RGame: A Video Game for Interactive Robot Learning

Micah Lapping - Carr
mlapping@cs.brown.edu
Department of Computer Science
Brown University, Providence, RI
Honors Thesis Submitted April 28th, 2009
Advisor: Chad Jenkins
Reader: Amy Greenwald

Abstract

We have investigated a major challenge for the success of robots in society: the difficulty end-users face in creating viable robot control policies. We have sought to address this challenge through interactive robot learning. We have developed a robot control interface using the Nintendo Wii remote that our users have informally said is intuitive and easy to learn. Users of this interface are able to control various kinds of mobile robots remotely. In addition, we built a networked game that lets users take part in interactive robot training. We see this work as one step towards facilitating the integration of robots into our world.

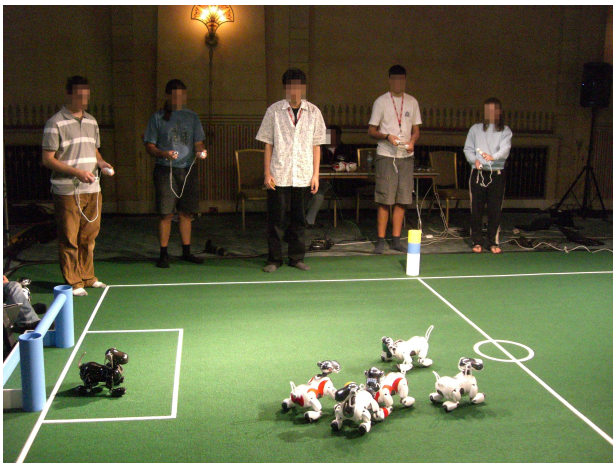

Figure 1: Users controlling Aibos with Wiimotes

1 Introduction

For most current robots, only the "technically elite" (programmers and engineers) are currently able to create the robot control policies they want, while the rest of the population must make do using the built-in policies (such as those on the iRobot Roomba, or WowWee Robotics' line of Robosapiens) included by the robot's creators. Through **interactive robot learning**, we aim to provide users of consumer robot technologies with a medium for transforming their desired robot behavior into executable control policies. Specifically, given the same situational awareness, a robot should make a decision similar to the one the creator of the policy would make.

While several paradigms exist for controlling robots (e.g. continuous teleoperation, speech and gesture-based instruction[7], text-based and visual computer programming, optimization/search), we remain confronted by a **human-robot divide**. This divide refers to the disparity between the needs and ideas of users in society, a population with a diverse set of technical abilities and creative design sensibilities, and their ability to instantiate robot control to meet their desired ends. If a **personal robotics revolution** is to come, there will need to exist applications that will make new forms of personal expression tangible, enhance personal productivity, and put this

new technology into the hands of users (analogous to the spreadsheet, web authoring, 3D virtual worlds, etc., on the PC).

In the future, we can envision off-the-shelf robots that can be "taught" by humans to perform **unknown tasks**, where no task-specific information needs to be hardcoded into the robot's decision making procedures. For instance, a user should be able to purchase a robot platform (let's say a robot dog) from their local electronics store and, without writing a single line of code, teach it to play soccer at a level competitive with hardcoded decision making. This makes the reasonable assumption that non-technical users are comparable or better creators of robot controllers, once technical barriers are relaxed. As examples, consider animated filmmaