathbf{p}_{X_i} \tag{3.108}$$

$$\mathbf{J}_{i,X_i} = \begin{pmatrix} -\mathbf{E} & \mathbf{r}_{X_i}^* \end{pmatrix} \tag{3.109}$$

$$\mathbf{J}_{i,Y_i} = \begin{pmatrix} \mathbf{E} & -\mathbf{r}_{Y_i}^* \end{pmatrix} \tag{3.110}$$

$$\dot{\mathbf{J}}_{X_i,\cdot}\mathbf{v}_{X_i} + \dot{\mathbf{J}}_{Y_i,\cdot}\mathbf{v}_{Y_i} = \omega_{Y_i} \times (\omega_{Y_i} \times \mathbf{r}_{Y_i}) - \omega_{X_i} \times (\omega_{X_i} \times \mathbf{r}_{X_i}) \tag{3.111}$$

$$m_i = 3. \tag{3.112}$$

We can implement the ball-and-socket joint by substituting these quantities into Equation (3.103) or Equation (3.104).

3.3.2.2 Saddle Joint

Definition. A saddle joint extends the ball-and-socket joint by removing one rotational degree of freedom from the articulated body such that the connected parts can only rotate about two remaining axes at the anchor location. The two axes are defined explicitly and are perpendicular to each other, where the first axis is attached to the first part X_i and the second axis is attached to the second part Y_i. The saddle joint i is defined by the positions of the anchor points $\mathbf{r}_{X_i}^b$ and $\mathbf{r}_{Y_i}^b$ and the directions $\mathbf{u}_{X_i}^b$ and $\mathbf{u}_{Y_i}^b$ of the two joint axes defined in the body spaces of the parts.

Formulation. Given the world space locations $\mathbf{p}_{X_i} = \mathbf{x}_{X_i} + \mathbf{R}_{X_i} \cdot \mathbf{r}_{X_i}^b$ and $\mathbf{p}_{Y_i} = \mathbf{p}_{Y_i} + \mathbf{R}_{Y_i} \cdot \mathbf{r}_{Y_i}^b$ of the anchor points and the world space directions $\mathbf{u}_{X_i} = \mathbf{R}_{X_i} \cdot \mathbf{u}_{X_i}^b$ and $\mathbf{u}_{Y_i} = \mathbf{R}_{Y_i} \cdot \mathbf{u}_{Y_i}^b$ of the joint axes, computed according to Equation (3.4), the joint requests the anchor points to occupy the same world space location and the axes to be perpendicular in world space,

$$\mathbf{C}_p^i := \begin{pmatrix} \mathbf{p}_{Y_i} - \mathbf{p}_{X_i} \\ -\mathbf{u}_{X_i} \cdot \mathbf{u}_{Y_i} \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ 0 \end{pmatrix}. \tag{3.113}$$

The constraint removes four degrees of freedom, $m_i = 4$, and is defined by four constraint rows $\left(C_p^i\right)_k = 0$, $k = 1, \ldots, 4$, of which the first three rows are due to the ball-and-socket joint from above. The fourth row is given by $\left(C_p^i\right)_4 := -\mathbf{u}_{X_i} \cdot \mathbf{u}_{Y_i} = 0$. The row produces

$$\left(\dot{C}_p^i\right)_4 := (\mathbf{u}_{X_i} \times \mathbf{u}_{Y_i}) \cdot (\omega_{Y_i} - \omega_{X_i}) = 0 \tag{3.114}$$

$$\left(\ddot{C}_p^i\right)_4 := (\mathbf{u}_{X_i} \times \mathbf{u}_{Y_i}) \cdot (\alpha_{Y_i} - \alpha_{X_i}) + ((\omega_{X_i} \times \mathbf{u}_{X_i}) \times \mathbf{u}_{Y_i} + \mathbf{u}_{X_i} \times (\omega_{Y_i} \times \mathbf{u}_{Y_i})) \cdot (\omega_{Y_i} - \omega_{X_i}) = 0. \tag{3.115}$$

Implementation. Utilizing the quantities from above, we can implement the saddle joint using Equation (3.103) or Equation (3.104) with the following values for the terms in the formulations

$$\mathbf{C}_p^i = \begin{pmatrix} \mathbf{p}_{Y_i} - \mathbf{p}_{X_i} \\ -\mathbf{u}_{X_i} \cdot \mathbf{u}_{Y_i} \end{pmatrix} \tag{3.116}$$

$$\mathbf{J}_{i,X_i} = \begin{pmatrix} -\mathbf{E} & \mathbf{r}_{X_i}^* \\ \mathbf{0}^T & -(\mathbf{u}_{X_i} \times \mathbf{u}_{Y_i})^T \end{pmatrix} \tag{3.117}$$

$$\mathbf{J}_{i,Y_i} = \begin{pmatrix} \mathbf{E} & -\mathbf{r}_{Y_i}^* \\ \mathbf{0}^T & (\mathbf{u}_{X_i} \times \mathbf{u}_{Y_i})^T \end{pmatrix} \tag{3.118}$$

$$\dot{\mathbf{J}}_{X_i,\cdot}\mathbf{v}_{X_i} + \dot{\mathbf{J}}_{Y_i,\cdot}\mathbf{v}_{Y_i} = \begin{pmatrix} \omega_{Y_i} \times (\omega_{Y_i} \times \mathbf{r}_{Y_i}) - \omega_{X_i} \times (\omega_{X_i} \times \mathbf{r}_{X_i}) \\ ((\omega_{X_i} \times \mathbf{u}_{X_i}) \times \mathbf{u}_{Y_i} + \mathbf{u}_{X_i} \times (\omega_{Y_i} \times \mathbf{u}_{Y_i})) \cdot (\omega_{Y_i} - \omega_{X_i}) \end{pmatrix} \tag{3.119}$$

$$m_i = 4. \tag{3.120}$$

From Equation (3.103), we see that the saddle joint works exactly like the ball-and-socket joint but additionally prohibits the rotation of the parts about the axis $\mathbf{u}_{X_i} \times \mathbf{u}_{Y_i}$. The value of $\left(\dot{C}_p^i\right)_4$ measures the relative angular velocity (rotation speed) about the axis $\mathbf{u}_{X_i} \times \mathbf{u}_{Y_i}$ and the corresponding velocity constraint row forces this value to zero.

3.3.2.3 Hinge Joint

Definition. A hinge joint extends the ball-and-socket joint by removing two rotational degrees of freedom from the articulated body so that the connected parts can only rotate about a single common axis (hinge axis) at the anchor. The hinge joint i is defined by the anchor positions $\mathbf{r}_{X_i}^b$ and $\mathbf{r}_{Y_i}^b$ in the body spaces of the parts and the directions $\mathbf{u}_{X_i}^b$ and $\mathbf{u}_{Y_i}^b$ of the hinge axis expressed in the body spaces of the parts. The joint works to align the anchor points and the axes in world space and permits only rotations of the parts about the hinge axis. In aligning the anchor points, it works exactly like the ball-and-socket joint.

Formulation. Because velocity constraints allow to naturally express requests on the axes that the parts can and cannot rotate about, we define the constraint equation for the joint directly on the velocity level. The constraint removes five degrees of freedom, $m_i = 5$, and is defined by five velocity constraint rows $\left(C_v^i\right)_k = 0$ with stabilization, $k = 1, \ldots, 5$, where the first three rows are due to the ball-and-socket joint from above. The remaining two rows $\left(C_v^i\right)_4 = 0$ and $\left(C_v^i\right)_5 = 0$ implement the constraint due to the hinge axis and the rows are defined as follows,

$$\left(C_v^i\right)_4 := \mathbf{g} \cdot (\omega_{Y_i} - \omega_{X_i}) - \mathbf{g} \cdot (\mathbf{u}_{Y_i} \times \mathbf{u}_{X_i}) \cdot \alpha = 0 \tag{3.121}$$

$$\left(C_v^i\right)_5 := \mathbf{h} \cdot (\omega_{Y_i} - \omega_{X_i}) - \mathbf{h} \cdot (\mathbf{u}_{Y_i} \times \mathbf{u}_{X_i}) \cdot \alpha = 0, \tag{3.122}$$

where $\mathbf{u}_{X_i} = \mathbf{R}_{X_i} \cdot \mathbf{u}_{X_i}^b$ and $\mathbf{u}_{Y_i} = \mathbf{R}_{Y_i} \cdot \mathbf{u}_{Y_i}^b$ are the world space directions of the hinge axis, as attached to the first and the second part, $\mathbf{g}$ and $\mathbf{h}$ are any two orthogonal vectors such that $\mathbf{g} \times \mathbf{h} = \mathbf{u}_{X_i}$ and α is a stabilization constant from Equation (3.103).

The last two rows can be formulated as constraints over accelerations, by taking the time derivative of the velocity-level formulation and adding the stabilization terms according to Equation (3.104),

$$\left(\mathbf{C}_a^i\right)_4 := \mathbf{g}\cdot(\alpha_{Y_i}-\alpha_{X_i})+(\omega_{X_i}\times\mathbf{g})\cdot(\omega_{Y_i}-\omega_{X_i})-\mathbf{g}\cdot(\mathbf{u}_{Y_i}\times\mathbf{u}_{X_i})\cdot\alpha-\mathbf{g}\cdot(\omega_{Y_i}-\omega_{X_i})\cdot\beta=0 \tag{3.123}$$

$$\left(\mathbf{C}_a^i\right)_5 := \mathbf{h}\cdot(\alpha_{Y_i}-\alpha_{X_i})+(\omega_{X_i}\times\mathbf{h})\cdot(\omega_{Y_i}-\omega_{X_i})-\mathbf{h}\cdot(\mathbf{u}_{Y_i}\times\mathbf{u}_{X_i})\cdot\alpha-\mathbf{h}\cdot(\omega_{Y_i}-\omega_{X_i})\cdot\beta=0, \tag{3.124}$$

because we can assume **g** and **h** are attached to the first part.

Implementation. Utilizing the quantities from above, we can implement the hinge joint using Equation (3.103) or Equation (3.104) with the following values for the terms in the formulations

$$\mathbf{C}_p^i = \begin{pmatrix} \mathbf{p}_{Y_i}-\mathbf{p}_{X_i} \\ \mathbf{g}\cdot(\mathbf{u}_{X_i}\times\mathbf{u}_{Y_i}) \\ \mathbf{h}\cdot(\mathbf{u}_{X_i}\times\mathbf{u}_{Y_i}) \end{pmatrix} \tag{3.125}$$

$$\mathbf{J}_{i,X_i} = \begin{pmatrix} -\mathbf{E} & \mathbf{r}_{X_i}^* \\ \mathbf{0}^T & -\mathbf{g}^T \\ \mathbf{0}^T & -\mathbf{h}^T \end{pmatrix} \tag{3.126}$$

$$\mathbf{J}_{i,Y_i} = \begin{pmatrix} \mathbf{E} & -\mathbf{r}_{Y_i}^* \\ \mathbf{0}^T & \mathbf{g}^T \\ \mathbf{0}^T & \mathbf{h}^T \end{pmatrix} \tag{3.127}$$

$$\dot{\mathbf{J}}_{X_i,\cdot}\mathbf{v}_{X_i}+\dot{\mathbf{J}}_{Y_i,\cdot}\mathbf{v}_{Y_i} = \begin{pmatrix} \omega_{Y_i}\times(\omega_{Y_i}\times\mathbf{r}_{Y_i})-\omega_{X_i}\times(\omega_{X_i}\times\mathbf{r}_{X_i}) \\ (\omega_{X_i}\times\mathbf{g})\cdot(\omega_{Y_i}-\omega_{X_i}) \\ (\omega_{X_i}\times\mathbf{h})\cdot(\omega_{Y_i}-\omega_{X_i}) \end{pmatrix} \tag{3.128}$$

$$m_i = 5. \tag{3.129}$$

Interpretation. The last two rows $\left(C_v^i\right)_4=0$ and $\left(C_v^i\right)_5=0$ in the velocity formulation of the constraint constrain the relative rotation of the body parts about the axes **g** and **h** that are perpendicular to $\mathbf{u}_{X_i}$ so that (1) only rotations about $\mathbf{u}_{X_i}$ are allowed (relative rotation speed about any axis perpendicular to $\mathbf{u}_{X_i}$ is constrained) and to (2) stabilize the orientation of the parts such that $\mathbf{u}_{X_i}=\mathbf{u}_{Y_i}$ in the future.

Specifically, if the parts are not aligned properly, $\mathbf{u}_{X_i}\neq\mathbf{u}_{Y_i}$, the parts need to rotate with respect to each other at the anchor in order to recover the alignment. This recovery rotation can be characterized by the vector $\mathbf{u}_{Y_i}\times\mathbf{u}_{X_i}$, where the direction of the vector determines the axis for the recovery rotation and the magnitude of the vector is proportional to the current alignment error and the rotation angle in the recovery rotation. We enforce the recovery rotation using the stabilization technique from Section 3.2.4.1, by requesting rotation speeds along the axes **g** and **h**.

3.4 Contact

In order to model the motion of the articulated body of the person in an environment, we need to (1) represent the environment and (2) account for the interactions of the body model with this environment. We are interested in two kinds of such interactions: preventing penetration of body parts with the geometry of the environment and accounting for friction.

Because the motion of the model is produced by physical simulation in our approach, we model these interactions through forces. We produce the forces from constraints that are set up based on the geometry of the parts and the environment. In modeling these interactions, we make use of non-penetration and friction constraints defined over pairs of contacting rigid bodies in the simulated world. The bodies in constraints may represent parts of the model of the person, objects in the static environment (*e.g.*, a ground plane) or any other dynamic objects in the world (eg, a falling brick or parts of a trampoline), allowing to model interactions between all of them. In particular, we use the same constraints to model non-penetration and friction between the parts of the body and the environment as well as the parts and other parts in the body model.

Environment. We model the (static) environment $\mathscr{G}$ as a set of rigid bodies that have infinite masses and remain fixed in the world. Each body in the environment has assigned a rigid shape that defines that part of the environment. For example, we use a single fixed rigid body to model the ground plane and a set of additional fixed bodies to model stairs. The shapes are defined as compositions of simple geometric primitives, including boxes, capsules, spheres and polytopes.

Body Parts. Parts of the model of the person are represented as rigid bodies with (finite) mass properties derived from the rigid shapes of the parts. For the purposes of simulation, we use the same geometry primitives as above. See Section 3.3 for more information on how we set up the geometry of the model.

Interactions. Motion of the model results from forces applied on the body. Consequently, all effects of the environment on the motion of the body have to be accounted for through forces. The forces have to be computed exactly, so that a desired effect is achieved. For example, when a foot hits the ground, an appropriate force has to be applied on the foot to prevent it from going down through the ground or the model falls through the ground. We account for two aspects of motion, non-penetration and contact with friction.

The aspect of *non-penetration* is to ensure that the shapes (geometries) of rigid bodies in the system do not penetrate when their positions are updated using equations of motion while, at the same time, the bodies properly respond to forces and impulses applied on them. We model non-penetration by introducing non-penetration constraints that we formulate as position-level inequality constraints. We implement non-penetration incrementally just like any other inequality constraints. That is, assuming that the shapes do not penetrate at the initial state, we apply non-penetration constraints at individual time steps to ensure that penetration will not occur in the future. The constraints are defined between the parts of the body and rigid bodies in the environment as well as parts and other parts. The other aspect we account for is that of *contact*, where the goal is to enforce non-penetration while realistically responding to body collisions and accounting for friction. We model contact with friction by combining non-penetration constraints with friction constraints.

In order to model contact of rigid bodies, we need (1) a collision detection library to detect contacts between the shapes of the bodies and (2) contact constraints to model the response to the contact. We discuss these concepts next. We omit derivations for the concepts and constraints and refer interested readers to related work [191, 194, 192, 19].

3.4.1 Collision Detection

Given a simulation state at a particular time, we use a collision detection library to find which pairs of rigid bodies are in contact, such that their shapes touch. In accordance with Section 3.2.4.2, if the collision detection finds that shapes already penetrate, the simulation state is rolled back to a previous state and simulation is restarted with a smaller simulation step size so that the penetration is avoided. In the practice, we use a sufficiently small step size for simulation so that penetrations do not occur.

Contact. We informally define *contacts* as relevant pairs of close points on two body shapes where contact forces or impulses should act in order to prevent penetration. For example, a ball resting on the ground may produce one contact point at the bottom of the ball while a brick may produce four contacts at the corners of the bottom face. The problem of finding contacts for shapes is a geometric problem that can be solved efficiently using specialized libraries. We assume contacts are described by tuples

$$(X_i, Y_i, \mathbf{p}_{X_i}, \mathbf{p}_{Y_i}, \mathbf{n}_i, \dot{\mathbf{n}}_i) \tag{3.130}$$

such that the contact i involves the rigid bodies X_i and Y_i that contact at the world space point $\mathbf{p}_i = (\mathbf{p}_{X_i} + \mathbf{p}_{Y_i})/2$, where $\mathbf{p}_{X_i}$ and $\mathbf{p}_{Y_i}$ are the corresponding close points attached to X_i and Y_i, the contact surface normal at $\mathbf{p}_i$ is given by a unit vector $\mathbf{n}_i$ pointing away from X_i toward Y_i and $\dot{\mathbf{n}}_i$ is the time derivative of $\mathbf{n}_i$. Note, when the normal is attached to X_i, $\dot{\mathbf{n}}_i = \omega_{X_i} \times \mathbf{n}_i$. We will use $\mathbf{r}_{X_i} = \mathbf{p}_{X_i} - \mathbf{x}_{X_i}$ and $\mathbf{r}_{Y_i} = \mathbf{p}_{Y_i} - \mathbf{x}_{Y_i}$.

Response to the contact of bodies is modeled solely based on the contact information reported by the collision detection library. This information is specific to the simulation state. We use the contact information to formulate non-penetration and friction constraints to constrain relative body motion at the contacting points, accounting for contact effects only at the contact points. We next look into modeling non-penetration.

3.4.2 Non-penetration

Formulation. We model non-penetration at the contact i by requesting

$$\mathbf{C}_n^i := \mathbf{n}_i \cdot (\mathbf{p}_{Y_i} - \mathbf{p}_{X_i}) \geq 0, \tag{3.131}$$

which is a one-dimensional position level inequality constraint. We use $\mathbf{n}_i$ to refer to the value of the constraint function due to non-penetration. The function measures separation distance of the body Y_i from the first body X_i along the contact normal $\mathbf{n}_i$ and the constraint requests the separation distance to be non-negative at the contact, such that the shapes do not penetrate locally at the contact point.

By taking the time derivatives of the constraint function and assuming that $\mathbf{p}_{X_i} = \mathbf{p}_{Y_i}$, we get the velocity-level and acceleration-level formulations

$$\dot{\mathbf{C}}_n^i = \mathbf{n}_i \cdot \mathbf{v}_{Y_i} + (\mathbf{r}_{Y_i} \times \mathbf{n}_i) \cdot \omega_{Y_i} - \mathbf{n}_i \cdot \mathbf{v}_{X_i} - (\mathbf{r}_{X_i} \times \mathbf{n}_i) \cdot \omega_{X_i} \geq 0 \tag{3.132}$$

$$\begin{aligned}\ddot{\mathbf{C}}_n^i = {} & 2 \cdot \dot{\mathbf{n}}_i \cdot (\mathbf{v}_{Y_i} + \omega_{Y_i} \times \mathbf{r}_{Y_i} - \mathbf{v}_{X_i} - \omega_{X_i} \times \mathbf{r}_{X_i}) + (\mathbf{n}_i \cdot \mathbf{a}_{Y_i} + (\mathbf{r}_{Y_i} \times \mathbf{n}_i) \cdot \alpha_{Y_i} - \mathbf{n}_i \cdot \mathbf{a}_{X_i} - \\ & (\mathbf{r}_{X_i} \times \mathbf{n}_i) \cdot \mathbf{a}_{X_i} + \mathbf{n}_i \cdot (\omega_{Y_i} \times (\omega_{Y_i} \times \mathbf{r}_{Y_i})) - \mathbf{n}_i \cdot (\omega_{X_i} \times (\omega_{X_i} \times \mathbf{r}_{X_i}))) \geq 0\end{aligned} \tag{3.133}$$

Implementation. Because the contact point is reported only when the separation distance at the contact is within the tolerance ε, we can implement non-penetration at the contact i using Equation (3.105) or Equation (3.106) using the rules from Section 3.2.4.2 with the following values for the terms in the formulations

$$\mathbf{C}_p^i = \mathbf{n}_i \cdot (\mathbf{p}_{Y_i} - \mathbf{p}_{X_i}) \tag{3.134}$$

$$\mathbf{J}_{i,X_i} = \begin{pmatrix} -\mathbf{n}_i^T & -(\mathbf{r}_{X_i} \times \mathbf{n}_i)^T \end{pmatrix} \tag{3.135}$$

$$\mathbf{J}_{i,Y_i} = \begin{pmatrix} \mathbf{n}_i^T & (\mathbf{r}_{Y_i} \times \mathbf{n}_i)^T \end{pmatrix} \tag{3.136}$$

$$\dot{\mathbf{J}}_{X_i,\cdot}\mathbf{v}_{X_i} + \dot{\mathbf{J}}_{Y_i,\cdot}\mathbf{v}_{Y_i} = 2 \cdot \dot{\mathbf{n}}_i \cdot (\mathbf{v}_{Y_i} + \omega_{Y_i} \times \mathbf{r}_{Y_i} - \mathbf{v}_{X_i} - \omega_{X_i} \times \mathbf{r}_{X_i}) + \tag{3.137}$$

$$\mathbf{n}_i \cdot (\omega_{Y_i} \times (\omega_{Y_i} \times \mathbf{r}_{Y_i}) - \omega_{X_i} \times (\omega_{X_i} \times \mathbf{r}_{X_i})) \tag{3.138}$$

$$m_i = 1. \tag{3.139}$$

Interpretation. The non-penetration constraint has the following direct interpretation. The value $\mathbf{C}_n^i$ measures *relative body separation* at the contact i along the contact normal, $\dot{\mathbf{C}}_n^i$ measures *normal velocity* at the contact and $\ddot{\mathbf{C}}_n^i$ measures *normal acceleration*.

If λ_i^n is the Lagrange multiplier due to the acceleration-level formulation of the constraint Equation (3.106), then λ_i^n equals the *normal force magnitude* at the contact and the force $-\mathbf{n}_i \cdot \lambda_i^n$ constraint force is applied on X_i at the point $\mathbf{p}_{X_i}$ and the force $\mathbf{n}_i \cdot \lambda_i^n$ is applied on Y_i at $\mathbf{p}_{Y_i}$. The LCP conditions from Equation (3.55 [LCP1]), Equation (3.56 [LCP2]) and Equation (3.57 [LCP3]) due to the non-penetration constraint thus specify that both the normal acceleration and the normal force magnitude have to be non-negative and complementary to each other.

If λ_i^n is the Lagrange multiplier due to the velocity-level formulation of the constraint Equation (3.105), then $-\dot{\mathbf{C}}_n^i(\lambda)$ measures the relative speed with which the bodies approach under the impulsive constraint force due to λ. The value of λ_i^n computed by the constraint solver equals the magnitude of the impulse along the normal of the contact that has to be applied to stop the bodies from approaching at the point $\mathbf{p}_i$ such that $\dot{\mathbf{C}}_n^i(\lambda) \geq 0$. We utilize this fact to model *impacts*.

3.4.3 Impacts

We model impacts of bodies by modifying the velocity-level formulation of the non-penetration constraint in Equation (3.132). Instead of requiring the bodies to stop approaching at the contact, we require the speed $-\dot{\mathbf{C}}_n^i(\lambda)$ of approach to be reversed such that the bodies will bounce off. We model this effect by using

$$\dot{\mathbf{C}}_{n^*}^i = \mathbf{n}_i \cdot \mathbf{v}_{Y_i} + (\mathbf{r}_{Y_i} \times \mathbf{n}_i) \cdot \omega_{Y_i} - \mathbf{n}_i \cdot \mathbf{v}_{X_i} - (\mathbf{r}_{X_i} \times \mathbf{n}_i) \cdot \omega_{X_i} \geq -r \cdot \dot{\mathbf{C}}_n^i(\mathbf{0}) \tag{3.140}$$

in place of Equation (3.132), where $-\dot{\mathbf{C}}_n^i(\mathbf{0})$ is the original speed of approach before constraint impulses are applied and r is a *coefficient restitution*, $0 \leq r \leq 1$. In our simulations, we use $r = 0$ most of the time.

This change results in the modification of the term c_i^{imp} from Equation (3.105) that implements the non-penetration constraint such that

$$c_i^{imp} = -\mathbf{C}_n^i \cdot \alpha + r \cdot \dot{\mathbf{C}}_n^i(\mathbf{0}). \tag{3.141}$$

3.4.4 Friction

Another important phenomenon that occurs at contacts is friction. Friction acts along directions tangential to the contact normals and works to resists sliding of bodies along those directions. We account for these effects by introducing friction constraints.

Friction is usually modeled according to Coulomb friction law, [180]. The Coulomb law defines friction forces that act along the tangential directions at contacts and extends the linear complementarity conditions on the normal acceleration and normal force magnitude from Equation (3.55 [LCP1]), Equation (3.56 [LCP2]) and Equation (3.57 [LCP3]) due to the non-penetration constraint by adding conditions on the friction force. The conditions on the friction force relate the force to the relative sliding velocity at the contact and the normal force magnitude. Because this relation is quadratic, it is usually linearized for efficient implementation, by considering two friction directions that the friction force can act along at each contact and implementing friction in each direction separately [180]. As such the friction cone in the Coulomb law is replaced with a friction pyramid. We follow this approximation in our work.

Definition. We approximate friction at the contact i by two additional *friction bounded-equality constraints* that constrain relative body motion in two tangential (friction) directions $\mathbf{t}_i^x$ and $\mathbf{t}_i^y$ that are attached to X_i and perpendicular to each other and to the contact normal $\mathbf{n}_i$, such that $\mathbf{t}_i^x \cdot \mathbf{t}_i^y = 0$ and $\mathbf{t}_i^x \times \mathbf{t}_i^y = \mathbf{n}_i$.

Friction constraints work as hard as possible to prevent relative body motion along the friction directions but are allowed to fail if stopping the bodies requires too much force. We can express this goal as position-level constraints

$$\mathbf{C}_x^i := \mathbf{t}^x \cdot (\mathbf{p}_{Y_i} - \mathbf{p}_{X_i}) = 0 \tag{3.142}$$

$$\mathbf{C}_y^i := \mathbf{t}^y \cdot (\mathbf{p}_{Y_i} - \mathbf{p}_{X_i}) = 0 \tag{3.143}$$

$$\tag{3.144}$$

that work subject to constraint force bounds specified below. Note that each of the two friction constraints is associated with one friction direction and works independently of the other constraint in constraining the motion of the contacting bodies. The friction constraints are implemented using bounded equality constraints, defined either on the velocity level, as $\dot{\mathbf{C}}_x^i = 0, \dot{\mathbf{C}}_y^i = 0$, or acceleration level, as $\ddot{\mathbf{C}}_x^i = 0, \ddot{\mathbf{C}}_y^i = 0$, with (impulsive) constraint force bounds dependent on the amount of the (impulsive) normal force applied at the contact. As such, the bounds are not constant and the constraints are "special".

To approximate coupling between the forces due to non-penetration and due to friction, as required by the Coulomb law, we use the velocity-level formulation of the friction constraints when the corresponding non-penetration constraint is implemented on the velocity level. Otherwise, we use the acceleration-level formulation. Constraint formulations with the friction bounds are as follows:

- If the non-penetration constraint from Equation (3.131) is implemented on the velocity level, we implement friction constraints for the contact i on the velocity level, by requesting

$$\dot{\mathbf{C}}_x^i = \mathbf{t}_i^x \cdot \mathbf{v}_{Y_i} + (\mathbf{r}_{Y_i} \times \mathbf{t}_i^x) \cdot \omega_{Y_i} - \mathbf{t}_i^x \cdot \mathbf{v}_{X_i} - (\mathbf{r}_{X_i} \times \mathbf{t}_i^x) \cdot \omega_{X_i} = 0 \tag{3.145}$$

$$\dot{\mathbf{C}}_y^i = \mathbf{t}_i^y \cdot \mathbf{v}_{Y_i} + (\mathbf{r}_{Y_i} \times \mathbf{t}_i^y) \cdot \omega_{Y_i} - \mathbf{t}_i^y \cdot \mathbf{v}_{X_i} - (\mathbf{r}_{X_i} \times \mathbf{t}_i^y) \cdot \omega_{X_i} = 0 \tag{3.146}$$

subject to the bounds $(\lambda_i^x)^{lo} = (\lambda_i^y)^{lo} = -\mu_i \cdot \lambda_i^n$ and $(\lambda_i^x)^{hi} = (\lambda_i^y)^{hi} = \mu_i \cdot \lambda_i^n$, where λ_i^x scales the impulsive friction force along $\mathbf{t}_i^x$ and λ_i^y along $\mathbf{t}_i^y$, $\mu_i \geq 0$ is an *impulsive friction coefficient* and λ_i^n is the normal impulse magnitude at the contact, [84].

- If the non-penetration constraint from Equation (3.131) is implemented on the acceleration level, we implement friction constraints for the contact i on the acceleration level, by requesting

$$\begin{aligned}\ddot{\mathbf{C}}_x^i &= 2 \cdot \dot{\mathbf{n}}_i \cdot (\mathbf{v}_{Y_i} + \omega_{Y_i} \times \mathbf{r}_{Y_i} - \mathbf{v}_{X_i} - \omega_{X_i} \times \mathbf{r}_{X_i}) + (\mathbf{t}_i^x \cdot \mathbf{a}_{Y_i} + (\mathbf{r}_{Y_i} \times \mathbf{t}_i^x) \cdot \alpha_{Y_i} - \mathbf{t}_i^x \cdot \mathbf{a}_{X_i} - \\ &\quad (\mathbf{r}_{X_i} \times \mathbf{t}_i^x) \cdot \mathbf{a}_{X_i} + \mathbf{t}_i^x \cdot (\omega_{Y_i} \times (\omega_{Y_i} \times \mathbf{r}_{Y_i})) - \mathbf{t}_i^x \cdot (\omega_{X_i} \times (\omega_{X_i} \times \mathbf{r}_{X_i}))) = 0 \qquad (3.147) \\ \ddot{\mathbf{C}}_y^i &= 2 \cdot \dot{\mathbf{n}}_i \cdot (\mathbf{v}_{Y_i} + \omega_{Y_i} \times \mathbf{r}_{Y_i} - \mathbf{v}_{X_i} - \omega_{X_i} \times \mathbf{r}_{X_i}) + (\mathbf{t}_i^y \cdot \mathbf{a}_{Y_i} + (\mathbf{r}_{Y_i} \times \mathbf{t}_i^y) \cdot \alpha_{Y_i} - \mathbf{t}_i^y \cdot \mathbf{a}_{X_i} - \\ &\quad (\mathbf{r}_{X_i} \times \mathbf{t}_i^y) \cdot \mathbf{a}_{X_i} + \mathbf{t}_i^y \cdot (\omega_{Y_i} \times (\omega_{Y_i} \times \mathbf{r}_{Y_i})) - \mathbf{t}_i^y \cdot (\omega_{X_i} \times (\omega_{X_i} \times \mathbf{r}_{X_i}))) = 0 \qquad (3.148)\end{aligned}$$

subject to the bounds $(\lambda_i^x)^{lo} = (\lambda_i^y)^{lo} = -\mu_i \cdot \lambda_i^n$ and $(\lambda_i^x)^{hi} = (\lambda_i^y)^{hi} = \mu_i \cdot \lambda_i^n$, where λ_i^x scales the friction force along $\mathbf{t}_i^x$ and λ_i^y along $\mathbf{t}_i^y$, $\mu_i \geq 0$ is a *static friction coefficient* and λ_i^n is the normal force magnitude at the contact.

Implementation. Utilizing the fact that all constraints can be formulated as bounded equality constraints with appropriate force bounds, the procedure from above can be implemented using a bounded equality constraint that uses Equation (3.105) or Equation (3.106) with the following values for the terms and the bounds for the constraint forces in the formulation,

$$\mathbf{C}_p^i = \begin{pmatrix} \mathbf{n}_i \cdot (\mathbf{p}_{Y_i} - \mathbf{p}_{X_i}) - \frac{r}{\alpha} \cdot \dot{\mathbf{C}}_n^i(\mathbf{0}) \\ 0 \\ 0 \end{pmatrix} \tag{3.149}$$

$$\mathbf{J}_{i,X_i} = \begin{pmatrix} -\mathbf{n}_i^T & -(\mathbf{r}_{X_i} \times \mathbf{n}_i)^T \\ -(\mathbf{t}_i^x)^T & -(\mathbf{r}_{X_i} \times \mathbf{t}_i^x)^T \\ -(\mathbf{t}_i^y)^T & -(\mathbf{r}_{X_i} \times \mathbf{t}_i^y)^T \end{pmatrix} \tag{3.150}$$

$$\mathbf{J}_{i,Y_i} = \begin{pmatrix} \mathbf{n}_i^T & (\mathbf{r}_{Y_i} \times \mathbf{n}_i)^T \\ (\mathbf{t}_i^x)^T & (\mathbf{r}_{Y_i} \times \mathbf{t}_i^x)^T \\ (\mathbf{t}_i^y)^T & (\mathbf{r}_{Y_i} \times \mathbf{t}_i^y)^T \end{pmatrix} \tag{3.151}$$

$$\dot{\mathbf{J}}_{X_i,\cdot}\mathbf{v}_{X_i} + \dot{\mathbf{J}}_{Y_i,\cdot}\mathbf{v}_{Y_i} = \begin{pmatrix} 2 \cdot \dot{\mathbf{n}}_i \cdot (\mathbf{v}_{Y_i} + \omega_{Y_i} \times \mathbf{r}_{Y_i} - \mathbf{v}_{X_i} - \omega_{X_i} \times \mathbf{r}_{X_i}) + \\ \mathbf{n}_i \cdot (\omega_{Y_i} \times (\omega_{Y_i} \times \mathbf{r}_{Y_i}) - \omega_{X_i} \times (\omega_{X_i} \times \mathbf{r}_{X_i})) \\ 2 \cdot (\omega_{Y_i} \times \mathbf{t}_i^x) \cdot (\mathbf{v}_{Y_i} + \omega_{Y_i} \times \mathbf{r}_{Y_i} - \mathbf{v}_{X_i} - \omega_{X_i} \times \mathbf{r}_{X_i}) + \\ \mathbf{t}_i^x \cdot (\omega_{Y_i} \times (\omega_{Y_i} \times \mathbf{r}_{Y_i}) - \omega_{X_i} \times (\omega_{X_i} \times \mathbf{r}_{X_i})) \\ 2 \cdot (\omega_{Y_i} \times \mathbf{t}_i^y) \cdot (\mathbf{v}_{Y_i} + \omega_{Y_i} \times \mathbf{r}_{Y_i} - \mathbf{v}_{X_i} - \omega_{X_i} \times \mathbf{r}_{X_i}) + \\ \mathbf{t}_i^y \cdot (\omega_{Y_i} \times (\omega_{Y_i} \times \mathbf{r}_{Y_i}) - \omega_{X_i} \times (\omega_{X_i} \times \mathbf{r}_{X_i})) \end{pmatrix} \tag{3.152}$$

$$\lambda_i^{lo} = \begin{pmatrix} 0 \\ -\mu_i \cdot \lambda_i^n \\ -\mu_i \cdot \lambda_i^n \end{pmatrix} \tag{3.153}$$

$$\lambda_i^{hi} = \begin{pmatrix} +\infty \\ \mu_i \cdot \lambda_i^n \\ \mu_i \cdot \lambda_i^n \end{pmatrix} \tag{3.154}$$

$$m_i = 3, \tag{3.155}$$

where $\lambda_i = (\lambda_i^n, \lambda_i^x, \lambda_i^y)$ are the Lagrange multipliers for the constraint force due to the constraint. This formulation accounts for non-penetration, impacts and friction at the contact i at the same time.

Handling Friction Force Bounds. Note that for the friction constraints rows, constraint force bounds are not constant and depend on the current value of λ_i^n. Consequently, to account for the friction constraints, we need to modify the solution procedure from Section 3.2.2.6 that computes λ. We employ an iterative approach, where we alternate between fixing and re-estimating the bounds for the LCP. We solve the LCP with fixed bounds in a loop, using the values of λ_i^n from the current iteration to update and fix bounds for the values of λ_i^x and λ_i^y in the next iteration. At the first iteration, we set the friction bounds to zero.

3.5 Joint Angle Limits

In order to make sure that physical simulation does not generate poses of the model of the person that are impossible for the real human body, we define constraints over the kinematic pose. For convenience, and to follow common practice in computer graphics and vision, we define these constraints as limits on the valid values of joint angles at the joints in the model. In defining the constraints, we utilize a representation of the body pose in reduced coordinates, where the pose parameters encode the position and orientation of the root part and the remaining parts are encoded recursively using the values of angles at the joints in the body. We formally define the bounding constraints as position inequality constraints over the individual joint angle values and implement these constraints incrementally by constraining joint angle velocities.

Implications for Motion Estimation. Joint angle limits as defined in this section restrict the set of kinematic poses that the model of the person can assume. The position constraints that achieve this kinematic effect have been explored previously in both computer graphics and vision. However, together with simulation, the constraints achieve a dynamic effect that has not been implemented in vision applications thus far. The kinematic bounding ensures that the poses produced by the simulation are not only possible kinematically, but also that they can be attained by a human without falling or having to fly, thus enforcing much stronger constraints.

We next describe the representation of the pose of the body using reduced coordinates. We then illustrate how we use this pose representation to constrain the poses.

3.5.1 Pose Representation

Pose. Assuming the parts in the articulated body are organized into a hierarchy (*e.g.*, as in Table 3.3), we characterize the spatial configuration of the parts in terms of parameters that automatically incorporate the articulation constraints from Section 3.3.2. Towards that end, we re-parameterize the positions and orientations of the parts in the articulated body recursively, relative to their parent parts, using the values of displacements and angles at joints in the body as the parameters. We concatenate these values into the *pose* vector $\mathbf{q} \in \mathbb{R}^N_{dofs}$, where N_{dofs} is the total number of parameters. This representation of pose parameterizes the pose using reduced coordinates.

The pose vector $\mathbf{q}$ describes the *pose* of the model and consists of the *root position* $\mathbf{q}^{pos}$, *root orientation* $\mathbf{q}^{rot}$ and *joint angles* $\mathbf{q}^{ang}$,

$$\mathbf{q} = \left(\mathbf{q}^{pos}, \mathbf{q}^{rot}, \mathbf{q}^{ang}\right), \tag{3.156}$$

where $\mathbf{q}^{pos} \in \mathbb{R}^3$ records the displacements at the root joint and encodes the 3D root position of the model, $\mathbf{q}^{rot} \in \mathbb{R}^3$ records the angles at the root joint and describes the 3D root orientation and $\mathbf{q}^{ang} \in \mathbb{R}^{N_{dofs}-6}$ stores joint angles along the hierarchy and describes the pose of the model relative to the root. We encode the values of rotational joint DOFs using Euler angles.

In encoding $\mathbf{q}^{ang}$, we assume each joint angle $\mathbf{q}^{ang}_i$, $i = 1, \ldots, N_{dofs} - 6$, is due to an angular joint DOF i characterized by the tuple

$$(X_i, Y_i, \mathbf{w}^b_{X_i,i}, \mathbf{q}^{ang}_i), \tag{3.157}$$

where X_i and Y_i are the body parts affected by the DOF, $\mathbf{w}^b_{X_i,i}$ is the direction of the joint axis of the DOF in the body space of X_i and $\mathbf{q}^{ang}_i$ is the current value of the DOF. We assume that the DOF is set up such that the world space joint axis $\mathbf{w}_{X_i,i} = \mathbf{R}_{X_i} \cdot \mathbf{w}^b_{X_i,i}$ is attached to the part X_i and that $\mathbf{q}^{ang}_i$ measures the angle that the part Y_i is rotated about this joint axis with respect to the part X_i.

We take the same approach and characterize the angular root DOFs i that encode the root orientation $\mathbf{q}^{rot}$ and the linear root DOFs i that encode the root position $\mathbf{q}^{pos}$ by the tuples

$$(X_i := ground, Y_i := root, \mathbf{w}_i, \mathbf{q}_i^{rot}) \tag{3.158}$$

$$(X_i := ground, Y_i := root, \mathbf{w}_i, \mathbf{q}_i^{pos}), \tag{3.159}$$

where $\mathbf{w}_i$ is the i-th world space axis attached at the world space origin to the rigid body X_i that represents the ground, $\mathbf{q}_i^{rot}$ measures the rotation of the root part about this axis and $\mathbf{q}_i^{pos}$ records the displacement of the position of the root part from the world space origin along the axis.

Velocity. Using the same parameterization, we recursively re-parameterize the linear and angular velocities of the parts in the articulated body and encode the velocities of the parts relative to the motion of the parent parts, obtaining the velocity of the body. We define *pose velocity* $\dot{\mathbf{q}}$ of the model as the time derivative of the pose,

$$\dot{\mathbf{q}} = \frac{\partial \mathbf{q}}{\partial t}. \tag{3.160}$$

The components of $\dot{\mathbf{q}}$ can be characterized using the descriptions of DOFs from Equation (3.157), Equation (3.158) and Equation (3.159). Specifically, if i is an index of an angular joint DOF, then

$$\dot{\mathbf{q}}_i^{ang} = \mathbf{w}_{X_i,i} \cdot (\omega_{Y_i} - \omega_{X_i}). \tag{3.161}$$

Similarly, if i is an index of an angular root DOF or a linear root DOF, then

$$\dot{\mathbf{q}}_i^{rot} = \mathbf{w}_i \cdot \omega_{root} \tag{3.162}$$

$$\dot{\mathbf{q}}_i^{pos} = \mathbf{w}_i \cdot \mathbf{v}_{root}. \tag{3.163}$$

Dynamic Pose. The pose $\mathbf{q}$ together with the pose velocity $\dot{\mathbf{q}}$ encode the *dynamic pose* $(\mathbf{q}, \dot{\mathbf{q}})$ of the model. The dynamic pose characterizes the kinetic state of the entire articulated body, while implicitly accounting for the effects of the articulation constraints. Specifically, it specifies how the parts of the body are placed and oriented in the world with respect to each other and how the parts move at the current time instant so that the articulation constraints hold. We refer to the pose $\mathbf{q}$ as the *kinematic pose* when we want to be specific about the fact that the pose $\mathbf{q}$ encodes only the kinematic configuration of the body and not the velocity.

Note that reduced coordinates approaches for the simulation of the articulated body formulate equations of motion for the body directly in terms of the dynamic pose. We use the dynamic pose only for parameterization convenience and always simulate the body in maximal coordinates so that (1) we can directly implement other types of constraints and (2) implement all constraints in a uniform way.

3.5.2 Angle Limits Constraints

Formulation. We define joint angle limits constraints as position-level inequality constraints that are specific for individual angular joint DOFs $i = 1, \ldots, N_{dofs} - 6$. Each DOF i produces two constraints

$$C_{p,lo}^i := \mathbf{q}_i^{ang} - \left(\mathbf{q}_{lo}^{ang}\right)_i \geq 0 \tag{3.164}$$

$$C_{p,hi}^i := -\mathbf{q}_i^{ang} + \left(\mathbf{q}_{hi}^{ang}\right)_i \geq 0, \tag{3.165}$$

where $\left(\mathbf{q}_{lo}^{ang}\right)_i$ is the lower bound for the joint angle due to the DOF and $\left(\mathbf{q}_{hi}^{ang}\right)_i$ is the upper bound. We set the bounds for the body models manually or from training motion capture data (when available), based on biomechanical principles.

We maintain the position-level constraints incrementally, using the technique from Section 3.2.4.2. In doing so, we utilize the velocity-level formulations of the constraints

$$\dot{C}_{p,lo}^{i} = \dot{\mathbf{q}}_i^{ang} \geq 0 \tag{3.166}$$

$$\dot{C}_{p,hi}^{i} = -\dot{\mathbf{q}}_i^{ang} \geq 0 \tag{3.167}$$

that are in effect when a joint angle bound is hit. According to Equation (3.161), we have $\dot{C}_{p,lo}^{i} = \mathbf{w}_{X_i,i} \cdot (\omega_{Y_i} - \omega_{X_i})$ and $\dot{C}_{p,hi}^{i} = -\mathbf{w}_{X_i,i} \cdot (\omega_{Y_i} - \omega_{X_i})$. Therefore, $\ddot{C}_{p,lo}^{i} = (\omega_{X_i} \times \mathbf{w}_{X_i,i}) \cdot (\omega_{Y_i} - \omega_{X_i}) + \mathbf{w}_{X_i,i} \cdot (\alpha_{Y_i} - \alpha_{X_i})$ and $\ddot{C}_{p,hi}^{i} = -(\omega_{X_i} \times \mathbf{w}_{X_i,i}) \cdot (\omega_{Y_i} - \omega_{X_i}) - \mathbf{w}_{X_i,i} \cdot (\alpha_{Y_i} - \alpha_{X_i})$.

Implementation. Consequently, we can implement the lower bound for the angular joint DOF *i* from Equation (3.164) using Equation (3.105) or Equation (3.106) according to the rules in Section 3.2.4.2 with the following values for the terms in the formulations

$$\mathbf{C}_p^i = C_{p,lo}^i \tag{3.168}$$

$$\mathbf{J}_{i,X_i} = \begin{pmatrix} \mathbf{0}^T & -\mathbf{w}_{X_i,i}^T \end{pmatrix} \tag{3.169}$$

$$\mathbf{J}_{i,Y_i} = \begin{pmatrix} \mathbf{0}^T & \mathbf{w}_{X_i,i}^T \end{pmatrix} \tag{3.170}$$

$$\dot{\mathbf{J}}_{X_i,\cdot}\mathbf{v}_{X_i} + \dot{\mathbf{J}}_{Y_i,\cdot}\mathbf{v}_{Y_i} = (\omega_{X_i} \times \mathbf{w}_{X_i,i}) \cdot (\omega_{Y_i} - \omega_{X_i}) \tag{3.171}$$

$$m_i = 1. \tag{3.172}$$

The upper bound for the DOF *i* from Equation (3.164) is implemented analogously, using the following values for the terms

$$\mathbf{C}_p^i = C_{p,hi}^i \tag{3.173}$$

$$\mathbf{J}_{i,X_i} = \begin{pmatrix} \mathbf{0}^T & \mathbf{w}_{X_i,i}^T \end{pmatrix} \tag{3.174}$$

$$\mathbf{J}_{i,Y_i} = \begin{pmatrix} \mathbf{0}^T & -\mathbf{w}_{X_i,i}^T \end{pmatrix} \tag{3.175}$$

$$\dot{\mathbf{J}}_{X_i,\cdot}\mathbf{v}_{X_i} + \dot{\mathbf{J}}_{Y_i,\cdot}\mathbf{v}_{Y_i} = -(\omega_{X_i} \times \mathbf{w}_{X_i,i}) \cdot (\omega_{Y_i} - \omega_{X_i}) \tag{3.176}$$

$$m_i = 1. \tag{3.177}$$

3.6 Actuation

Constraints from previous sections can only generate a motion of the model that is passive. The motion complies with the geometry of the environment, articulation constraints and kinematic constraints on the pose, but the simulated character appears unconscious and falls straight down on the ground during simulation. To generate motions that are voluntary and realize particular desired behaviors, the model needs be actuated.

We actuate the model by equipping the articulated body with motors and by activating these motors using a controller. In this section, we focus on the presentation of a general *low-level motion control* (body actuation) paradigm within the controller for the character, where the desired behavior for the model is described by actions. We define actions momentarily and illustrate how the actions can be performed by powering the motors. We refine the actuation paradigm in following chapters, when we talk about particular high-level control processes in controllers that generate actions.

We next describe the motors, actions and powering of the motors to perform the actions. In explaining the body actuation, we assume actions are generated by a high-level decision-making process in the controller that we overviewed in Section 1.4.3.1. We discuss particular decision processes in Chapter 4 and Chapter 5.

Joint Motors. We employ a humanoid body model for actuation, where parts of the body are actuated by motors integrated with the joints in the articulated body. The motors directly control angular joint DOFs in the body model (*e.g.*, an angle at the knee), by producing torques about the corresponding joint axes. We generate the torques from position-level actuation constraints that specify desired values for the individual DOFs.

We implement body actuation as an incremental stabilization of the actuation constraints, using the approach from Section 3.2.4.1. The actuation constraints are maintained as soft velocity-level or acceleration-level bounded-equality constraints from Equation (3.103) and Equation (3.104) with constraint force bounds.

Actions. Actions characterize instantaneous desired actuation of the body. The parameters of an action include an encoding of a single *desired (target) pose* $\mathbf{q}_d$ for the model that the motors attempt to follow and a parameterization of the *strategy* to achieve the pose. In actuating the body, actions encode simpler control tasks. More complex behaviors of the model are produced by switching actions using a high-level control process (see Section 1.4.3.1).

3.6.1 Actuation Constraints

Assuming we have an action that the motors should perform, we realize the actuation by defining actuation constraints for the individual degrees of freedom in the body.

Overview. We implement actuation using constraint-based springs. The springs are realized as stabilized soft constraints with constraint force bounds (see Section 3.2.4.1), where the constraints work to stabilize the current pose $\mathbf{q}$ of the body towards a desired pose $\mathbf{q}_d$ in the action and the parameters of the constraints encode the actuation strategy. The strategy is described in terms of parameters α of the stabilization mechanism that stabilizes the constraints, softnesses $\mathbf{s}$ of the constraints and constraint force bounds λ^{lo}, λ^{hi}:

- The stabilization mechanism powers the body and actuates it towards the desired pose $\mathbf{q}_d$. It uses an exponential decay to reduce the difference between the current pose and the desired pose, based on the discussion in Section 3.2.4.1. Parameters α of the stabilization mechanism in the constraint formulations control the speed of stabilization and correspond to gains of springs.

- Softnesses $\mathbf{s}$ of the constraints damp the stabilization process and make the body appear less stiff. Softness parameters in the formulation correspond to damping parameters of springs.

- Constraint force bounds λ^{lo} and λ^{hi} account for the powering limits of the motors and make the generated motions appear more human-like such that super-human forces can not be generated.

By using soft constraints and constraints with bounds for the actuation of the body, we explicitly request the pose of the body oscillate around the desired pose such that it does not reach the desired pose exactly in one simulation step. By using appropriate parameters for the constraints, encoded in the action, we can encode behaviors that last for many frames. Consequently, our actions need not change with every frame and we can characterize the desired behavior for the model in a sparse way, using only a few actions. We utilize this fact extensively in Chapter 5.

Joint DOF Actuation. We now define the actuation constraints. Our goal is to control all degrees of freedom in the body such that the desired pose $\mathbf{q}_d$ is achieved, but in doing so, only joint angles can be powered directly. Consequently, the root part remains unactuated (*i.e.*, there are no magic forces that hold and drive the root) and the position and orientation of the root results from the motion of the rest of the body and the interactions of the body with the environment.

We formulate low-level motion control of the character as a collection of position-level equality constraints i, such that the i-th constraint actuates the i-th angular joint DOF in the pose representation. Utilizing the representation of the pose and the characterization of the DOF from Section 3.5.1, the i-th DOF is actuated by the constraint

$$C^i_{p,ang} := \mathbf{q}^{ang}_i - (\mathbf{q}^{ang}_d)_i = 0, \tag{3.178}$$

where $(\mathbf{q}^{ang}_d)_i$ is the desired value for the DOF, as given by the desired pose $\mathbf{q}_d$. This position-level constraint is similar to the joint angle limit constraint from Equation (3.164), except that it is an equality constraint. Similarly to before, it produces $\dot{C}^i_{p,ang} = \mathbf{w}_{X_i,i} \cdot (\omega_{Y_i} - \omega_{X_i})$ and $\ddot{C}^i_{p,ang} = (\omega_{X_i} \times \mathbf{w}_{X_i,i}) \cdot (\omega_{Y_i} - \omega_{X_i}) + \mathbf{w}_{X_i,i} \cdot (\alpha_{Y_i} - \alpha_{X_i})$.

Implementation. Using the formulations from above, we can implement the constraint in terms of Equation (3.103) or Equation (3.104). To maintain the semantics we promised, we take these equations as the definitions of soft bounded-equality constraints (see Section 3.2.2.6) with the softness parameter s_i and constraint force bounds λ^{lo}_i and λ^{hi}_i. We also force the use of the stabilization parameter α_i. The constraint produces the following terms for the soft bounded-equality variants of Equation (3.103) and Equation (3.104),

$$\mathbf{C}_p^i = C_{p,ang}^i \tag{3.179}$$

$$\mathbf{J}_{i,X_i} = \begin{pmatrix} \mathbf{0}^T & -\mathbf{w}_{X_i,i}^T \end{pmatrix} \tag{3.180}$$

$$\mathbf{J}_{i,Y_i} = \begin{pmatrix} \mathbf{0}^T & \mathbf{w}_{X_i,i}^T \end{pmatrix} \tag{3.181}$$

$$\dot{\mathbf{J}}_{X_i,\cdot}\mathbf{v}_{X_i} + \dot{\mathbf{J}}_{Y_i,\cdot}\mathbf{v}_{Y_i} = (\omega_{X_i} \times \mathbf{w}_{X_i,i}) \cdot (\omega_{Y_i} - \omega_{X_i}) \tag{3.182}$$

$$m_i = 1 \tag{3.183}$$

$$\alpha = \alpha_i \tag{3.184}$$

$$\mathbf{s}_i = \mathbf{s}_i \tag{3.185}$$

$$\lambda_i^{lo} = -\gamma \cdot \frac{m_{X_i} + m_{Y_i}}{2} \tag{3.186}$$

$$\lambda_i^{hi} = \gamma \cdot \frac{m_{X_i} + m_{Y_i}}{2}. \tag{3.187}$$

We set the bounds for the constraint force based on the values of the masses m_{X_i} and m_{Y_i} such that

$$\lambda_i^{lo} := -\gamma \cdot \frac{m_{X_i} + m_{Y_i}}{2} \leq \lambda_i \leq \lambda_i^{hi} := \gamma \cdot \frac{m_{X_i} + m_{Y_i}}{2}, \tag{3.188}$$

where λ_i is the multiplier due to the constraint force due to the actuation constraint i and γ is a single parameter for the entire body that controls the strength of the motors in the body. Intuitively, these bounds implement the notion that heavier body parts contain larger muscle volume and, hence, can apply larger torques about the joint axis. To account for the fact that the two parts X_i and Y_i connected at the joint may have disproportional mass, we take an average of m_{X_i} and m_{Y_i}. This bounding is an approximation to the answer that would take into account moment of inertia of the affected parts.

Balance. The actuation mechanism described here only actuates joint angles such that $\mathbf{q}^{ang}$ approaches $\mathbf{q}_d^{ang}$. In doing so, and because $\mathbf{q}^{ang}$ encodes the pose of the body relative to the root part, it does not take into consideration whether the root tracks the desired root position and orientation in $\mathbf{q}_d$. Consequently, the orientation of the root can deviate from the desiredata and the simulated model can stumble and/or fall.

To address this problem, a form of balance has to be integrated with body actuation. We assume that balancing is realized by high-level control processes (discussed *e.g.*, in Chapter 5) that dynamically adjust the desired joint angles in actions based on the state of the model such that the joint angles are tracked and the model stays balanced. We employ a feedback-based method of Yin *et al.*[212] for balance control in Chapter 5 that adjusts the desired pose and actuates the body such that both the desired joint angles and the desired root orientation in $\mathbf{q}_d$ are tracked.

Action Parameterization. We refer to actions by $\mathbf{a}$. We characterize each action by the desired pose $\mathbf{q}_d$, balance parameters ϑ, specific to the used high-level balancing strategy, and constraint stabilization parameters α_i and constraint softness parameters s_i for each angular joint DOF i that we concatenate into α and $\mathbf{s}$,

$$\mathbf{a} = (\mathbf{q}_d, \vartheta, \alpha, \mathbf{s}). \tag{3.189}$$

For notation convenience, we collect α and $\mathbf{s}$ into the vector θ,

$$\theta = (\alpha, \mathbf{s}), \tag{3.190}$$

such that θ encodes the control strategy and ϑ encodes the balance strategy. Using this notation, we can parameterize the action by an equivalent vector

$$\mathbf{a} = (\mathbf{q}_d, \vartheta, \theta). \tag{3.191}$$

We illustrate the parameterization in Figure 3.2.

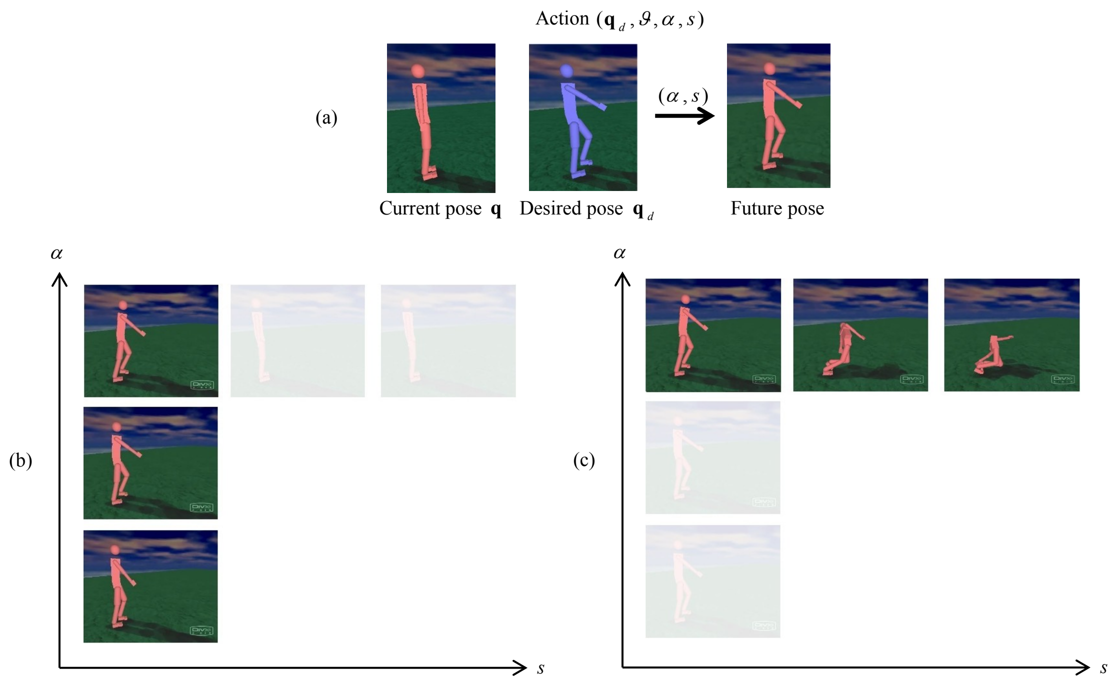

Action $(\mathbf{q}_d, \vartheta, \alpha, s)$
(a)
(α, s)
Current pose $\mathbf{q}$
Desired pose $\mathbf{q}_d$
Future pose
α
(b)
s
α
(c)
s

Figure 3.2: **Parameterization of Actions.** We parameterize actions uniformly using the vector $(\mathbf{q}_d, \vartheta, \alpha, \mathbf{s})$, where $\mathbf{q}_d$ encodes the desired pose for the character that should be reached when the action is active, α describes the speed with which the pose is approached and $\mathbf{s}$ characterizes the softness of the actuation constraints that do so. The vector ϑ encodes parameters of a balancing strategy that dynamically adjusts $\mathbf{q}_d$ such that the character does not fall and we discuss this strategy in Chapter 5. Here, we illustrate actuation results when no active balancing strategy is employed such that $\vartheta = \emptyset$. The first row in the figure shows a character in the current pose $\mathbf{q}$, a desired pose $\mathbf{q}_d$ that is to be reached and the pose that the character assumes in 1 second for a certain setting of α and $\mathbf{s}$. The second row illustrates how the assumed poses change as a function of either α (b) or $\mathbf{s}$ (c), when all other parameters of the action remain fixed.

3.7 Constrained Simulation

Simulation. We simulate the articulated model of the person using the method from Section 3.1.4 that we apply to the state $\mathbf{y}$ of the entire model. Specifically, we use $\mathbf{y} = (\mathbf{y}_1, \ldots, \mathbf{y}_n)$ to represent the model state, where $\mathbf{y}_j$ is the state of the part j in maximal coordinates and n is the number of parts, and use the integration

Aspect	Constraint	Formulation Type	Section
Articulation	Ball-and-socket Joint	Position Equality	Section 3.3.2.1
Articulation	Saddle Joint	Position Equality	Section 3.3.2.2
Articulation	Hinge Joint	Position Equality	Section 3.3.2.3
Non-penetration, Friction	Contact	Special Bounded Equality	Section 3.4.4
Possible Poses	Joint DOF Limits	Position Inequality	Section 3.5.2
Body Actuation	Joint DOF Actuation	Position Equality	Section 3.6.1

Table 3.4: **Summary of Constraints.** Summary of constraints that we employ to build our model of the person. We model the person as a collection of constrained rigid bodies where constraints implement various aspects of motion control. We list the aspects and point to sections that define the constraints that model them.

function $f : \mathbf{y}_t \rightarrow \mathbf{y}_{t+1}$ to update the state from the time t to the time $t+1$. The function operates as in Section 3.1.4, except that it updates states of all parts at the same time, based on the forces and impulses that are applied on the parts.

We use constraints on acceleration and velocities[5] of the parts in the model to produce forces for the parts such that the parts stay connected at joints (see Section 3.3.2), contact of parts and the environment is accounted for (see Section 3.4.4), kinematically impossible poses of the model are avoided (see Section 3.5.2) and joint angles in the body are actuated as desired (see Section 3.6.1). These constraints are always formulated with respect to the current state of the model and we summarize the constraints in Table 3.4.

We apply the following forces and impulses on the model:

- constraint impulse $\mathbf{F}_{imp}^{constraint}$ due to current velocity-level constraints,
- constraint force $\mathbf{F}^{constraint}$ due to current acceleration-level constraints, and
- gravity force $\mathbf{F}^{gravity}$ from Equation (3.15) that is applied on each body part separately.

Inverse Dynamics. We compute constraint forces and impulses using a constraint solver and the built-in support for inverse dynamics. The support takes as an input a potential state $\mathbf{y}$ of the model and formulations of velocity-level and acceleration-level constraints from Section 3.2.2 and Section 3.2.3 to be satisfied in that state, and outputs the constraint impulse $\mathbf{F}_{imp}^{constraint}$ and the constraint force $\mathbf{F}^{constraint}$ that implement the constraints.

The support computes the constraint impulse $\mathbf{F}_{imp}^{constraint}$ with respect to the state $\mathbf{y}$ by reducing the formulations of velocity constraints to a LCP using the method described in Section 3.2.3. It applies the impulses on the state and solves for the constraint force $\mathbf{F}^{constraint}$ to satisfy acceleration constraints formulated with respect to the updated state. Constraint forces are obtained via an analogous reduction to a similar LCP, as explained in detail in Section 3.2.2.7. We handle LCP constraints with non-constant force bounds for modeling friction using the approach from Section 3.4.4.

We use this support for inverse dynamics in our controllers in Chapter 4 and Chapter 5 to compute the actuation forces for the model and to determine contact forces so that the controllers can reason about which

[5]We request most constraints on the velocity level and implement them on the acceleration level, according to discussion in Section 3.2.3, when all velocity constraints are already satisfied.

parts of the body are currently supported by the environment. We also use this contact force information to generate contact events for the controllers.

Implementation. We refer interested readers to [191, 192] for detailed information on the implementation of the simulator.

Chapter 4

Physics-based Particle Filtering

This chapter describes our basic approach towards addressing physical plausibility in motion estimation. In this chapter, we integrate physically plausible motion predictions produced by a trajectory tracking controller with pose tracking by a Particle Filter. We use the Particle Filter to estimate a desired pose trajectory for the character as an unstructured sequence of desired poses specified for every frame in the sequence and we use a trajectory tracking controller to realize the motion encoded by the sequence and produce the interpretation of the images.

In doing so, we pursue the general controller estimation approach outlined in Section 1.5.1 that we apply to a trajectory tracking controller model. A trajectory tracking controller (see the part (c) in Figure 1.13) takes the representation of the desired motion for the character as an unstructured sequence of desired poses and tracks the poses in the sequence as closely as possible. We use optimization by a Particle Filter to recover the desired poses in the sequence incrementally from image observations using analysis-by-synthesis such that the motion produced by the tracking controller with the estimated sequence matches image observations the best. The approach uses the general method from Section 1.5.1 for the estimation of the desired poses for the tracking controller. In doing so, it makes use of the character model from Chapter 3.

In order to facilitate efficient inference of the desired pose trajectories, we utilize an exemplar-based method for control that biases the trajectories to pose sequences similar to training motions. The resulting simulation-based method can estimate human motion from a single-view or multiple-view imagery. It uses an explicit knowledge of the environment to model physical ground-person interactions as well as other physical laws encoded by the underlying physical model (see Chapter 3) to regularize tracking. We show, both quantitatively and qualitatively, that our physics-based method improves upon other Bayesian filtering methods with standard motion priors and enables motion estimation in presence of strong person interactions with the ground.

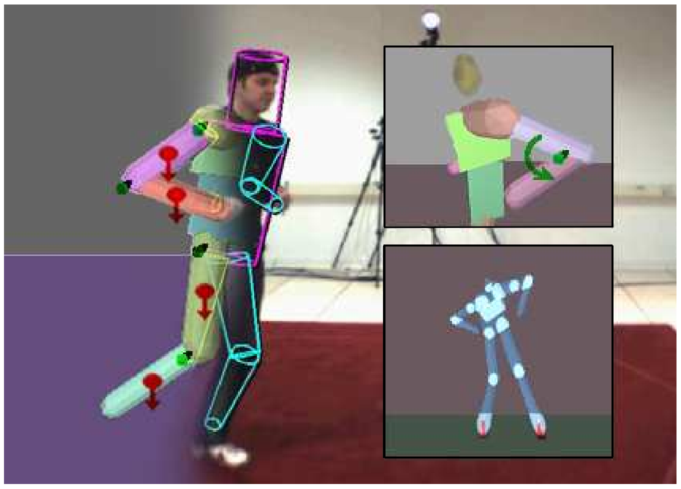


Figure 4.1: **Incorporating Physics-based Dynamic Simulation With Joint Actuation and Dynamic Interaction Into Particle Filtering.** We represent the person as a simulated character. The motion of the character is determined by general physical laws, actuation forces at joints (top) and interaction of shape surfaces at contacts (bottom).

4.1 Introduction

Our method integrates full body physical simulation of the model of the person with pose tracking by a Particle Filter. We take the humanoid character model from Chapter 3 and actuate it using a trajectory tracking controller that we introduce in this chapter to generate pose predictions for the tracking. Pose predictions are generated using the simulation and control loop from Section 1.4.3.2, where the trajectory controller provides desired poses for low-level motion control of the character from Section 3.6.

Our method can be interpreted in two ways: (1) as a special case of the general controller estimation approach from Section 1.5.1 that is applied for the estimation a trajectory tracking controller and (2) as a generalization of standard kinematic pose tracking using a Particle Filter, where simulation and the control loop of the physical model replaces the kinematic motion model in the filter. We discuss each of these two interpretations next. In later exposition, we refer to any of these interpretations as convenient.

4.1.1 Relation to Controller Estimation

From the point of view of the controller estimation approach from Chapter 1, our goal in this chapter is to estimate a trajectory tracking controller for the observed motion and use it to re-synthesize the poses in the image sequence. We characterize the controller and the controller estimation process as follows.

Actions. The trajectory tracking controller uses actions from Section 3.6 that switch with every frame in the sequence and that are parameterized by Equation (3.189) each. Actions encode strategies for low-level motion control of the character that drives the pose of the character towards desired poses represented by the actions.

Because actions are short-time in this case, we can assume all aspects of low-level motion control can be encoded by varying the desired pose for the character between frames. As such, the controller does not employ any explicit strategy for body balance and it uses fixed constraint stabilization constants α and softness parameters s for the actuation constraints. These parameters are shared by all joint angle DOFs and all actions. Consequently, only the encoding of the desired pose $\mathbf{q}_d$ within Equation (3.189) is relevant for each action and we use desired poses as representations of actions.

Action Selection. Actions in our trajectory tracking controller switch regularly, with an ability to perform a "passive" action that does not produce any actuation forces when the model incurs a heavy impact. Consequently, the state-machine for actions (as defined in Section 1.4.3.1) is not maintained explicitly.

Controller Estimation. Given this characterization of actions and the process that selects actions, the desired motion for the trajectory tracking controller is fully determined by the sequence $\Theta = \{(\mathbf{q}_d)_t\}_{t=1}^{T}$ of the desired poses $(\mathbf{q}_d)_t$ for the frames, where t refers to frames and T is the number of frames in the sequence. Here, Θ encodes the full parameters of the controller and the estimation of the controller amounts to the estimation of the desired pose for the character in every frame of the sequence.

The high dimensionality of Θ ($T \times N_{dofs}$ dimensions) makes a batch optimization for the entire pose trajectory difficult. To address this problem, we optimize Θ incrementally using a Particle Filter, where each iteration in the filter recovers a desired pose for one additional frame in the sequence (see (c) in Figure 4.2). For practical reasons, we also implement a hybrid method that adds noise to the poses produced by the simulation in order to achieve a more accurate motion reconstruction (see (b) in Figure 4.2). In doing so, we reflect on the fact that the inference of Θ alone is prone to overfitting to individual frames and usually fails to achieve an accurate overall fit. By allowing the reconstructions to deviate from the simulated motion, we can get a more accurate motion interpretation but we lose physical plausibility.

4.1.2 Relation to Kinematic Trackers

From the point of view of Bayesian kinematic trackers, our goal is to clean pose predictions within the trackers so that the predictions alone would be physically plausible (see Figure 4.2).

Kinematic Trackers. *Kinematic trackers* (see (a) in Figure 4.2) realize direct inference over kinematic poses in the image sequence. They are typically implemented by Particle Filters. The filter maintains hypotheses about possible interpretations of the current frame in terms of the pose $\mathbf{q}$ and propagates these hypotheses incrementally from one frame to another. The update mechanism is illustrated in Figure 4.2 (a), where a pose hypothesis $\mathbf{q}_t$ at the current frame is updated for the next frame by applying a kinematic motion model to predict a possible pose $\mathbf{q}_{t+1}$ for the character at the next frame. The new hypothesis is subjected to the addition of a random noise to account for the dynamics that is not modeled by the motion model and the validity of the hypothesis (weight) is evaluated with respect to the image observations at the next frame. This evaluation may include *e.g.*, measuring of the overlap between the observed silhouette of the person in the next frame and the hypothesized silhouette, as generated from the pose hypothesis and the geometry of the character using camera projection (see Section 1.5.1). Because the kinematic motion model does not account for physics directly, it may generate predictions that are not physically plausible, in a way that they simply

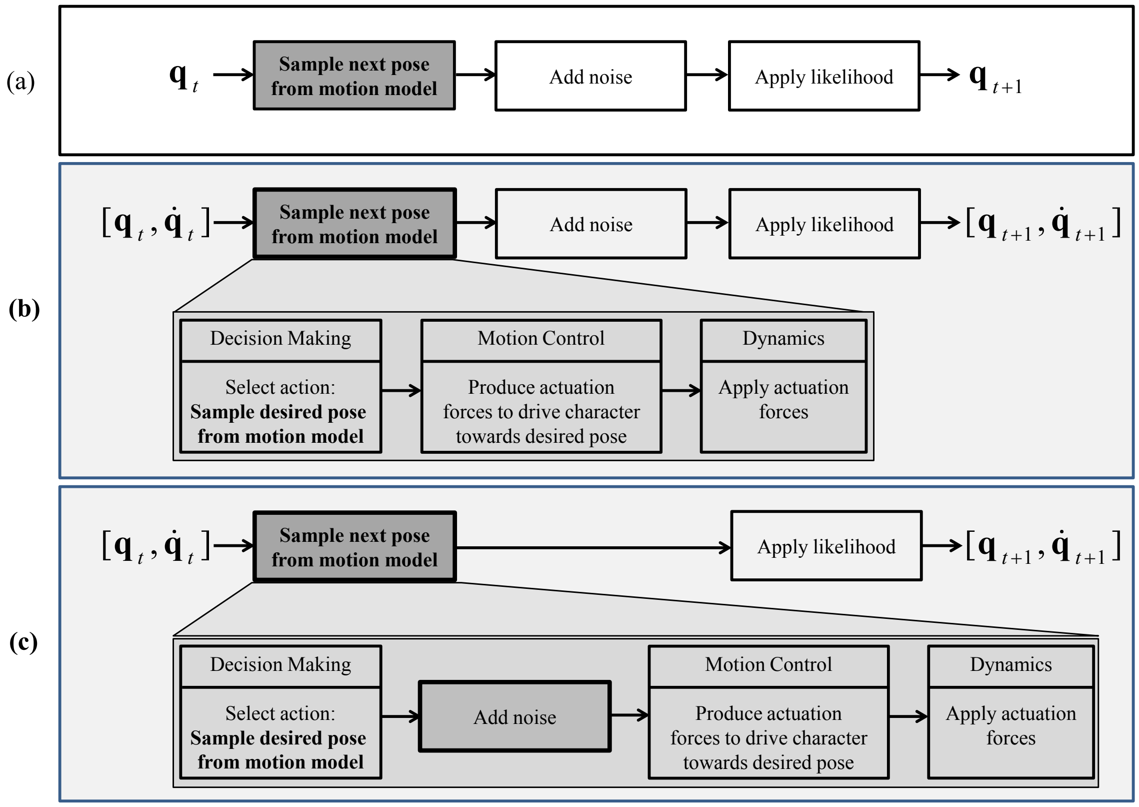

(a)
$\mathbf{q}_t$
Sample next pose from motion model
Add noise
Apply likelihood
$\mathbf{q}_{t+1}$
(b)
$[\mathbf{q}_t, \dot{\mathbf{q}}_t]$
Sample next pose from motion model
Add noise
Apply likelihood
$[\mathbf{q}_{t+1}, \dot{\mathbf{q}}_{t+1}]$
Decision Making
Select action: Sample desired pose from motion model
Motion Control
Produce actuation forces to drive character towards desired pose
Dynamics
Apply actuation forces
(c)
$[\mathbf{q}_t, \dot{\mathbf{q}}_t]$
Sample next pose from motion model
Apply likelihood
$[\mathbf{q}_{t+1}, \dot{\mathbf{q}}_{t+1}]$
Decision Making
Select action: Sample desired pose from motion model
Add noise
Motion Control
Produce actuation forces to drive character towards desired pose
Dynamics
Apply actuation forces

Figure 4.2: **Kinematic Trackers And Our Trackers.** Comparison of the general pipeline of standard kinematic trackers (a) to our physics-based hybrid tracker (b) and our physically-plausible tracker (c). Kinematic trackers (a) take an estimate of the pose from the current frame and update it through the kinematic motion model to produce a prediction for the pose for the next frame for the tracker. Because the kinematic motion model is not aware of physical constraints, the predicted pose can violate physical constraints. Contrary, with our hybrid method (b), implemented in Section 4.2.4.2, we take the prediction as a target for simulation with a trajectory tracking controller, simulate the character towards the desired pose and output the pose the simulation ends in as a corrected prediction. As such, our hybrid method filters predictions from the kinematic motion model, making sure the predictions are physically plausible, such that the character can move from the current pose to the predicted pose. However, because the hybrid method adds noise to the poses produced by the simulation, the estimated pose sequence may not be physically plausible. Our physically plausible tracker (c), implemented in Section 4.2.4.1, addresses this problem by adding noise only to the desired poses, before the low-level motion control takes place.

penetrate the environment or it is not biomechanically possible for a human to move to the predicted pose from the current pose.

Our Hybrid Tracker. To improve physical plausibility of predictions and the recovered motions, we propose a hybrid pose tracker that integrates physical simulation with a trajectory tracking controller into the prediction module in the Particle Filter (see (b) in Figure 4.2).

The purpose of the simulation is to correct next pose predictions from the kinematic motion model such that the predictions would be consistent with physical laws and the poses could be reached from the current pose by a human. Towards that end, the physics-based tracker uses the kinematic motion model to sample desired poses for the trajectory controller and uses physical simulation with the controller to drive the character model towards them. The poses produced by the simulation are then used as new/corrected predictions for the filter. As such, predictions made by our physics-based motion model can be seen as results of a filtering process built on top of the more traditional kinematic motion model that makes predictions physically plausible.

Our Physically-plausible Tracker. The controller estimation approach, as applied to the trajectory-tracking controller model in this chapter, is implemented as a desired pose tracker. This tracker differs from the hybrid tracker by adding noise only to the desired poses generated by the kinematic motion model (see (c) in Figure 4.2). The desired poses followed by the low-level motion control process in Figure 4.2 (c) encode the desired pose trajectory $\Theta = \{(\mathbf{q}_d)_t\}_{t=1}^{T}$ for a trajectory-tracking controller that reconstructs the entire observed motion in simulation.

Prior over Desired Poses. We sample desired poses for control from a kinematic prior that can be used by any kinematic pose tracker. We use this prior to limit the effective search space for the desired poses and allow tractable inference of poses. Although we can use any kinematic prior for this purpose, we make use of an exemplar-based prior similar to [155]. Priors similar to ours have been successfully used for articulated pose estimation in [155, 159, 186], dynamically adaptive animation [215], and humanoid robot imitation [75]. Here, we utilize the previous exemplar methods [155] in a new domain, to deal with exemplars that account for single-frame kinematics and dynamics of human motion.

Control. Our exemplar-based prior provides desired poses for a constraint-based trajectory-tracking controller for an articulated body similar to that of Kokkevis *et al.*[87], where joint DOF trajectories are defined on a frame-by-frame basis from a database of motion capture data.

Benefits. Our simulation-based approach has a number of benefits over the kinematic trackers: (1) predicted poses are implicitly biased towards physically plausible interpretations and (2) reasonable predictions can be made even when there is not enough training data available due to the direct incorporation of physics.

We demonstrate these benefits in quantitative and qualitative tracking experiments that involve subjects under effects of dynamic human-environment interactions. We illustrate that the proposed physics-based pose tracker achieves a better tracking accuracy than previous kinematic trackers and is able to better generalize to certain new environments and physical interactions. In order to facilitate a better understanding of this model and promote the use of physical simulation for tracking, we have made the source code of our controller and the simulation-based motion model available on our project website at `http://brown-robotics.org/wp/projects/current/dynamical_tracking/`. The project website also features videos that supplement experiments from this chapter.

4.1.3 Particle Filter for State Inference

In order to formulate our tracker, we first provide a background on general tracking of a model state based on indirect observations of the state using a *Particle Filter* (PF). We use this general mechanism for the inference of the pose. In the context of this chapter, our goal is to jointly track the pose of the simulated character and the state of the underlying control process between frames, encoded into the state vector $\mathbf{x}$, utilizing silhouettes from the video frames, encoded in $\mathbf{y}$.

Tracking with PF is formulated as an estimation of the *posterior* $p(\mathbf{x}_t|\mathbf{y}_{1:t})$ for frames $t = 1, \ldots, T$, where $\mathbf{x}_t$ is the state of the model at the frame t and $\mathbf{y}_{1:t}$ is the set of observations up to and including the frame t. The posterior characterizes the density over potential model states at the frame t and provides the estimation of the true state at that frame. We wish to recover this density for every frame t in the sequence.

This density is in PF approximated using a set of (typically) weighted samples/particles and is computed recursively from the representation at the previous frame,

$$\underbrace{p(\mathbf{x}_t|\mathbf{y}_{1:t})}_{\text{Posterior at } t} \propto \underbrace{p(\mathbf{y}_t|\mathbf{x}_t)}_{\text{Likelihood}} \underbrace{\int \overbrace{p(\mathbf{x}_t|\mathbf{x}_{t-1})}^{\text{Temporal Prior}} \overbrace{p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1})}^{\text{Posterior at } t-1} \, \mathrm{d}\mathbf{x}_{t-1}}_{\text{Predictive Density}} . \tag{4.1}$$

In this formulation, $p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1})$ is the posterior from the previous frame and $p(\mathbf{y}_t|\mathbf{x}_t)$ is the *likelihood* that measures how well a hypothesis at the frame t explains the observations at the frame. *The density* $p(\mathbf{x}_t|\mathbf{x}_{t-1})$ *is often referred to as the* temporal prior*, or* motion model*, and is the main focus of our work in this chapter.*

Temporal Priors. Temporal prior encodes temporal relationship of model states between frames and is usually represented as a first or a second order linear dynamical system with Gaussian noise [13, 46, 157]. For example, in [13, 46] the non-informative smooth prior,

$$p(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_{t-1}, \Sigma), \tag{4.2}$$

which facilitates continuity in the recovered motions, was used; alternatively, constant velocity temporal priors of the form:

$$p(\mathbf{x}_t|\mathbf{x}_{t-1}, \gamma_{t-1}) = \mathcal{N}(\mathbf{x}_{t-1} + \gamma_{t-1}, \Sigma), \tag{4.3}$$

where γ_{t-1} is scaled velocity learned or inferred (*e.g.*, $\gamma_{t-1} = \mathbf{x}_{t-1} - \mathbf{x}_{t-2}$), have also been proposed [157] and shown to have favorable properties when it comes to monocular imagery. However, human motion, in general, is non-linear and non-stationary. Kinematic data-driven models (*e.g.*, see Section 2.1.5.2) attempt to capture these aspects implicitly, by learning models of human motion from training motion capture data. In contrast, we argue such aspects result from physical laws and interactions of the body with an environment, and can be represented explicitly by incorporating these laws into the prior through physical simulation.

Inference Using PF. General inference with a Particle Filter proceeds incrementally by updating the representation of $p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1})$ from the frame $t-1$ to the frame t using Equation (4.1). In doing so, the motion model within the filter is first applied onto the particles from the frame $t-1$ to generate (predict) hypotheses for the new model states at the frame t by sampling from $p(\mathbf{x}_t|\mathbf{x}_{t-1})$. The new hypotheses are then scored and re-weighted using the likelihood model, based on how well they match the observations $\mathbf{y}_t$ at the frame

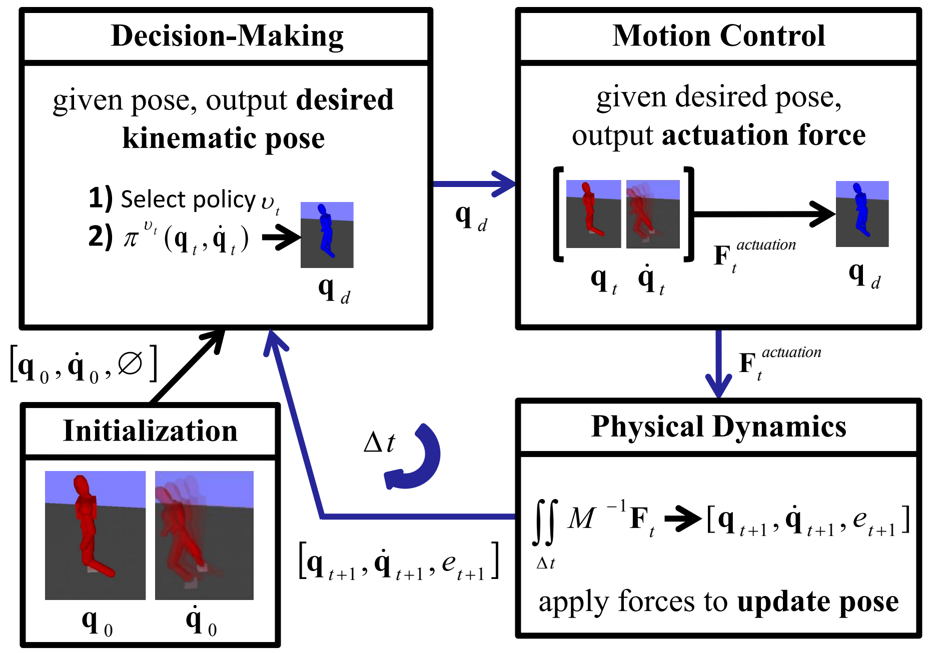

Decision-Making
given pose, output desired kinematic pose
1) Select policy υ_t
2) $\pi^{\upsilon_t}(\mathbf{q}_t, \dot{\mathbf{q}}_t)$
$\mathbf{q}_d$
Motion Control
given desired pose, output actuation force
$\mathbf{q}_t$ $\dot{\mathbf{q}}_t$
$\mathbf{F}_t^{actuation}$
$\mathbf{q}_d$
$[\mathbf{q}_0, \dot{\mathbf{q}}_0, \varnothing]$
Initialization
$\mathbf{q}_0$ $\dot{\mathbf{q}}_0$
Δt
$[\mathbf{q}_{t+1}, \dot{\mathbf{q}}_{t+1}, e_{t+1}]$
Physical Dynamics
$\iint_{\Delta t} M^{-1}\mathbf{F}_t \rightarrow [\mathbf{q}_{t+1}, \dot{\mathbf{q}}_{t+1}, e_{t+1}]$
apply forces to update pose

Figure 4.3: **Motion Model: Simulation and Control Loop.** Each iteration advances the model state $[\mathbf{q}, \dot{\mathbf{q}}, \upsilon]$ by the time Δt by (1) generating a desired kinematic pose $\mathbf{q}_d$ for the character to follow (decision-making process in the controller), (2) computing actuation forces needed to approach the desired pose (low-level motion control process in the controller) and (3) applying actuation forces on the character to update the current dynamic pose (physical dynamics). Contact feedback information e records significant impacts of the character with the environment and affects selection of the control policy for the next iteration.

t. Assuming the initial state $\mathbf{x}_1$ is given, this process can be repeatedly applied on $\mathbf{x}_1$ to estimate $p(\mathbf{x}_t|\mathbf{y}_{1:t})$ for any $t = 2, \ldots, T$.

In our pose tracker in this chapter, we use a variant of PF that augments the representation of particles. Each particle stores not only a potential hypothesis for the model state at the current frame, but also the full history of the previous hypotheses from the previous frames that the particle was generated from. We use this information to trace how a given model state evolved from the initial state and to report the most-likely consistent motion interpretation of the entire sequence that is physically plausible when tracking finishes. From now on, we assume that the representation of each state hypothesis $\mathbf{x}_t$ at the frame t implicitly records the entire history of the states $\mathbf{x}_{1:t} = \{\mathbf{x}_i\}_{i=1}^t$ for the frames up to and including the frame t, such that each $\mathbf{x}_i$ in the history was produced by sampling from $p(\mathbf{x}_i|\mathbf{x}_{i-1})$ for $i = 2, \ldots, T$.

4.2 Tracking with Physical Dynamics

We realize pose tracking by utilizing the general tracking approach from Section 4.1.3 with the temporal prior $p(\mathbf{x}_t|\mathbf{x}_{t-1})$ based on simulation of physical dynamics of the model of the person. We run our simulations in a world model consisting of a known static environment, $\mathscr{G}$, and a character model from Chapter 3 that represents the person (Figure 4.4). We assume "physical properties" (mass, inertial properties and shapes) are known for each part of the body.

Our temporal prior is implemented as a feedback control loop from Section 1.4.3.2 that we integrate with a trajectory tracking controller. As motivated in Section 4.1.1, the controller uses actions that switch at the beginning of each frame. These actions are parameterized by desired poses that the low-level motion control process attempts to reach in simulation, and the process applies the same control strategy for performing all actions. At any frame t, the decision- making process in the controller determines the desired pose that should be approached and the low-level motion control process computes the actuation forces that do so. We use the control loop to draw samples from the temporal prior in order to make predictions for next model states in pose tracking. We illustrate this loop, integrated with the trajectory tracking controller, in Figure 4.3.

From the point of view of a kinematic tracker, predictions of new model states using the control loop can be explained as a filtering process that attempts to clean up pose predictions from the kinematic motion model (see Section 4.1.2). This filtering process takes next pose proposals from the kinematic prior as an input. The proposed poses are corrected through *simulation* of the character model in the environment towards the proposals, using the pose outputs by the simulation as new pose predictions. The corrected poses are, by definition, physically plausible and transitions between them feasible. We abstract kinematic priors by control policy functions. Control policy functions map current poses to next intended (desired) poses and implicitly encode control intentions (or objectives) of the model. We permit a collection of (possibly motion-specific) control policies over a variety of objectives and allow inference over which policy to use at any given time. For example, the system can incorporate two different policies for actuated motions, one for walking and another one for jogging, or, it can provide one policy to account for voluntary motions and another for involuntary (passive) body responses. We switch control policies probabilistically, (optionally) conditioned on contact feedback, e.

Pose inference in our framework takes the form of a Particle Filter implemented by Algorithm 1 with the motion (lines $3-9$, Section 4.2.2), likelihood (line 12, Section 4.2.3) and noise (lines 5 and 10, Section 4.2.4) models explained next.

4.2.1 Model State Space

Model state $\mathbf{x}$ encodes the information we wish to track over time and that we need to estimate from the video for every frame in the sequence. It consists of the parameterization of the dynamic pose of the character and the state of the controller. We use a controller model that can switch between control policies and so, consequently, the state of the controller keeps track of which policy is currently in effect. Formally, the *model state* is represented by a vector $\mathbf{x} = [\mathbf{q}, \dot{\mathbf{q}}, \upsilon]$, where $\mathbf{q} \in \mathbb{R}^{31}$ is the kinematic pose of the character, $\dot{\mathbf{q}} \in \mathbb{R}^{31}$ is the time derivative of the kinematic pose (pose velocity), and υ is a discrete identifier designating the control

Algorithm 1 Update particle set for the next frame

```
Input: Weighted particle set $\{\mathbf{x}_t^{(i)}, w_t^{(i)}\}_{i=1}^N$ for frame $t$ and geometry of the scene $\mathscr{G}$ (note $\mathbf{x}_t^{(i)} = [\mathbf{q}_t^{(i)}, \dot{\mathbf{q}}_t^{(i)}, \upsilon_t^{(i)}]$)
Output: Weighted particle set $\{\mathbf{x}_{t+1}^{(i)}, w_{t+1}^{(i)}\}_{i=1}^N$ for frame $t+1$
1: $\{\tilde{\mathbf{x}}_t^{(i)}, \frac{1}{N}\}_{i=1}^N :=$ resample($\{\mathbf{x}_t^{(i)}, w_t^{(i)}\}_{i=1}^N$)    // resample particle set
2: for $i := 1$ to $N$ do
3:     if $\tilde{\upsilon}_t^{(i)} = A$ then
4:         $\mathbf{q}_d := \pi^A([\tilde{\mathbf{q}}_t^{(i)}, \tilde{\dot{\mathbf{q}}}_t^{(i)}])$    // predict desired pose
5:         $\mathbf{q}_d \mathrel{+}= \eta_d$    // add noise: $\eta_d \sim \mathscr{N}(\mathbf{0}, \Sigma_d)$
6:     else
7:         $\mathbf{q}_d := \emptyset$
8:     end if
9:     $[\mathbf{q}_{t+1}^{(i)}, \dot{\mathbf{q}}_{t+1}^{(i)}, e] :=$ simulate($[\tilde{\mathbf{q}}_t^{(i)}, \tilde{\dot{\mathbf{q}}}_t^{(i)}], \mathbf{q}_d, \mathscr{G}$)
10:    $[\mathbf{q}_{t+1}^{(i)}, \dot{\mathbf{q}}_{t+1}^{(i)}] \mathrel{+}= \eta_s$    // add noise: $\eta_s \sim \mathscr{N}(\mathbf{0}, \Sigma_s)$
11:    $\upsilon_{t+1}^{(i)} \sim p(\upsilon_{t+1}^{(i)} | \tilde{\upsilon}_t^{(i)}, e)$    // sample next policy
12:    $w_{t+1}^{(i)} := p(\mathbf{y}_{t+1} | [\mathbf{q}_{t+1}^{(i)}, \dot{\mathbf{q}}_{t+1}^{(i)}])$    // apply image likelihood
13: end for
```

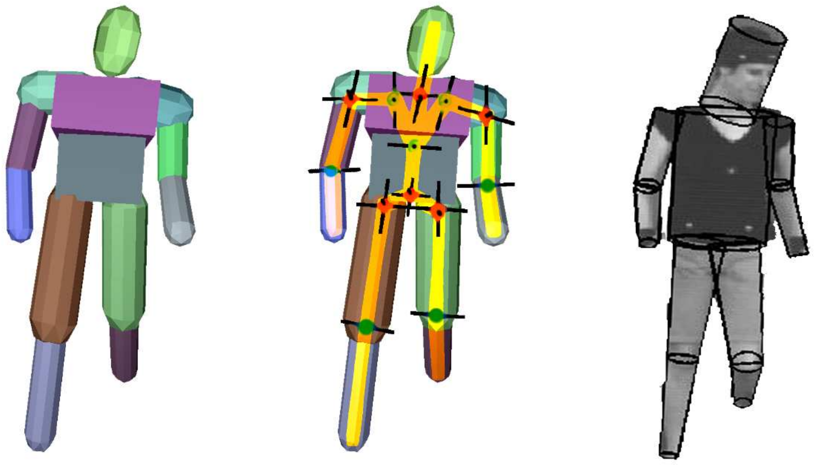


Figure 4.4: **Body and Shape Model.** 31 degree-of-freedom (DOF) character model with simulation shapes of the parts (left), the joints and skeletal structure (middle) and the image shapes in an image projection (right). Most joints have 3 DOFs, except for the knee and elbow joints (1 DOF), spine joint and the clavicle joints (2 DOFs) and the root joint (6 DOFs).

policy in use.

We use a body model for the simulated character that consists of 13 rigid parts and that has a total of 31 degrees of freedom (DOFs), as illustrated in Figure 4.4. Parts have associated mass properties and shapes that are used for simulation and evaluation of pose likelihoods, as discussed later. We set this model up manually, as described in Chapter 3. According to Section 3.5.1, we collect joint DOF values encoding the kinematic configuration of the body into $\mathbf{q}$ in order to describe the *kinematic pose* of the character. We also define the *dynamic pose* $[\mathbf{q}, \dot{\mathbf{q}}]$ to describe the dynamic state of the character, consisting of the kinematic configuration of the body and the speeds with which the body parts translate and rotate. We consider a given pose *invalid* if it causes self-penetration of body parts and/or penetration with the environment (detected by the collision detection library within the simulator), or if the joint DOF values are outside of the valid ranges. These constraints on the kinematic pose allow us to reject invalid model states early in the filtering process.

Control policy identifier υ identifies the policy function to be used for generating next pose proposals for control. The policy function $\pi^{\upsilon} : [\mathbf{q}, \dot{\mathbf{q}}] \mapsto \mathbf{q}_d$ determines the next desired (intended) kinematic pose $\mathbf{q}_d$ that the character attempts to reach from $[\mathbf{q}, \dot{\mathbf{q}}]$ during simulation and is typically obtained by sampling from a kinematic motion prior associated with the policy function. We implement two control policies, an *active* motion policy (A) for actuated motions, where $\mathbf{q}_d$ is obtained from kernel regression on training motion capture data, and a *passive* motion policy (P) for unactuated motions, where no particular desired pose is proposed and, consequently, no actuation forces are applied during simulation. Consequently, $\upsilon \in \{A, P\}$ is binary (see Section 4.2.2.1 and Figure 4.5 for an illustration).

4.2.2 Motion Model and Control Loop

In order to realize pose tracking, we need to encode how model states evolve between frames. Motion models characterize this evolution probabilistically (see Section 2.1.5.2).

We describe state evolution in a generative way by prescribing a computational procedure for drawing samples for new model states given a current state. Sampling from our physics-based motion model is realized by executing the control loop. For every state hypothesis $\mathbf{x} = [\mathbf{q}, \dot{\mathbf{q}}, \upsilon]$ at one frame[1], one loop iteration is taken to produce a hypothesis at the next frame, as illustrated in Figure 4.3 and Algorithm 1.

Control Loop. The sampling procedure uses the current control policy function π^{υ} to propose a next desired kinematic pose $\mathbf{q}_d = \pi^{\upsilon}([\mathbf{q}, \dot{\mathbf{q}}])$ for the character to approach (*decision-making process*, see Section 4.2.2.1). *Physical dynamics* (see Section 4.2.2.3) moves the character towards the desired pose and outputs the resulting dynamic pose as the next pose for the character at the next frame. During simulation, the current dynamic pose $[\mathbf{q}, \dot{\mathbf{q}}]$ is guided towards $\mathbf{q}_d$, considering biomechanical and environment constraints, dictated by the scene geometry $\mathscr{G}$. The guidance is realized through application of appropriate actuation forces $\mathbf{F}^{actuation}$ on the body of the character, generated by the low-level motion control process (see Section 4.2.2.2) from $[\mathbf{q}, \dot{\mathbf{q}}]$ and $\mathbf{q}_d$ using inverse dynamics. In case no desired kinematic pose was proposed by the policy, $\mathbf{F}^{actuation} = \mathbf{0}$, no actuation forces are generated and the character is let move passively during simulation. As a simpler alternative to inverse dynamics, the low-level motion control process could generate actuation forces directly by *e.g.*, a *proportional-derivative servo* [206].

Feedback. In order to optionally allow the control process to react to "external events" that took place during simulation, we record contact event feedback information, e, from the simulator and use it in the decision process to help choose the control policy for the next frame (see Section 4.2.2.1). We currently restrict ourselves to modeling reactions to unanticipated heavy impacts (*e.g.*, as in Figure 4.18) that are unlikely to be represented well in the training motion capture set. Hence, our feedback information consists of only

[1]Where unnecessary, for clarity of notation, we omit subscripts for frame identity and superscripts for hypothesis identity.

	$p(\upsilon_{f+1} = A \mid \upsilon_f, e = 0)$					
	0.95	0.9	0.75	0.5	0.25	0.0
Error (in mm)	32.1	30.2	35.0	37.1	38.3	70.8

Table 4.1: **Tracking Errors As A Function Of $p(\upsilon_{t+1} = A \mid \upsilon_t, e = 0)$ (L1 Walking).** For details on the error metric see Section 4.3.

a binary indicator variable recording whether the body has collided with the environment, detected when a relative velocity at a body contact exceeds a threshold of 1 m/s. This information is extracted from the contact information provided by the simulator (see Section 3.4 and Section 3.4.2).

Randomness. Our control policy functions are fully deterministic such that the desired poses they output depend only on the current pose of the character. In order to model many possible evolutions of the motion from the current pose (*e.g.*, the character may want to step forward from a neutral pose with either a right foot or a left foot), we add the noise Σ_d to desired poses before they are followed by the low-level motion control process (see Section 4.2.4). Note that all pose sequences produced by sampling from the control loop satisfy the equations of motion of the underlying physical model and are thus physically plausible. However, different sampling passes produce different pose sequences. This randomness comes from two facts: (1) control policies are selected by a stochastic process and (2) we add noise to desired poses generated by our policy functions.

4.2.2.1 Decision-making Process In Control

The decision-making component in the control loop implements a process that selects actions for the low-level motion control. The process consists of (1) applying the current control policy to propose a next intended kinematic pose $\mathbf{q}_d$ to be corrected by simulation and (2) determining which policy the current policy should switch to after the simulation completes, utilizing the contact event feedback information from the simulation.

We switch policies by a stochastic process in which the new policy υ_{t+1} is sampled from simple $p(\upsilon_{t+1} \mid \upsilon_t, e)$ distributions that do not take pose or velocity information into account. In practice, we assume that $p(\upsilon_{t+1} = A \mid \upsilon_t = A, e = 0) = p(\upsilon_{t+1} = A \mid \upsilon_t = P, e = 0)$ and estimate the value of $p(\upsilon_{t+1} = A \mid \upsilon_t, e = 0) = 1 - p(\upsilon_{t+1} = P \mid \upsilon_t, e = 0) = 0.9$ using cross validation. The behavior of the tracker as a function of $p(\upsilon_{t+1} = A \mid \upsilon_t = A, e = 0)$ is illustrated in Table 4.1. The value of $p(\upsilon_{t+1} = A \mid \upsilon_t = A, e = 1) = p(\upsilon_{t+1} = A \mid \upsilon_t = P, e = 1)$ is set by hand as in our data impacts of desired magnitude happen very infrequently and hence learning (even using cross-validation) is inconclusive. Motivated by [215], we let $p(\upsilon_{t+1} = A \mid \upsilon_t, e = 1) = 0$.

Passive Motion (P). This policy applies no actuation forces, as if the character was unconscious. As a result, no $\mathbf{q}_d$ is generated and no actuation takes place when the policy is in effect. The purpose of passive motion policy is to account for unmodeled dynamics in the active motion policy and it should typically be activated for short periods of time or when the body is in the free fall.

Active Motion (A). Our active motion-capture based policy generates desired kinematic poses so that the proposed motion would look similar to training motion capture. We take an exemplar based approach resembling [75, 155, 215] and extend it to work with dynamic poses. To that end, we first form a database of observed

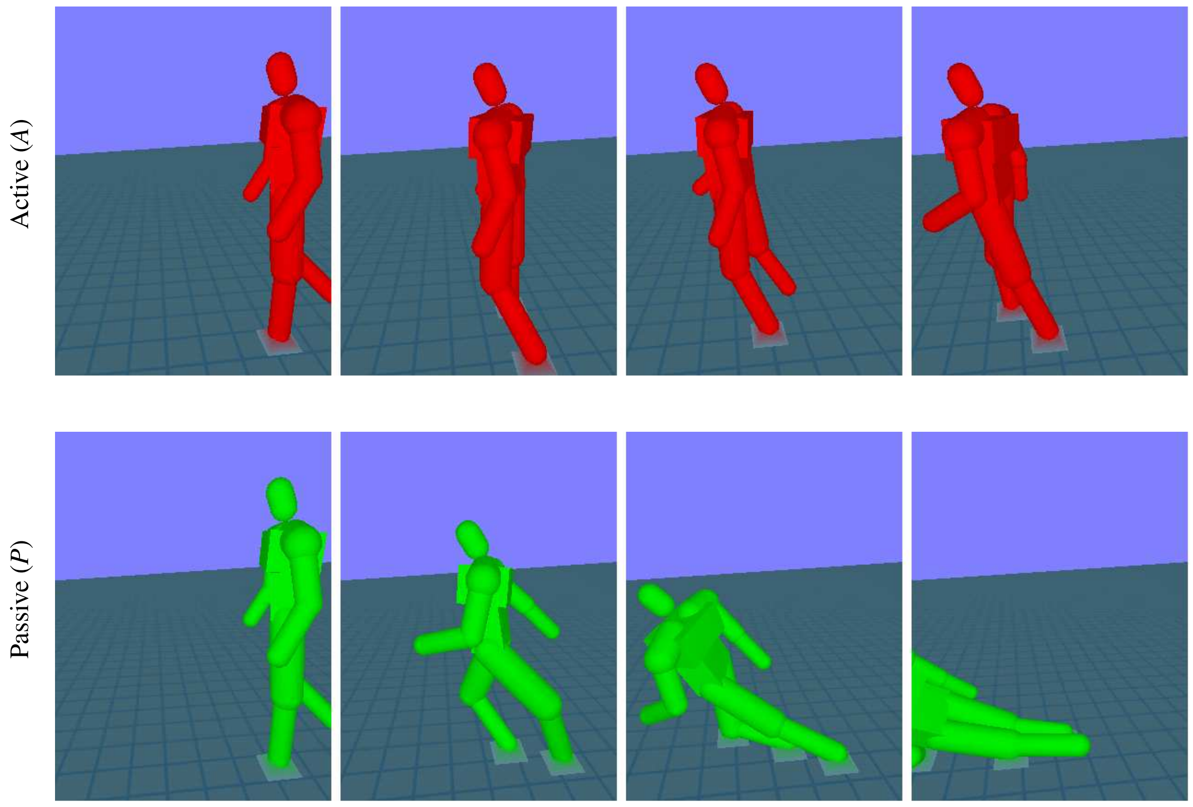


Figure 4.5: **Control Policies.** Predictions made by the control loop from a given initial dynamic pose (top and bottom left) for the time duration of 2 seconds. The top row shows poses generated by control with the active motion policy, the bottom row shows the poses generated by the passive policy.

input-output pairs (from training motion capture data) between a dynamic pose at frame t and a kinematic pose at frame $t+1$, $\{[\mathbf{q}_t^*, \dot{\mathbf{q}}_t^*], \mathbf{q}_{t+1}^*\}_{t=1}^n$. For pose invariance to global position and heading, corresponding degrees of freedom are removed from $\mathbf{q}_t^*$ and $\dot{\mathbf{q}}_t^*$.

Given this database, that can span training data from multiple subjects and activities, and a new query dynamic pose $[\mathbf{q}, \dot{\mathbf{q}}]$, our objective is to determine the next intended kinematic pose $\mathbf{q}_d$. We formulate this objective as in [155] using a k nearest neighbors (k-NN) kernel regression method, where a set of similar prototypes/exemplars to the query point $[\mathbf{q}, \dot{\mathbf{q}}]$ is first found in the database and then $\mathbf{q}_d$ is obtained by weighted averaging over their corresponding outputs; the weights are set proportional to the similarity of the prototype/exemplar to the query point. This inference can be formally written as,

$$\mathbf{q}_d = \frac{1}{Z_K} \sum_{[\mathbf{q}_t^*, \dot{\mathbf{q}}_t^*] \in neighborhood[\mathbf{q}, \dot{\mathbf{q}}]} K(d_t([\mathbf{q}_t^*, \dot{\mathbf{q}}_t^*], [\mathbf{q}, \dot{\mathbf{q}}])) \cdot \mathbf{q}_{t+1}^*, \tag{4.4}$$

where Z_K is a normalizing constant, $d_t([\mathbf{q}_t^*, \dot{\mathbf{q}}_t^*], [\mathbf{q}, \dot{\mathbf{q}}])$ is the similarity measure and K is the *kernel* function that determines the weight falloff as a function of distance from the query point.

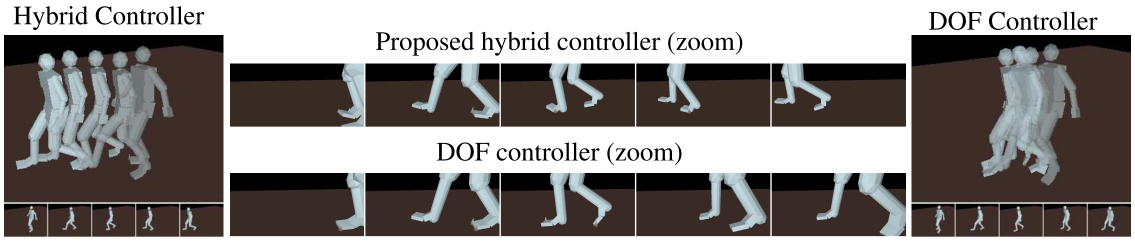


Figure 4.6: **Locomotion.** Comparisons of motions generated by a joint DOF tracking controller based on Kokkevis *et al.*[87] (right) and our hybrid controller (left) when the controllers follow the same trajectories specified by the motion capture data. Differences in simulated dynamics from the captured dynamics result in impacts between the right foot and the ground plane (see the middle frame of the zoomed bottom animation). The impacts prevent the joint DOF tracking controller from performing the motion correctly, resulting in the undesired step backwards, making the character stay to the right of the camera view. Our hybrid controller on the other hand (see the zoomed top animation) is more robust to such artifacts, allowing the body to faithfully follow the desired motion.

We use a similarity measure that is a linear combination of positional and velocity information,

$$d_t([\mathbf{q}_t^*, \dot{\mathbf{q}}_t^*], [\mathbf{q}, \dot{\mathbf{q}}]) = w \cdot d_M(\mathbf{q}, \mathbf{q}_t^*) + (1-w) \cdot d_M(\dot{\mathbf{q}}, \dot{\mathbf{q}}_t^*), \tag{4.5}$$

where $d_M(\cdot)$ denotes a Mahalanobis distance between $\mathbf{q}$ and $\mathbf{q}_t^*$, and $\dot{\mathbf{q}}$ and $\dot{\mathbf{q}}_t^*$, respectively with covariance matrices learned from the training data, $\{\mathbf{q}_t^*\}_{t=1}^n$ and $\{\dot{\mathbf{q}}_t^*\}_{t=1}^n$; the value of $w = 0.9$ accounts for the relative weighting of the two terms and is determined empirically using cross-validation. For the kernel function, we use a simple Gaussian, $K = \mathcal{N}(\mathbf{0}, \sigma)$, with empirically determined variance σ^2.

The method discussed above can be interpreted as a form of a more traditional kinematic prior learned from a database of motion-capture exemplars. While we opted for a simple and robust approach with kernel regression, other regression methods can be used in this context; for example, Gaussian Processes Regression [143] or Conditional Mixture of Experts [23, 167]. The former is closely related to kernel regression, but in addition produces the measure of uncertainty for the prediction; the latter allows for multi-modal predictions. Because we are conditioning on both the kinematics and velocity information, the multi-modality does not seem to be as abundant as with pure kinematic models [23, 167]. Furthermore, as with traditional kinematic motion priors, it is reasonable to assume that the underlying degrees of freedom are much lower than those encoded by the full kinematic state. With that in mind, low-dimensional motion priors (*e.g.*, Latent Variable Gaussian Process Latent Variable Models [185, 183, 186] or Mixture of Factor Analyzers [105]) are likely to facilitate more efficient inference methods. The use of such latent variable models in this context remains future work.

4.2.2.2 Low-level Motion Control

The low-level motion control component computes actuation forces for the character to move it from the current dynamic pose $[\mathbf{q}, \dot{\mathbf{q}}]$ towards the desired kinematic pose $\mathbf{q}_d$ in simulation. Because $\mathbf{q}_d$ is generated using a statistical model, kernel regression, the pose is not guaranteed to be free of self and world penetrations. Consequently, if used as a prediction for tracking, the motion reconstruction may suffer from artifacts. In our model, low-level motion control together with physical simulation is responsible for fixing such predictions so that the penetrations will not occur. Low-level motion control and simulation essentially correct $\mathbf{q}_d$, by finding a new next pose for the character that is close to $\mathbf{q}_d$, can be physically reached from the current pose and use that pose as a prediction for the pose tracking in the next frame.

Overview. Following Section 3.6, we formulate low-level motion control as solving for actuation forces $\mathbf{F}^{actuation}$ that satisfy control/actuation laws. We define these laws as Lagrange multiplier-based constraints on $\mathbf{q}$ (see Section 3.2.4) and $\dot{\mathbf{q}}$ (see Section 3.2.3) from Section 3.6 and Section 3.5, considering the desired pose $\mathbf{q}_d$ to be approached. Each constraint in this set is defined as an equality or inequality equation with a softness constant determining the tradeoff between the amount of the constraint force that is allowed to act at the constraint and the exact satisfaction of the constraint (see Section 3.2.2.6). Magnitudes of the constraint forces are bounded to account for limits of the true human body.

Control Laws. We set up the control laws similarly to the method of Kokkevis *et al.*[87] and to the general actuation approach explained in Section 3.6. We use the desired pose as the encoding of the action from Equation (3.189), where $\vartheta = \emptyset$ and the values for α and $\mathbf{s}$ are set manually and are shared by all actions and use the constraints from Section 3.6.1 to actuate the body. Our control in this chapter differs from the general method and the method of Kokkevis *et al.*in a few aspects:

- With respect to the method of Kokkevis *et al.*, we only actuate angular DOFs of the character so that the trajectory traced by the root part results from interactions of the feet with the environment. As such, we do not apply forces on the root part directly.

  We still actuate the orientation of the root part, by applying "magic" torques on the root. This strategy facilitates balance of the body but is not physically correct because the applied torques are not generated by the joints in the body and the orientation of the root can change regardless of the support from the rest of the body. We use this strategy to enable longer-time predictions required by some of our experiments[2].

- With respect to our general actuation method from Section 3.6, we dynamically adjust targets for control based on the geometry of the environment and dynamics of the character so that the body is not forced to move "against" the ground.

  The control from Section 3.6 assumes the desired poses in actions are compliant with the character model and the environment such that the simple joint DOF tracking method from Equation (3.178) produces intended behavior of the character. However, because our desired poses come from the kinematic prior, the desired poses may not only match the simulated dynamics but can also violate geometric constraints, by *e.g.*, requesting the feet to be actuated down through the ground, causing problems. We illustrate one such a problem in Figure 4.6 (right) and (bottom). Consider the case where the desired kinematic pose $\mathbf{q}_d$ is infeasible (*e.g.*, causing penetration with the environment). Actuation towards the pose causes undesired impacts during simulation, which affects the motion adversely. For

[2]Maintaining balance exactly is not critical, given we use the hybrid motion estimation approach from Section 4.2.4.2 that does not guarantee exact physical plausibility.

example, impacts at the end of the walking cycle (see the impact of the right foot in the middle frame of Figure 4.6 (bottom)) forces the character to step back instead of forward.

To address this problem, we adjust the targets for low-level motion control so that the impacts are avoided. Rather than using inverse kinematics and the knowledge of the environment to correct the representation of the desired pose, we use inverse dynamics to solve for desired velocity of the character that will avoid the impact and track this velocity using motors in simulation.

Formulation. We propose a hybrid constraint-based control method (see Figure 4.7) that can track both desired joint angle trajectories as well as trajectories of selected points on the surface geometry of the body that we generally refer to as markers. We use this controller for tracking the desired positions of toes (computed with respect to the desired kinematic pose $\mathbf{q}_d$ using forward kinematics) that we adjust in order to avoid penetrations with the environment. Consequently, our markers are attached to the locations of the toes.

Following the approach from Section 3.6, our aim is to stabilize the current positions of markers and the values of joint angles toward the control targets. Similarly to the method from Section 3.2.4, we formalize these tracking objectives as constraints on the velocities

$$\dot{\mathbf{z}}^j = -\alpha_{marker} \cdot (\mathbf{z}^j - \mathbf{z}_d^j) \tag{4.6}$$

$$\dot{\mathbf{q}}^k = -\alpha_{dof} \cdot (\mathbf{q}^k - \mathbf{q}_d^k), \tag{4.7}$$

where $\mathbf{z}^j$ are the locations of the tracked toe markers j attached to the body, $\mathbf{z}_d^j$ are their corresponding corrected desired locations, k are the tracked angular DOFs and $\alpha_{marker} > 0, \alpha_{dof} > 0$ are parameters[3] that determine how fast the controller should approach the desired values. These parameters are equivalent to the position stabilization constants from Section 3.2.4.

Implementation. Ideally, we would like to submit the tracking objectives from Equation (4.6) and Equation (4.7) as constraints for inverse dynamics and use them to compute $\mathbf{F}^{actuation}$. However, in our constraint model, these objectives can not be satisfied directly. The marker tracking objectives prescribe desired values for the marker velocities $\dot{\mathbf{z}}^j$ in the global frame and, consequently, could be satisfied by changing the velocity of the root segment through "magic" forces, resulting in an undesired motion.

To avoid this problem, we satisfy these objectives using two inverse dynamics passes. We use the first inverse dynamics pass to reformulate the objectives as constraints over velocities, where we compute desired velocities for the character that realize the tracking of both the desired joint angle values as well as the desired marker positions. In the second inverse dynamics pass, we solve for actuation forces to track the computed desired velocities in simulation. When related to the actuation method from Section 3.6.1, the purpose of the first inverse dynamics pass is to produce velocity constraints for low-level motion control that are more informed than the constraints due to Equation (3.178). The second inverse dynamics pass just solves for the actuation forces to satisfy the velocity constraints from the first pass.

Implementation: First Pass. We first compute desired velocities $\dot{\mathbf{q}}_d$ for the character that satisfy the original

[3]We manually set $\alpha_{marker} = 0.5$ and $\alpha_{dof} = 0.2$ so that the controller can replay training motions in simulation.

tracking objectives. Towards that end, we first augment our objective set by

$$\dot{\mathbf{q}}^{root} = (\dot{\mathbf{q}}^{root})_t \tag{4.8}$$

$$\mathbf{q}^i \geq \mathbf{q}^i_{min}, \mathbf{q}^i \leq \mathbf{q}^i_{max}, \tag{4.9}$$

where *root* refers to the linear and angular DOFs of the root part, $(\dot{\mathbf{q}}^{root})_t$ are the current DOF values at the root, i iterates over the remaining angular DOFs in the body and $\mathbf{q}^i_{min}$ and $\mathbf{q}^i_{max}$ are joint angle limits. The objective in Equation (4.8) fixes the velocity of the root part such that the actuation of the body due to the tracking objectives in Equation (4.6) and Equation (4.7) can not be realized by directly actuating the root. We then use the augmented objectives as constraints for first-order inverse dynamics from Section 3.7 to solve for angular velocities $\dot{\mathbf{q}}_d$ of the character that satisfy the objectives.

Implementation: Second Pass. We use the computed desired velocities $\dot{\mathbf{q}}_d$ as targets for the actuation of the character. The desired velocities produce the velocity constraints

$$\dot{\mathbf{q}} = \dot{\mathbf{q}}_d \tag{4.10}$$

that can be encoded for inverse dynamics similarly to the constraints from Section 3.6.1. We use the second inverse dynamics pass to compute the actuation forces $\mathbf{F}^{actuation}$ to stabilize the above velocity constraints in simulation. We summarize our low-level motion control algorithm in Algorithm 2.

Algorithm 2 Compute actuation forces

```
Input: Current dynamic pose $[\mathbf{q}, \dot{\mathbf{q}}]$, desired pose $\mathbf{q}_d$ and geometry of the scene $\mathscr{G}$
Output: Actuation forces $\mathbf{F}^{actuation}$
1: Formulate control laws from Equation (4.6), Equation (4.7), Equation (4.8) and Equation (4.9)
2: Use inverse dynamics to solve for the desired pose velocity $\dot{\mathbf{q}}_d$ that satisfies the control laws
3: Use inverse dynamics to solve for actuation forces $\mathbf{F}^{actuation}$ to track $\dot{\mathbf{q}}_d$
```

4.2.2.3 Physical Dynamics

Physical dynamics updates the dynamic pose $[\mathbf{q}, \dot{\mathbf{q}}]$ forward in time for the time duration of one frame, Δt seconds, following Newtonian equations of motion with actuation forces $\mathbf{F}^{actuation}$. We approximate the dynamics using simulation with the simulation model from Chapter 3. The collision detection from the simulator is used to check for body penetrations.

4.2.3 Likelihood Model

The likelihood function measures how well a particular model state hypothesis explains image observations (see Figure 4.8). We first define likelihoods for kinematic poses and then extend the approach to handle dynamic poses by considering velocity information.

Kinematic Poses. For a *kinematic pose* $\mathbf{q}_t$, we employ a relatively generic likelihood model $p(\mathbf{I}_t|\mathbf{q}_t)$ from [14] that tries to maximize the similarity between the projection of the hypothesized model and the observed silhouette extracted from the image $\mathbf{I}_t$ (see Figure 1.12 and Figure 4.8).

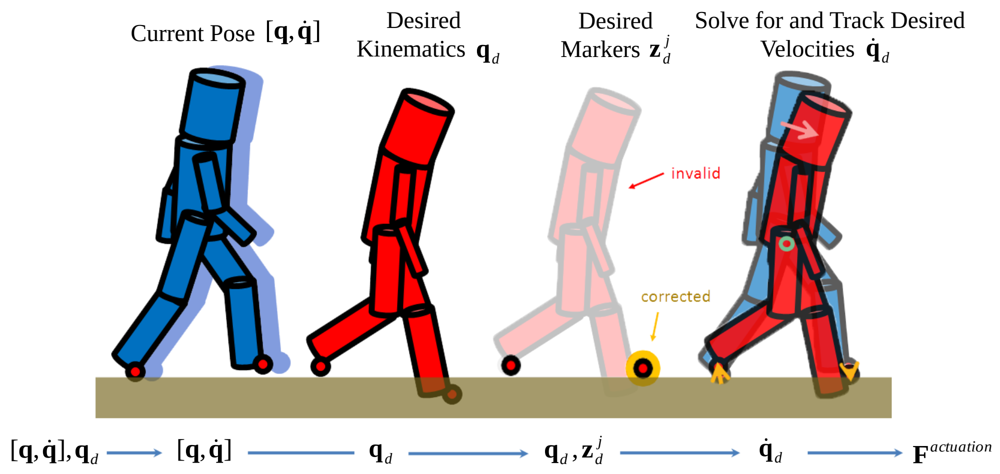

Current Pose $[\mathbf{q}, \dot{\mathbf{q}}]$
Desired Kinematics $\mathbf{q}_d$
Desired Markers $\mathbf{z}_d^j$
Solve for and Track Desired Velocities $\dot{\mathbf{q}}_d$
invalid
corrected
$[\mathbf{q}, \dot{\mathbf{q}}], \mathbf{q}_d \rightarrow [\mathbf{q}, \dot{\mathbf{q}}] \rightarrow \mathbf{q}_d \rightarrow \mathbf{q}_d, \mathbf{z}_d^j \rightarrow \dot{\mathbf{q}}_d \rightarrow \mathbf{F}^{actuation}$

Figure 4.7: **Low-level Motion Control.** The input kinematic pose $\mathbf{q}$ determines the positions $\mathbf{z}^j$ of markers on the feet (1), the desired kinematic pose $\mathbf{q}_d$ defines their desired positions $\mathbf{z}_d^j$ (2). The desired positions are adjusted to prevent penetration with the ground (3) and constraints on the marker world space velocities $\dot{\mathbf{z}}^j$ and relative joint DOF velocities $\dot{\mathbf{q}}^k$ are formed. Constraints are solved for desired velocities $\dot{\mathbf{q}}_d$, to be followed by the character during simulation, using inverse dynamics. The velocities are followed by applying actuation forces $\mathbf{F}^{actuation}$, computed by another inverse dynamics pass.

Assuming that $\mathbf{M}(x,y)$ is the binary image foreground silhouette at the frame t (obtained from the image $\mathbf{I}_t$ through background subtraction), and $\hat{\mathbf{M}}_t(x,y)$ is the binary silhouette obtained by projecting our model under the pose $\mathbf{q}_t$, we are interested in maximizing the overlap between the two binary silhouette images,

$$\mathbf{C}_t = \sum_{(x,y)\in|\mathbf{I}_t|} \mathbf{M}(x,y)\hat{\mathbf{M}}_f(x,y), \tag{4.11}$$

while at the same time minimizing the non-overlapping regions:

$$\mathbf{A}_t = \sum_{(x,y)\in|\mathbf{I}_t|} \mathbf{M}_t(x,y)(1-\hat{\mathbf{M}}_t(x,y)), \tag{4.12}$$

$$\mathbf{B}_t = \sum_{(x,y)\in|\mathbf{I}_t|} \hat{\mathbf{M}}_t(x,y)(1-\mathbf{M}_t(x,y)). \tag{4.13}$$

The likelihood that can achieve this objective can be formulated as follows,

$$p(\mathbf{I}_t|\mathbf{q}_t) \propto \prod_{views} \exp\left(-\frac{\mathbf{A}_t}{\mathbf{A}_t+\mathbf{C}_t} - \frac{\mathbf{B}_t}{\mathbf{B}_t+\mathbf{C}_t}\right). \tag{4.14}$$

A slightly more general form of this likelihood was defined in [14]. Alternatively, a very similar likelihood can be defined using an XOR operator as in [15].

Dynamic Pose. For a *dynamic pose* $[\mathbf{q}_t, \dot{\mathbf{q}}_t]$, we need to consider information extracted from both the current and the next frame so that velocity information could be implicitly measured and compared against $\dot{\mathbf{q}}_t$. Towards this end, we set up the *coupled observation* $\mathbf{y}_t = [\mathbf{I}_t, \mathbf{I}_{t+1}]$ and define the likelihood of the *dynamic pose*

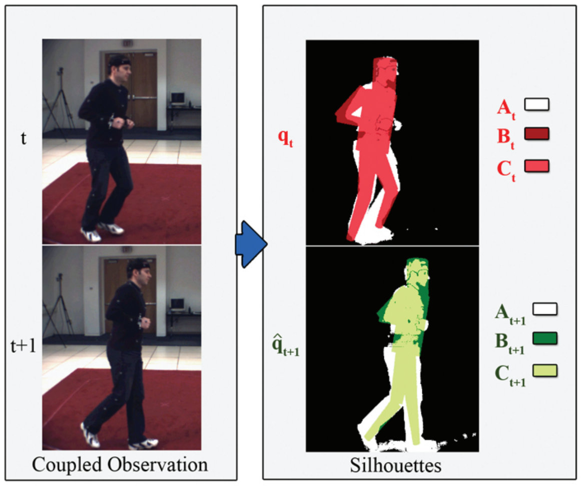


Figure 4.8: **Image Likelihood.** The *coupled observation* $\mathbf{y}_t$, consisting of two consecutive frames $\mathbf{I}_t$ (upper left) and $\mathbf{I}_{t+1}$ (lower left), explains the dynamic pose $[\mathbf{q}_t, \dot{\mathbf{q}}_t]$ well (see Equation (4.15)) if silhouettes at the frame t fit the kinematic pose $\mathbf{q}_t$ (red pose) and silhouettes at the frame $t+1$ fit the kinematic pose $\hat{\mathbf{q}}_{t+1}$ (green pose), where $\hat{\mathbf{q}}_{t+1} = \mathbf{q}_t + \Delta t \cdot \dot{\mathbf{q}}_t$ is predicted from the kinematic pose $\mathbf{q}_t$ and the velocities $\dot{\mathbf{q}}_t$ using the constant velocity model. For the definition of $\mathbf{A}_t$, $\mathbf{B}_t$ and $\mathbf{C}_t$ see Equation (4.12), Equation (4.13) and Equation (4.11) respectively.

with respect to the coupled observation as a weighted product of two kinematic likelihoods

$$p(\mathbf{y}_t|[\mathbf{q}_t, \dot{\mathbf{q}}_t]) \propto p(\mathbf{I}_t|\mathbf{q}_t)p(\mathbf{I}_{t+1}|\hat{\mathbf{q}}_{t+1}), \tag{4.15}$$

where $\hat{\mathbf{q}}_{t+1} = \mathbf{q}_t + \Delta t \cdot \dot{\mathbf{q}}_t$ is the estimate of the kinematic state/pose at the next frame. Alternatively, one can formulate a likelihood measure that explicitly computes the velocity information [27] (*e.g.*, using optical flow) and compares it to the corresponding velocity components of the state vector.

The remaining portions of our state $\mathbf{x}_t$, such as the control policy, are inherently unobservable and are assumed to have uniform probability with respect to the likelihood function[4]. Hence, we define our likelihood function $p(\mathbf{y}_t|\mathbf{x}_t)$ as $p(\mathbf{y}_t|\mathbf{x}_t) = p(\mathbf{y}_t|[\mathbf{q}_t, \dot{\mathbf{q}}_t])$.

4.2.4 Motion Reconstruction

We reconstruct the motion in the video as a sequence of poses of the simulated character that match image observations. We use a Particle Filter implementation with the motion and likelihood models from above as an inference mechanism (see Algorithm 1). This PF implementation, discussed in Section 4.1.3, produces the distributions $p(\mathbf{x}_t|\mathbf{y}_{1:t})$ for the individual frames $t = 1, \dots, T$ and we need to choose a *representative pose* from each such a distribution to provide an interpretation of the entire sequence. Our goal is to choose representative poses such that the resulting pose sequence is physically plausible.

[4]The resulting dual-counting of observations, only makes the unnormalized likelihood more peaked, and can formally be handled as in [27].

Finding such a sequence is difficult, because Particle Filter, as any other Bayesian filtering method, requires noise (or a diffusion process) to be present in the motion model to account for disturbances and subtleties of a particular activity being tracked (see Figure 4.2). As such, the control loop from Section 4.2.2 needs to incorporate noise.

In kinematic pose trackers, such as [13, 15, 46, 33, 112, 157], it is customary to employ motion models that perform deterministic prediction of the pose and then directly add noise to the predicted state (see (a) in Figure 4.2). Most often, the deterministic motion model is an identity function $\mathbf{q}_t = \mathbf{q}_{t-1}$ and state diffusion is implemented by replacing the predicted $\mathbf{q}_t$ with a sample from $\mathcal{N}(\mathbf{q}_t, \Sigma)$, where the covariance Σ controls the amount of noise added. Adding noise to the state, in our case, results in the loss of physical plausibility (see (b) in Figure 4.2), because the recovered pose trajectory, in general, cannot be simulated by the physical model exactly. Consequently, to output poses that are physically plausible, we need to take a different approach that only adds noise to the desired pose (see (c) in Figure 4.2).

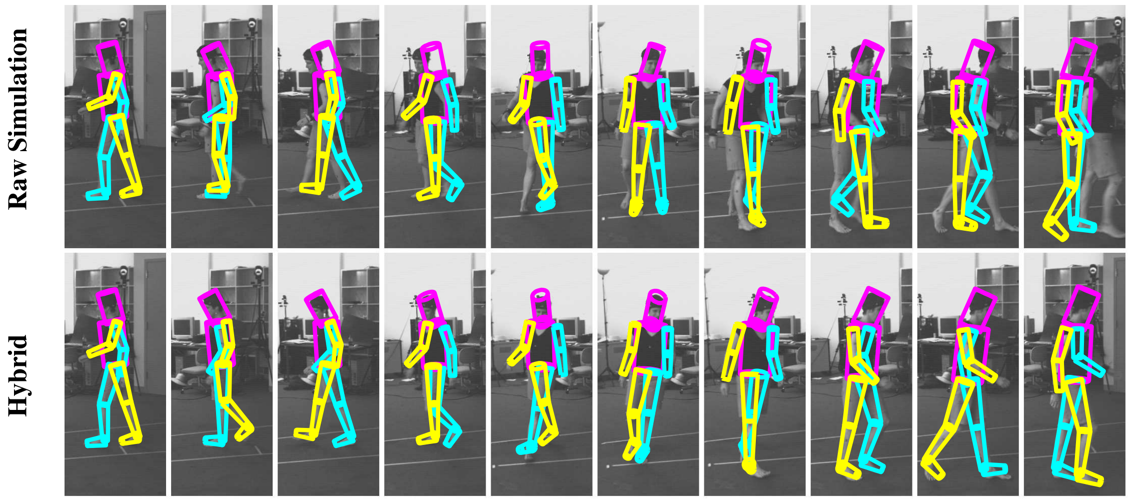


Figure 4.9: **Raw Simulation Results (Image Projections, Multi-view L1 Walking).** Illustration of motion reconstruction results on the multi-view L1 walking sequence obtained with our two physics-based motion estimation methods. The odd rows (white character) show a motion reconstruction using raw physical simulation. The method produces a physically plausible motion interpretation that does not feature artifacts, but fails to fit images accurately due to the high-dimensionality of the optimization space and the parameterization of desired motion. Results produced by the hybrid physics-based method are visualized in the even rows (red character). The hybrid method can achieve a better fit to the observations (by *e.g.*, translating the torso in a non-physical way), but can not guarantee the estimated pose sequence can be simulated by the physical model. As such, it may include artifacts. In both cases, we use an extended body model with extra parts for the feet so that the character can properly push off the ground while walking, as required by the first method that produces raw simulation results. See text in Section 4.2.4 for additional information on the two methods.

Next, we discuss general options for modeling of the noise in tracking and the corresponding consequences on the motion reconstruction and physical plausibility. We study two possible methods for motion reconstruction: a method that is guaranteed to output physically plausible interpretations (see Section 4.2.4.1 and (c) in Figure 4.2) and a method that biases interpretations towards the plausible results (see Section 4.2.4.2

and (b) in Figure 4.2). For each method, we illustrate what *noise model* we use, how we integrate the noise model with the control loop and how we select the representative poses from the posteriors. We also discuss implications on the ability of the method to generalize the estimated motion to new situations.

4.2.4.1 Physically Plausible Estimation by Raw Simulation

We first consider a motion estimation strategy that outputs physically plausible pose sequences (see (c) in Figure 4.2). These sequences are defined as raw results of physical simulations of the character model from an initial state $\mathbf{x}_1$. Consequently, the poses $\mathbf{q}_1, \ldots, \mathbf{q}_T$ in the reconstructed sequences must be generated from a single particle, as a history of poses that this particle represented as it was being updated from the initial frame until the last frame by the Particle Filter. In other words, the pose sequence must correspond to a trajectory of a single particle in the pose space, where poses are produced by simulation.

Noise Model. In order to produce particle trajectories in the Particle Filter that are entirely generated by simulation and control in the control loop (see Section 4.2.2), noise can only be added to the desired kinematic pose, $\mathbf{q}_d$, before the body actuation and dynamic simulation takes place (see Algorithm 1, line 5, and (c) in Figure 4.2).

Towards that end, we use a noise model, where we add $\mathscr{N}(\mathbf{0}, \Sigma_d)$ to the desired pose generated by deterministic control policy functions and we *add no noise to the predicted state*. We choose Σ_d to be proportional to the maximum expected difference in kinematic pose between the prediction by simulation and the true observation as in [13, 46]. Because we do not wish any noise to be added to the pose generated by the simulation, we let $\Sigma_s = \mathbf{0}$ for Algorithm 1 in this case.

Representative Poses. In order to obtain an interpretation of the entire motion, we consider full state histories of the particles from the last frame and report the pose sequence produced by the *most likely (MAP)* history, such that the cumulative likelihood value for the poses in the sequence is maximized. The returned pose sequence is physically plausible, because it records an evolution of a single particle (single simulation) from an initial state (see Figure 4.9 (top) and Figure 4.10). See Section 4.1.3 on how we represent state histories in particles.

Adaptation to New Environments. Because we are able to reconstruct a trajectory-tracking controller using this method, we can apply the estimated controller to simulated characters in new situations. Abilities of these generalizations are, however, limited because the control feedback is restricted to only responses to impacts. Furthermore, because desired poses for control are prescribed for every frame in the sequence, the produced motion is not able to deviate from the desired motion in simulation much (a well known fact for trajectory-tracking controllers).

4.2.4.2 Hybrid Estimation

While, in principle, physically plausible motion estimation is a desired alternative, it assumes that (1) our physical simulation is rich enough to generate the motion we are observing exactly and (2) that the likelihood model is strong enough to ensure that the physical state of the system can be inferred accurately at all times using the PF.

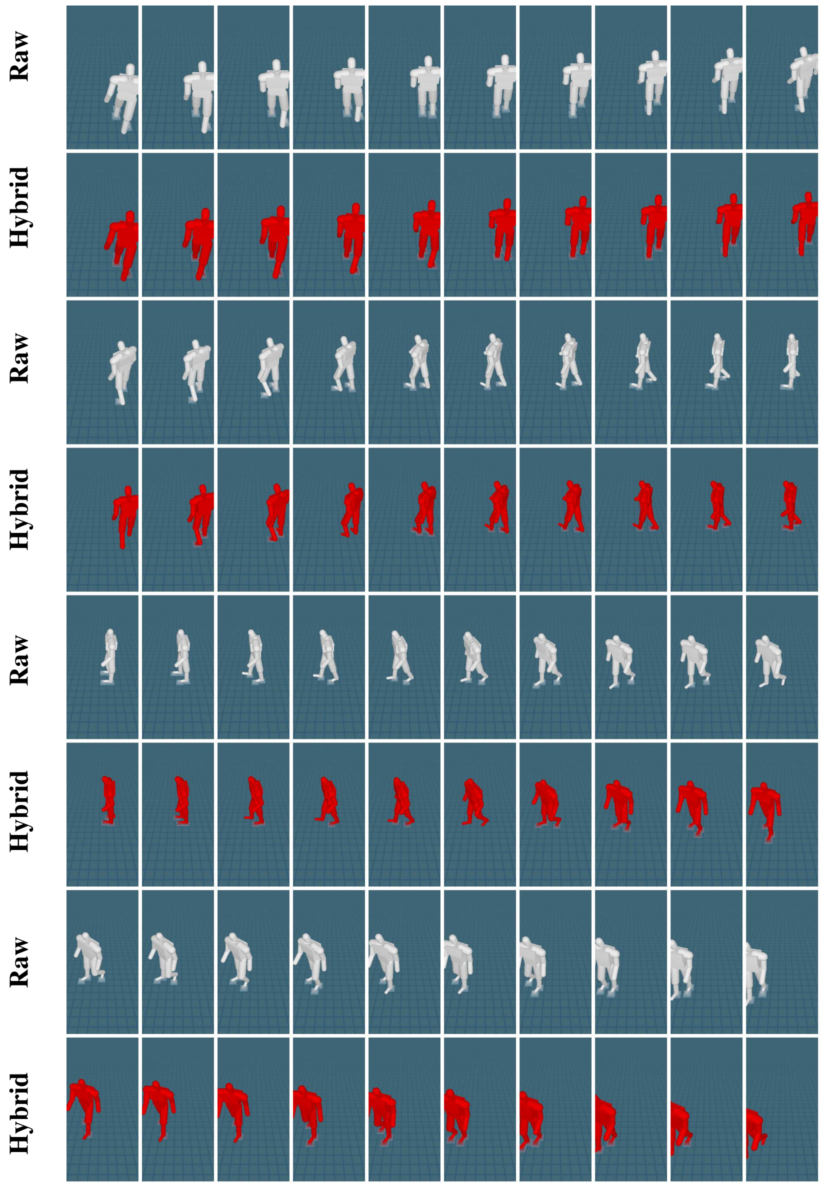


Figure 4.10: **Raw Simulation Results (Pose Renderings from A Different View, Multi-view L1 Walking).** Visualization of motion reconstruction results on the multi-view L1 walking sequence obtained with our physics-based motion estimation methods, rendered from a view not used for the estimation. Motion of the white character was produced using raw physical simulation. Motion of the red character was reconstructed using the hybrid physics-based method. In both the cases, we used an extended body model with extra parts to represent feet. See text in Section 4.2.4 for additional information on the methods.

In practice, neither of the two assumptions holds exactly, which causes problems when we use PF as an inference mechanism. In particular, often, because of the noise in the image observations, the moment when the foot hits the ground cannot be detected well using the implemented likelihood model and, consequently,

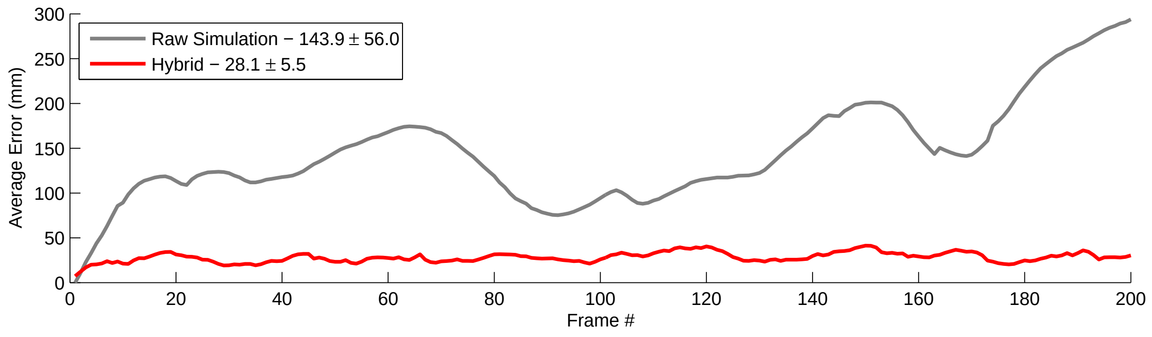


Figure 4.11: **Raw Simulation Results (Tracking Errors, Multi-view L1 Walking).** Pose reconstruction errors for results on the multi-view L1 walking sequence obtained with our physics-based motion estimation methods and an extended body model with extra parts to represent feet. See text in Section 4.2.4 for additional information on the methods.

PF can easily overfit the desired pose to the observations in the current frame (see Figure 4.9, top). Specifically, because the PF proceeds strictly incrementally and never reoptimizes particles from previous frames given future observations, it is not able to correct mistakes it has made earlier. As a result, foot sometimes hits the ground too early (or too late) requiring the model to either take a longer or shorter step later to compensate and keep up with subsequent image observations, or to lag behind for the duration of the walk cycle (until the foot support transfers). Both options result in suboptimal performance and require using too many particles to circumvent (each particle provides one potential motion interpretation for the entire sequence and there are only as many possible interpretations as particles in the last frame). On the other hand, these errors could often be fixed by either translating the whole model or the toe(s) in the model by a small amount along the floor. Such moves fix the tracking errors locally, but they are physically invalid and are explicitly disallowed by the physical model. If allowed, artifacts emerge and *e.g.*, foot will start sliding on the floor in the reconstruction. As a practical compromise, we propose to take a hybrid approach, where we add noise to both desired poses and the poses produced by the simulation (see (b) in Figure 4.2).

Noise Model. In this approach, we add a fraction Σ_d of the required total noise Σ to the desired pose (see Algorithm 1, line 5) and a fraction Σ_s to the predicted state (see Algorithm 1, line 10), where the standard deviation in Σ is set to be proportional to the maximum expected difference in state between the prediction by simulation and the true observation and $\Sigma = \rho\Sigma_d + (1-\rho)\Sigma_s$ for a factor ρ.

Representative Poses. Because the addition of noise to the simulated pose loses physical plausibility, we are no longer constrained to report motion interpretations that are generated as histories of single particles. Instead, for each frame t in the sequence, we output the *expected pose*, defined as the mean of the distribution $p(\mathbf{x}_t|\mathbf{y}_{1:t})$ estimated by the PF.

Because each such a reported pose is obtained by averaging poses from the particle distribution (*i.e.*, poses from different simulations) and the poses in the posterior themselves are corrupted with noise, proportional to Σ_s, the reported poses are not guaranteed to honor kinematic constraints exactly and may be temporally

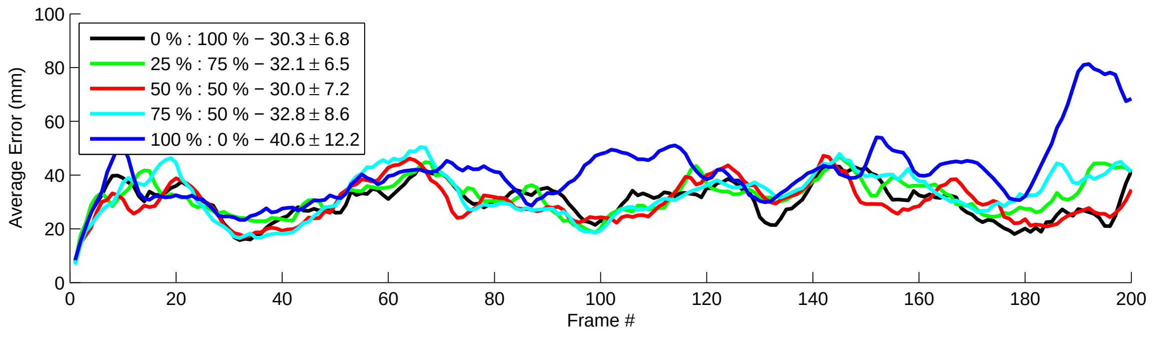


Figure 4.12: **Tracking Errors For Different Noise Distributions (Multi-view L1 Walking).** Effects of different ratios of noise added to desired poses vs to predicted poses on tracking errors. All cases produce similar tracking results, except for the case 100 % - 0 %, when all noise is added only to the desired poses and the tracker can not fix global translation errors until the end of the stride.

incoherent. However, they can provide a good fit to the image observations locally and explain individual images well.

As illustrated in Figure 4.9 (bottom), Figure 4.11 and Figure 4.12 (see experiments for the description of the dataset and error metric), this hybrid strategy results in a plausible compromise between physical plausibility and the ability to subdue the confusion that arises from the lack of good image observations and the incremental nature of the inference. It is worth noting that even though the resulting motion obtained by this strategy cannot be guaranteed to be physical simulation from our model, it is *very* strongly biased by the model towards physically plausible solutions, because the amount of noise Σ_s added to the pose is relatively small with respect to Σ_d. We assume this motion estimation strategy for the rest of our experiments.

Adaptation to New Environments. The hybrid method is not able to produce generalizations of the motion reconstructions for new environments directly. The result of the reconstruction is a sequence of poses, not a controller, in this case, and the pose sequence is not necessarily simulated.

4.3 Experiments

We evaluate our method both in terms of its ability to track common human activities from monocular and multi-view video (see Table 4.2) as well as to predict human motion alone.

4.3.1 Data Sets and Performance Metrics

Datasets. In our experiments, we make use of two publicly available datasets [13] and [161], containing synchronized motion capture (Mocap) and video data from multiple cameras. We also collected our own custom monocular sequences that contain no associated motion capture. The use of the synchronized data, in the former datasets, allows us to (1) perform baseline experiments that quantitatively analyze performance,

(2) obtain data for training the motion and noise models and to (3) get reasonable initial poses for the first frame of the sequence from which tracking is initiated. Each public dataset contains disjoint training and testing data, that we use accordingly.

L1. The first public dataset, used in [13], contains a single subject (L1) performing a walking motion with stopping, imaged with 4 grayscale cameras.

S1 - S3. The second public dataset, HUMANEVA-I[161], contains three subjects (S1 to S3) performing a variety of motions (*e.g.*, walking, jogging, boxing) imaged with 7 cameras (we, however, make use of the data from at most 3 color and 1 grayscale cameras for our experiments).

M. The custom sequences contain monocular footage of a subject (M) exhibiting more complex dynamical interactions with the environment like walking up the stairs and jumping of a ledge. The footage was taken by a low-quality stock digital camera for which no ground truth Mocap information is available. The images were captured at 640×480 resolution at 30 frames per second. Due to the low quality of the camera some of the frames were dropped, resulting in a variable frame rates as low as 15 frames per second. Rough camera calibration was extracted directly from sequences. Geometry of the environment (*e.g.*, ground plane, stairs) was built by hand based on the recovered calibration. For each sequence, initial poses and parameters of the body and shape model (limb lengths) were tuned manually. In all cases, the Mocap for training L1 sequence was used to train motion priors for the M sequences.

Motion Reconstruction. For all our tracking experiments in this section, we use the hybrid strategy from Section 4.2.4.2 to estimate distributions over model states for frames in the sequence and select representative poses. For each frame, we report the expected pose of such a distribution.

Performance Metrics. To quantitatively evaluate the performance on standard datasets we make use of the metric employed in [13] and [161], where pose error is computed as an average distance between a set of 15 markers defined at the key joints and end points of the limbs. In 3D, this error has an intuitive interpretation of the average joint distance, in (mm), between the ground truth and a recovered pose (*absolute error*). In our monocular experiments and specific motion model experiments, we use an adaptation of this error that measures the average joint distance with respect to the position of the pelvis (*relative error*) to avoid biases that may arise due to depth ambiguities and to avoid penalizing of other competing motion priors that do not model changes in 3D position and orientation of the body. For tracking experiments, we report the error of the expected pose.

Method Comparison. For comparison with our Physics-based method (**Physics**), we implemented two alternative standard Bayesian filtering approaches, Particle Filter[5] (**PF**) and Annealed Particle Filter[5] with 5 levels of annealing (**APF 5**), each with two priors: a smooth prior (**Smooth**) and a kinematic motion capture exemplar prior (**Mocap**). The kinematic motion capture prior takes the form of

$$\begin{aligned} p(\mathbf{q}_t|\mathbf{q}_{t-1},\dot{\mathbf{q}}_{t-1}) &= \mathcal{N}(\mathbf{q}_d,\Sigma) \\ &= \mathcal{N}(\pi^A([\mathbf{q}_{t-1},\dot{\mathbf{q}}_{t-1}]),\Sigma). \end{aligned} \tag{4.16}$$

[5]We make use of the public implementation by Balan *et al.*[13] available from `http://www.cs.brown.edu/people/alb/`.

Subject	Activity	Source	# Cameras	# Frames	Section
L1	Walking	[13]	4	200	4.3.3
S3	Jogging	[161]	4	200	4.3.4
L1	Walking	[13]	1	200	4.3.5
S3	Jogging	[161]	1	200	4.3.6
M	Stairs	Custom	1	150	4.3.7
M	Jumping	Custom	1	35	4.3.8

Table 4.2: **Sequences And Data Sets Used In Tracking Experiments.**

For uniformity with the implementation of our body actuation, the kinematic motion capture prior does not predict the motion of the root; rather it relies on the sampling covariance of the noise model for the positional degrees of freedom of the root segment. To make the comparison as fair as possible we always use the same number of particles, 250 for multi-view sequences, 1000 for monocular sequences[6], same likelihoods, same noise model and same interpenetration and joint limit constraints in all cases; joint limit constraints are learned from training data.

4.3.2 Motion Prediction with Ground Interaction

We first evaluate the proposed motion model (see Section 4.2.2) alone. The key aspect of our model is the ability to perform accurate physically-plausible predictions of the future state based on the current state estimates. We demonstrate this ability through quantitative comparisons with predictions made by the standard temporal prior models based on stationary linear dynamics (described in Section 4.1.3) and exemplar-based predictions.

Figure 4.14 (right) shows performance of the *smooth* prior (No Prediction, see Equation (4.2)), *constant velocity* prior (see Equation (4.3)), *kinematic motion capture* prior (see Equation (4.16)) and individual predictions based on the two control policies implemented within our physics-based prediction module. For all 5 methods we use 200 frames of motion capture data from the L1 sequence to predict poses from 0.05 to 0.5 seconds ahead. To make sure the experiment results are not biased by the effects of noise in the control loop (see Section 4.2.4), we only use deterministic priors with $\Sigma = \mathbf{0}$ and a single control policy function for the whole sequence in one experiment. We then compare our predictions to the poses observed by motion capture data at corresponding times and report prediction errors using the metric from above.

For short temporal predictions all methods perform well; however, once the predictions are made further into the future, our *active* motion control policy, that filters predictions from the exemplar-based Mocap method, significantly outperforms the competitors. Overall, the active control policy achieves 29 % lower error over the constant velocity prior (averaged over the range of prediction times from 0.05 to 0.5 seconds).

Figure 4.14 (left) shows the effect of noise on the predictions. For a fixed prediction time of 0.25 seconds, a zero mean Gaussian noise is added to each of the initial ground truth dynamic poses before the prediction is made. The performance is then measured as a function of the noise variance. While performance of the

[6]In APF, we use 250 particles for each annealing layer in multi-view sequences and 1000 particles for each layer in monocular sequences.

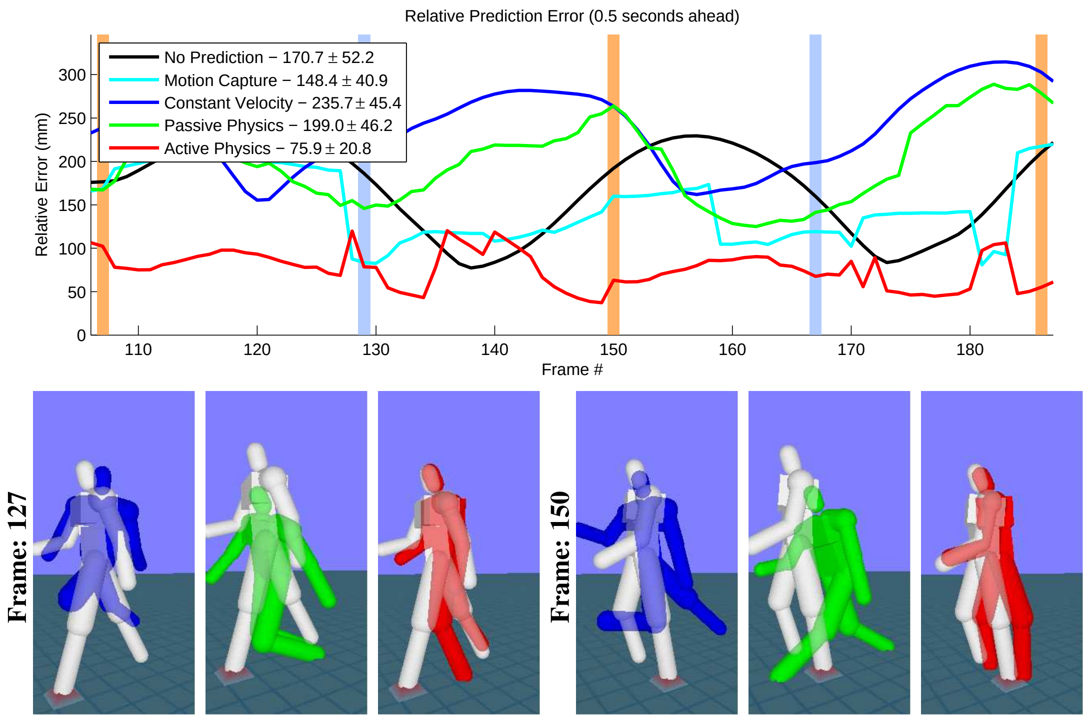


Figure 4.13: **Prediction Error.** Errors in predictions (0.5 seconds ahead) are analyzed as a function of one walking cycle. Vertical bars illustrate different phases of walking motion: light blue – foot pushes on the ground, light orange – change in the direction of the arm swing.

constant velocity prior and passive motion prior degrade with noise, the performance of our active motion prediction stays low and flat.

Notice that in both plots in Figure 4.14 (right) the constant velocity prior performs similarly to the passive motion; intuitively, this performance makes sense because the constant velocity prior is an approximation to the passive motion dynamics that does not account for environment interactions. Because such interactions happen infrequently and we are averaging over 200 frames, the differences between the two methods are not readily observed, but are important at the key instants when they occur (see Figure 4.13). For example, predictions of passive motion model tend to be more accurate when a foot strikes the ground. Consequently, when change in the direction in the arm swing occurs, inertia tends to allow the motion to continue in the initial direction, making passive motion model predictions less accurate.

4.3.3 Tracking with Multiple Views (L1 Walking)

As our first tracking experiment, we analyze the tracking of the multi-view sequence of L1 (see supplemental video from the project website for qualitative analysis). The quantitative results are illustrated in Figure 4.15 (top left). Our method has 63 % lower error than PF and 27 % lower error than APF with smooth prior; 59 % lower error than PF and 18 % lower error than APF with the kinematic motion capture prior. In all cases, our

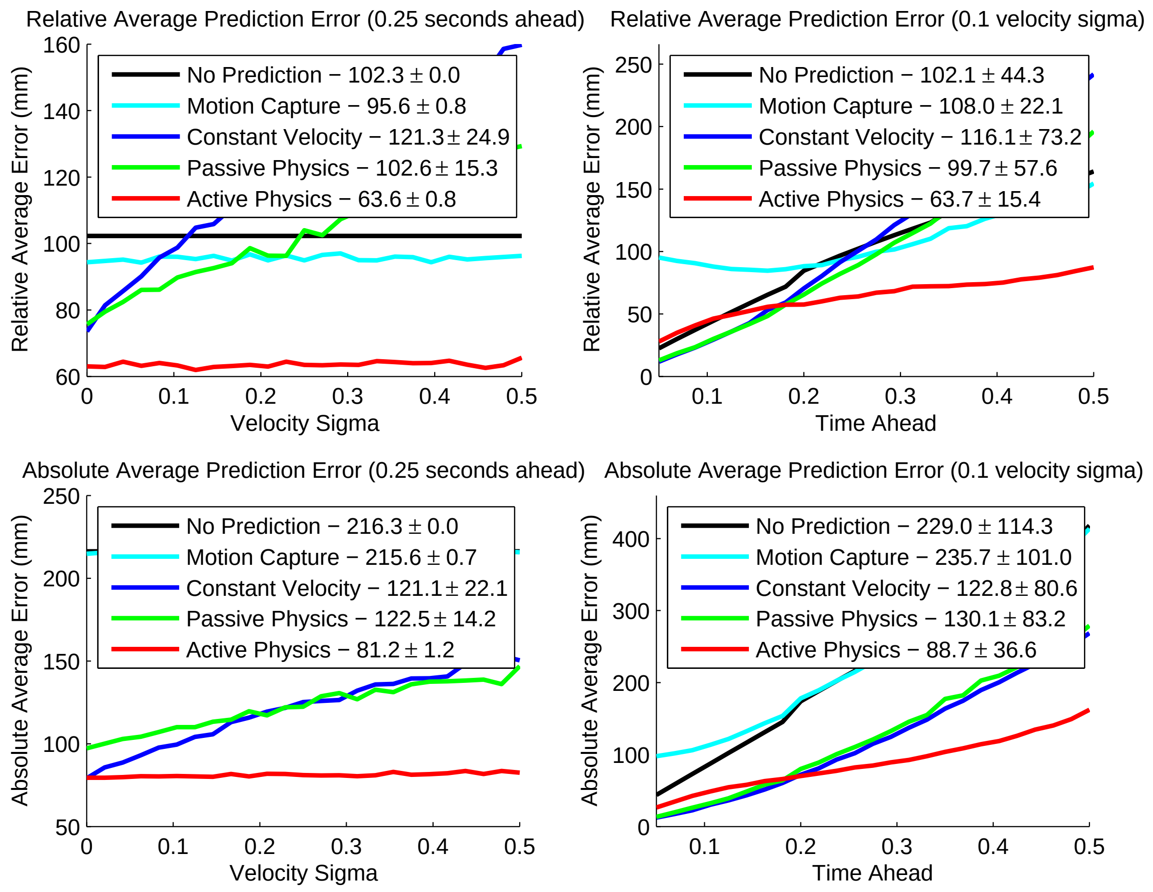


Figure 4.14: **Average Prediction Error.** Illustrated, on the right, is the quantitative evaluation of the 5 different dynamical priors for human motion: smooth prior (No Prediction), motion capture based prior without physics (Motion Capture), constant velocity prior and (separately) the active and passive physics-based priors implemented here. On the left, the performance in the presence of noise added to the initial ground truth poses is explored. For completeness, top row shows the relative error and the bottom illustrates absolute error measure. See text for further details.

	Test: L1 Train: L1	Test: L1 Train: S1-S3
Our method	30.3 (mm)	47.2 (mm)
Our method ([207] likelihood)	33.9 (mm)	52.4 (mm)
Xu and Li [207]	49.0 (mm)	55.3 (mm)
PF (smooth prior, [207] likelihood)	120.7 (mm)	
PF (smooth prior)	80.0 (mm)	
APF 5 (smooth prior, [207] likelihood)	63.5 (mm)	
APF 5 (smooth prior)	42.8 (mm)	

Table 4.3: **Comparison With Other Methods (Multi-view L1 Walk).**

	Mean (std.) in (mm)
Our method	71.5 (19.7)
APF 5 (250 particles/layer)	132.5 (35.5)
APF 5 (500 particles/layer) [104]	111.82 (47.91)
APF 5 GPLVM (500 particles/layer) [104]	99.05 (21.90)
MHT GCMFA [104]	70.13 (21.34)

Table 4.4: **Comparison With Other Methods (Multi-view S3 Jog).**

method also resulted in a considerably lower variance.

We have also tested how the performance of our method degrades with larger training sets that come from other subjects performing similar (walking) motions (see Physics S1-S3 L1). It can be seen that additional training data does not noticeably degrade the performance of our method (only by 0.6 mm on an average), which suggests that our approach is able to scale to larger datasets with multiple subjects. We also test whether or not our approach can generalize, by training on data of subjects from the HUMANEVA-Idataset and running on a different subject, L1, from the dataset of [13] (Physics S1-S3). The results are encouraging in that we can still achieve reasonable performance that has lower error than PF with either of the two alternative priors, but performs marginally worse than APF with smooth prior. Comparison with APF using kinematic motion capture prior is not meaningful in this case, because it is trained using subject specific motion data of L1. Our experiments tend to indicate that our approach can generalize within observed classes of motions given sufficient amount of training data (for generalizations to execution in different environments see Section 4.3.7 and Section 4.3.8).

We also compare the performance of our Bayesian tracking method with the physics-based prior to that of [207]. In [207], a more informative kinematic prior model was proposed (as compared to the smooth prior), that explicitly learns correlations between the parts of the body in a coordinated motion (*e.g.*, walking). This prior is then used in the context of a more efficient Rao-Blackwellised Particle Filter (RBPF). In [207], however, a weaker likelihood model was used (which we employed in an earlier variant of this work [193]), so we report the performance with both the types of likelihoods (see Table 4.3). It is also worth mentioning that in [207], 1000 particles were used, instead of 250 here; yet, our method still performs favorably. Furthermore, from the table one can see that while the models that utilize smooth priors tend to be very sensitive to the quality of the likelihoods (gaining over 30 % performance increase with the better likelihoods utilized here), our model is much less sensitive to those aspects.

4.3.4 Tracking with Multiple Views (S3 Jogging)

To illustrate that our method is not limited to motions of any particular type (*e.g.*, walking), we conducted a similar experiment to above but on the jogging sequence of subject S3 from the HUMANEVA-Idataset. All the parameters of the tracker are set as before, except that the prior was trained on jogging sequences of the subject S3 (disjoint of the test set). Performance on sample frames is illustrated in the supplemental video and quantitatively analyzed in Figure 4.15 (bottom left). The proposed model once again achieves the lowest error against all competing methods.

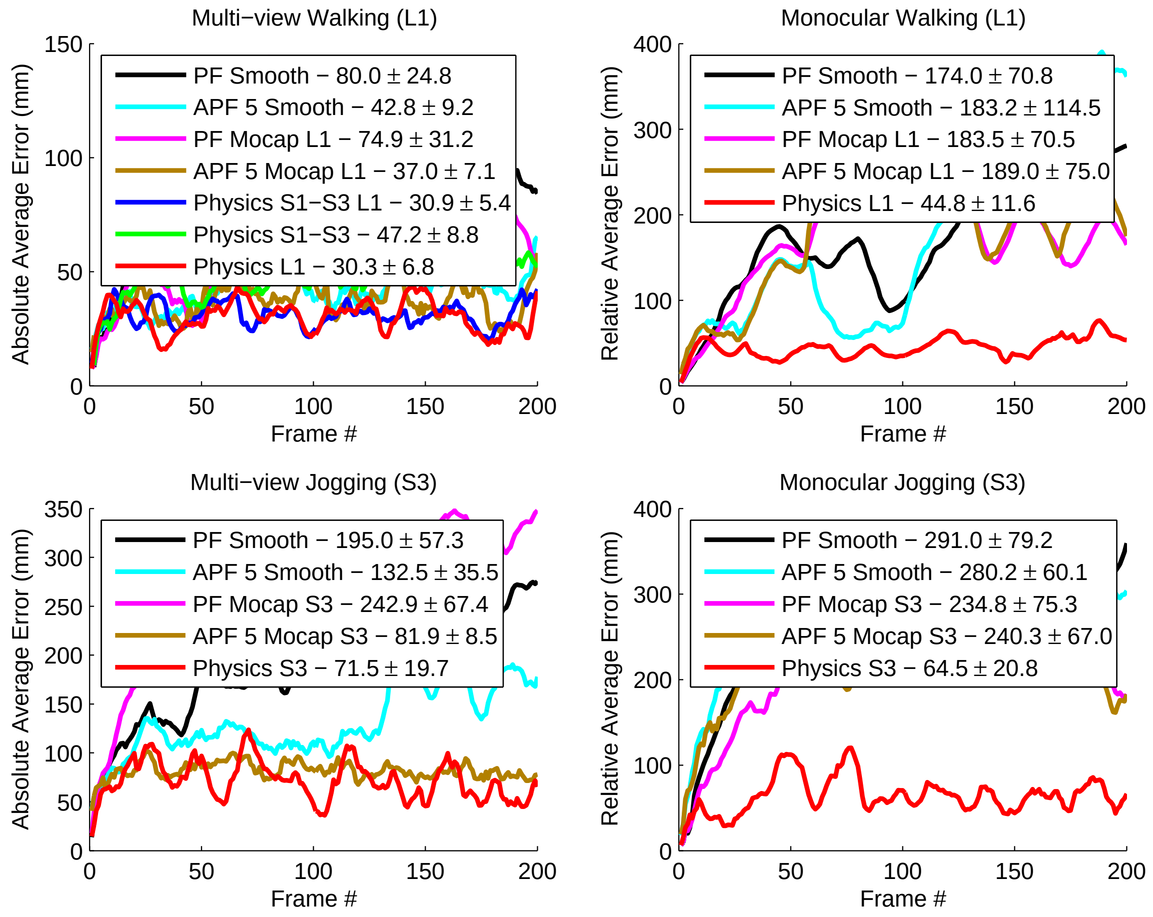


Figure 4.15: **Quantitative Tracking Performance.** Illustration of the performance of the proposed physics-based prior method (Physics) with different testing and training datasets versus standard Particle Filter (PF) and Annealed Particle Filter (APF 5) with 5 layers of annealing. Multi-view tracking performance with 4 cameras and 250 particles is shown on the left, monocular tracking performance with 1000 particles on the right.

We also compare our performance on this sequence to other methods published in the literature (see Table 4.4). In particular we compare the performance, to the results reported in [104], where a Multiple Hypothesis Tracker with a kinematic Globally Coordinated Mixture of Factor Analyzers (MHT GCMFA) prior was presented and compared to an independent implementation of Annealed Particle Filter with 5 layers of annealing with (1) a smooth prior (APF 5) and (2) a kinematic Gaussian Processes Latent Variable Model prior (APF 5 GPLVM). In this case, the comparison may not be direct because we are not certain as to the exact form of the likelihood used in [104]; in addition, [104] uses 500 particles per layer of APF (as opposed to 250 in our implementation). The difference in the number of particles/samples may explain slightly lower performance of our APF implementation (132.5 versus 111.82 (*mm*)). Quantitatively, performance of our method is on par with that of [104] and is more accurate than that of APF 5 GPLVM, despite the fact that we are using half as many samples/particles and a relatively simple kinematic proposal process for $\mathbf{q}_d$. In addition, we expect our method to produce more physically plausible motions.

4.3.5 Monocular Tracking (L1 Walking)

The most significant benefit of our approach is that it can deal with monocular observations. Physical constraints encoded in our prior help to properly place hypotheses and avoid overfitting of monocular image evidence that lacks 3D information (see Figure 4.16 (Physics) and supplemental video); the results from PF and APF on the other hand suffer from these problems, resulting in physically implausible 3D hypotheses (see Figure 4.16 (APF 5) bottom) and lead to more severe problems with local optima (see Figure 4.16 (APF 5) top). Figure 4.16 (Physics) bottom, illustrates the physical plausibility of the recovered 3D poses using our approach. Quantitatively, our model has 74 % lower error than PF and 76 % lower error than APF with smooth prior; 76 % lower error than both PF and APF with kinematic motion capture prior, with a considerably lower (roughly $\frac{1}{6}$) variance (see Figure 4.15 at top right).

4.3.6 Monocular Tracking (S3 Jogging)

Similar results can be seen for the jogging sequence of the subject S3 from the HUMANEVA-Idataset. The results are quantitatively analyzed in Figure 4.15 (bottom right). See supplemental video for the qualitative results. While it may seem that the tracking with a single view performs better than with multiple views, this is not the case. In the monocular case, we only report marker errors with respect to the position of the pelvis, thereby ignoring errors in the global translation of the body.

To be fair, we would like to also acknowledge that the lowest error of 18.9 (mm), with the standard deviation of 7.5 (mm), on this sequence was reported by Urtasun *et al.* in [182]. The method of [182], however, utilizes a very different class of discriminative models. Discriminative models tend to produce quantitatively accurate performance, but result in poses that are extremely noisy over time (no temporal continuity); these models are also difficult to generalize to new motions and/or poses that, in part, result from environment interactions (something we address with our model in the next two experiments).

4.3.7 Monocular Tracking (M Stairs)

A key benefit of the proposed model is its ability to generalize to complex interactions with the environment. In Figure 4.17 and supplemental video, we illustrate the performance of our tracker, trained using level-ground walking of L1, on a new subject walking up a set of stairs. Despite the fact that the prior is trained with clearly very different motion from the one observed, our method is able to successfully track the lower body as it interacts with the stairs in order to support the motion of the person. To our knowledge, no other kinematic prior method is able to illustrate such generalization. Clearly, the knowledge of the environment together with the ability to reason about interactions of the feet with the stairs through physics-based predictions are responsible for the resulting performance.

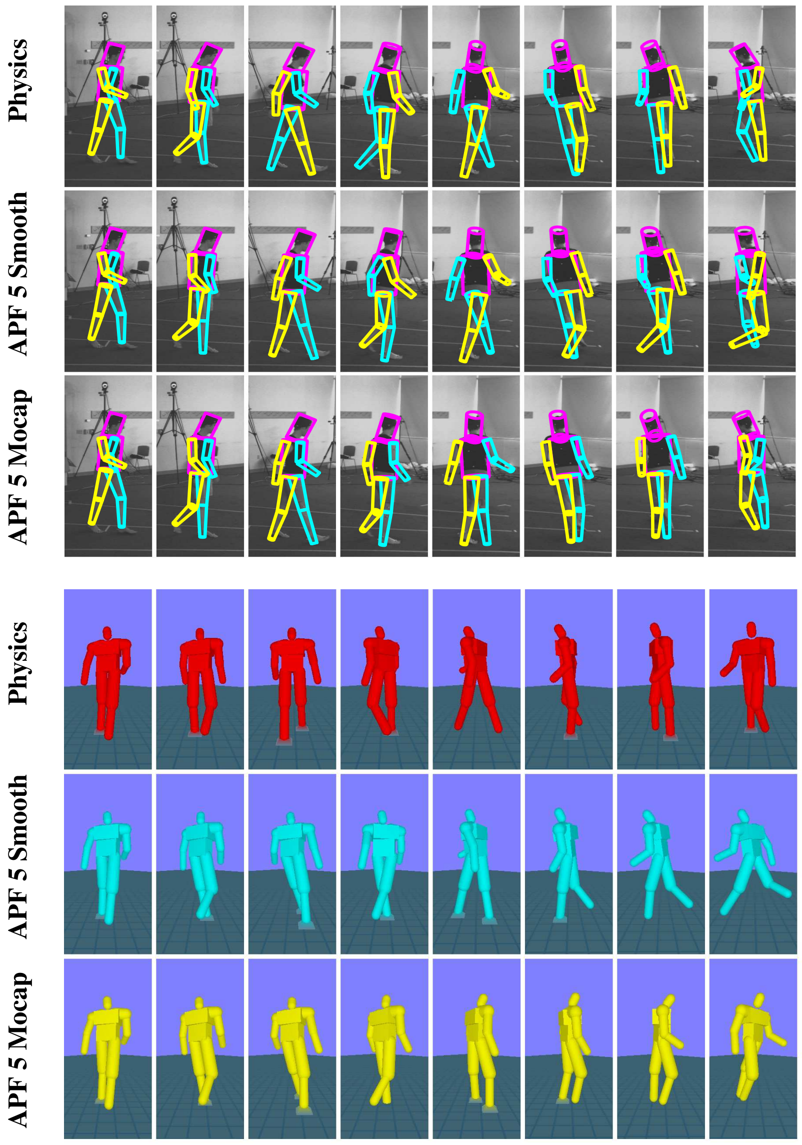


Figure 4.16: **Monocular Tracking (Walking).** Visualization of performance on a monocular walking sequence of subject L1. Illustrated is the performance of the proposed method (Physics) versus the Annealed Particle Filter (APF 5) with the smooth and kinematic motion capture prior; in all cases with 1000 particles. The top row shows projections (into the view used for inference) of the resulting 3D poses at 20-frame increments; the bottom row shows the corresponding rendering of the model in 3D, from a different view, along with the ground contacts. Our method, unlike APF with either prior, does not suffer from out-of-plane rotations, has consistent ground contact pattern and can estimate correct heading of the person that is consistent with the direction of motion. APF, on the other hand, produces poses that tend to drift along the ground plane and face in an opposite direction (APF 5 Mocap). For quantitative evaluation see Figure 4.15 (top right).

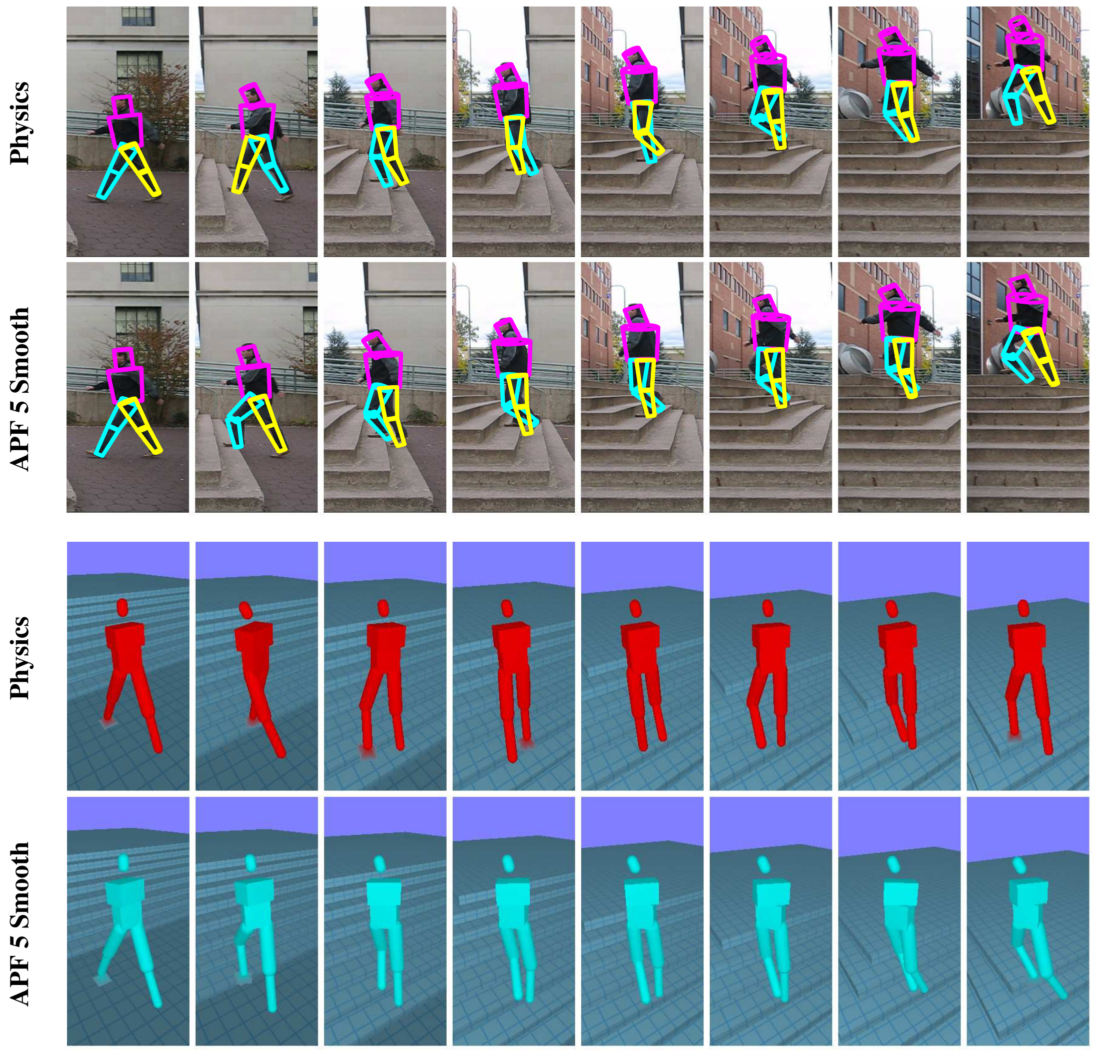


Figure 4.17: **Monocular Tracking (Stairs).** An ability of the proposed model to generalize to complex interactions with the environment is tested. Despite the fact that the prior is trained only using level-ground walking, the tracker still performs well on this more challenging terrain and is able to track the person as they walk up the stairs. Annealed particle filter with the kinematic motion capture prior (APF 5 Mocap), on the other hand, fails to track this sequence completely. As soon as the person approaches the stairs, the kinematic motion capture prior starts producing state hypotheses that penetrate the stairs, which prevents the tracker from further tracking. See text for additional details and discussion.

4.3.8 Monocular Tracking (M Jumping)

In the final experiment (Figure 4.18), we illustrate the performance of the tracker on the fast motion of a person jumping off a ledge. Because we know that this motion is mostly ballistic, we tuned the parameters of the decision process in the control loop to always select the passive motion control policy (for more efficient inference). The physics-based model is able to track the person reasonably well, even though the

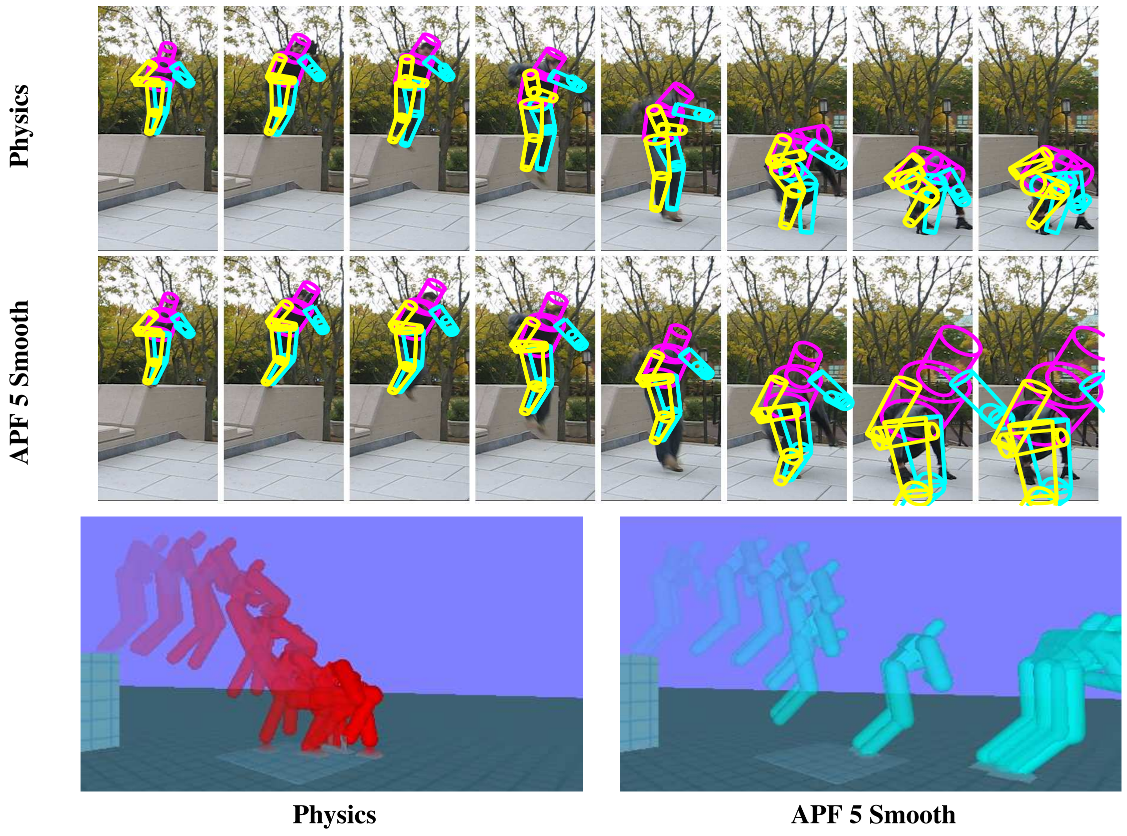


Figure 4.18: **Monocular Tracking (Jump).** We illustrate an ability of the proposed model to generalize to new activities in new environments. The set up for this experiment is similar to Figure 4.17, except no motion capture data is used for training and the model relies on passive motion control policy for predictions. Because the physical effects of the environment are computationally accounted for in our model, the proposed tracker is able to properly predict the motion of the person as the body hits the ground.

footage is of poor quality and the motion is extremely fast, producing large changes in the body pose between frames and motion blur (see supplemental video). Of particular interest is the natural way body crouches as it hits the ground (unlike what happens with more traditional kinematic priors, *e.g.*, Figure 4.18 bottom right). Consequently, the pose changes cannot be predicted by simpler smooth (*e.g.*, a constant-velocity or no-motion) motion prior models, because the noise required to account for the fast motion is simply too large to allow efficient inference. Because this motion was not part of the motion capture training set, comparison with the motion capture based prior is not possible (or meaningful).

4.4 Conclusions

Summary. In this chapter, we have presented a method for incorporating full-body physical simulation with control into a temporal prior within a Particle Filter for video-based pose tracking. Using the simulation and control, we are able to account for non-linear non-stationary dynamics of the human motion and interactions of the body with the environment (*e.g.*, ground contact). To allow tractable inference, we introduce two control policies: an active motion policy, which uses motion-capture data as targets for body actuation using a hybrid controller, and a passive motion policy, which does not compute any actuation forces. Our method can be easily integrated with existing kinematic pose trackers. Our approach can be interpreted as a filter built on top of any statistical motion model within the pose tracker (kernel regression, in this case) that corrects pose predictions from the kinematic model to make them physically plausible. The corrections are realized by actuating the character model from the current pose towards the proposed pose from the kinematic model using simulation and control within the control loop. Using these tools, we illustrate that our approach can better model the dynamical process that underlies human motion. We show that the resulting performance of the tracker is more accurate than the results obtained using standard Bayesian filtering methods such as a Particle Filter (PF) or Annealed Particle Filter (APF) with kinematic priors and the tracker can better generalize for different environments. In order to promote the use of physical models for tracking, we have made the source code of our controllers and the simulation-based prior available on our website.

Discussion. Using physical simulation for correcting predictions in the Particle Filter is effective for pose tracking but has its limitations. The nice properties are the ability of the physical model to generalize and make physically plausible predictions for the character under interactions with the environment. On the other hand, we have to infer a desired pose for the character in every frame of the sequence, which is difficult to do accurately given the observation model and the inference mechanism we use. Consequently, we can not reconstruct motions that are fully physically plausible and we can only bias the pose interpretations to those produced by the simulation.

We conjecture this deficiency comes from the modeling choice of the inference method that is prone to overfitting and the controller model that requires inference over the entire desired pose trajectory. We discuss these aspects in more detail next.

Incremental Inference and Overfitting. First, because the inference by PF proceeds strictly incrementally from one frame to the next frame by one frame increments and never reoptimizes, using only the observations from the current frame to estimate the model state for the frame, the inference is essentially driven by local image observations. As such, the estimation process can easily overfit and estimation errors made earlier in the process can not be fixed later. That is, once a distribution over model states is computed for a particular frame by the PF, it remains fixed, regardless of the information that becomes available in the future. These properties have important implications on pose tracking.

If we restrict pose tracking by PF to the tracking of only the desired pose, so we can obtain a physically plausible motion interpretation of the entire sequence, our method fails to provide a good overall fit to the images and requires too many particles to converge. Consequently, as a practical compromise, we resort to using a hybrid estimation strategy that adds noise to the poses predicted by simulation. The addition of noise breaks physical plausibility in both predictions and motion reconstructions but allows for a better fit to the data (the noise allows to shift the simulated pose in the world arbitrarily in order to better fit the particular observed silhouette). The motion estimation results that we demonstrate in our experiments in this chapter are all produced using this hybrid approach, where *the reconstructed pose trajectories result from simulated* *trajectories and noise.*

High-dimensional Representation of Controller. Second, because we use a trajectory tracking controller model, parameterized by an unstructured sequence of desired poses, a desired pose for the simulated character has to be estimated for every frame in the sequence. In this implementation, we represent desired poses in a high-dimensional full-body space. Consequently, the number of variables to be estimated in each frame is proportional to the number of kinematic DOFs in the body model and the inference over the desired pose trajectory results in a difficult, high-dimensional optimization problem. We rely on training motion capture data to make this inference possible.

Towards Next Chapter. In the next chapter, we introduce a new method that addresses the challenges from above and always produces motion interpretations that are raw results of physical simulations. The new method differs from the PF-based pose tracking in the following key aspects:

- An inference mechanism:

  The new method employs an inference mechanism that is based on direct optimization. The mechanism is still incremental, but reoptimizes estimation results from past frames to reduce the chance of overfitting.

- A controller model:

  The new method replaces the trajectory tracking controller with a state-space controller model that encodes the desired pose trajectory for the character in a sparse way, utilizing an underlying structure of the motion. As a result, there is much fewer variables to be estimated per frame during the optimization. In addition, the assumption of structured motion better constrains the optimization problem (see Section 1.3.2 and Section 1.5.1) and each optimization pass in the incremental process is reinforced by information from many frames.

- A character model:

  The new method makes use of an improved model for the simulated character. The model uses a more elaborate body structure, can generate a wide variety of self-balanced motions in simulation (*e.g.*, walking, jumping, running, gymnastics), employs a proper balancing strategy and does not rely on training motion capture data for simulation.

Chapter 5

Controller Capture

In this chapter, we address shortcomings of our previous method for motion estimation from Chapter 4. Our goal is to improve upon the previous method such that (1) we can provide physically plausible motion interpretations of the observed motions that are raw results of physical simulations of the character model and (2) represent the observed motions in such a way so that they can be directly adapted to new situations. In order to realize this goal, we employ a state-space controller model that parameterizes the desired motion for the character sparsely. Consequently, we can integrate information from many frames when recovering targets for control, which results in improved robustness of estimation, brings the guarantee of physical plausibility and lets the control deviate from the captured poses in simulation in response to perturbations of the environment and character.

We implement our method by following the controller estimation approach outlined in Section 1.5.1. We apply this approach to the state-space controller model from Section 1.4.3.1 that we equip with a balance feedback mechanism. The control is encoded as a sequence of actions. Transitions among actions are triggered on time and on proprioceptive events (*e.g.*, contact). Inference takes the form of optimal control where we optimize (incrementally and/or in a batch) a high-dimensional vector of control parameters and the structure of the controller based on an objective function that compares the resulting simulated motion with input observations. We use a gradient-free genetic optimizer, Covariance Matrix Adaptation (CMA), to estimate the control parameters. We make use of the character model from Chapter 3 for simulation and body actuation.

We illustrate our approach by automatically estimating controllers for a variety of motions directly from monocular video. To decouple errors introduced from tracking in video from errors introduced by the controllers, we also test on reference motion capture data. We show that estimation of controller structure through incremental optimization and refinement leads to controllers that are more stable and that better approximate the reference motion. We demonstrate the power of this approach by capturing sequences of walking, jumping, and gymnastics. We evaluate the results through qualitative and quantitative comparisons to video and motion capture data.

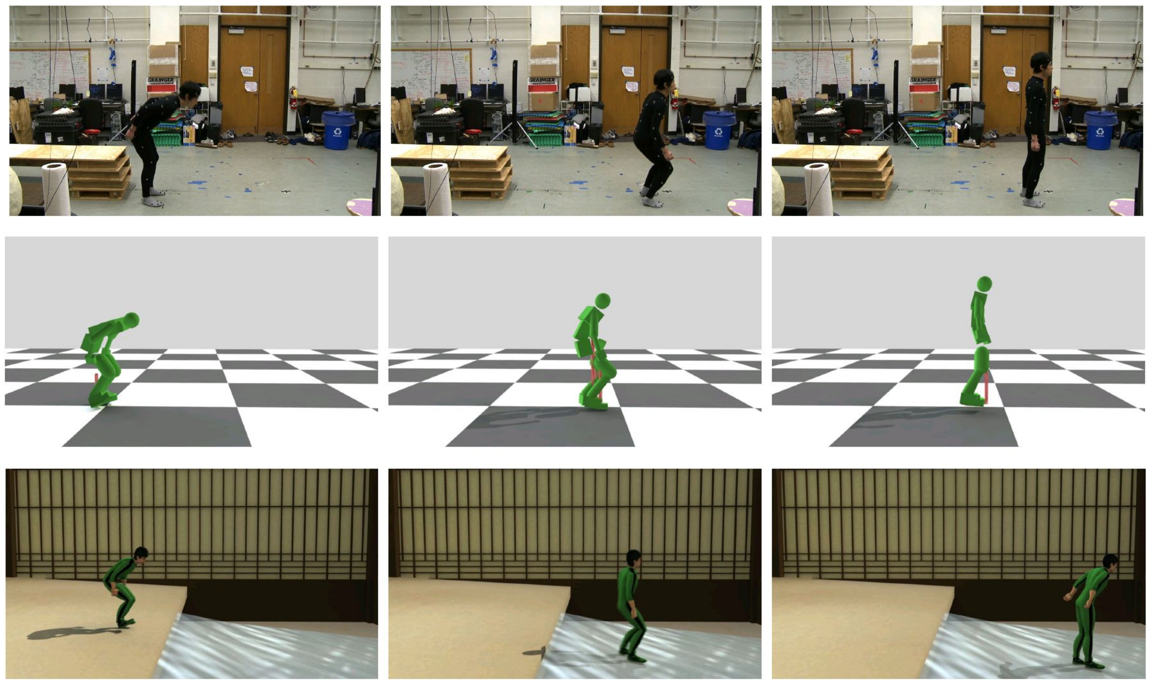


Figure 5.1: **Controller Reconstruction from Video.** We estimate biped controllers from monocular video sequences (top) together with a physics-based responsive character (bottom left) that we can simulate in new environments (bottom right).

5.1 Introduction

We formulate motion estimation as a capture of control mechanisms for a simulated physics-based character model. Our approach consists of a model for an actuated character and an inference mechanism that uses simulation of the character to optimize for the control.

Controller capture can be seen as a generalization of the physics-based particle filtering from Chapter 4, where we replace the trajectory tracking controller within the Particle Filter with a sparse state-space controller from this chapter. In doing so, we are motivated by the fact that the desired pose trajectory for the tracking controller can only be reliably estimated in presence of strong likelihoods. Rather than focusing on employing stronger observation models, we approximate the desired pose trajectory as a sequence of longer-term actions selected by a state machine. We illustrate controller capture by showing how it relates to the previous particle filtering method.

5.1.1 Relation to Physics-based Particle Filtering

Controller capture differs from physics-based particle filtering in the control method, inference and character model.

Control. The method from Chapter 4 uses a controller that can be encoded as a chain of short actions,

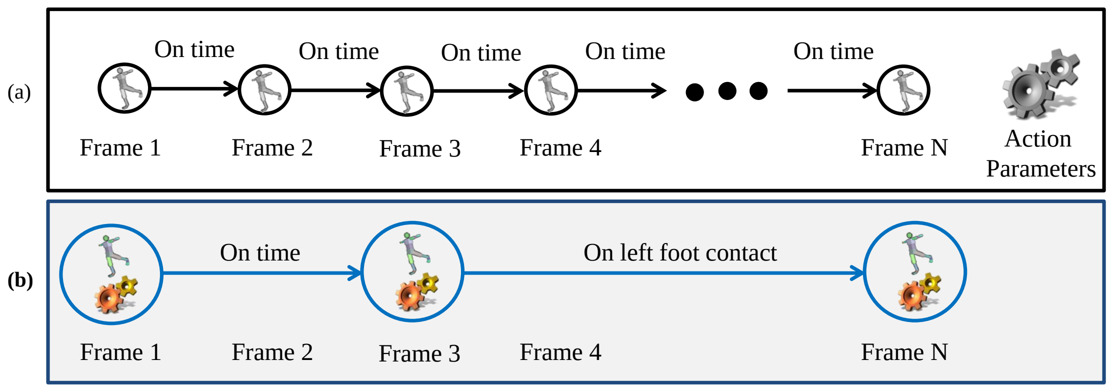


Figure 5.2: **Comparison of Controllers in Controller Capture and Physics-based Particle Filtering.** Physics-based particle filtering (a) uses a trajectory tracking controller that switches actions with every frame and uses the same parameters for "actuation programs" in every frame. Controller capture, on the other hand, uses longer-term actions with own parameters for the actions (b). The parameters encode control and balance laws for the character, as appropriate for the given action. Switching of actions in the machine can be conditioned on distinct physical events that happen in the simulated world, such as a change of left foot contact.

where actions lasts for the time duration of one frame and switch regularly at the end of each frame (see (a) in Figure 5.2). Each of the actions was parameterized as a desired pose that the low-level motion control process attempted to reach and all actions shared the same control parameters to set up the control laws.

In contrast, the controller model that we use in this chapter uses actions that last for many frames (see (b) in Figure 5.2) and the actions can switch either based on time or contact events. The switching process is explicitly encoded by a state machine with states corresponding to actions and transitions encoding the switching rules. Compared to the previous model, control tasks associated with the actions are also more complex. Each action uses its own set of parameters to set up the tasks, in terms of Equation (3.189), and the control tasks also employ an active balancing strategy for the body.

State-space controllers allow concise representation of motion dynamics through a sparse set of desired poses and control parameters, in essence allowing a key-frame-like representation of the original motion. Because the controller structure is sparse, we are able to integrate information locally from multiple (tens of) image frames which induces smoothness in the resulting motion and resolves some of the ambiguities that arise in monocular video-based capture, without requiring a user-in-the-loop.

Inference. The method from Chapter 4 uses a Particle Filter implementation to incrementally recover desired poses for the trajectory controller. The inference proceeds strictly incrementally, is prone to ambiguities and noise in likelihoods and often overfits. We had to resort to adding noise to the poses generated by simulation in order to facilitate tracking (see Section 4.2.4.2).

In controller capture, on the other hand, we recover a controller for the input motion and generate poses in the sequence implicitly, by computing actuation forces for the simulated character using the controller and integrating them over time in simulation. The use of a controller eliminates unrealistic behaviors such as ground or body-part penetrations which are often found in results obtained by typical video-based motion capture approaches and guarantees physically plausible motion reconstructions. We argue that by restricting kinematic motion to instances generated by a control structure (parameters of which we optimize), we can approximate general principles of motion, such as balance and contact, allowing response behaviors to emerge automatically.

In order to estimate a controller for the input motion, we need to recover the structure of the motion (as a state machine) and parameters of the actions and transitions within the machine. We formulate this estimation as an optimization over actions in the controller. This optimization is difficult because the number of actions in the motion can be high for many behaviors and the number of parameters to optimize large. The optimization is realized over both discrete variables, that encode the structure of the controller, in terms of the number of states and types of transitions in the state machine, as well as continuous. The continuous parameters encode the behavior of the controller given its structure. The parameters encode actions within the controller, in terms of Equation (3.189), and encode desired poses in actions, parameters of constraints that realize low-level motion control and parameters of balance laws and the initial pose.

Character Model. Compared to Chapter 4, the controller capture method employs a more elaborate model for the simulated character that can actively balance based on simulation feedback.

5.1.2 Contributions

To our knowledge, ours is the first method to propose a method for recovery of full-body bipedal controllers directly from single-view video. Our method is capable of automatically recovering controllers for a variety of complex, three-dimensional, full-body human motions, including walking, jumping, kicking and gymnastics. The inference is cast as a problem of optimal control, where the goal is to maximize the reward function measuring consistency of the simulated 3D motion with image observations (or motion capture). The sparse nature of our controller is specifically designed to allow estimation of parameters from noisy image observations without overfitting. As a result our method not only addresses the traditional 3D pose tracking problem, but also allows estimation of responsive bipedal characters directly and automatically from video without requiring user intervention or intermediate kinematic motion representation. Recovered controllers also contain a feedback balancing mechanism that allows replay in different environments and under physical perturbations. With respect to prior work in character bipedal control, we make three key contributions: (1) we optimize the structure of the controller along with the control parameters, (2) we propose a new low-dimensional control within the state-space controller paradigm that we argue is more robust and efficient, and (3) we optimize the controllers based on single-view video.

5.2 Overview

The goal of our method is to estimate motion and controller from monocular video (or reference motion capture data). We optimize the controller structure and parameters such that the resulting motion produces a good match to image observations. Our approach has three important components:

Body Model and Motion. We model the human body, its actuation and interactions with the environment using the character model from Chapter 3. According to the formalisms in Section 3.5.1, we represent the 3D kinematic pose of an articulated human skeleton at the time t by a state vector $\mathbf{q}_t = [\mathbf{q}_t^{pos}, \mathbf{q}_t^{rot}, \mathbf{q}_t^{ang}]$, comprising the root position ($\mathbf{q}_t^{pos}$), the root orientation ($\mathbf{q}_t^{rot}$) and joint angles ($\mathbf{q}_t^{ang}$). Articulated rigid-body dynamics and integration from Section 3.7 provide a function for mapping the *dynamic* pose, $[\mathbf{q}_t, \dot{\mathbf{q}}_t]$, at the time t to the pose at the next time instant, $[\mathbf{q}_{t+1}, \dot{\mathbf{q}}_{t+1}]$ (where $\dot{\mathbf{q}}_t$ is the time derivative of $\mathbf{q}_t$),

$$[\mathbf{q}_{t+1}, \dot{\mathbf{q}}_{t+1}] = f([\mathbf{q}_t, \dot{\mathbf{q}}_t], \tau_t). \tag{5.1}$$

This function is a numerical approximation of the continuous integration of internal joint torques in the humanoid body model, τ_t, with respect to the current dynamic pose, $[\mathbf{q}_t, \dot{\mathbf{q}}_t]$. Joint torques approximate body actuation by muscles.

Actuation. We make use of a state-space controller from Section 1.4.3.1 for the actuation of the body that divides the control into a sequence of simple actions, transitions among which occur based on time or contact (see Section 5.4.1). We generate the motion of the character using the simulation and control loop from Section 1.4.3.2 where actuation forces for actions are produced by the low-level motion control process within the controller. We formulate the control process based on constraints, where the constraints drive the simulated character towards a pre-defined posture encoded within the current action. We implement the low-level motion control using the method from Section 3.6.

We produce the joint torques, τ_t, in Equation (5.1) using a *controller g*,

$$\tau_t = g([\mathbf{q}_t, \dot{\mathbf{q}}_t]; \mathscr{S}_M, \Theta), \tag{5.2}$$

where $\mathscr{S}_M$ is the structure of the controller, M is the number of states in the control structure (see Section 5.4.1) and Θ is a vector of controller parameters. A particular controller structure $\mathscr{S}_M$ induces a family of controllers, where the parameters Θ define the behavior of the controller. The simulation and control proceeds by iteratively applying Equation (5.1) and Equation (5.2), resulting in a sequence of kinematic poses, $\mathbf{q}_{1:T}$. Because this formulation is recursive, initial kinematic pose $\mathbf{q}_0$ and velocities $\dot{\mathbf{q}}_0$ are needed to bootstrap integration.

Estimating Controllers. The key aspect of our method is optimization of the controller structure ($\mathscr{S}_M^*$), the parameters of the controller (Θ^*), the initial pose ($\mathbf{q}_0^*$) and the velocities ($\dot{\mathbf{q}}_0^*$) in order to find a motion interpretation that best explains the observed behavior. This optimization can be written as a minimization of the energy function,

$$[\Theta^*, \mathscr{S}_M^*, \mathbf{q}_0^*, \dot{\mathbf{q}}_0^*] = \text{argmin}_{\Theta, \mathscr{S}_M, \mathbf{q}_0, \dot{\mathbf{q}}_0} E(\mathbf{y}_{1:T}), \tag{5.3}$$

that measures the inconsistency between the poses produced by the dynamic simulation (integration) with the controller within simulation and control loop and the image (or reference motion capture) observations, $\mathbf{y}_{1:T}$. In order to produce controllers that can generalize, the energy function also measures the quality of the controller itself in terms of its robustness and stability.

Because optimizing both controller structure and parameters is difficult, we first formulate a batch optimization over the controller parameters assuming a known and fixed controller structure and then introduce

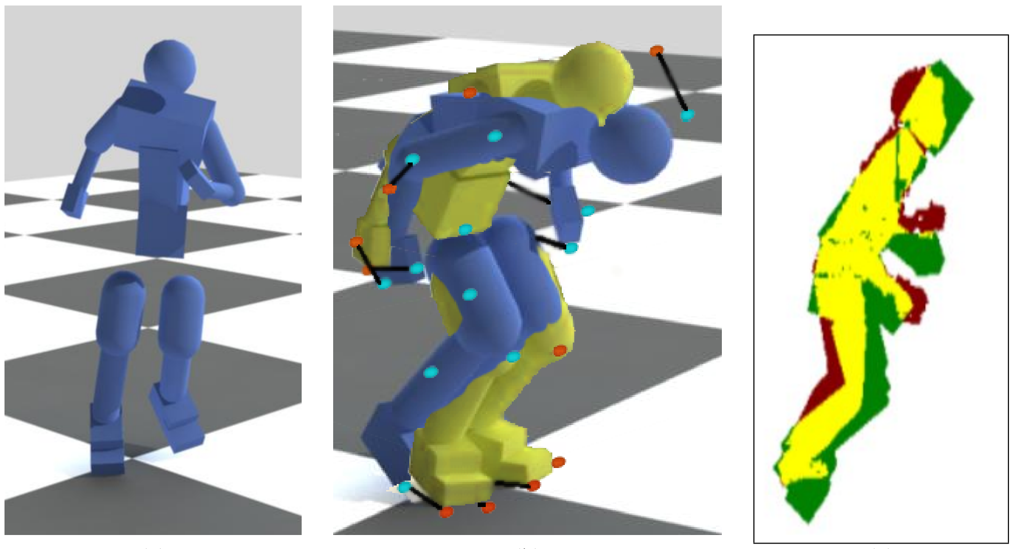

(a) (b) (c)

Figure 5.3: **Body Model and Likelihoods.** Illustration of the body model and the collision geometry used for simulation is shown in (a) (see Section 5.3 for details); (b) and (c) illustrate the motion capture and image based likelihoods respectively (see Section 5.5 for more details).

an incremental optimization procedure over the structure itself by utilizing the batch optimization as a subroutine. We further refine the structure of the resulting controllers, by applying structural transformations and re-optimizing the parameters.

The exact formulation of our objective function is given in Section 5.5.

5.3 Body Model and Motion

We use a generic model of the human body from Chapter 3 that represents an average person. We encode the human body using 18 rigid body parts (with upper and lower body parts for arms and legs, hands, two part feet, three part torso, and head). The center of mass and inertial properties of the parts are taken from population averages and biomechanical statistics [45]. For the purposes of simulation, each body part is parameterized by position and orientation in 3D space and has an associated collision geometry composed of geometric primitives (see (a) in Figure 5.3). Physical and environment restrictions are encoded using constraints. In particular, we impose (i) *joint articulation constraints* (see Section 3.3.2) that ensure that body parts stay attached at the joints and have only certain relative degrees of freedom (*e.g.*, we model the knee and elbow as 1 DOF hinge joints), (ii) *joint angle limit constraints* (see Section 3.5) that ensure that degrees of freedom are only allowed to articulate within allowable ranges (*e.g.*, to prevent hyperextension at elbows and knee joints), (iii) *contact constraints* (see Section 3.4.4) that model contact and friction between parts of the body, and parts and the environment. All three types of constraints are automatically generated by the simulator. The pose $\mathbf{q}_t = [\mathbf{q}_t^{pos}, \mathbf{q}_t^{rot}, \mathbf{q}_t^{ang}] \in \mathbb{R}^{40}$ of the model encodes the root position, root orientation and joint angles, respectively.

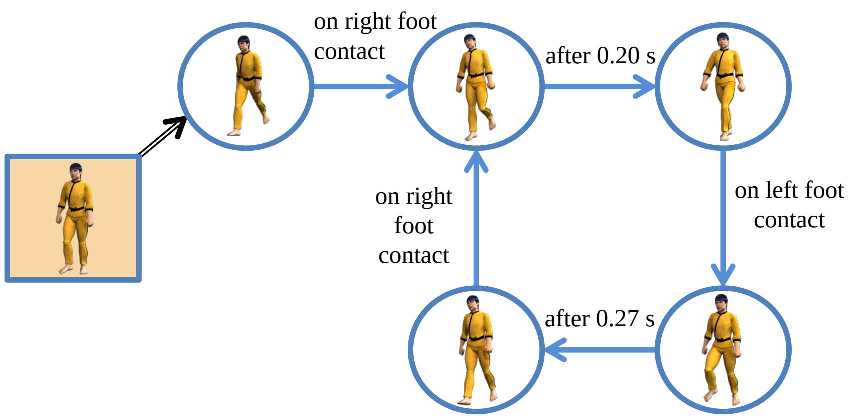


Figure 5.4: **State-space Controller.** Walking motion as an iterative progression of simple control tasks with tasks illustrated as circles, transitions as arrows and the initial pose as a box. The shown controller structure and controller parameters were obtained by Algorithm 6 applied on the fast walk motion from our dataset.

These constraints result in a large system of equations that can be expressed as a *lo-hi linear complementarity problem* (Mixed LCP), see Section 3.2.2.7 and [134, 181]. LCP solvers can be used to solve this problem efficiently, yielding a set of forces (and torques) required to satisfy all of the above mentioned constraints; we use a publicly available implementation [190].

5.4 Actuation

We formulate our controller using constraints and directly integrate these constraints with the body and simulator-level constraints discussed above. As a result, the augmented lo-hi LCP solution takes the actuation torques into account to ensure that the constraint forces anticipate the impact of actuation on the system.

We assume a SIMBICON [213] style controller, where a motion can be expressed as a progression of simple control tasks. Such paradigms have also been proposed in biomechanics (*e.g.*, [154]) and amount to a hypothesis that the control consists of higher level primitives that compose to create the desired motion.

5.4.1 State-space Controller

State-space controllers abstract desired motion dynamics through a sparse set of target poses and parameters; in essence allowing a key-frame-like representation [201] of the original motion. The use of this sparse representation allows more robust inference that is not as heavily biased by the noise in the individual video frames. Despite the fact that this is a sparse representation, however, it still allows the expressiveness necessary to model variations in style and speed [199] that we may observe in video.

An action-based (state-space) controller can be expressed as a finite state machine (see Figure 5.4) with M

states, where states correspond to actions and transitions encode the rules under which the actions switch. Actions switch based on timing or contact events (*e.g.*, foot-ground contact). For example, a walking controller that can be expressed in this way (and estimated by our method) is shown in Figure 5.4.

Controller Structure. We denote the structure of the controller by $\mathscr{S}_M$. The structure describes which actions need to be executed and how the actions switch in order to produce a desired motion for the character. We characterize the structure using a schematic representation that consists of (1) actions, denoted by $\odot_i$, (2) the types of transitions among those actions, which are denoted by arrows with associated discrete variables κ_i, $i \in [1, M]$, indicating when the transition can be taken (*e.g.*, $\kappa_i = \mathscr{T}$ for a transition on time, $\kappa_i = \mathscr{L}$ for a transition on left foot contact, and $\kappa_i = \mathscr{R}$ for a transition on right foot contact), and (3) a symbol $\oplus \longrightarrow$ to indicate the initial pose and velocity at the simulation start. A chain controller for one single walk cycle in our representation may look something like this,

$$\mathscr{S}_4 = \{\oplus \longrightarrow \odot_1 \overset{\kappa_1=\mathscr{L}}{\longrightarrow} \odot_2 \overset{\kappa_2=\mathscr{T}}{\longrightarrow} \odot_3 \overset{\kappa_3=\mathscr{R}}{\longrightarrow} \odot_4\}. \tag{5.4}$$

Controller Parameters. Given the structure $\mathscr{S}_M$, the behavior of the controller is determined by controller parameters Θ describing the actions and transitions within the controller structure.

We make use of parameterized actions from Equation (3.189) defined in Section 3.6.1. Assuming the $\mathbf{a}_i$ parameterizes the action in the state $\odot_i$ and ϕ_i denotes the time of the transition $\overset{\kappa_i=\mathscr{T}}{\longrightarrow}$, when the transition happens based on time ($\kappa_i = \mathscr{T}$), we encode the controller behavior using the parameter vector Θ,

$$\Theta = [\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_M, \phi_1, \phi_2, \ldots]. \tag{5.5}$$

According to Equation (3.189), we characterize the action in $\odot_i$ by the *target pose* $\mathbf{q}_i^d$ that the controller tries to achieve when executing the state $\odot_i$ and the parameters θ_i, ϑ_i of the corresponding control and balance laws, respectively, used to do so. We encode the target pose $\mathbf{q}_i^d$ in a potentially low-dimensional space using $\mathbf{t}_i$ such that $\mathbf{q}_i^d = g(\mathbf{t}_i)$. The simplest case is where the encoding of the target pose is the same as $\mathbf{t}_i$, $\mathbf{q}_i^d = g(\mathbf{t}_i) = \mathbf{t}_i$; a more effective representation is discussed in Section 5.4.1.2[1]. Consequently,

$$\mathbf{a}_i = [\mathbf{t}_i, \theta_i, \vartheta_i]. \tag{5.6}$$

As such, $\mathbf{a}_i$ corresponds to the parameters of $\odot_i$ in $\mathscr{S}_M$ and ϕ_i to the parameters of the transitions $\overset{\kappa_i=\mathscr{T}}{\longrightarrow}$. Note, typically, it is not necessary to *hit* $\mathbf{q}_i^d$ to achieve the desired motion.

5.4.1.1 Low-level Motion Control

Assuming the state machine is in the state $\odot_i$, the function of *low-level motion control* is to compute torques necessary to drive the pose of the character towards the target pose $g(\mathbf{t}_i)$ while keeping the body balanced. We extended the low-level motion control approach from Section 3.6 and the approach of Yin and colleagues [212] for control and balance to set up the control laws.

As in [213], we maintain balance of the body by (1) using an active feedback mechanism to adjust the target pose and (2) applying torques on the body in order to follow the prescribed target orientation for the root

[1]We only encode desired angular configuration of the body in $\mathbf{q}_i^d$.

part. To make the method more robust, we replace Yin's Proportional Derivative (PD)-servo with actuation based on inverse dynamics. The behavior of the actuation process is determined by the values of the control parameters θ_i and balance parameters ϑ_i. Intuitively, these parameters encode how the target pose should be adjusted by the balance feedback mechanism and define the speed with which the adjusted pose should be reached. We use the control principles from Section 3.6.1 to realize the body actuation.

Actuation. We formulate our low-level motion control based on soft constraints (similar to [181]) from Section 3.2.3. Assuming that the body is currently in a pose $\mathbf{q}_t$, and the state machine is executing an action from the finite state $\odot_i$, we realize low-level motion control for the action using *pose difference stabilization*. A stabilization mechanism imposes constraints on the angular velocity of the root, $\dot{\mathbf{q}}_t^{rot}$, and joint angles, $\dot{\mathbf{q}}_t^{angles}$, [36] in order to reduce the difference between the current pose and the adjusted target pose $[\mathbf{q}_d^{rot}, \mathbf{q}_d^{angles}]$, where $[\mathbf{q}_d^{rot}, \mathbf{q}_d^{angles}]$ is obtained by modifying the target pose $g(\mathbf{t}_i)$ from the machine state by the balance feedback. As a result, the low-level motion control defines these soft constraints,

$$\{\dot{\mathbf{q}}_t^{rot} = -\alpha_i^{root} \cdot (\mathbf{q}_t^{rot} - \mathbf{q}_d^{rot}) \qquad \text{with softness } \mathbf{s}_i^{root}\} \tag{5.7}$$

$$\{\dot{\mathbf{q}}_t^{angles} = -\alpha_i^{angles} \cdot (\mathbf{q}_t^{ang} - \mathbf{q}_d^{angles}) \qquad \text{with softness } \mathbf{s}_i^{angles}\}, \tag{5.8}$$

parameterized by α_i and $\mathbf{s}_i$, where $\theta_i = \{\alpha_i, \mathbf{s}_i\}$ according to Equation (3.190). The parameters α_i encode the speed of the stabilization mechanism and $\mathbf{s}_i$ control the softness of the constraints. Semantics of the stabilization mechanism is explained in Section 3.2.4.1 and Section 3.6.1. To reduce the number of control parameters that are optimized, the left side and the right side of the body share the same parameter values. Note, these constraints are obtained by applying the constraint stabilization mechanism from Section 3.2.4.1 on the position constraints from Equation (3.178) and applying the same constraints on the root.

Underactuation. The formulation above directly controls the global orientation of the root, which is essential for balance. As in Yin and colleagues [213], it does so by directly applying forces on the root part. However, actuation forces should only be produced by joints in the body model and the root should be unactuated. Consequently, this control is only valid if the net torque on the root part is equal and opposite to the sum of torques of all connected parts,

$$\{\tau_t^{root} = -\sum_{k \in \text{Neighbor}(root)} \tau_t^k\}. \tag{5.9}$$

Because we cannot express Equation (5.9) directly as a constraint with Equation (5.7) and Equation (5.8), we employ a prioritized approach. This approach satisfies Equation (5.9) exactly and uses a "best effort" strategy to satisfy the other two constraints (Equation (5.7) and Equation (5.8)). Towards that end, we use the method from Yin and colleagues [213] to decompose extra torques applied at the root to additional reaction torques applied at the thighs (see Algorithm 3).

We first solve for the torques τ_t that satisfy Equation (5.7) and Equation (5.8) using inverse dynamics (ignoring Equation (5.9)) and apply the torques to the character. We then compute τ_{error}^{root} — the amount of violation in Equation (5.9). The value of τ_{error}^{root} equals exactly the "extra" torque received by the root part that is not "explained" by any of the joints attached to the root. To alleviate this error, we apply the corresponding reaction torque $-\tau_{error}^{root}$ to the swing upper leg part (to ensure that Equation (5.9) holds, see Figure 5.5). This approach is similar to Yin and colleagues [213]. As a result of this correction, Equation (5.9) and

Algorithm 3 : Application of control forces

```
1: Solve for $\tau_t$ using inverse dynamics to satisfy Equation (5.7) and Equation (5.8) and apply $\tau_t$
2: Determine by how much Equation (5.9)) is violated:
       $\tau^{root}_{error} = \tau^{root}_t + \sum_{k \in \text{Neighbor}(root)} \tau^k_t$
3: Apply $-\tau^{root}_{error}$ to the swing upper leg
4: Solve for $\tau_t$ using inverse dynamics to satisfy Equation (5.8)) and apply $\tau_t$
```

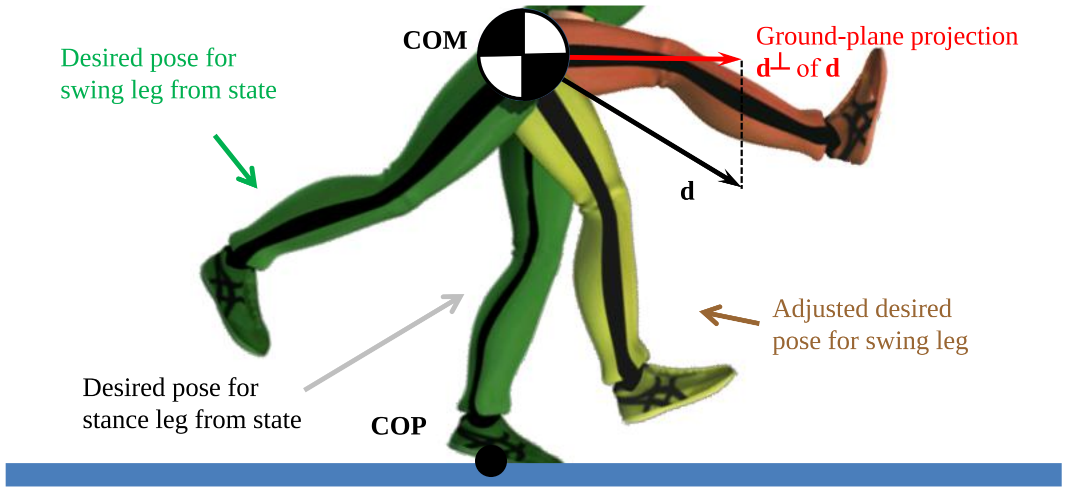

COM
Ground-plane projection
$\mathbf{d}^\perp$ of $\mathbf{d}$
Desired pose for swing leg from state
$\mathbf{d}$
Adjusted desired pose for swing leg
Desired pose for stance leg from state
COP

Figure 5.5: **Balance Feedback.** Target orientation of the swing upper leg (yellow) is the result of blending between the target orientation stored in the state (green) and an orientation where the leg points along the ground-plane projection $\mathbf{d}^\perp$ of $\mathbf{d}$ (red), produced by the feedback mechanism.

Equation (5.7) hold exactly and Equation (5.8) may slightly be violated due to the $-\tau^{root}_{error}$ torque applied at the swing upper leg.

Balance Feedback. The constraints on the root part attempt to maintain the world orientation of the character. Active feedback mechanisms, however, may be necessary to prevent falling when the environment changes or perturbations are applied. We generalize the balance feedback mechanism of Yin and colleagues [213] to dynamically adjust the target orientation of the swing upper leg from $g(\mathbf{t}_i)$. The feedback mechanism uses three additional control parameters encoded into a vector $\vartheta_i \in \mathbb{R}^3$.

Intuitively, the purpose of the balance control law is to affect the motion of the swing foot in order to move the character toward a balanced pose where the center of mass (**COM**) is above the center of pressure (**COP**). The feedback is implemented by synthesizing a new target orientation for the swing upper leg and using it as a target (see Figure 5.5). The controller will then track an adjusted target pose $[\mathbf{q}^{rot}_d, \mathbf{q}^{angles}_d]$, where the target orientation $\mathbf{q}^{angles,SUL}_d$ for the swing upper leg is set as

$$\mathbf{q}^{angles,SUL}_d = \text{slerp}(g(\mathbf{t}_i)^{k,SUL}, \text{orientation}(\mathbf{d}^\perp), \vartheta_{i,3}), \tag{5.10}$$

where slerp is a spherical linear interpolation function, SUL refers to the DOFs representing the orientation of the swing upper leg in the pose,

$$\mathbf{d} = \vartheta_{i,1}(\mathbf{COM} - \mathbf{COP}) + \vartheta_{i,2}(\dot{\mathbf{COM}} - \dot{\mathbf{COP}}), \tag{5.11}$$

$\mathbf{d}^{\perp}$ is $\mathbf{d}$ projected to the ground plane and orientation($\mathbf{d}^{\perp}$) produces a leg orientation such that the leg's medial axis points along $\mathbf{d}^{\perp}$. The remaining DOFs within $[\mathbf{q}_d^{rot}, \mathbf{q}_d^{angles}]$ are copied from $g(\mathbf{t}_i)$.

Force Limits. To ensure that the controller does not apply super-human forces, we limit the joint torques via limits on the solution to the lo-hi LCP problem using Equation (3.188) with $\gamma = 120$.

Discussion. When we solve for control torques, control constraints are automatically combined with all other constraints in our simulator to anticipate effects of body articulation and contact. Our constraint-based control is different from a PD-servo. A PD-servo assumes that the control torques are linear functions of the differences between the current and the target poses of the body. It also assumes that each degree of freedom can be controlled independently and relies on a feedback mechanism to resolve the resulting interactions within the system. In contrast, our control mechanism solves for the torques necessary to approach the target pose, by explicitly taking into consideration constraints present among rigid body parts and the environment. Because we may not want to reach the target pose immediately, parameters α_i modulate the fraction of the pose difference to be reduced in one simulation step. As such, the difference in pose is reduces with an exponential decay (see Section 3.2.4.1). More exact control and less reliance on feedback allow us to simulate at much lower frequencies (*e.g.*, 120Hz). We use support from Section 3.7 to realize inverse dynamics.

5.4.1.2 Parameterization for Target Pose

Encoding of Target Pose, PCA Space. The vector $\mathbf{t}_i$ encodes the representation of the target pose $\mathbf{q}_i^d = g(\mathbf{t}_i)$ for the machine state $\odot_i$, where the function $g(\mathbf{t}_i)$ maps $\mathbf{t}_i$ into the angle representation of $[\mathbf{q}^{rot}, \mathbf{q}^{ang}]$. For a given class of motions, it is reasonable to assume that $g(\mathbf{t}_i)$ should lie within a manifold of poses likely for that motion. Hence, we propose a PCA prior for $g(\mathbf{t}_i)$ and represent target poses in a PCA space. In doing so, we are motivated by the fact that the PCA representation lets us reduce the dimensionality of the control parameter space for optimization and also implicitly accounts for important correlations between poses of body parts that we can not observe in silhouettes.

Training PCA Spaces. We train activity-specific PCA priors from marker-based motion capture obtained off-line. We use motion capture from a single subject performing walk, jump, twist jump, spin kick and back handspring. For each activity, we learn a linear basis $\mathbf{U}$ from the corresponding training motion capture data[2] using SVD and let,

$$g(\mathbf{t}_i) = \mathbf{U} \cdot \mathbf{t}_i + \mathbf{b}. \tag{5.12}$$

We then only need to store PCA coefficients $\mathbf{t}_i$ in the parameter vector Θ. We keep enough principal components to account for 95% of the variance in the data. The PCA representation of target poses significantly reduces the dimensionality of our pose search space from $40 \cdot M$ (where M is the number of states in the state-space controller) to roughly $8M$ for most motions. We also found this method to lead to faster and more

[2]We encode each angle as $[sin(\cdot), cos(\cdot)]$ to avoid wrap around.

robust convergence. We add a uniform prior on $\mathbf{t}_i$ such that the coefficients are within $\pm 4\sigma$. Similarly, we assume that the initial kinematic pose can be encoded in the same PCA space, so that $\mathbf{q}_0 = \mathbf{U} \cdot \mathbf{t}_0 + \mathbf{b}$, and optimize $\mathbf{t}_0$ instead of $\mathbf{q}_0$.

PCA Spaces at Testing. Using the procedure from above, we obtain a family of models $\{\mathbf{U}_i, \mathbf{b}_i\}$ where i is over the set of activities considered as well as an activity-independent model $\{\mathbf{U}_0 = \mathbf{E}_{40\times 40}, \mathbf{b}_0 = \mathbf{0}_{40}\}$. Because, in practice, we do not know the type of activity being performed at test time, we run our framework with each prior and choose the resulting controller that best fits observations according to the objective function $E(\mathbf{y}_{1:T})$. We used data from disjoint trials for training and testing.

Unlike traditional kinematic prior models, we use PCA priors in a very limited way — to parameterize target poses in the controller so that we can capture important correlations between joint angles that otherwise may not be observable, or well constrained, by the image observations. In doing so, we do not make use of any temporal information present in the data and allow targets to significantly exaggerate training poses. We found PCA priors unnecessary when optimizing from less ambiguous data (*e.g.*, reference motion capture or favorable videos).

5.4.1.3 Transitions

In our framework, state transitions within a state-space controller can happen based on timing ($\kappa_i = \mathscr{T}$) or contact events ($\kappa_i \in \{\mathscr{L}, \mathscr{R}\}$). Transitions on timing happen once the simulation time spent in the state is $\geq \phi_i$ for the current state $\odot_i$. Transitions on contact events happen when the associated body part becomes supported by a contact force, as determined by the simulator.

5.5 Estimating Controllers

In order to obtain a controller that approximates the motion in the video (or in the reference motion capture), we need to estimate both the structure of the controller appropriate for the given motion, $\mathscr{S}_M$ (including the number of states M and the types of transitions κ_i) and parameters, Θ, of the controller optimized to fit the observations and the priors. We also need to optimize the initial pose, $\mathbf{q}_0$, and velocities, $\dot{\mathbf{q}}_0$. This optimization amounts to minimizing the objective function, in Equation (5.3) with respect to $\mathscr{S}_M$, Θ, $\mathbf{q}_0$ and $\dot{\mathbf{q}}_0$. Our objective function,

$$E(\mathbf{y}_{1:T}) = \lambda_l E_{\text{like}} + \lambda_s E_{\text{stability}} + \lambda_p E_{\text{prior}} + \lambda_c E_{\text{contact}} + \lambda_t E_{\text{torque}}, \tag{5.13}$$

contains five terms measuring the inconsistency of the simulated motion produced by the controller with the image-based (or reference motion capture) observations and the quality of the controller itself; where λ_i are weighting factors designating the overall importance of the different terms. We now describe the five error terms.

Image-based Likelihood. The key component of the objective function is the likelihood term, E_{like}. In the image case, it measures the inconsistency of the simulated motion, $\mathbf{q}_{1:T}$, with the foreground silhouettes, $\mathbf{y}_{1:T}$, $\mathbf{q}_{1:T}$ is obtained using the simulation and control loop. Assuming that the likelihood is independent at each frame, given the motion of the character, we can measure the inconsistency by adding up contributions from each frame. We determine this contribution by projecting a simulated character into the image (assuming a known camera projection matrix) and computing the difference from image features at each pixel. We adopt a simple silhouette likelihood [161] and define a symmetric distance between the estimated binary silhouette mask, $\hat{\mathbf{M}}_s$ (see Figure 5.3 (c), green blob), at the time s, and the image binary silhouette mask, $\mathbf{M}_t$ (see Figure 5.3 (c), red blob), at the time t. This approach results in the following formulation for the energy term that we optimize as part of the overall objective:

$$E_{\text{like}} = \sum_{t=1}^{T} \frac{\mathbf{B}_{t,t}}{\mathbf{B}_{t,t} + \mathbf{C}_{t,t}} + \frac{\mathbf{A}_{t,t}}{\mathbf{A}_{t,t} + \mathbf{C}_{t,t}}, \tag{5.14}$$

where the $\mathbf{C}_{t,s}$, $\mathbf{A}_{t,s}$ and $\mathbf{B}_{t,s}$ terms count pixels in the $\mathbf{M}_t \wedge \hat{\mathbf{M}}_s$, $\mathbf{M}_t \wedge \neg\hat{\mathbf{M}}_s$ and $\neg\mathbf{M}_t \wedge \hat{\mathbf{M}}_s$ masks, that is,

$$\mathbf{C}_{t,s} = \sum_{(x,y)} \mathbf{M}_t(x,y)\hat{\mathbf{M}}_s(x,y) \tag{5.15}$$

$$\mathbf{A}_{t,s} = \sum_{(x,y)} \mathbf{M}_t(x,y)\left[1 - \hat{\mathbf{M}}_s(x,y)\right] \tag{5.16}$$

$$\mathbf{B}_{t,s} = \sum_{(x,y)} \left[1 - \mathbf{M}_t(x,y)\right]\hat{\mathbf{M}}_s(x,y). \tag{5.17}$$

We estimate silhouettes in video using a standard background subtraction algorithm [47]. The background model comprised the mean color image and an intensity gradient, along with a single 5D covariance matrix (estimated over the entire image).

Motion-capture-based Likelihood. To use reference motion capture data instead of video data to estimate controllers, we define a motion capture likelihood as a sum of squared differences between the markers attached to the observed skeleton and the simulated motion. In particular, we attach three markers to every body part and let

$$E_{\text{like}} = \sum_{t=1}^{T} \sum_{j=1}^{18} \sum_{k=1}^{3} ||m_{t,j,k} - \hat{m}_{t,j,k}||^2 \tag{5.18}$$

where $m_{t,j,k} \in \mathbb{R}^3$ is the location of the k-th marker attached to the j-th body part at the time t (computed using forward kinematics) from the reference motion capture and $\hat{m}_{t,j,k} \in \mathbb{R}^3$ is the location of the k-th marker attached to the j-th part at time t of the simulated character.

Stability. The likelihood defined above will result in controllers that may fail near the end of the sequence because the optimization has a short horizon. We make an assumption that subjects in our video sequence end up in a stable posture. To ensure that the controller ends up in a similar, statically stable pose, we add an additional term that measures inconsistency of the simulation for the time ΔT past the time T, where T is the time of the last observation. We reuse the formulation of E_{like} in Equation (5.14) and define this term as

$$E_{\text{stability}} = \sum_{t=T+1}^{T+\Delta T} \frac{\mathbf{B}_{T,t}}{\mathbf{B}_{T,t} + \mathbf{C}_{T,t}} + \frac{\mathbf{A}_{T,t}}{\mathbf{A}_{T,t} + \mathbf{C}_{T,t}}. \tag{5.19}$$

Prior. To bias the solution towards more likely interpretations, we propose a prior over the state-space control parameters. Generally, there are four types of parameters that we optimize:

- representation of the target poses for the low-level motion control process, $\mathbf{t}_i$,
- parameters of control laws, consisting of the constraint stabilization parameters, α_i, and constraint softness parameters, $\mathbf{s}_i$,
- balance feedback parameters, ϑ_i, and
- transition times, ϕ_i,.

In addition, we optimize the initial pose $\mathbf{q}_0$ and velocities $\dot{\mathbf{q}}_0$.

For $\alpha_i, \mathbf{s}_i, \phi_i, \vartheta_i$, we use uninformative uniform priors over the range of possible values. These priors are

$$\begin{aligned} (\alpha_i)_j &\sim \mathscr{U}(0.001, 0.2) && (5.20)\\ (\mathbf{s}_i)_j &\sim \mathscr{U}(0.00001, 0.05) && (5.21)\\ \phi_i &\sim \mathscr{U}(0.05, 1.0) && (5.22)\\ \vartheta_{i,1} &\sim \mathscr{U}(-1, 1) && (5.23)\\ \vartheta_{i,2} &\sim \mathscr{U}(-1, 1) && (5.24)\\ \vartheta_{i,3} &\sim \mathscr{U}(0, 1), && (5.25) \end{aligned}$$

where j indexes DOFs within vectors. We also impose a uniform prior on $\mathbf{t}_i$,

$$(\mathbf{t}_i)_j \sim \mathscr{U}(-4\sigma, 4\sigma). \tag{5.26}$$

The uniform priors over parameters are encoded using a linear penalty on the values of the parameters that are outside the valid range. In particular, for every variable, v, with a uniform prior $v \sim \mathscr{U}(a, b)$, we add the following term to the prior, $E_{\text{prior}}(\Theta)$

$$|\max(0, v - b)| + |\min(0, v - a)|. \tag{5.27}$$

Contact. In our experience, controllers optimized with these objectives often result in frequent contact changes. This issue is particularly problematic for low clearance motions like walking. For example, a walking controller may stumble slightly (see (a) in Figure 5.8) if that helps to produce a motion that is more similar to what is observed in video or reference motion capture. This behavior hinders the ability of the controller to be robust to perturbations in the environment. To address this issue, we assume that there is no more than one contact state change between any two consecutive actions in the state-space controller. This policy is motivated by the observation that contact state change yields discontinuity in the dynamics and hence must be accompanied by a state transition. State transitions, however, may happen for other reasons (*e.g.*, performance style). To encode this knowledge in our objective, we define a contact state change penalty term (see (b) in Figure 5.8 for the illustration of the effects of the term):

$$E_{\text{contact}} = \sum_{i=i}^{M-1} c(i), \tag{5.28}$$

where

$$c(i) = \begin{cases} 0 & \text{0 or 1 contact change between } \odot_i \text{ and } \odot_{i+1} \\ 10,000 & > 1 \text{ contact change between } \odot_i \text{ and } \odot_{i+1} \end{cases}$$

We define a contact state change as one or more body parts changing their contact state with respect to the environment (environment ceasing to apply forces, or vice versa).

Energy Expenditure. In order to produce motions that appear smoother and do not always utilize the maximum power that the joint torque bounds allow, we add an additional term to the objective function that penalizes the energy expenditure and encourages the motion to use less power,

$$E_{\text{torque}} = \sqrt{\sum_{t=1}^{T} ||\tau_t||^2}, \tag{5.29}$$

where $||\tau_t||^2$ approximates the energy expenditure at the time t and τ_t is the joint torque. Smoothness of the produced motion can be further encouraged by incorporating additional terms in the objective function that attempt to minimize *e.g.*, the sum of squared joint accelerations [151] or jerk (rate of change of acceleration) [54].

5.5.1 Optimization

Now that we have defined our objective function, we need to optimize it with respect to $\mathscr{S}_M$, Θ, $\mathbf{q}_0$, $\dot{\mathbf{q}}_0$ to obtain the controller of interest. The parameters Θ are in general closely tied to the structure of the controller $\mathscr{S}_M$. We first formulate a batch optimization over the controller parameters assuming a known and fixed controller structure $\mathscr{S}_M$ and then introduce an iterative optimization over the structure itself by utilizing the batch optimization as a sub-routine.

Two-phase Optimization Process. Our incremental optimization over the structure and the parameters of the controller is done in two phases (see Figure 5.6).

In the first phase (see Figure 5.6, (a)), we use incremental optimization from Section 5.5.1.2 to construct a state chain with all transitions based on time. In this phase, we optimize in stages over an expanding motion window and greedily add states to the end of the chain as long as the addition improves fit to the observations. The obtained chain controller provides an initial interpretation of the motion, but is limited to the environment where the capture took place, because it does not react to contact events that happen in simulation.

In the second phase (see Figure 5.6, (b)), we refine the structure of the controller using the method from Section 5.5.1.3. We either replace transitions on time, happening close to actual contact events (as encouraged by Equation (5.28)), with the corresponding transitions on contact, or remove states and add a loop if we find a close-enough duplicate state that produces the same low-level motion control. These changes make the controller more robust to external perturbances and more compact. By forcing the transitions to happen on contact events, we are able to improve the ability of the controller to generalize, because it is able to react to these physical events in simulation, even if they happen at different (unexpected) time instants. Adding loops allows us to recover cyclic controllers for motions that are inherently cyclic (*e.g.*, walking), such that they can be simulated beyond the optimization horizon.

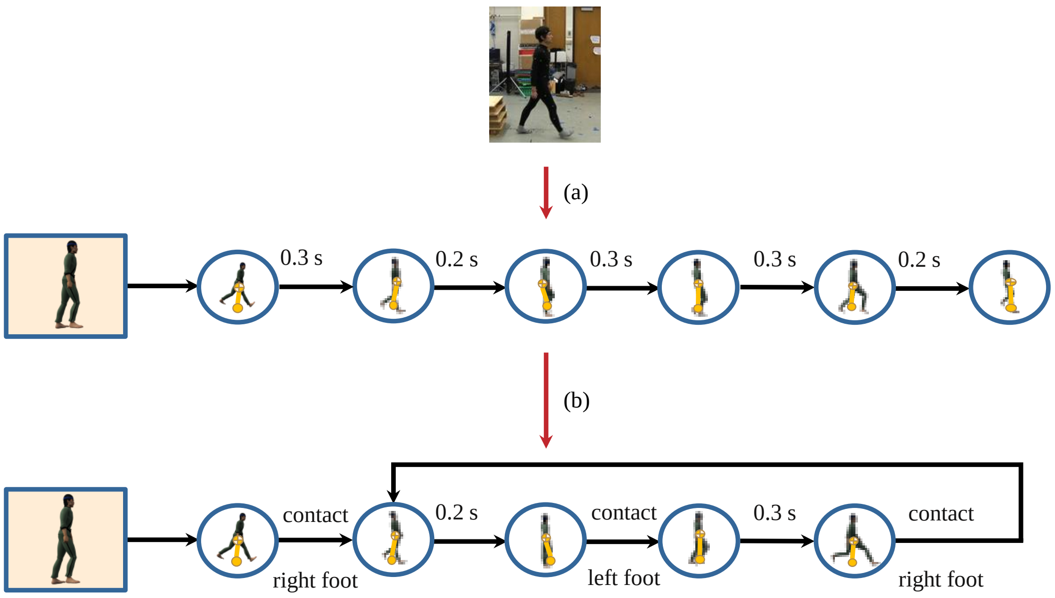


Figure 5.6: **Optimization in Two Phases.** Illustration of the optimization of fast walk in two phases. In the first phase (a), greedy incremental optimization produces a state chain with all transition based on time such that the input video (top) is reproduced. This controller provides a motion reconstruction of the observed video in the original environment. In the second phase (b), the structure of the state chain is refined and the controller re-optimized, forcing transitions to happen on contact and to recover cyclicity, where appropriate.

Covariance Matrix Adaptation. For optimizations of parameters with respect to a particular structure, we use gradient-free Covariance Matrix Adaptation (CMA) [67]. CMA is an iterative genetic optimization algorithm that maintains a Gaussian distribution over parameter vectors. The Gaussian is initialized with a mean, μ, encoding initial parameters, and diagonal spherical covariance matrix with variance along each dimension equal to σ^2. CMA proceeds by: (1) sampling a set of random samples from this Gaussian, (2) evaluating the objective $E(\mathbf{q}_{1:T})$ for each of those samples (this step involves simulation), and (3) producing a new Gaussian based on the most promising samples and the mean. The number of samples to be evaluated and to be used for the new mean is chosen automatically by the algorithm based on the problem dimensionality.

5.5.1.1 Batch Optimization of Controller Parameters

Given a state-space controller structure, $\mathscr{S}_M$, batch optimization infers the controller parameters Θ

$$\Theta = [(\mathbf{t}_1, \alpha_1, \mathbf{s}_1, \vartheta_1), (\mathbf{t}_2, \alpha_2, \mathbf{s}_2, \vartheta_2), \ldots, (\mathbf{t}_M, \alpha_M, \mathbf{s}_M, \vartheta_M), \phi_1, \phi_2, \ldots] \tag{5.30}$$

and the initial pose $\mathbf{q}_0$ and velocity $\dot{\mathbf{q}}_0$ of the character by minimizing the objective function $E(\mathbf{y}_{1:T})$. The optimization procedure (integrated with CMA) is illustrated in Algorithm 4.

Algorithm 4 : Controller batch optimization using CMA $[\Theta, \mathbf{q}_0, \dot{\mathbf{q}}_0, E] = \textbf{BatchOp}\ (\mathscr{S}_M, \mathbf{q}_0, \mathbf{Y}, \mathbf{U}, \mathbf{b}, \Theta, \dot{\mathbf{q}}_0)$

```
Input: State-space controller structure ($\mathscr{S}_M$); initial pose ($\mathbf{q}_0$); PCA prior ($\mathbf{U}$, $\mathbf{b}$); observations / image features ($\mathbf{Y} = \{\mathbf{y}_0, \mathbf{y}_1, \dots, \mathbf{y}_T\}$)
 Optional Input: Controller parameters ($\Theta$); initial velocity ($\dot{\mathbf{q}}_0$);
Output: Controller parameters ($\Theta$); initial pose ($\mathbf{q}_0$); initial velocity ($\dot{\mathbf{q}}_0$) objective value ($E$)
1: Project initial pose onto PCA space: $\mathbf{t}_0 := \mathbf{U}^{-1} \cdot (\mathbf{q}_0 - \mathbf{b})$
2: if $\dot{\mathbf{q}}_0 = \emptyset$, $\Theta = \emptyset$ then
3:     Initialize initial velocity: $\dot{\mathbf{q}}_0 := \mathbf{0}$
4:     Initialize controller parameters ($\Theta$):
           $\mathbf{t}_i := \mathbf{t}_0, \alpha_i := \text{replicate}(0.1), \mathbf{s}_i := \text{replicate}(0.0001), \vartheta_i := [0,0,0], \phi_i := 0.25\ \forall i \in [1, M]$
5: end if
6: Initialize variance: $\Sigma := \mathbf{E}\sigma$
7: Initialize mean: $\mu := [\Theta, \mathbf{t}_0, \dot{\mathbf{q}}_0]^T$
8: for $i := 1 \to N_{ITERATIONS}$ do
9:     for $j := 1 \to N_{POPULATION}$ do
10:        Sample controller parameters and initial pose:
               $[\Theta^{(j)}, \mathbf{t}_0^{(j)}, \dot{\mathbf{q}}_0^{(j)}] \sim \mathcal{N}(\mu, \Sigma)$
11:        Reconstruct initial pose:
               $\mathbf{q}_0^{(j)} := \mathbf{U}\mathbf{t}_0^{(j)} + \mathbf{b}$
12:        for $t := 1 \to T + \Delta T$ do
13:            Control and simulation:
               $\begin{bmatrix} \mathbf{q}_t^{(j)} \\ \dot{\mathbf{q}}_t^{(j)} \end{bmatrix} := f\left(\begin{bmatrix} \mathbf{q}_{t-1}^{(j)} \\ \dot{\mathbf{q}}_{t-1}^{(j)} \end{bmatrix}, g\left(\begin{bmatrix} \mathbf{q}_{t-1}^{(j)} \\ \dot{\mathbf{q}}_{t-1}^{(j)} \end{bmatrix}, \Theta^{(j)}\right)\right)$
14:        end for
15:        Compute objective:
               $E^{(j)} := \lambda_l E_{\text{like}} + \lambda_s E_{\text{stability}} + \lambda_p E_{\text{prior}} + \lambda_c E_{\text{contact}} + \lambda_t E_{\text{torque}}$
16:    end for
17:    $[\mu, \Sigma] := \text{CMA\_update}\ (\mu, \Sigma, \{\Theta^{(j)}, \mathbf{t}_0^{(j)}, \dot{\mathbf{q}}_0^{(j)}, E^{(j)}\})$
18: end for
19: Let $j^* := \text{argmin}_j E^{(j)}$
20: return $\Theta^{(j^*)}$, $\mathbf{q}_0^{(j^*)}$, $\dot{\mathbf{q}}_0^{(j^*)}$, $E^{(j^*)}$
```

Batch optimization is useful when we have a reasonable guess for the controller structure. Batch optimization of cyclic controllers is particularly beneficial because weak observations for one motion cycle can be reinforced by evidence from other cycles, making the optimizations less sensitive to observation noise and less prone to overfitting to local observations. We make use of this fact during controller structure refinement in Section 5.5.1.3.

Reliance of Θ on the controller structure $\mathscr{S}_M$ suggests an approach where we enumerate the controller structures, estimate parameters in a batch, as above, and choose the best structure based on the overall objective value. Unfortunately, such an enumeration strategy is impractical because the number of possible controllers is exponential in M and each structure may yield many controller parameters (approximately $50 \cdot (M+1)$ continuous variables).

For example, see Figure 5.13 for an illustration of a complicated gymnastics motion that we estimate to require $M = 20$ states. In addition, without a good initialization, in our experience, optimization of the high-dimensional parameter vector is difficult and often gets stuck in local optima. To alleviate these problems,

Algorithm 5 : Incremental optimization of chain controller with transitions based on time $[\mathscr{S}_M, \Theta, \mathbf{q}_0, \dot{\mathbf{q}}_0, E] = \textbf{IncremOp}\ (\mathbf{q}_0, \mathbf{Y}, \mathbf{U}, \mathbf{b})$

```
Input: Initial pose ($\mathbf{q}_0$); PCA prior ($\mathbf{U}$, $\mathbf{b}$); observations / image features ($\mathbf{Y} = \{\mathbf{y}_0, \mathbf{y}_1, \ldots, \mathbf{y}_T\}$)
Output: State-space controller structure ($\mathscr{S}_M$); controller parameters ($\Theta$); initial pose ($\mathbf{q}_0$); initial velocity
    ($\dot{\mathbf{q}}_0$); objective value ($E$)
1: Number of observations to add per stage:
       $N_{STAGES} := T/T_s$
2: Initialize controller structure:
       $M := 1 \qquad \mathscr{S}_1 := \{\oplus \longrightarrow \odot_1\}$
3: Optimize parameters:
       $[\Theta, \mathbf{q}_0, \dot{\mathbf{q}}_0, E] := \textbf{BatchOp}\ (\mathscr{S}_1, \mathbf{q}_0, \mathbf{y}_{1:T_s}, \mathbf{U}, \mathbf{b})$
4: for $i := 2 \to N_{STAGES}$ do
5:     Re-optimize parameters:
           $[\Theta, \mathbf{q}_0, \dot{\mathbf{q}}_0, E] := \textbf{BatchOp}\ (\mathscr{S}_M, \mathbf{q}_0, \mathbf{y}_{1:i \cdot T_s}, \mathbf{U}, \mathbf{b}, \Theta, \dot{\mathbf{q}}_0)$
6:     Try to add a state:
           $\mathscr{S}_M^+ := \{\mathscr{S}_M \overset{\kappa_M = \mathscr{T}}{\longrightarrow} \odot_{M+1}\}$
           $[\Theta^+, \mathbf{q}_0^+, \dot{\mathbf{q}}_0^+, E^+] := \textbf{BatchOp}\ (\mathscr{S}_M^+, \mathbf{q}_0, \mathbf{y}_{1:i \cdot T_s}, \mathbf{U}, \mathbf{b}, \Theta, \dot{\mathbf{q}}_0)$
7:     if $E^+ < E$ then
8:         $\mathscr{S}_{M+1} := \mathscr{S}_M^+ \qquad M = M + 1$
9:         $[\Theta, \mathbf{q}_0, \dot{\mathbf{q}}_0, E] := [\Theta^+, \mathbf{q}_0^+, \dot{\mathbf{q}}_0^+, E^+]$
10:    end if
11: end for
```

we propose an approach for estimating the controller structure incrementally that also gives us an ability to have better initializations for optimization of control parameters.

5.5.1.2 Incremental Optimization of Controller Structure

Our key observation is that we can optimize controllers incrementally, such that the controller structure can be estimated locally and simultaneously with the estimation of control parameters. Our approach greedily selects the structure and the parameters of the controller as new observations are added (see Figure 5.7). The basic idea is to divide the hard high-dimensional batch optimization over the entire sequence into a number of easier lower-dimensional optimization problems over an expanding motion window. In doing so, we are motivated by the fact that (1) when the optimization window is short, we can assume a fixed controller structure and optimize only control parameters, and (2) we can optimize longer windows by reusing optimization results from shorter windows. In particular, optimization results over a current window can provide a good initialization for optimizations over the expanded window.

To this end, we start with a simple generic initial controller with one state and incrementally update this initial controller in order to construct a state chain that eventually explains the entire motion observed in the video. We realize this optimization in stages. At each stage, we expand the current window by T_s frames and re-optimize the current controller to fit the frames in the expanded window, adding new states to the chain as is necessary (see Algorithm 5).

Optimization in Stages. The incremental optimization proceeds in stages. At the first stage, the first T_s frames of the video sequence (the initial motion window) are optimized using the batch optimization algorithm (see Algorithm 5, lines 2-3), assuming a fixed initial controller structure with only one state, $\mathscr{S}_1 = \{\oplus \longrightarrow \odot_1\}$ to produce an initial controller. At all later stages, the current motion window is expanded by the subsequent T_s frames from the video sequence (or reference motion capture) and the current controller is re-optimized to fit this new window until the entire motion is processed.

In fitting the controller to the expanded window at a stage, we have two main options for the update of the controller. We can either attempt to explain the new frames with a controller that has an unchanged structure, assuming the new frames are a continuation of the last action in the chain, or we can change the controller structure. For example, the new frames may constitute a new action and so a new state may need to be added to the end of the chain. Because we do not know which hypothesis is correct, we try them all. More generally, we re-optimize the current controller using a *number of local optimizations*. The purpose of the local optimizations is to propose possible updates to the current controller structure (mainly the addition of a state to the current chain), test the validity of the updates by re-optimizing the current controller with the new structures and choose the new controller for the next stage based on its objective value after re-optimization. In particular, we select the controller that produces the lowest value for E. See Figure 5.7 for an illustration of the process.

Re-optimizations, Multiple Seeds. To avoid the optimization of longer and longer chains and to make the optimizations efficient, each of the local optimizations only optimizes the last one or two states in the controller (this approximation gives us a fixed compute time regardless of the number of states in the controller structure). Note, this approximation is not illustrated in Algorithm 5 due to the complexity of notation it would entail (but it is illustrated in Figure 5.7).

The ability of the local optimizations to re-optimize past states in the chain is critical. A process that proceeds strictly incrementally and only optimizes the last state can easily overfit to images in one stage, get stuck and fail to explain future images in later stages. To alleviate this problem, we *re-optimize previous states in the chain* given images in the expanded window. In addition, we found it effective to run multiple **BatchOp** optimizations with different seeds for stochastic sampling in CMA when executing Algorithm 5, because optimizations by CMA, in our case, do not find a global optimum.

As such, for every instance of **BatchOp** in Algorithm 5, we run six **BatchOp** optimizations: optimizing the last one or two states, each with three different seeds, and choose the best. Note, the number of recovered states in the state chain and the performance of the incremental optimization depends on the number of frames added in the optimization window in one optimization stage. We analyze this dependency in Section 5.6.3.

5.5.1.3 Controller Structure Refinement

The incremental optimization described in the previous section allows us to fit control to video or reference motion capture. The chain structure of the controller and transitions based on timing make the controller well behaved and easier to optimize. However, this control structure may not be *optimal* in terms of stability or compactness. It may also hinder the ability of the controller to generalize because the controller is not able to respond to physical events that happen in simulation. Therefore, we propose an additional refinement on the controller structure.

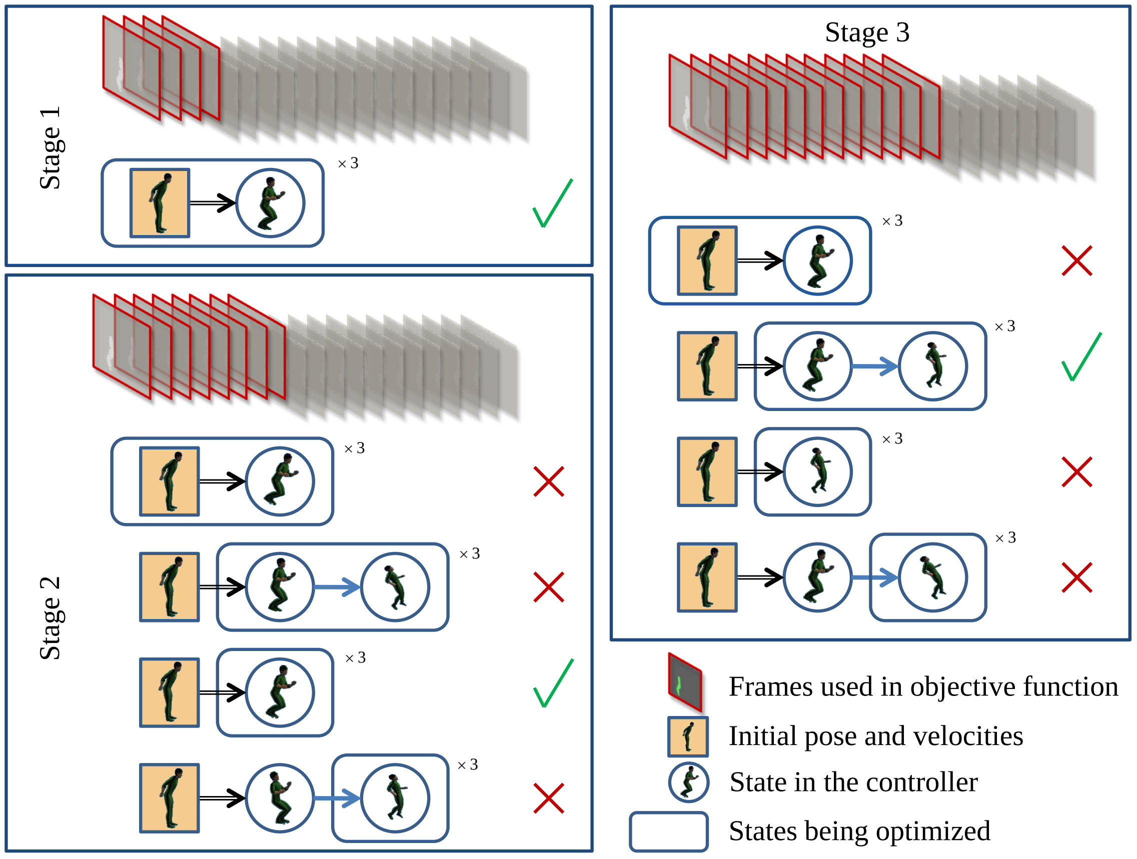


Figure 5.7: **Incremental Optimization.** Illustration of the incremental construction of the state chain. At the first stage, an initial controller consisting of a single state is defined. Parameters of this controller are optimized to explain the initial optimization window at the first stage with the first T_s frames of the input sequence. At all later stages, the optimization window is expanded by T_s frames, local changes are proposed to the structure of the current controller and parameters of the new controller structures are re-optimized. With our current implementation, the structure can only change in one way: we either add a new state to the end of the chain at this stage or we do not. Independently of whether we add the state, we also choose how many past states in the chain we are going to re-optimize during the update: either one or two. The new controller with the lowest objective value (denoted with the green check mark) is chosen to be current for the next stage. Each new controller structure is re-optimized three times to decrease the risk of getting stuck in a local minima.

We make the following observation: there exists an equivalence class of controllers all of which can simulate the same motion. For example, a one-and-a-half cycle walking controller can be represented in at least three different ways:

(1) using a chain controller with transitions base on timing:

$\mathscr{S}_6 = \{\oplus \longrightarrow \odot_1 \xrightarrow{\kappa_1=\mathscr{T}} \odot_2 \xrightarrow{\kappa_2=\mathscr{T}} \odot_3 \xrightarrow{\kappa_3=\mathscr{T}} \odot_4 \xrightarrow{\kappa_4=\mathscr{T}} \odot_5 \xrightarrow{\kappa_5=\mathscr{T}} \odot_6\}$

(2) using a chain controller with some transitions on contact:

$\mathscr{S}'_6 = \{\oplus \longrightarrow \odot_1 \xrightarrow{\kappa_1=\mathscr{L}} \odot_2 \xrightarrow{\kappa_2=\mathscr{T}} \odot_3 \xrightarrow{\kappa_3=\mathscr{R}} \odot_4 \xrightarrow{\kappa_4=\mathscr{T}} \odot_5 \xrightarrow{\kappa_5=\mathscr{L}} \odot_6\}$, or

Algorithm 6 : Complete optimization with structure refinement $[\mathscr{S}_M, \Theta, \mathbf{q}_0, \dot{\mathbf{q}}_0, E] =$ **IncremPlusRefinement** $(\mathbf{q}_0, \mathbf{Y}, \mathbf{U}, \mathbf{b})$

```
Input: Initial pose ($\mathbf{q}_0$); PCA prior ($\mathbf{U}$, $\mathbf{b}$); observations / image features ($\mathbf{Y} = \{\mathbf{y}_0, \mathbf{y}_1, \dots, \mathbf{y}_T\}$)
Output: State-space controller structure ($\mathscr{S}_M$); controller parameters ($\Theta$); initial pose ($\mathbf{q}_0$); initial velocity
    ($\dot{\mathbf{q}}_0$); objective value ($E$)
1: Incremental optimization:
       $[\mathscr{S}_M, \Theta, \mathbf{q}_0, \dot{\mathbf{q}}_0, E] :=$ IncremOp $(\mathbf{q}_0, \mathbf{Y}, \mathbf{U}, \mathbf{b})$
2: Structure transformation (for contact transitions):
       $\mathscr{S}_M' := \mathbb{T}_\perp(\mathscr{S}_M)$
       $[\Theta', \mathbf{q}_0', \dot{\mathbf{q}}_0', E'] :=$ BatchOp $(\mathscr{S}_M', \mathbf{q}_0, \mathbf{Y}, \mathbf{U}, \mathbf{b}, \Theta, \dot{\mathbf{q}}_0)$
3: if $E' < E \cdot (1+\delta)$ then
4:     $[\mathscr{S}_M, \Theta, \mathbf{q}_0, \dot{\mathbf{q}}_0, E] := [\mathscr{S}_M', \Theta', \mathbf{q}_0', \dot{\mathbf{q}}_0', E']$
5: end if
6: Structure transformation (for cyclicity):
       $\mathscr{S}_M' := \mathbb{T}_\infty(\mathscr{S}_M)$
       $[\Theta', \mathbf{q}_0', \dot{\mathbf{q}}_0', E'] :=$ BatchOp $(\mathscr{S}_M', \mathbf{q}_0, \mathbf{Y}, \mathbf{U}, \mathbf{b}, \Theta, \dot{\mathbf{q}}_0)$
7: if $E' < E \cdot (1+\delta)$ then
8:     $[\mathscr{S}_M, \Theta, \mathbf{q}_0, \dot{\mathbf{q}}_0, E] := [\mathscr{S}_M', \Theta', \mathbf{q}_0', \dot{\mathbf{q}}_0', E']$
9: end if
```

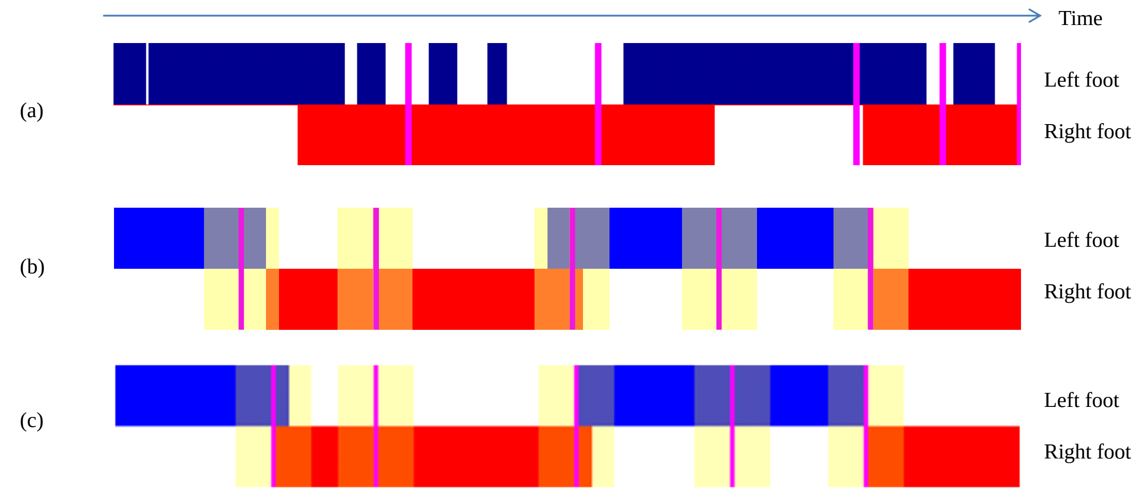


Figure 5.8: **Foot Contact Pattern.** Illustration of foot contact patterns for different optimizations of fast walk in our data set. The first row (a) shows contact pattern for optimizations from reference motion capture with the E_{contact} term disabled. Blue bars indicate when the character is supported by the left foot. Red bars indicate support by the right foot. Transitions between states are highlighted by purple bars. The second row (b) illustrates a pattern for an optimization with the E_{contact} term enabled before the refinement of the structure takes place. During the refinement, contact events that happen within 0.2 seconds of the transition (windows highlighted in yellow) are forced to happen on the corresponding contact event. The contact pattern for the refined structure and the reoptimized controller is illustrated in the bottom row (c).

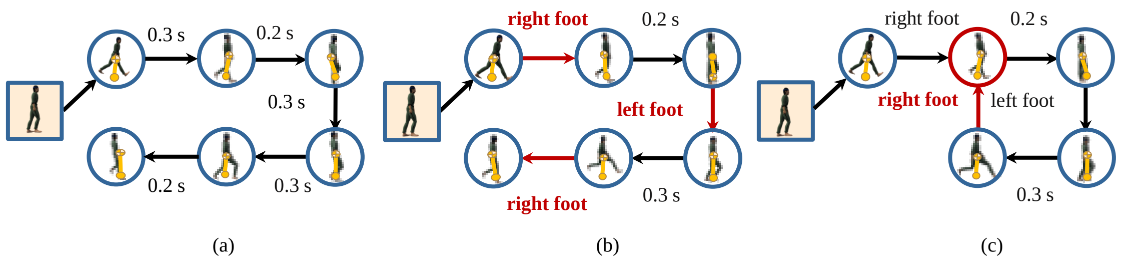


Figure 5.9: **Refinement of Controller Structure.** Illustration of the refinement of the controller structure for fast walk. The initial chain structure $\mathscr{S}_M$ in (a) is first transformed in order to replace transitions based on time with those on contact, if an actual contact event happens close to the time of the transition. The obtained controller with the structure $\mathbb{T}_\perp(\mathscr{S}_M)$ in (b) is more robust to external disturbances because the controller can react to explicitly to change of foot contact. This controller is further transformed in order to add a loop and remove duplicate states. The cyclic controller with the structure $\mathbb{T}_\infty(\mathbb{T}_\perp(\mathscr{S}_M))$ in (c) is now able to produce longer cyclic motions. See Figure 5.10 for the illustration of reconstruction results for the fast walk sequence with the cyclic controller obtained by the refinement.

(3) using a cyclic controller:
$\mathscr{S}_4 = \{\oplus \longrightarrow \odot_1 \xrightarrow{\kappa_1=\mathscr{L}} \odot_2 \xrightarrow{\kappa_2=\mathscr{T}} \odot_3 \xrightarrow{\kappa_3=\mathscr{R}} \odot_4 \xrightarrow{\kappa_4=\mathscr{T}} \odot_1\}$.
All of these controllers produce exactly the same simulation results with the same low-level motion control for actions (assuming stable gait and that transitions based on time in $\mathscr{S}_6$ are chosen coincident with contact events in $\mathscr{S}_6'$ and $\mathscr{S}_4$). However, $\mathscr{S}_6'$ and $\mathscr{S}_4$ are more robust and $\mathscr{S}_4$ is more compact in terms of representation; so in practice we would likely prefer $\mathscr{S}_4$ over the alternatives.

In the chain controllers obtained in the previous section, we often see that transitions based on time encoded in the control structure coincide with contact events (within a few frames). In fact, the contact term in our objective function from Equation (5.28) implicitly helps to enforce such transitions. We take advantage of this observation and propose two transformation functions that can take the chain controllers and transform their structures to make them more robust and compact as above (see Algorithm 6 and Figure 5.9).

Transitions on Contact. We define a transformation $\mathscr{S}_M' = \mathbb{T}_\perp(\mathscr{S}_M)$ that replaces transitions based on timing with appropriate transitions on contact in $\mathscr{S}_M$ if the timing transition is within 0.2 s of the contact (and there is only one contact change within the 0.2 s window). Because timing events often do not happen exactly on contact, but close to it (hence, the time threshold), we also re-optimize parameters using Algorithm 4 with using the Θ obtained in Section 5.5.1.2 as the initial guess. See Figure 5.9 (b) and Figure 5.8 (b) and (c) for illustrations.

Adding a Loop. Similarly, we define a transformation that greedily looks for cyclicity $\mathscr{S}_M' = \mathbb{T}_\infty(\mathscr{S}_M)$ by comparing the type of transition, target pose and control and balance parameters to previous states. We allow to add a loop from $\odot_i$ to a previous state $\odot_j$, if the states encode almost identical actions, such that $\mathbf{a}_i \approx \mathbf{a}_j$. Again, after the transformation is applied, the parameters are re-optimized with a good initial guess to account for minor misalignment. See Figure 5.9 (c).

The new transformed controllers are chosen instead of the simple chain controller if the resulting objective value is within $\delta = 15\%$ of the original. This use of δ approximates the minimum description length criterion that allows us to trade off fit to the data for the compactness of representation. The full algorithm is outlined in Algorithm 6.

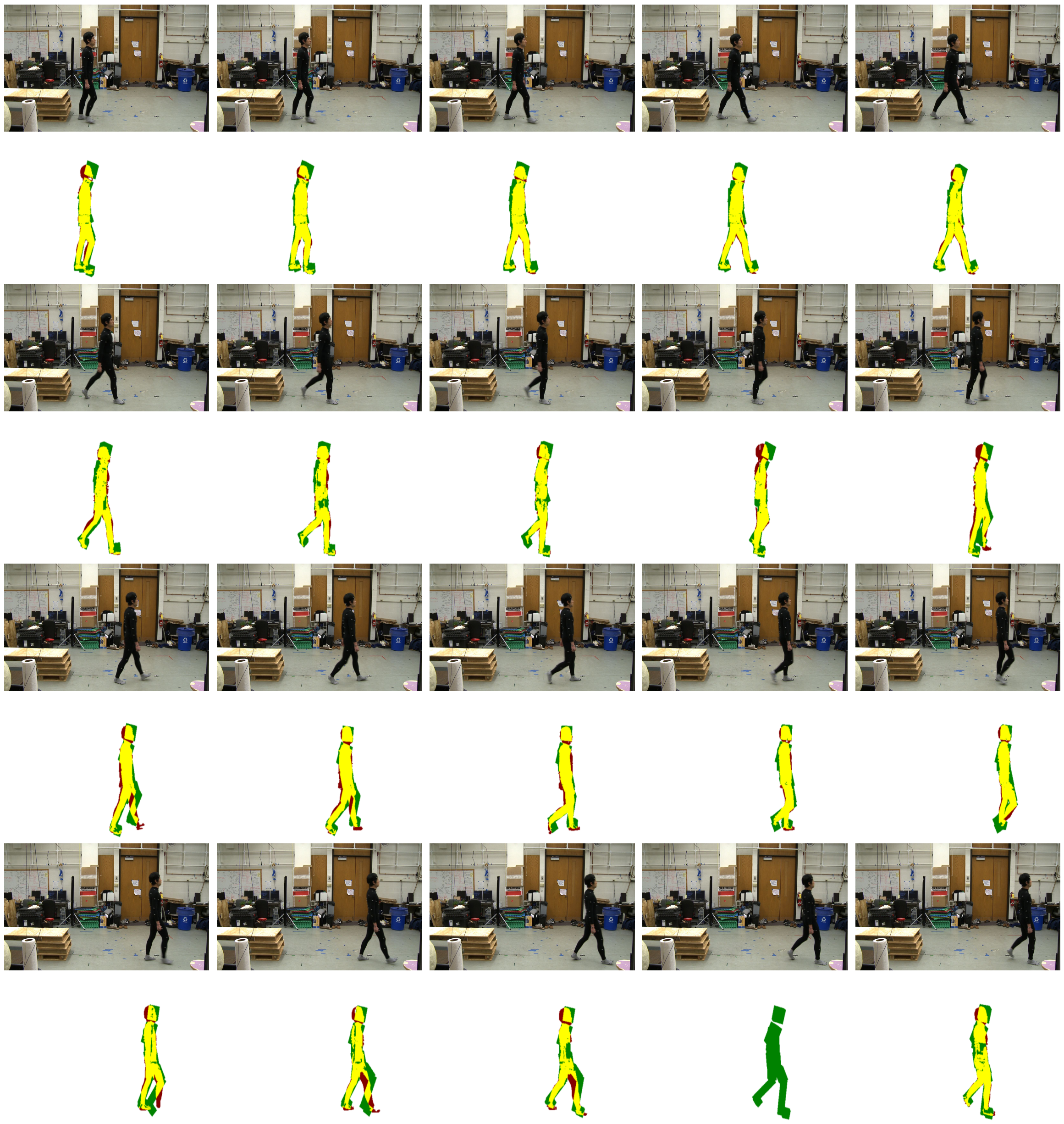


Figure 5.10: **Fast Walk from Video.** Fast walk input video and likelihoods of the reconstructed motion simulated by the cyclic controller estimated from the video.

5.6 Experiments

We have collected a dataset consisting of motions with synchronized video and motion capture data. Motion capture data was recorded at 120Hz and video at 60Hz using a progressive scan 1080P camera. To speed up the likelihood computations, we sub-sample observed silhouette images to 160×90[3]. Motions in this dataset include: jump, twist jump, spin kick, back handspring, cartwheel and fast walk. To enable comparison with previous work, we also make use of the 30Hz jump sequence from VideoMocap [201] in our experiments. For sequences where calibration is unavailable, we solve for calibration using an annotation of joint locations in the first frame, assuming a fixed skeleton for the character. While the appearance of the subject in all these sequences is relatively simple, we want to stress that we make no use of this aspect in our likelihood or optimization and only use generic foreground silhouettes as observations. All PCA models were learned from a single adult male subject[4], but are applied to other subjects in our test set (*e.g.*, jump from [201]).

We have estimated controllers from motion capture and video. We test performance of the recovered controllers in their ability to reproduce observed motions as well as to synthesize new motions in novel environments and under the effect of external disturbances. In all our experiments, we only rely on manual specification of a rough estimate of the initial pose for the character and, for the video sequences, corresponding camera calibration information. We use the same generic model of the human body with the same mass properties in all cases. We employ a marker-based error metric from [161] to evaluate accuracy of our pose reconstructions against the ground truth, and report Euclidean joint position distance averaged across joints and frames in cm. We first show results using reference motion capture as an input and then illustrate our algorithm on monocular videos.

5.6.1 Controller Optimization from Reference Motion Capture

We compare performance of the batch (**Batch**) method, that optimizes parameters given a known controller structure $\mathscr{S}_M$ (but no initialization except for the initial pose), to the incremental (**Inc**) method and the incremental method with refinement (**Refined**) that recover the structure automatically (see Figure 5.11). Despite the fact that the batch method contains more information (controller structure), our incremental method is substantially more accurate and the refinement step further improves the performance. Notice that for the back handspring, we were unable to run the batch method, as it required optimization over a prohibitive number of states and never converged. Furthermore, for a complicated motion, knowing or guessing the controller structure *a priori* is difficult. Our incremental method can estimate this structure automatically. The structure of the resulting controllers, after refinement, are illustrated in Figure 5.11 (the last column). As a result of our optimization, we automatically build controllers with varying numbers of states (from 4 to 20) and the controllers can be both linear or cyclic (as in fast walk).

[3]Subsampling does not degrade accuracy of the reconstructions given we use a simplistic shape model for the character, where we model hypothesized silhouettes as compositions of rigid boxes. We obtain as good results for optimizations from full resolution images, except that such optimizations are slower. We expect that high resolution observations would be useful, and improve reconstruction accuracy substantially, if we replaced the box model with a more faithful deformable model of the human shape.

[4]We took motion capture from the subject performing fast walk, jump, twist jump and spin kick to train our PCA models. For those sequences, we used data from disjoint trials for training and testing.

We obtain controllers that represent the motion well, even in the case of a long and complex gymnastic sequence in Figure 5.13 (top and middle). In some cases, the quantitative errors are higher; in our experience these errors are due to the sparse nature of our controller (that inadvertently approximates the motion) and timing. Despite quantitative differences, in all cases the results are dynamically and visually similar (see accompanying video at our project website at `http://brown-robotics.org/wp/projects/current/video-based-3d-motion-capture-through-biped-control/`). We are able to get a more accurate performance by reducing the length of our stages, where the solution in the limit approximates inverse dynamics. However, controllers obtained this way are less responsive and are unusable for video because they severely overfit.

Robustness to Changes in Environment. We test robustness of the recovered controllers by altering the environment, as shown in Figure 5.13 (bottom). Note that despite the dynamic environment and different geometry, the character is able to adapt and perform the desired motion. Further environment alteration tests are illustrated in the next section, where we recover controllers from video.

Robustness to Pushes. We quantify robustness of our controllers to external disturbances with push experiments; after [198, 98]. Because we have a different model for actuation and dynamics, it is difficult to match conditions to [198, 98] exactly. For this experiment, we choose a cyclic walking controller that results from optimization with reference motion capture as an input. Despite the fact that we only optimize the controller to match roughly 1.5 cycles of the motion, the controller is able to produce a mostly stable gait (it sometimes stumbles after 7 to 10 walk cycles, but always recovers afterwards). We perform a test where we apply a push force to the center of mass of the torso for 0.4seconds once every 4seconds. The controller passes the test if the character still walks after 40seconds. Our controller can resists pushes of up to 210N, 294N, 273N and 294N from front, rear, right and left sides respectively when we use relatively large joint torque bounds. When the controller has 75% of the joint torque limits (just enough for the controller to walk without substantially altering the gait), the character survives pushes of up to 42N, 67N, 33N and 42N respectively.

5.6.2 Controller Optimization from Single-view Video

Qualitative Performance. We have evaluated the ability of our method to reconstruct observed motions from single-view video in Figure 5.10, Figure 5.16, Figure 5.18, Figure 5.14. See the accompanying video for additional visualizations.

Quantitative Performance. We report quantitative performance for controller optimization from single-view video in Figure 5.12. The reconstruction error is on average only 80% higher than that of optimizations from motion capture, despite a substantially more complex problem and optimization. We see this result as very encouraging. We also see a similar trend between batch optimizations and incremental optimizations, where the batch optimizations perform considerably worse. For the batch optimizations, we report the best results out of 15 independent optimization runs. In many cases, other runs produced much inferior results. Our incremental method is much more repeatable. Unlike optimizations from reference motion capture, we see less of a benefit for structure refinement in reducing quantitative error when optimizing from video. The

Motion (mocap)	**Frames**	**Batch**	**Inc**	**Refined**	**Structure**
Fast walk	207	4.8 *cm*	2.1 *cm*	2.1 *cm*	$\oplus \longrightarrow \odot_1 \xrightarrow{\kappa_1=\mathscr{R}} \odot_2 \xrightarrow{\kappa_2=\mathscr{T}} \odot_3 \xrightarrow{\kappa_3=\mathscr{L}} \odot_4 \xrightarrow{\kappa_4=\mathscr{T}} \odot_5 \xrightarrow{\kappa_5=\mathscr{R}} \odot_2$
Jump	241	9.2 *cm*	5.6 *cm*	5.0 *cm*	$\oplus \longrightarrow \odot_1 \xrightarrow{\kappa_1=\mathscr{T}} \odot_2 \xrightarrow{\kappa_2=\mathscr{T}} \odot_3 \xrightarrow{\kappa_3=\mathscr{T}} \odot_4 \xrightarrow{\kappa_4=\mathscr{T}} \odot_5$
Twist jump	226	7.9 *cm*	5.1 *cm*	5.1 *cm*	$\oplus \longrightarrow \odot_1 \xrightarrow{\kappa_1=\mathscr{T}} \odot_2 \xrightarrow{\kappa_2=\mathscr{T}} \odot_3 \xrightarrow{\kappa_3=\mathscr{T}} \odot_4 \xrightarrow{\kappa_4=\mathscr{L}} \odot_5 \xrightarrow{\kappa_5=\mathscr{T}} \odot_6$
Spin kick	131	13.5 *cm*	8.9 *cm*	8.1 *cm*	$\oplus \longrightarrow \odot_1 \xrightarrow{\kappa_1=\mathscr{T}} \odot_2 \xrightarrow{\kappa_2=\mathscr{T}} \odot_3 \xrightarrow{\kappa_3=\mathscr{T}} \odot_4$
Cartwheel	175	25.4 *cm*	10.7 *cm*	10.7 *cm*	$\oplus \longrightarrow \odot_1 \xrightarrow{\kappa_1=\mathscr{T}} \odot_2 \xrightarrow{\kappa_2=\mathscr{T}} \odot_3 \xrightarrow{\kappa_3=\mathscr{R}} \odot_4 \xrightarrow{\kappa_4=\mathscr{T}} \odot_5 \xrightarrow{\kappa_5=\mathscr{L}} \odot_6 \xrightarrow{\kappa_6=\mathscr{T}} \odot_7$
Back handspring	775	—	4.9 *cm*	4.9 *cm*	$\oplus \longrightarrow \odot_1 \xrightarrow{\kappa_1=\mathscr{T}} \odot_2 \xrightarrow{\kappa_2=\mathscr{T}} \ldots \xrightarrow{\kappa_{18}=\mathscr{T}} \odot_{19} \xrightarrow{\kappa_{19}=\mathscr{T}} \odot_{20}$

Figure 5.11: **Controller Optimization from Reference Motion Capture.** Incremental optimization performs considerably better than batch optimization, and refinement step further improves the fit as well as results in a more robust controller structure.

objective function is multimodal in this case, with the modes far apart (a well known issue in the marker-less motion capture literature). While the incremental optimization is very effective in finding good local optima, escaping those optima is difficult through refinement.

Robustness to Changes in Environment. To illustrate the robustness of our controllers, we made a number of alterations to the environment.

Jump: For a jump (see Figure 5.16, top), we changed the geometry of the ground plane, attached 3.5kg skis to the feet and dialed down the coefficient of friction by a factor of $2,000$; these alterations emulate a very slippery skiing slope. The results of such alterations can be seen in Figure 5.1, Figure 5.17 (top) and the accompanying video. Notice that the character is able to perform the desired motion while dynamically adjusting to the changed environment and maintaining balance at the end. Similarly, we can make the character jump from a tilted plane and have it land on the level plane (see (bottom) in Figure 5.17). Despite the initial perturbation, the character is still able to land and counteract the impact.

Twist jump: In this case, we give the character a 13kg mass to carry with its right arm as it performs the jump. This modification alters the upper body motion throughout the sequence. At the end of the simulation the character leans forward to counter-balance the weight in the hand, which needs to be held behind the torso according to the last controller state. These results can be seen in the video.

Comparison to Wei and Chai [201]. We also compare the performance of our method to the one described in [201]. The two methods have a similar goal, but estimate the motion in very different ways. In [201] six keyframe annotations, specification of the contact states and four additional 2D refinement constraints have to be manually specified for the tested jump sequence. In contrast, our approach is fully automatic and requires only specification of a rough initial pose at the first frame; the contact states are recovered implicitly through optimization (see Figure 5.15). While we are not able to recover the hand motion (in part because the jumps in our PCA model are quite different from the one in video), we are able to generate a controller capable of replaying the motion with all the intricacies of the dynamics. Furthermore, because we are recovering a controller, we are able to put the character in a different environment (on a trampoline) and see the motion under physical and, in this case, dynamic perturbations (Figure 5.14). Our fully automatic method achieves average reconstruction errors as low as 4.2cm which is comparable to the average error of 4.5cm reported in [201].

Motion (video)	**Frames**	**Batch**	**Inc**	**Refined**
Jump	241	9.5 *cm*	6.8 *cm*	6.8 *cm*
Twist jump	226	11.9 *cm*	9.6 *cm*	9.6 *cm*
Back handspring	154	33.3 *cm*	17.6 *cm*	17.5 *cm*
Fast walk	207	6.7 *cm*	4.2 *cm*	4.2 *cm*
VideoMocap jump	91	N/A	N/A	N/A

Figure 5.12: **Controller Optimization from Monocular Video.** Incremental optimization performs considerably better than batch optimization.

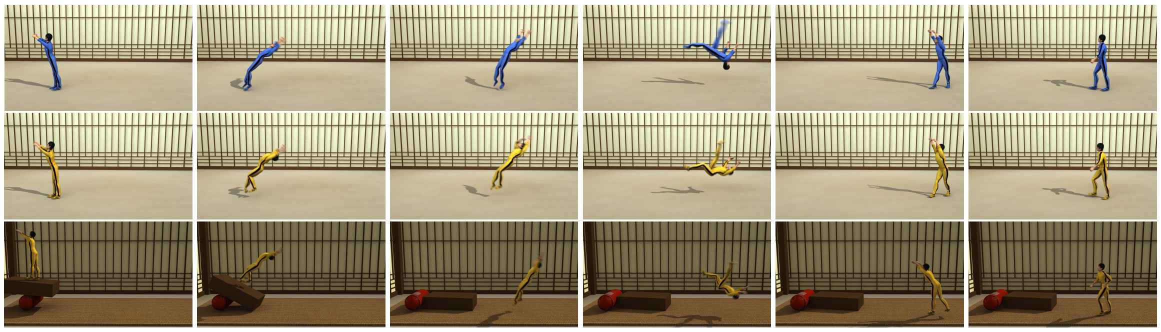


Figure 5.13: **Motion Capture Result.** Input reference motion capture (top), reconstructed motion simulated with our controller (middle), simulated motion in a new environment (bottom).

5.6.3 Number of Stages and States

The performance of the controller and the incremental optimization in Section 5.5.1.2 depend on the number of stages N_{STAGES} in the optimization. In this section, we analyze the behavior of the optimization as a function of $N_{STAGES} = T/T_s$ and T_s. Note, at each stage of the optimization, at most one new state is added to the end of the current state chain and the state is added greedily as long as the addition improves the fit. Consequently, the length of the chain can vary between 1 and N_{STAGES}, where N_{STAGES} refers to the number of stages in the optimization. Besides this theoretical bound, we demonstrate the actual numbers of recovered states as well as the reconstruction errors that we obtain with our optimization algorithm for optimizations with different values of T_s. We perform these experiments on fast walking from our data set, because walking is sensitive to the variation of T_s due to frequent changes of contact. We show our results in Figure 5.19.

Optimizations from Single-view Video. When performing optimizations from video, we found shorter stages improve reconstruction accuracy. However, very short stages tend to produce controllers that overfit, such that only a few frames from the video contribute to the optimization of a single state in the chain. Optimizing states in this case is difficult, because our image observations are noisy and only a few frames can be used to constrain the estimation. Furthermore, we also can not perceive the contact state of the body reliably and detect when contact events happen. Consequently, a character performing a walk may end up stumbling and skipping instead of locomoting properly. This behavior of optimizations matches our initial expectations, because controller capture approaches the method from Chapter 4 when stages are very

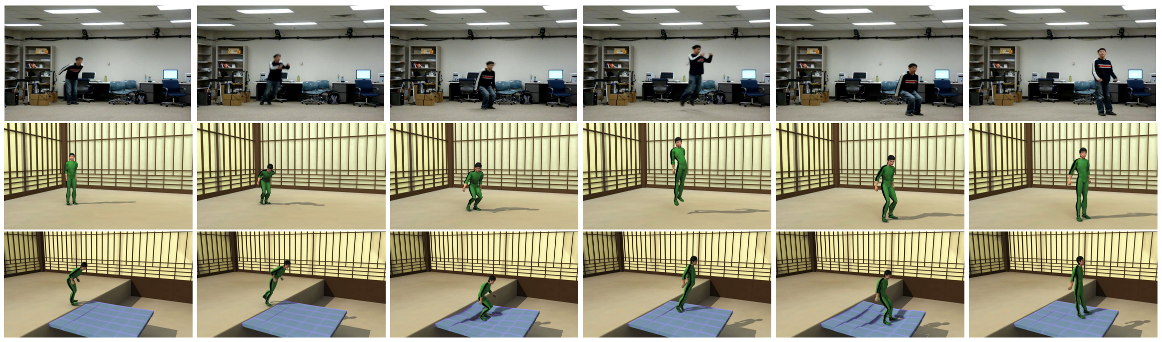


Figure 5.14: **Video Result: Jump from VideoMocap.** Input jump sequence from VideoMocap (top), reconstructed motion simulated by the controller estimated from the video (middle), new motion simulated in a modified environment (bottom).

short, replacing the sparse parameterization of the desired motion with a high-dimensional dense sequence of desired poses with a desired pose specified for every frame in the video. With that said, our key motivation for controller capture is to exploit longer stages and sparse motion parameterizations so that information from many frames can reinforce optimizations of states. Note that too long stages, on the other hand, hinder the ability of the controller and optimizer to fit the motion. The representation of the desired motion for control is either unable to represent the motion at a sufficient level of detail, in that case, or the optimizer fails to find a control strategy to reproduce motion in such a long window.

Based on our experience, we set the number of stages to $N_{STAGES} = T/T_s$ such that we add $T_s = 40$ new frames to the optimization window at each stage. This decision corresponds to expanding the optimization window by roughly 0.3s of video data. We summarize this discussion and analyze the performance of the optimization process as a function of T in Figure 5.19 (left).

Optimizations from Reference Motion Capture. Optimizations from reference motion capture data are slightly different (see Figure 5.19, right). With the full observability of the state and no observation ambiguities, which the reference data ensures, optimizations with shorter stages do not have problems with local optima and in the limit approach inverse dynamics. However, short stages may produce controllers that are less responsive such that they can not much deviate from the desired motion. This aspect may limit the ability of the controllers to generalize. Note that motion capture optimizations self-regularize the numbers of used states, keeping the numbers low. We hypothesize that this behavior is caused by the fact that each state in the chain induces a change of body actuation, which introduces discontinuity in control, generated joint torques and the body momentum. However, our energy term from Equation (5.29) in the optimization objective penalizes motions that are not smooth, suggesting new states should only be added when necessary. Note that while we could use the minimum description length criterion to explicitly request a tradeoff between the compactness of the representation of the controller and the accuracy of the fit, our current implementation does not employ this technique and rather relies on the smoothness property. We use $T = 20$ for motion capture optimizations. See Figure 5.19 (right) for an illustration.

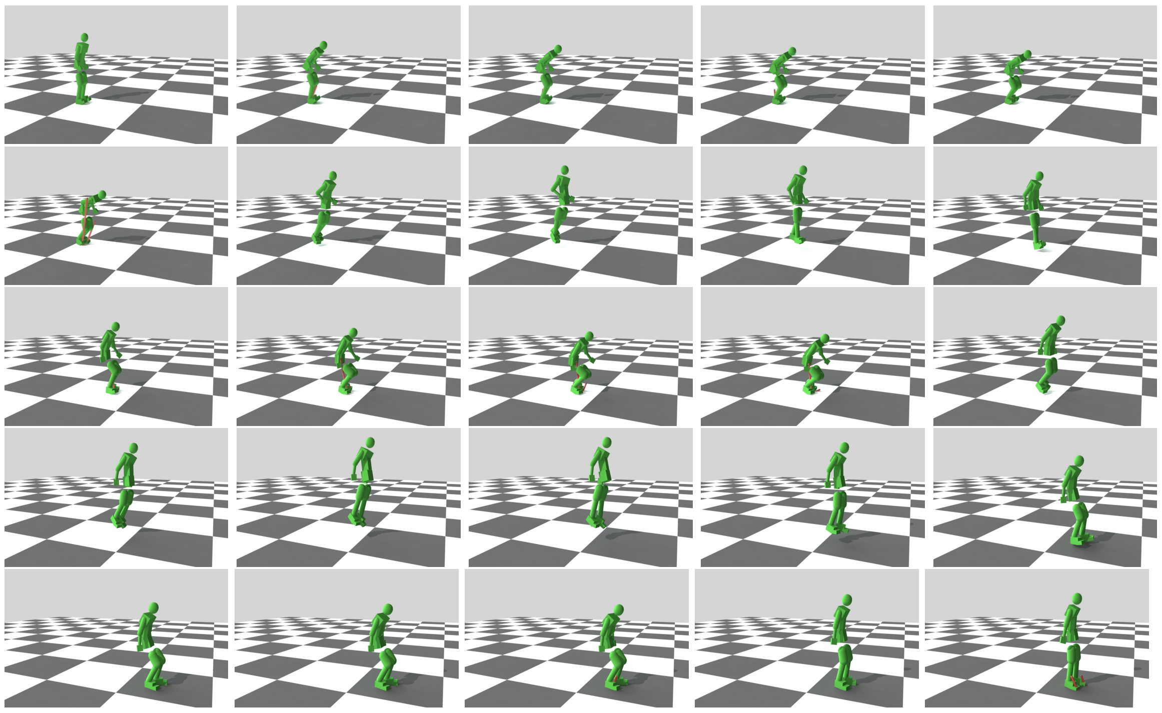


Figure 5.15: **Video Result: Recovered Motion and Contact Forces for Jump from VideoMocap.** Illustration of the recovered motion and contact forces on the jump sequence from VideoMocap. Note that our method, unlike VideoMocap [201], recovers the poses, contact forces and foot contact automatically without user supervision and requires only specification of a rough estimate of the initial pose. See Figure 5.14 for the input video.

5.6.4 Computation Time

Incremental optimization for 200 frames of motion capture data takes about 2.75 hours; the refinement procedure takes 15 minutes (single eight-core machine with a 2.8GHz Core i7 CPU and 4GB of RAM) — approximately 3 hours total. Video optimizations take 40% longer due to use of more expensive likelihoods.

5.7 Discussion

We present a novel approach for capturing bipedal controllers directly from single-view video and in the process of reconstructing the human motion from a single video stream. In doing so we simultaneously tackle two very challenging problems: bipedal control from noisy data and marker-less motion capture. We believe that by formulating the problem in such a way as to solve the two problems simultaneously, we are better leveraging the information present in the video of human motion and thereby simplifying the marker-less tracking problem. We are able to estimate controllers for a variety of complex and highly dynamic motions. Controllers from both sources of data are robust and can be used in new environments with novel terrain and dynamic objects.

While we are able to estimate controllers that are capable of generating motion, for a variety of human

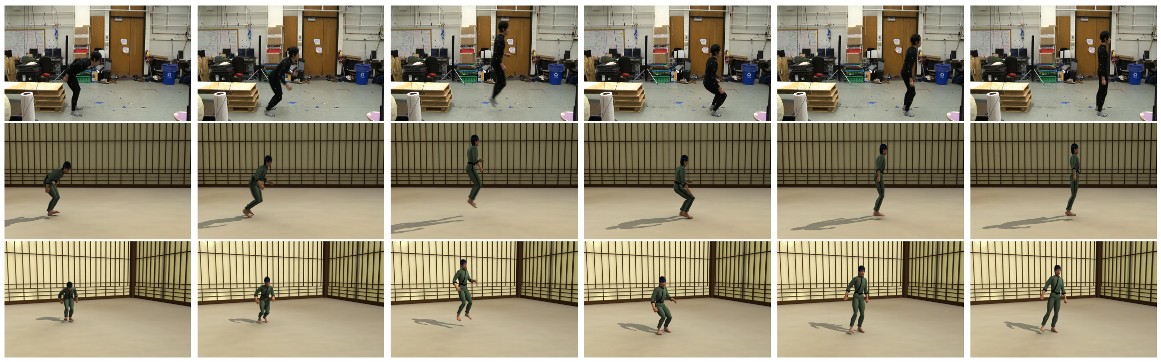


Figure 5.16: **Video Result: Jump (Pose Reconstruction).** Input jump sequence (top), reconstructed motion simulated by the controller estimated from the video as rendered from the camera view (middle), reconstructed motion rendered from a different view (bottom). See Figure 5.17 for the adaptations of the recovered motion to new environments.

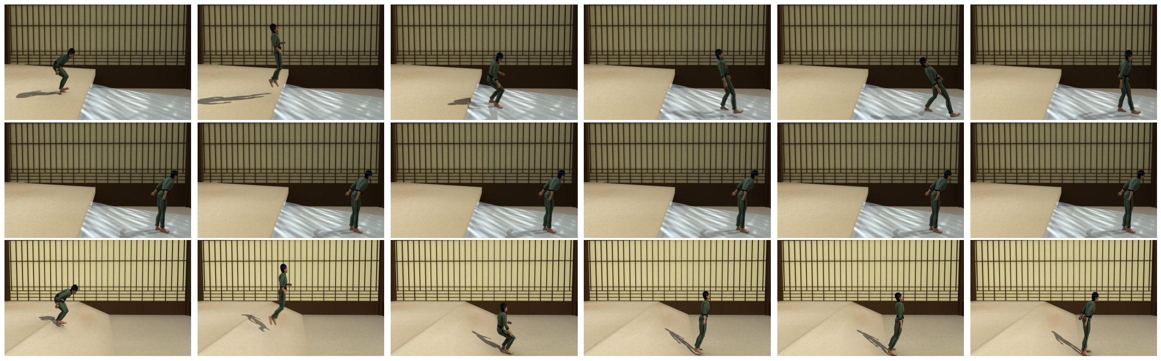


Figure 5.17: **Video Result: Jump (Adaptation).** Adaptations of the recovered jump motion from Figure 5.16 to two new environments: a jump on a tilted slippery surface (top), and a jump from a tilted ground (bottom). Notice the difference in the time durations of the adapted motions.

behaviors, some of the subtleties in the motion are not accurately reconstructed; the motions tend to look robotic and overpowered. We believe there are three main sources of errors that are contributing to these artifacts. First, the end effectors (feet and hands) are hard to observe in video and are not well constrained by the image likelihood model. From a sagittal view, it is also difficult to disambiguate single stance from double stance. Second, parameters found by CMA are not guaranteed to be optimal, because we are solving a very difficult high-dimensional optimization problem. Third, we believe the overpowered balancing behavior observed is due to relatively high joint torque bounds — our character by design has very strong joints, including ankles and knees. Higher bounds make the optimization easier. Improvements could probably be achieved for locomotion and other highly dynamic behaviors by having more sophisticated objectives that include biomechanical constraints, as in [199]; we plan to explore these objectives in the future.

At present, we loosely limit joint torques. Consequently, some of the strategies adopted by the character to accomplish a task require unnaturally strong muscles at, for example, the ankle joint. And finally, our

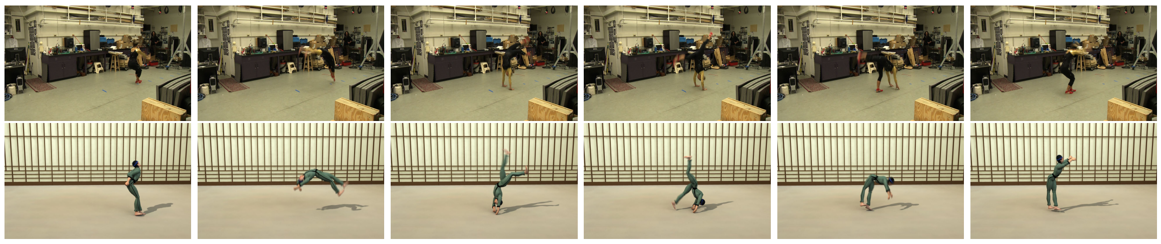


Figure 5.18: **Video Result: Back Handspring.** Input sequence (top), reconstructed motion simulated by the controller estimated from the video (bottom). Despite this reconstruction is obtained from only a single camera view and the image observations are low-quality and blurry, we are able to capture the essence of the motion.

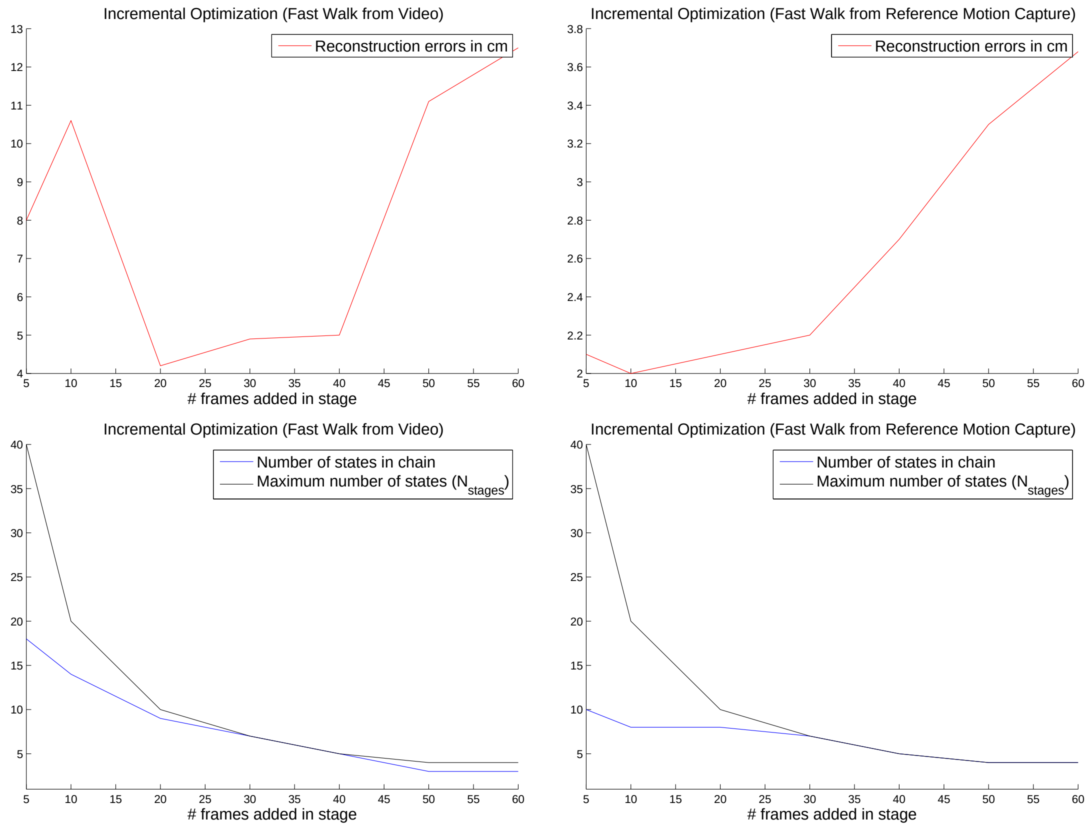


Figure 5.19: **Performance of Incremental Optimization.** Illustrated is the performance of incremental optimization on fast walk when optimized using the incremental approach without refinement. Reconstruction errors and the numbers of states in the chains are reported for optimizations from single-view video on the left and from reference motion capture on the right as a function of number of frames added in the optimization window in one stage.

simple model of a block character may be hindering the quality of the motion that we can obtain from the video. For example, foot contacts with the ground may happen at slightly misplaced locations because the model's feet do not match the much smaller feet of the subject.

While we took a puristic approach of only using single-view video, in practice multiple views (or Xbox's Kinect data) will undoubtedly simplify the problem and lead to an improved ability to capture 3D motion. While we do not utilize user-in-the-loop constraints, as we opted for a completely automatic method, those can easily be added to further refine the motions as was done by Wei and Chai [201]. Preliminary experiments show that we can indeed achieve better performance by having the user click on the joint locations in the image. Our experiments with motion capture data (perfect 3D data) further demonstrate this point.

The PCA priors that we introduced for representing the target poses for the state proved to be quite useful. However, they are limited to the motions we trained them on and not all stylistic variations can be captured by a discrete set of such learned models. In the future, we hope to explore refinement schemes where we can estimate controllers with the motion prior and then perhaps refine them by removing the prior term from the objective.

Chapter 6

Summary and Discussion

In this chapter, we summarize our formulation for vision-based motion capture and highlight the most important contributions. We also discuss the current challenges and provide more broad future directions for general research that can be based on this work.

6.1 Summary

In this thesis, we have proposed a new paradigm for vision-based human motion capture. This paradigm goes beyond traditional capture of kinematic poses and extends pose capture in a couple of aspects. First, it provides guarantees of physical plausibility for the motion reconstructions, where physical plausibility means that the reconstructions can be directly reproduced in simulations by applying actuation forces on the model of the person. In practical terms, physical plausibility implies that the estimated motions are smooth and can be realized by a humanoid character without falling, sliding or having to fly. Second, the paradigm provides mechanisms for adaptation of the captured motions to new environments. Adaptability allows to ask "what if" questions and hypothesize how the estimated motions could look like if they were captured in different environments or with different actors. We achieve these benefits by estimating controllers for simulated physics-based characters from (potentially monocular) images.

Controllers encode motions implicitly, based on their "underlying physical principles" and reconstruct the observed motions through simulation. On one hand, simulations with controllers in the original environments provide physically plausible interpretations of the original motions. On the other, contact feedback within the controllers allows application of the captured principles in modified environments, providing an ability to adapt the motions to external events, perturbations, and different characters. Both these benefits improve current applications for vision-based motion capture (*e.g.*, more accurate reconstruction without artifacts) as well as open possibilities for future applications (*e.g.*, direct control of physics-based characters in video games, building of motion libraries for character skills, *etc.*).

Formulation. We have formulated motion estimation as an optimization over controllers such that motions in the video are reproduced. In following our approach, we simultaneously address two difficult problems:

controller estimation from noisy image data and marker-less motion capture. We believe that by integrating these problems together, we are better leveraging the information present in the video of human motion and thereby simplifying the marker-less motion capture problem. We are able to estimate controllers for a variety of complex and highly dynamic motions using this approach.

In our formulation, we assume the observed motion is structured and can be generated by physical simulation of a character model with actuation forces produced by a controller. Our approach to motion estimation takes as an input a sequence of images from one or more calibrated cameras and estimates a controller for the observed motion. This controller implicitly encodes the underlying structure of the motion and is estimated such that the motion synthesized by the controller explains pose evidence in the images. We currently use rigid body approximations of human dynamics and methods for humanoid control to make our approach tractable.

We have applied this approach to simulated physics-based characters with two kinds of controller, trajectory tracking controllers and state-space controllers. These controllers differ in the ways they parameterize desired motion for control, which results in different controller optimization problems, different challenges and different abilities of the obtained controllers to generalize.

Dense Motion Parameterization: Trajectory-tracking Control. We first explored the trajectory tracking controller model that parameterizes the desired motion as a sequence of desired poses specified for every frame in the video. We use a Particle Filter implementation to recover the desired poses in this sequence incrementally from image observations and then simulate the character model towards them. We can recover the desired pose sequence incrementally and produce pose estimates that do not suffer from common artifacts such as excessive jitter, out-of-plane rotations or foot skate. With that said, the process is prone to overfitting and results in a high-dimensional dense representation of the desired motion for control that requires strong image likelihoods for inference.

Because a desired pose for the character has to be estimated for every frame, we employ an exemplar-based pose prior that biases desired pose trajectories to motions observed during training in order to make the inference easier. The method can be seen as a filter built on top of any statistical motion model (kernel regression, in our case) that corrects pose predictions from the statistical model using simulation so that the predictions are physically plausible. Despite this prior, our likelihoods and inference are not able to estimate desired poses accurately enough such that we could generate the motion interpretation by raw simulation with the estimated controller. Although the individual predictions made by simulation in the incremental Particle Filter (PF) are guaranteed to be physically plausible in our method, we still need to add noise to the poses generated by simulation in order to achieve a better fit to observations and avoid overfitting by the PF. These adjustments translate and rotate the body and introduce visual artifacts. By adding the noise, we also lose the ability to generalize the captured motion to new situations.

Sparse Motion Parameterization: State-space Control. In order to allow motion reconstructions that are raw results of physical simulations, we next replace our dense trajectory-tracking controller model with a sparse state-space controller and use direct optimization for inference. The sparse controller model represents desired pose trajectory for the character as a piece-wise constant function, where constant segments correspond to different constituent phases in the motion selected by an associated state machine. This abstraction is not only more compact and captures the underlying structure of the motion intrinsically, but is also easier to estimate from images and enables robust incremental inference.

Specifically, the assumption of a structured motion makes the inference better constrained. Because the controller structure is sparse, we are able to integrate information locally from multiple (tens of) image frames which induces smoothness in the resulting motion and resolves some of the ambiguities that arise in monocular video-based capture, without requiring a user-in-the-loop. The use of this sparse representation also allows inference that is not as heavily biased by the noise in the individual video frames. Despite the fact that this is a sparse representation, however, it still allows the expressiveness necessary to model variations in style and speed that we may observe in video. Lastly, the direct inference mechanism allows to correct earlier misestimations (for already processed frames) when later image observations are available through re-optimization, unlike the previous motion estimation method with a Particle Filter. Consequently, the process is less suspectible to overfitting and we do not need to add noise to the poses produced by the simulation.

The sparse controllers that our approach estimates can be applied on characters under perturbations to generate motion adaptations. The adaptations are enabled by the structure of the controllers that allows the motion to deviate from the desired pose trajectory and by the feedback within control that can respond to certain physical events that happen in the simulation (*e.g.*, foot contact).

Relation between Control Models. The estimation of controller with the trajectory-tracking control model can be interpreted as a special case of the state-space method, both in terms of implementation and its properties, where actions in the trajectory control switch regularly with every frame and recovery of the controller is implemented using incremental optimization over an expanding window with one frame increments between optimization stages. According to our experiments, using short optimization stages for state-space controllers degrades estimation accuracy and re-introduces the problems from trajectory tracking.

Results. We illustrate in our results that by computationally accounting for the physical properties of the human body, environment and control, our approach improves quality of recovered poses and results in physically plausible performance (*e.g.*, interaction forces explicitly prevent physically impossible hypotheses and enable more appropriate reactions to the environment). We also evaluate our approach quantitatively. We illustrate the tracking performance of our methods on a number of standard sequences, such as HumanEva, using standard error metrics, as well as on custom sequences involving highly dynamic floor gymnastics and jumping and compare the attained accuracy to the state-of-the-art. Our methods attain better or comparable results, but achieve more plausible motion interpretations.

6.2 Open Problems

While we are able to estimate controllers that are capable of generating the motion in video, some of the subtleties in motions are not accurately reconstructed and our results contain certain artifacts, such as the lack of detailed arm movement or stiff/robotic appearance. We believe there are two main sources of errors that are contributing to these artifacts: observation models and high-dimensionality of the optimization problem. We discuss these two sources next.

Observations. For the most part, the abovementioned subtleties that are not captured pertain to the end effectors and to the smaller body parts (head, hands or feet). This limitation is the result of the simple image observation models that we use. These parts are hard to observe in video and their motion is not easily distinguishable in the silhouette observations. Consequently, our image features do not provide sufficient constraints for the motion reconstruction process and physical laws and force bounds constraints alone are not rich enough to capture the need for their movement.

Improvements could probably be achieved by using more sophisticated objectives that include biomechanical objectives, as in [199], or by using stronger observation models [6] that model appearance of body parts more faithfully. We could also supplement/estimate the information missing from the observations by using stronger priors over valid poses that can be reconstructed, such as those previously used in computer vision literature to restrict direct search for poses during tracking. To support these hypotheses, we note that optimizations from reference motion capture data produce substantially better results, both quantitatively and qualitatively. In the case of motion capture data, we have the courtesy to work with clean unambiguous motion capture observations. This fact lets us further penalize joint torque expenditure in the optimization objective and estimate controllers that use less power and generate smoother motions. By using better observation models and/or additional constraints, we hypothesize we should be able to produce as good reconstructions directly from video.

High Dimensionality. Second, because we are solving a very difficult high-dimensional optimization problem, controller structures and their parameters found by either PF or CMA are not guaranteed to be optimal. Consequently, the motions we recover may appear robotic, overpowered and stiff. To address the high-dimensionality of the general inference problem, we employed a sparse controller model that uses compact piecewise-constant representations of desired pose trajectories and a PCA space model to represent the desired poses. However, the resulting space might still be too high-dimensional for the current optimization model. To further reduce dimensionality of the problem, we may need to impose stronger priors over the motion that can be reconstructed. For example, we may penalize individual poses in the reconstructed sequence if the poses deviate substantially from training motion capture data.

Furthermore, in order to not restrict controller optimizer much in finding feasible strategies for executing the observed motion (because the optimization is performed in a high-dimensional space), we define looser bounds for actuation forces. Consequently, some of the strategies adopted by the character to accomplish a task may require unnaturally strong muscles at, for example, ankles and knees and may produce motions that appear unrealistic. That said, such high force bounds make the optimization possible and are required in our current implementation so that the optimizer can find a feasible strategy for the task. We plan to include an iterative re-optimization that would gradually reduce the force bounds and re-optimize the controller with smaller and smaller force bounds.

6.3 Future Directions

We envision a few potential research directions for extending the “motion capture through controller capture” paradigm and its implementation that we have demonstrated. We think of two general directions. One, that attempts to improve motion capture and focuses on the vision aspect of the paradigm, and another, that further develops the idea of controller capture and goes beyond motion capture even further.

In order to improve the motion capture aspect of the paradigm and build a motion capture system that can rival the performance of current commercial solutions in estimation accuracy, it may be necessary to model appearance of the person in images more faithfully as well as to use a more faithful physical model of the person and a more advanced control model. For the same reason, we may also need to model dynamic objects in the environment in order to enable motion estimation in dynamic environments. Furthermore, in order to achieve specific goals beyond pose capture, we may want to integrate additional constraints with the controller estimation.

Shape and Appearance. For example, rather than using boxes to represent the shape, we may need to adopt a deformable shape model that is capable of capturing fine details in the deformation of the flesh in the human body as well as clothes. Such detailed models [14, 57, 110], based on statistical principles, have been used in computer vision literature before and have been demonstrated to be important for the improvement of the pose and motion estimation. For better modeling of the appearance, it may be necessary to combine shape estimation, pose estimation and foreground silhouette segmentation [65, 110]. We can also use additional image features that are more robust to changes in illumination and appearance of the person, and that allow operation in cluttered environments [142, 6].

Character Model. With respect to the simulated model of the body, we may want to model feet and hands with a greater detail (and perhaps as soft bodies) so that contact with environment (or objects within the environment) can be modeled in a more biomechanically faithful way. We may also want to simulate clothing in order to feed the shape and appearance models, as well as motion blur in the appearance of the body in images due to fast moving motions.

Furthermore, we can extend the repertoir of simulation events that the controller can react to. Ideally, all state transitions within a controller should be conditioned on certain physical events, and not just based on the time, so that the controller can better generalize and respond to those events exactly when they happen. Modeling of better contact and/or ability to grip may be necessary for capturing people manipulating objects in the environment (*e.g.*, picking up a mug) or just performing actions in environments that require the use of hands (*e.g.*, bar exercises in gymnastics, climbing a ladder). In order to better approximate desired motion for control, we can also refine parameterization of the desired motion, where we integrate both sparse and dense parameterization. For example, we can first fit a sparse state-space controller to obtain a base approximation of the motion and then layer a dense “style” controller on top of the base controller to encode the fine details of the pose trajectory. We can also combine reactive control with more deliberate motion planning and body actuation for better balance, such as [123]. We can also employ more realistic balancing strategies [113, 43] that do not implement balance by adjusting target poses but explicitly control the momentum of the body.

Environment Model. Our current implementation of controller capture is limited to static environments with known geometry observed by static calibrated cameras. Another research direction can focus on joint estimation of the motion, physical properties of the environment (*e.g.*, gravity, friction, as was proposed in [28]) and the geometry of the environment. We may also want to explore appearance models that do not require explicit camera calibration and permit non-stationary cameras [6].

In our approach, we have demonstrated the importance of modeling the effects of the environment on the motion of the body. We can bring this concept further and model dynamic objects in the environment. Such modeling would enable physically plausible motion estimation of subjects manipulating objects in an environment or subjects navigating dynamic environments (*e.g.*, walking on a trampoline or jumping of a diving board). In considering these cases, we are motivated by an example of a person carrying a heavy suitcase. The motion of the person, in this case, is largely determined by the weight and shape of the suitcase. An ability to model these factors directly would not only provide precise constraints for the motion reconstruction, of both the person and the suitcase, but also allow, *e.g.*estimation of the weight of the suitcase. This feature can be important for applications in surveillance.

Input Modalities. In our current implementation, we have only considered two input modalities: images (in the form of pre-segmented foreground silhouettes) and reference motion capture markers/poses. In principle, there is nothing that limits this method to these modalities and we can employ the same approach to *e.g.*, estimate controllers from depth maps (*e.g.*, Kinect data) and other sources.

User in the Loop. While estimating controllers, we may want to incorporate additional constraints specified by an user in order to constrain the estimation process. For example, we may want to realize a specific motion that can not be demonstrated by an actor directly, because it is too dangerous, costly or not technically feasible. To address this problem, the actor may demonstrate a different motion that is "close enough" and use constraints to directly edit the demonstrated motion as appropriate, as a part of the controller estimation process, such that the intended motion is achieved. As such, controller capture could be used as a motion editing tool.

Generalization of Control. Motion adaptions generated by our methods are obtained through application of the captured control mechanisms to new situations. In many cases (and as demonstrated in our experiments), these mechanisms produce motions that do not appear human-like. Such unplausible behaviors can be, at least in part, attributed to the fact that our control mechanisms do not adapt to the new situations and make use of full actuation power to recover from disturbances, unlike humans. Consequently, when giving answers to "what-if" questions, we are not showing what a human would do in the particular situation, but rather what our control strategy would do. Both can be different. Often, the control strategy does not know how to react appropriately on the situation. Depending on the disturbances and how big they are, the generated reactions may either look good or bad.

We believe these aspects can probably be addressed in a few possible ways. Because our controllers are reactive, and operate by reacting on certain events and based on balance feedback, we may try to expose the controller to typical disturbances and events in training (during the controller estimation) that can happen later during testing (during motion adaptation) so that the controller would learn how to react to these situations. For example, we may perturb the simulated character with random forces applied at the various parts of the body at random times and use the perturbations as a base for learning of the balancing mechanism, as in [199]. In order to make this strategy effective, we may need to introduce additional types for events and transitions in the state machine so that the controller can respond to various features of the motion.

We may also use more biomechanically accurate methods for body actuation, such that the employed low-level motion control strategies would directly correspond to the activations of real muscles in the human body. Specifically, rather than applying joint torques for the actuation of the humanoid character model with motors at joints, we may use *e.g.*, biologically-inspired muscle actuators [200]. These actuators can better approximate muscles in the human body, and, consequently, force the controllers to explore actuation strategies that are similar to those of humans [200].

In a more general way, and as a grand goal, we may attempt to take the observed motion as a single demonstration of a desired behavior and use techniques from learning from demonstration in order to recover the objective of the motion. We can then use this objective for motion generation. Given the current progress in the techniques in learning from demonstration though, as of now, this goal is, however, very far. Nevertheless, it may provide a good motivation for future research.

6.4 Conclusions

The core objective of this thesis was to demonstrate that it is technically feasible to realize marker-less motion capture from a single camera view, as defined in our problem statement, and with the ability to produce physically plausible motion interpretations that can generalize beyond the estimated motion. We achieved this objective by learning controllers for the observed motions and we have proven our thesis statement, which we re-state here again for clarity:

> Physical models of human dynamics and methods for humanoid control can be used to estimate physically plausible human motions from image sequences, where physically plausible refers to motion reconstructions produced by physical simulations of a character model from an initial pose.

Possible directions for future research include improving the features of the current method towards building a fully functional general vision-based motion capture system as well as focusing on controller estimation aspect of the method in order to provide new applications.

Bibliography

[1] P. Abbeel, D. Dolgov, A. Ng, and S. Thrun. Apprenticeship learning for motion planning with application to parking lot navigation. In *Intelligent Robots and Systems, 2008. Proceedings. 2008 IEEE/RSJ International Conference on*, pages 1083–1090, September 2008.

[2] A. Agarwal and B. Triggs. Recovering 3d human pose from monocular images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(1):44–58, January 2006.

[3] B. Akgun, M. Cakmak, J.-W. Yoo, and A. Thomaz. Trajectories and keyframes for kinesthetic teaching: A human-robot interaction perspective. In *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, pages 391–398. ACM, 2012.

[4] M. Andriluka, S. Roth, and B. Schiele. People-tracking-by-detection and people-detection-by-tracking. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. Proceedings of the 2008 IEEE Computer Society Conference on*, pages 1–8, June 2008.

[5] M. Andriluka, S. Roth, and B. Schiele. Pictorial structures revisited: People detection and articulated pose estimation. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. Proceedings of the 2009 IEEE Computer Society Conference on*, pages 1014–1021, June 2009.

[6] M. Andriluka, S. Roth, and B. Schiele. Monocular 3d pose estimation and tracking by detection. *Computer Vision and Pattern Recognition, 2010. CVPR 2010. Proceedings of the 2010 IEEE Computer Society Conference on*, pages 623–630, 2010.

[7] M. Andriluka and L. Sigal. Human context: Modeling human-human interactions for monocular 3d pose estimation. In F. Perales, R. Fisher, and T. Moeslund, editors, *Articulated Motion and Deformable Objects*, volume 7378 of *Lecture Notes in Computer Science*, pages 260–272. Springer Berlin Heidelberg, 2012.

[8] B. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469 – 483, 2009.

[9] O. Arikan and D. Forsyth. Interactive motion generation from examples. *ACM Transactions on Graphics (TOG)*, 21(3):483–490, July 2002.

[10] O. Arikan, D. Forsyth, and J. O'Brien. Motion synthesis from annotations. In *ACM Transactions on Graphics (TOG)*, SIGGRAPH '03, pages 402–408, New York, NY, USA, 2003. ACM.

[11] N. Badler and S. Smoliar. Digital representations of human movement. *ACM Computing Surveys (CSUR)*, 11:19–38, March 1979.

[12] A. Balan, M. Black, H. Haussecker, and L. Sigal. Shining a light on human pose: On shadows, shading and the estimation of pose and shape. In *Computer Vision, 2007. ICCV 2007. 2007 IEEE International Conference on*, pages 1–8, October 2007.

[13] A. Balan, L. Sigal, and M. Black. A quantitative evaluation of video-based 3d person tracking. In *IEEE VS-PETS*, pages 349–356, 2005.

[14] A. Balan, L. Sigal, M. Black, J. David, and H. Haussecker. Detailed human shape and pose from images. In *Computer Vision and Pattern Recognition, 2007. CVPR 2007. Proceedings of the 2007 IEEE Computer Society Conference on*, pages 1–8, June 2007.

[15] J. Bandouch, F. Engstler, and M. Beetz. Accurate human motion capture using an ergonomics-based anthropometric human model. In F. Perales and R. Fischer, editors, *Articulated Motion and Deformable Objects*, volume 5098 of *Lecture Notes in Computer Science*, pages 248–258. Springer Berlin / Heidelberg, 2008.

[16] D. Baraff. *Dynamic Simulation of Non-penetrating Rigid Bodies*. Phd dissertation, Cornell University, 1992.

[17] D. Baraff. Fast contact force computation for nonpenetrating rigid bodies. *ACM Transactions on Graphics (TOG)*, pages 23 – 34, July 1994.

[18] D. Baraff. Linear-time dynamics using lagrange multipliers. In *ACM Transactions on Graphics (TOG)*, SIGGRAPH '96, pages 137–146, New York, NY, USA, 1996. ACM.

[19] D. Baraff, A. Witkin, and M. Kass. An introduction to physically-based modelling. *SIGGRAPH Course Notes*, 1997.

[20] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(4):509–522, April 2002.

[21] D. Bentivegna. *Learning from Observation using Primitives*. PhD thesis, Georgia Institute of Technology, 2004.

[22] L. Bo and C. Sminchisescu. Twin gaussian processes for structured predictions. *International Journal of Computer Vision*, 87(1):28–52, 2010.

[23] L. Bo, C. Sminchisescu, A. Kanaujia, and D. Metaxas. Fast algorithms for large scale conditional 3d prediction. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. Proceedings of the 2008 IEEE Computer Society Conference on*, pages 1–8, June 2008.

[24] C. Bregler and J. Malik. Tracking people with twists and exponential maps. In *Computer Vision and Pattern Recognition, 1998. CVPR 1998. Proceedings of the 1998 IEEE Computer Society Conference on*, pages 8–15, June 1998.

[25] C. Bregler, J. Malik, and K. Pullen. Twist based acquisition and tracking of animal and human kinematics. *International Journal of Computer Vision*, 56:179–194, 2004.

[26] M. Brubaker and D. Fleet. The kneed walker for human pose tracking. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. Proceedings of the 2008 IEEE Computer Society Conference on*, pages 1–8, 2008.

[27] M. Brubaker, D. Fleet, and A. Hertzmann. Physics-based person tracking using simplified lower-body dynamics. In *Computer Vision and Pattern Recognition, 2007. CVPR 2007. Proceedings of the 2007 IEEE Computer Society Conference on*, pages 1–8, June 2007.

[28] M. Brubaker, L. Sigal, and D. Fleet. Estimating contact dynamics. In *Computer Vision, 2009. ICCV 2009. 2009 IEEE International Conference on*, pages 2389–2396, 2009.

[29] J. Butterfield, S. Osentoski, G. Jay, and O. Jenkins. Learning from demonstration using a multi-valued function regressor for time-series data. In *Humanoid Robots (Humanoids), 2010. 2010 IEEE-RAS International Conference on*, pages 328–333, December 2010.

[30] S. Calinon and A. Billard. Incremental learning of gestures by imitation in a humanoid robot. In *Human-robot Interaction (HRI), 2007. Proceedings of the 2007 ACM/IEEE International conference on Human-robot interaction*, pages 255–262. ACM, 2007.

[31] J. Chai and J. Hodgins. Constraint-based motion optimization using a statistical dynamic model. *ACM Transactions on Graphics (TOG)*, 26(3), July 2007.

[32] N. Chakraborty, S. Berard, S. Akella, and J. Trinkle. A fully implicit time-stepping method for multi-body systems with intermittent contact. *Robotics: Science and Systems*, 2007.

[33] L. Chan-Su and A. Elgammal. Modeling view and posture manifolds for tracking. In *Computer Vision, 2007. ICCV 2007. 2007 IEEE International Conference on*, pages 1–8, October 2007.

[34] S. Chernova and M. Veloso. Interactive policy learning through confidence-based autonomy. *Journal of Artificial Intelligence Research*, 34(1):25, 2009.

[35] D. Chu, A. Shapiro, B. Allen, and P. Faloutsos. A dynamic controller toolkit. In *ACM SIGGRAPH Video Game Symposium (Sandbox)*, pages 21–26, 2007.

[36] M. Cline. Rigid body simulation with contact and constraints. Master's thesis, The University of British Columbia, 2002.

[37] S. Collins, A. Ruina, R. Tedrake, and M. Wisse. Efficient bipedal robots based on passive-dynamic walkers. *Science*, 307(5712):1082–1085, 2005.

[38] S. Coros, P. Beaudoin, and M. van de Panne. Robust task-based control policies for physics-based characters. In *ACM Transactions on Graphics (TOG)*, pages 170:1–170:9, 2009.

[39] L. Csató and M. Opper. Sparse on-line gaussian processes. *Neural Computation*, 14(3):641–668, 2002.

[40] M. da Silva, Y. Abe, and J. Popović. Interactive simulation of stylized human locomotion. *ACM Transactions on Graphics (TOG)*, 27(3):82:1–82:10, August 2008.

[41] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. Proceedings of the 2005 IEEE Computer Society Conference on*, volume 1, pages 886–893 vol. 1, June 2005.

[42] T. Darrell, P. Maes, B. Blumberg, and A. Pentland. A novel environment for situated vision and behavior, 1994.

[43] M. de Lasa, I. Mordatch, and A. Hertzmann. Feature-based locomotion controllers. *ACM Transactions on Graphics (TOG)*, 29(4):131:1–131:10, July 2010.

[44] Q. Delamarre and O. Faugeras. 3d articulated models and multi-view tracking with silhouettes. In *Computer Vision, 1999. ICCV 1999. 1999 IEEE International Conference on*, volume 2, pages 716–721 vol.2, 1999.

[45] W. T. Dempster. Space requirements of the seated operator: Geometrical, kinematic, and mechanical aspects of the body with special reference to the limbs. Technical report, Wright-Patterson Air Force Base 55-159, 1955.

[46] J. Deutscher and I. Reid. Articulated body motion capture by stochastic search. *International Journal of Computer Vision*, 61:185–205, 2005.

[47] A. Elgammal, D. Harwood, and L. Davis. Non-parametric model for background subtraction. In David Vernon, editor, *Proceedings of the 2002 European conference on Computer Vision*, volume 1843 of *Lecture Notes in Computer Science*, pages 751–767. Springer Berlin / Heidelberg, 2000.

[48] A. Elgammal and C. Lee. Inferring 3d body pose from silhouettes using activity manifold learning. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–681–II–688 vol.2, June 2004.

[49] K. Erleben. Rigid body simulation, 2002. Lecture notes. `www.diku.dk/forskning/image/teaching/Courses/e02/RigidBodySimulation/`.

[50] A. Fang and N. Pollard. Efficient synthesis of physically valid human motion. In *ACM Transactions on Graphics (TOG)*, SIGGRAPH ’03, pages 417–426, New York, NY, USA, 2003. ACM.

[51] P. Felzenszwalb and D. Huttenlocher. Pictorial structures for object recognition. *International Journal of Computer Vision*, 61:55–79, 2005.

[52] V. Ferrari, M. Marin-Jimenez, and A. Zisserman. Progressive search space reduction for human pose estimation. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. Proceedings of the 2008 IEEE Computer Society Conference on*, pages 1–8, June 2008.

[53] M. Fischler and R. Elschlager. The representation and matching of pictorial structures. *Computers, IEEE Transactions on*, C-22(1):67 – 92, January 1973.

[54] T. Flash and N. Hogan. The coordination of arm movements: An experimentally confirmed mathematical model. *The journal of Neuroscience*, 5(7):1688–1703, 1985.

[55] O. Freifeld, A. Weiss, S. Zuffi, and M. Black. Contour people: A parameterized model of 2d articulated human shape. In *Computer Vision and Pattern Recognition, 2010. CVPR 2010. Proceedings of the 2010 IEEE Computer Society Conference on*, pages 639–646, June 2010.

[56] J. Gall, B. Rosenhahn, T. Brox, and H.-P. Seidel. Optimization and filtering for human motion capture. *International Journal of Computer Vision*, 87:75–92, 2010.

[57] J. Gall, C. Stoll, E. de Aguiar, C. Theobalt, B. Rosenhahn, and H.-P. Seidel. Motion capture using joint skeleton tracking and surface estimation. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. Proceedings of the 2009 IEEE Computer Society Conference on*, pages 1746–1753, June 2009.

[58] D. Gavrila and L. Davis. 3-d model-based tracking of humans in action: A multi-view approach. In *Computer Vision and Pattern Recognition, 1996. CVPR 1996. Proceedings of the 1996 IEEE Computer Society Conference on*, pages 73–80, June 1996.

[59] S. Grassia. Practical parameterization of rotations using the exponential map. *Journal of Graphics Tools*, 3:29–48, 1998.

[60] K. Grauman, G. Shakhnarovich, and T. Darrell. Inferring 3d structure with a statistical image-based shape model. In *Computer Vision, 2003. ICCV 2003. 2003 IEEE International Conference on*, pages 641–647 vol.1, October 2003.

[61] D. Grollman and O. Jenkins. Dogged learning for robots. In *Robotics and Automation, 2007. Proceedings. ICRA 2007. IEEE International Conference on*, pages 2483–2488. IEEE, 2007.

[62] D. Grollman and O. Jenkins. Sparse incremental learning for interactive robot control policy estimation. In *Robotics and Automation, 2008. Proceedings. ICRA 2008. IEEE International Conference on*, pages 3315–3320. IEEE, 2008.

[63] D. Grollman and O. Jenkins. Can we learn finite state machine robot controllers from interactive demonstration? In *From Motor Learning to Interaction Learning in Robots*, pages 407–430. Springer, 2010.

[64] D. Grollman and O. Jenkins. Incremental learning of subtasks from unsegmented demonstration. In *Intelligent Robots and Systems, 2010. Proceedings. 2010 IEEE/RSJ International Conference on*, pages 261–266, October 2010.

[65] P. Guan, O. Freifeld, and M. Black. A 2d human body model dressed in eigen clothing. In K. Daniilidis, P. Maragos, and N. Paragios, editors, *Proceedings of the 2010 European conference on Computer Vision*, volume 6311 of *Lecture Notes in Computer Science*, pages 285–298. Springer Berlin / Heidelberg, 2010.

[66] P. Guan, A. Weiss, A. Balan, and M. Black. Estimating human shape and pose from a single image. In *Computer Vision, 2009. ICCV 2009. 2009 IEEE International Conference on*, pages 1381 – 1388, September 2009.

[67] N. Hansen. The cma evolution strategy: A comparing review. *Towards a New Evolutionary Computation. Advances on Estimation of Distribution Algorithms*, pages 75–102, 2006.

[68] R. Heck and M. Gleicher. Parametric motion graphs. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*, I3D '07, pages 129–136, New York, NY, USA, 2007. ACM.

[69] A. Hilton, D. Beresford, T. Gentils, R. Smith, and S. Wei. Virtual people: Capturing human models to populate virtual worlds. In *Computer Animation, 1999. Proceedings*, pages 174–185, 1999.

[70] J. Hodgins, W. Wooten, D. Brogan, and J. O'Brien. Animating human athletics. In *ACM Transactions on Graphics (TOG)*, pages 71–78, 1995.

[71] David Hogg. Model-based vision: A program to see a walking person. *Image and Vision Computing*, 1(1):5 – 20, 1983.

[72] C. Hu, Q. Yu, Y. Li, and S. Ma. Extraction of parametric human model for posture recognition using genetic algorithm. In *Automatic Face and Gesture Recognition, 2000. Proceedings. 2000 IEEE International Conference on*, pages 518–523, 2000.

[73] Q. Huang, K. Yokoi, S. Kajita, K. Kaneko, H. Arai, N. Koyachi, and K. Tanie. Planning walking patterns for a biped robot. *Robotics and Automation, IEEE Transactions on*, 17(3):280–289, June 2001.

[74] M. Isard and A. Blake. Condensation-conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29:5–28, 1998.

[75] O. Jenkins and M. Matarić. Performance-derived behavior vocabularies: Data-driven acquisition of skills from motion. *International Journal of Humanoid Robotics*, 1(2):237–288, 2004.

[76] O. Jenkins and M. Matarić. A spatio-temporal extension to isomap nonlinear dimension reduction. In *Proceedings of the 2004 Conference on Machine Learning*, ICML '04, page 56, New York, NY, USA, 2004. ACM.

[77] S. Johnson and M. Everingham. Combining discriminative appearance and segmentation cues for articulated human pose estimation. In *Computer Vision Workshops (ICCV Workshops), 2009. 2009 IEEE International Conference on*, pages 405–412, October 2009.

[78] M. Jones and J. Rehg. Statistical color models with application to skin detection. In *Computer Vision and Pattern Recognition, 1999. CVPR 1999. Proceedings of the 1999 IEEE Computer Society Conference on*, volume 1, pages 2 vol. (xxiii+637+663), 1999.

[79] S. Ju, M. Black, and Y. Yacoob. Cardboard people: A parameterized model of articulated image motion. In *Automatic Face and Gesture Recognition, 1996. Proceedings. 1996 IEEE International Conference on*, pages 38–44, October 1996.

[80] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa. Biped walking pattern generation by using preview control of zero-moment point. In *Robotics and Automation, 2003. Proceedings. ICRA 2003. IEEE International Conference on*, volume 2, pages 1620 – 1626 vol.2, September 2003.

[81] Y. Kameda, M. Minoh, and K. Ikeda. Three dimensional pose estimation of an articulated object from its silhouette image. *Asian Conference on Computer Vision*, 1993.

[82] L. Kavan, S. Collins, J. Žára, and C. O'Sullivan. Geometric skinning with approximate dual quaternion blending. *ACM Transactions on Graphics (TOG)*, 27(4):105:1–105:23, November 2008.

[83] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, August 1996.

[84] K. Kawachi, H. Suzuki, and F. Kimura. Simulation of rigid body motion with impulsive friction force. *Proceedings of IEEE International Symposium on Assembly and Task Planning*, pages 182 – 187, August 1997.

[85] S. Khan and M. Shah. Tracking people in presence of occlusion. In *Asian Conference on Computer Vision*, pages 1132–1137, 2000.

[86] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5:90–98, March 1986.

[87] E. Kokkevis. Practical physics for articulated characters. *Game Developers Conference*, 2004.

[88] J. Kolter, J. Zico, P. Abbeel, and A. Ng. Hierarchical apprenticeship learning with application to quadruped locomotion. *Advances in Neural Information Processing Systems (NIPS)*, 20:769–776, 2008.

[89] L. Kovar and M. Gleicher. Automated extraction and parameterization of motions in large data sets. In *ACM Transactions on Graphics (TOG)*, SIGGRAPH '04, pages 559–568, New York, NY, USA, 2004. ACM.

[90] L. Kovar, M. Gleicher, and F. Pighin. Motion graphs. In *ACM Transactions on Graphics (TOG)*, SIGGRAPH '02, pages 473–482, New York, NY, USA, 2002. ACM.

[91] J. Kuffner and S. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA 2000. IEEE International Conference on*, volume 2, pages 995–1001 vol.2, 2000.

[92] J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue. Footstep planning among obstacles for biped robots. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 1, pages 500–505 vol.1, 2001.

[93] A. Kuo. Energetics of actively powered locomotion using the simplest walking model. *Journal of Biomechanical Engineering*, 124(1):113–120, 2002.

[94] M. Lee and I. Cohen. Proposal maps driven mcmc for estimating human body pose in static images. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–334 – II–341 vol.2, June 2004.

[95] M. Lee and R. Nevatia. Human pose tracking using multi-level structured models. In *Proceedings of the 2006 European conference on Computer Vision*, ECCV'06, pages 368–381, Berlin, Heidelberg, 2006. Springer-Verlag.

[96] Y. Lee, S. Kim, and J. Lee. Data-driven biped control. *ACM Transactions on Graphics (TOG)*, 29:129:1–129:8, July 2010.

[97] Y. Lee, S. Lee, and Z. Popović. Compact character controllers. In *ACM Transactions on Graphics (TOG)*, SIGGRAPH Asia '09, pages 169:1–169:8, New York, NY, USA, 2009. ACM.

[98] Y. Lee, K. Wampler, G. Bernstein, J. Popović, and Z. Popović. Motion fields for interactive character locomotion. *ACM Transactions on Graphics (TOG)*, 29(6):138:1–138:8, December 2010.

[99] F. Lerasle, G. Rives, and M. Dhome. Tracking of human limbs by multiocular vision. *Computer Vision and Image Understanding*, 75(3):229 – 246, 1999.

[100] T. Leung and J. Malik. Representing and recognizing the visual appearance of materials using three-dimensional textons. *International Journal of Computer Vision*, 43:29–44, 2001.

[101] S. Levine and V. Koltun. Continuous inverse optimal control with locally optimal examples. In *Proceedings of the 2012 Conference on Machine Learning*, 2012.

[102] S. Levine, Y. Lee, V. Koltun, and Z. Popović. Space-time planning with parameterized locomotion controllers. *ACM Transactions on Graphics (TOG)*, 30(3):23:1–23:11, May 2011.

[103] S. Levine, Z. Popović, and V. Koltun. Nonlinear inverse reinforcement learning with gaussian processes. *Advances in Neural Information Processing Systems (NIPS)*, 24, 2011.

[104] R. Li, T. Tian, S. Sclaroff, and M. Yang. 3d human motion tracking with a coordinated mixture of factor analyzers. In *International Journal of Computer Vision*, pages 137–150, 2010.

[105] R. Li, T-P. Tian, and S. Sclaroff. Simultaneous learning of nonlinear manifold and dynamical models for high-dimensional time series. In *Computer Vision, 2007. ICCV 2007. 2007 IEEE International Conference on*, pages 1–8, October 2007.

[106] R. Li, M. Yang, S. Sclaroff, and T. Tian. Monocular tracking of 3d human motion with a coordinated mixture of factor analyzers. In *Proceedings of the 2006 European conference on Computer Vision*, pages 702–718, 2006.

[107] Y. Li, S. Ma, and H. Lu. Human posture recognition using multi-scale morphological method and kalman motion estimation. In *Pattern Recognition, 1998. Proceedings. 1998 Internation Conference on*, volume 1, pages 175–177, August 1998.

[108] K. Liu, A. Hertzmann, and Z. Popović. Learning physics-based motion style with nonlinear inverse optimization. *ACM Transactions on Graphics (TOG)*, 24(3):1071–1081, July 2005.

[109] L. Liu, K. Yin, M. van de Panne, T. Shao, and W. Xu. Sampling-based contact-rich motion control. *ACM Transactions on Graphics (TOG)*, 29(4):128:1–128:10, July 2010.

[110] Y. Liu, C. Stoll, J. Gall, H.-P. Seidel, and C. Theobalt. Markerless motion capture of interacting characters using multi-view image segmentation. In *Computer Vision and Pattern Recognition, 2011. CVPR 2011. Proceedings of the 2011 IEEE Computer Society Conference on*, pages 1249–1256, June 2011.

[111] D. Lowe. Object recognition from local scale-invariant features. In *Computer Vision, 1999. ICCV 1999. 1999 IEEE International Conference on*, volume 2, pages 1150–1157 vol.2, 1999.

[112] Z. Lu, M. Carreira-Perpiňán, and C. Sminchisescu. People tracking with the laplacian eigenmaps latent variable model. *Advances in Neural Information Processing Systems (NIPS)*, pages 1705–1712, 2007.

[113] A. Macchietto, V. Zordan, and C. Shelton. Momentum control for balance. *ACM Transactions on Graphics (TOG)*, 28(3):80:1–80:8, July 2009.

[114] J. Malik, S. Belongie, J. Shi, and T. Leung. Textons, contours and regions: Cue integration in image segmentation. In *Computer Vision, 1999. ICCV 1999. 1999 IEEE International Conference on*, volume 2, pages 918–925 vol.2, 1999.

[115] D. Marr and H. Nishihara. Representation and recognition of the spatial organization of three-dimensional shapes. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 200(1140):269–294, 1978.

[116] J. McCann, N. Pollard, and S. Srinivasa. Physics-based motion retiming. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation, pages 205–214, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[117] T. McGeer. Passive dynamic walking. *The International Journal of Robotics Research*, 9(2):62–82, 1990.

[118] D. Metaxas and D. Terzopoulos. Shape and nonrigid motion estimation through physics-based synthesis. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 15(6):580–591, June 1993.

[119] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(10):1615–1630, October 2005.

[120] M. Miziguchi, J. Buchanan, and T. Calvert. Data driven motion transitions for interactive games. *Eurographics 2001 Short Presentations*, 2(3):6, 2001.

[121] T. Moeslund and E. Granum. A survey of computer vision-based human motion capture. *Computer Vision and Image Understanding*, 81(3):231 – 268, 2001.

[122] T. Moeslund, A. Hilton, and V. Krüger. A survey of advances in vision-based human motion capture and analysis. *Computer Vision and Image Understanding*, 104(2-3):90 – 126, 2006.

[123] I. Mordatch, M. de Lasa, and A. Hertzmann. Robust physics-based locomotion using low-dimensional planning. *ACM Transactions on Graphics (TOG)*, 29(4):71:1–71:8, July 2010.

[124] G. Mori, X. Ren, A. Efros, and J. Malik. Recovering human body configurations: Combining segmentation and recognition. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–326 – II–333 Vol.2, June-July 2004.

[125] U. Muico, Y. Lee, J. Popović, and Z. Popović. Contact-aware nonlinear control of dynamic characters. In *ACM Transactions on Graphics (TOG)*, SIGGRAPH '09, pages 81:1–81:9, New York, NY, USA, 2009. ACM.

[126] Y. Nakamura and K. Yamane. Dynamics computation of structure-varying kinematic chains and its application to human figures. *Robotics and Automation, IEEE Transactions on*, 16(2):124 –134, April 2000.

[127] J. Nakanishi, J. Morimoto, G. Endo, G. Cheng, S. Schaal, and M. Kawato. Learning from demonstration and adaptation of biped locomotion. *Robotics and Autonomous Systems*, 47(2):79–91, 2004.

[128] M. Nicolescu, O. Jenkins, A. Olenderski, and E. Fritzinger. Learning behavior fusion from demonstration. *Interaction Studies*, 9(2):319–352, 2008.

[129] M. Nicolescu and M. Matarić. Natural methods for robot task learning: Instructive demonstrations, generalization and practice. In *Proceedings of the 2003 International joint conference on Autonomous agents and Multiagent systems*, pages 241–248. ACM, 2003.

[130] S. Niekum, A. Barto, and L. Spector. Genetic programming for reward function search. *Autonomous Mental Development, IEEE Transactions on*, 2(2):83–90, June 2010.

[131] S. Niekum, S. Osentoski, G. Konidaris, and A. Barto. Learning and generalization of complex tasks from unstructured demonstrations. In *Intelligent Robots and Systems, 2012. Proceedings. 2012 IEEE/RSJ International Conference on*, pages 5239–5246. IEEE, 2012.

[132] H. Ning, W. Xu, Y. Gong, and T. Huang. Discriminative learning of visual words for 3d human pose estimation. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. Proceedings of the 2008 IEEE Computer Society Conference on*, pages 1–8, June 2008.

[133] B. North, A. Blake, M. Isard, and J. Rittscher. Learning and classification of complex dynamics. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(9):1016 –1034, September 2000.

[134] J. Pang and F. Facchinei. *Finite-Dimensional Variational Inequalities and Complementarity Problems (I)*. Springer Verlag, 2003.

[135] S. Park and J. Aggarwal. Simultaneous tracking of multiple body parts of interacting persons. *Computer Vision and Image Understanding*, 102(1):1 – 21, 2006.

[136] V. Pavlovic, J. Rehg, T. Cham, and K. Murphy. A dynamic bayesian network approach to figure tracking using learned dynamic models. In *Computer Vision, 1999. ICCV 1999. 1999 IEEE International Conference on*, volume 1, pages 94–101, 1999.

[137] D. Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97, 1991.

[138] Z. Popović and A. Witkin. Physically based motion transformation. In *ACM Transactions on Graphics (TOG)*, SIGGRAPH '99, pages 11–20, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[139] R. Poppe. Vision-based human motion analysis: An overview. *Computer Vision and Image Understanding*, 108(1-2):4 – 18, 2007.

[140] J. Pratt. *Exploiting Inherent Robustness and Natural Dynamics in the Control of Bipedal Walking Robots*. PhD thesis, Massachusetts Institute Of Technology, 2000.

[141] D. Ramanan. Using segmentation to verify object hypotheses. In *Computer Vision and Pattern Recognition, 2007. CVPR 2007. Proceedings of the 2007 IEEE Computer Society Conference on*, pages 1–8, June 2007.

[142] Deva Ramanan. Learning to parse images of articulated bodies. *Advances in Neural Information Processing Systems (NIPS)*, 19:1129–1136, 2007.

[143] C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[144] N. Ratliff, J. Bagnell, and S. Srinivasa. Imitation learning for locomotion and manipulation. In *Humanoid Robots (Humanoids), 2007. 2007 IEEE-RAS International Conference on*, pages 392–397, November 2007.

[145] X. Ren and J. Malik. Learning a classification model for segmentation. In *Computer Vision, 2003. ICCV 2003. 2003 IEEE International Conference on*, pages 10–17 vol.1, October 2003.

[146] T. Roberts, S. McKenna, and I. Ricketts. Human pose estimation using learnt probabilistic region similarities and partial configurations. In T. Pajdla and J. Matas, editors, *Proceedings of the 2004 European conference on Computer Vision*, volume 3024 of *Lecture Notes in Computer Science*, pages 291–303. Springer Berlin / Heidelberg, 2004.

[147] B. Rosenhahn, C. Schmaltz, T. Brox, J. Weickert, D. Cremers, and H.-P. Seidel. Markerless motion capture of man-machine interaction. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. Proceedings of the 2008 IEEE Computer Society Conference on*, pages 1–8, June 2008.

[148] B. Rosenhahn, C. Schmaltz, T. Brox, J. Weickert, and H.-P. Seidel. Staying well grounded in markerless motion capture. In *Pattern Recognition*, volume 5096 of *Lecture Notes in Computer Science*, pages 385–395. Springer, June 2008.

[149] P. Rybski, K. Yoon, J. Stolarz, and M. Veloso. Interactive robot task training through dialog and demonstration. In *Human-robot Interaction (HRI), 2007. Proceedings of the 2007 ACM/IEEE International conference on Human-robot interaction*, pages 49–56, 2007.

[150] A. Safonova and J. Hodgins. Construction and optimal search of interpolated motion graphs. *ACM Transactions on Graphics (TOG)*, 26(3), July 2007.

[151] A. Safonova, J. Hodgins, and N. Pollard. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM Transactions on Graphics (TOG)*, 23(3):514–521, August 2004.

[152] J. Saunders, C. Nehaniv, and K. Dautenhahn. Teaching robots by moulding behavior and scaffolding the environment. In *Proceedings of the 2006 ACM SIGCHI/SIGART conference on Human-robot Interaction*, HRI '06, pages 118–125, New York, NY, USA, 2006. ACM.

[153] S. Schaal and C. Atkeson. Constructive incremental learning from only local information. *Neural Computation*, 10(8):2047–2084, 1998.

[154] S. Schaal and N. Schweighofer. Computational motor control in humans and robots. *Current Opinion in Neurobiology*, 15(6):675–682, 2005.

[155] G. Shakhnarovich, P. Viola, and T. Darrell. Fast pose estimation with parameter-sensitive hashing. In *Computer Vision, 2003. ICCV 2003. 2003 IEEE International Conference on*, pages 750–757 vol.2, October 2003.

[156] K. Shoemake. Animating rotation with quaternion curves. *ACM Transactions on Graphics (TOG)*, 19(3):245–254, 1985.

[157] H. Sidenbladh and M. Black. Learning image statistics for bayesian tracking. In *Computer Vision, 2001. ICCV 2001. 2001 IEEE International Conference on*, volume 2, pages 709–716 vol.2, 2001.

[158] H. Sidenbladh, M. Black, and D. Fleet. Stochastic tracking of 3d human figures using 2d image motion. In D. Vernon, editor, *Proceedings of the 2000 European conference on Computer Vision*, volume 1843 of *Lecture Notes in Computer Science*, pages 702–718. Springer Berlin / Heidelberg, 2000.

[159] H. Sidenbladh, M. Black, and L. Sigal. Implicit probabilistic models of human motion for synthesis and tracking. In *Proceedings of the 2002 European conference on Computer Vision*, volume 2350 of *Lecture Notes in Computer Science*, pages 784–800. Springer Berlin / Heidelberg, 2002.

[160] H. Sidenbladh, F. De la Torre, and M. Black. A framework for modeling the appearance of 3d articulated figures. In *Automatic Face and Gesture Recognition, 2000. Proceedings. 2000 IEEE International Conference on*, pages 368–375, 2000.

[161] L. Sigal, A. Balan, and M. Black. Humaneva: Synchronized video and motion capture dataset and baseline algorithm for evaluation of articulated human motion. *International Journal of Computer Vision*, 87(1-2):4–27, 2010.

[162] L. Sigal, S. Bhatia, S. Roth, M. Black, and M. Isard. Tracking loose-limbed people. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 1, pages I–421 – I–428, June-July 2004.

[163] L. Sigal and M. Black. Measure locally, reason globally: Occlusion-sensitive articulated pose estimation. In *Computer Vision and Pattern Recognition, 2006. CVPR 2006. Proceedings of the 2006 IEEE Computer Society Conference on*, volume 2, pages 2041 – 2048, 2006.

[164] L. Sigal and M. Black. Predicting 3d people from 2d pictures. In F. Perales and R. Fisher, editors, *Articulated Motion and Deformable Objects*, volume 4069 of *Lecture Notes in Computer Science*, pages 185–195. Springer Berlin / Heidelberg, 2006.

[165] W. Smart and L. Kaelbling. Effective reinforcement learning for mobile robots. In *Robotics and Automation, 2002. Proceedings. ICRA 2002. IEEE International Conference on*, volume 4, pages 3404–3410. IEEE, 2002.

[166] C. Sminchisescu and A. Jepson. Generative modeling for continuous non-linearly embedded visual inference. In *Proceedings of the 2004 Conference on Machine Learning*, ICML '04, page 96, New York, NY, USA, 2004. ACM.

[167] C. Sminchisescu, A. Kanaujia, L. Zhiguo, and D. Metaxas. Discriminative density propagation for 3d human motion estimation. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. Proceedings of the 2005 IEEE Computer Society Conference on*, volume 1, pages 390 – 397 vol. 1, June 2005.

[168] C. Sminchisescu and B. Triggs. Building roadmaps of local minima of visual models. In A. Heyden, G. Sparr, M. Nielsen, and P. Johansen, editors, *Proceedings of the 2002 European conference on Computer Vision*, volume 2350 of *Lecture Notes in Computer Science*, pages 566–582. Springer Berlin / Heidelberg, 2002.

[169] R. Smith. Open dynamics engine, 2006. `http://www.ode.org/`.

[170] K. Sok, M. Kim, and J. Lee. Simulating biped behaviors from human motion data. *ACM Transactions on Graphics (TOG)*, 26(3):107, July 2007.

[171] M. Spong, S. Hutchinson, and M. Vidyasagar. *Robot Modeling and Control (Hardcover)*. Wiley, Hoboken, NJ, 2006.

[172] J. Starck and A. Hilton. Model-based multiple view reconstruction of people. *Computer Vision, 2003. ICCV 2003. 2003 IEEE International Conference on*, 2:915–922, 2003.

[173] J. Starck and A. Hilton. Surface capture for performance-based animation. *IEEE Computer Graphics and Applications*, 27:21–31, 2007.

[174] C. Stauffer and W. Grimson. Adaptive background mixture models for real-time tracking. In *Computer Vision and Pattern Recognition, 1999. CVPR 1999. Proceedings of the 1999 IEEE Computer Society Conference on*, volume 2, pages 2 vol. (xxiii+637+663), 1999.

[175] P. Stone and M. Veloso. Beating a defender in robotic soccer: Memory-based learning of a continuous function. Technical report, DTIC Document, 1995.

[176] P. Stone and M. Veloso. Layered learning. In *Machine Learning: ECML 2000*, pages 369–381. Springer, 2000.

[177] G. Taylor, L. Sigal, D. Fleet, and G. Hinton. Dynamic binary latent variable models for 3d human pose tracking. In *Computer Vision and Pattern Recognition, 2010. CVPR 2010. Proceedings of the 2010 IEEE Computer Society Conference on*, pages 631–638, 2010.

[178] T-P. Tian and S. Sclaroff. Fast globally optimal 2d human detection with loopy graph models. In *Computer Vision and Pattern Recognition, 2010. CVPR 2010. Proceedings of the 2010 IEEE Computer Society Conference on*, pages 81–88, June 2010.

[179] A. Treuille, Y. Lee, and Z. Popović. Near-optimal character animation with continuous control. *ACM Transactions on Graphics (TOG)*, 26(3), July 2007.

[180] J. Trinkle, J. Pang, S. Sudarsky, and G. Lo. On dynamic multi-rigid-body contact problems with coulomb friction. *Zeitschrift für Angewandte Mathematik und Mechanik*, pages 267 – 279, 1997.

[181] Y.-Y. Tsai, K. Cheng, W. Lin, J. Lee, and T.-Y. Lee. Real-time physics-based 3d biped character animation using an inverted pendulum model. *IEEE Transactions on Visualization and Computer Graphics*, 16(2):325–337, 2010.

[182] R. Urtasun and T. Darrell. Sparse probabilistic regression for activity-independent human pose inference. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. Proceedings of the 2008 IEEE Computer Society Conference on*, pages 1–8, June 2008.

[183] R. Urtasun, D. Fleet, and P. Fua. 3d people tracking with gaussian process dynamical models. In *Computer Vision and Pattern Recognition, 2006. CVPR 2006. Proceedings of the 2006 IEEE Computer Society Conference on*, volume 1, pages 238 – 245, June 2006.

[184] R. Urtasun, D. Fleet, and P. Fua. Temporal motion models for monocular and multiview 3d human body tracking. *Computer Vision and Image Understanding*, 104(2-3):157 – 177, 2006.

[185] R. Urtasun, D. Fleet, A. Geiger, J. Popović, T. Darrell, and N. Lawrence. Topologically-constrained latent variable models. In *Proceedings of the 2008 Conference on Machine Learning*, ICML '08, pages 1080–1087, New York, NY, USA, 2008. ACM.

[186] R. Urtasun, D. Fleet, A. Hertzmann, and P. Fua. Priors for people tracking from small training sets. In *Computer Vision, 2005. ICCV 2005. 2005 IEEE International Conference on*, volume 1, pages 403–410, 2005.

[187] S. Vijayakumar, A. D'souza, and S. Schaal. Incremental online learning in high dimensions. *Neural Computation*, 17(12):2602–2634, 2005.

[188] S. Vijayakumar and S. Schaal. Locally weighted projection regression: An o(n) algorithm for incremental real time learning in high dimensional space. In *Proceedings of the 200 Conference on Machine Learning*, volume 1, pages 288–293, 2000.

[189] D. Vlasic, I. Baran, W. Matusik, and J. Popović. Articulated mesh animation from multi-view silhouettes. *ACM Transactions on Graphics (TOG)*, 27(3):97:1–97:9, August 2008.

[190] M. Vondrák. *Crisis Physics Engine*. `http://crisis.sourceforge.net/`.

[191] M. Vondrák. Articulated body dynamics. Master's thesis, Charles University, Prague, 2006.

[192] M. Vondrák. *Rigid Body Dynamics With Constraints*. LAP LAMBERT Academic Publishing, 2013.

[193] M. Vondrák, L. Sigal, and O. Jenkins. Physical simulation for probabilistic motion tracking. *Computer Vision and Pattern Recognition, 2008. CVPR 2008. Proceedings of the 2008 IEEE Computer Society Conference on*, pages 1–8, 2008.

[194] M. Vondrák, L. Sigal, and O. C. Jenkins. *Dynamics and Control of Multibody Systems*. IN-TECH, 2009.

[195] M. Vukobratović and B. Borovac. Zero-moment point - thirty five years of its life. *International Journal of Humanoid Robotics*, 1(1):157–173, 2004.

[196] J. Wang, D. Fleet, and A. Hertzmann. Gaussian process dynamical models. *Advances in Neural Information Processing Systems (NIPS)*, 18:1441, 2006.

[197] J. Wang, D. Fleet, and A. Hertzmann. Multifactor gaussian process models for style-content separation. In *Proceedings of the 2007 Conference on Machine Learning*, ICML '07, pages 975–982, New York, NY, USA, 2007. ACM.

[198] J. Wang, D. Fleet, and A. Hertzmann. Optimizing walking controllers. In *ACM Transactions on Graphics (TOG)*, pages 168:1–168:8, 2009.

[199] J. Wang, D. Fleet, and A. Hertzmann. Optimizing walking controllers for uncertain inputs and environments. In *ACM Transactions on Graphics (TOG)*, pages 73:1–73:8, 2010.

[200] J. Wang, S. Hamner, S. Delp, and V. Koltun. Optimizing locomotion controllers using biologically-based actuators and objectives. *ACM Transactions on Graphics (TOG)*, 31(4):25, 2012.

[201] X. Wei and J. Chai. Videomocap: Modeling physically realistic human motion from monocular video sequences. *ACM Transactions on Graphics (TOG)*, 29(4):42:1–42:10, 2010.

[202] X. Wei, J. Min, and J. Chai. Physically valid statistical models for human motion generation. *ACM Transactions on Graphics (TOG)*, 30(3):19:1–19:10, May 2011.

[203] A. Witkin and M. Kass. Spacetime constraints. *ACM Transactions on Graphics (TOG)*, 22:159–168, 1998.

[204] C. Wren, A. Azarbayejani, T. Darrell, and A. Pentland. Pfinder: Real-time tracking of the human body. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(7):780–785, July 1997.

[205] C. Wren and A. Pentland. Dynamic models of human motion. In *Automatic Face and Gesture Recognition, 1998. Proceedings. 1998 IEEE International Conference on*, pages 22–27, 1998.

[206] P. Wrotek, O. Jenkins, and M. McGuire. Dynamo: Dynamic, data-driven character control with adjustable balance. In *ACM SIGGRAPH Video Game Symposium (Sandbox)*, Sandbox '06, pages 61–70, New York, NY, USA, 2006. ACM.

[207] X. Xu and B. Li. Learning motion correlation for tracking articulated human body with a rao-blackwellised particle filter. In *Computer Vision, 2007. ICCV 2007. 2007 IEEE International Conference on*, pages 1–8, October 2007.

[208] M. Yagi and V. Lumelsky. Biped robot locomotion in scenes with unknown obstacles. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 1, pages 375–380 vol.1, 1999.

[209] K. Yamane and J. Hodgins. Simultaneous tracking and balancing of humanoid robots for imitating human motion capture data. *Intelligent Robots and Systems, 2009. Proceedings. 2009 IEEE/RSJ International Conference on*, pages 2510–2517, 2009.

[210] K. Yamane and Y. Nakamura. Robot kinematics and dynamics for modeling the human body. *International Symposium on Robotics Research*, pages 49–60, 2007.

[211] Y. Yang and D. Ramanan. Articulated pose estimation with flexible mixtures-of-parts. In *Computer Vision and Pattern Recognition, 2011. CVPR 2011. Proceedings of the 2011 IEEE Computer Society Conference on*, pages 1385–1392, June 2011.

[212] K. Yin, S. Coros, P. Beaudoin, and M. van de Panne. Continuation methods for adapting simulated skills. *ACM SIGGRAPH*, pages 81:1–81:7, 2008.

[213] K. Yin, K. Loken, and M. van de Panne. Simbicon: Simple biped locomotion control. In *ACM Transactions on Graphics (TOG)*, 2007.

[214] B. Ziebart, A. Maas, J. Bagnell, and A. Dey. Human behavior modeling with maximum entropy inverse optimal control. In *AAAI Spring Symposium on Human Behavior Modeling*, pages 92–97, 2009.

[215] V. Zordan, A. Majkowska, B. Chiu, and M. Fast. Dynamic response for motion capture animation. *ACM Transactions on Graphics (TOG)*, 24(3):697–701, July 2005.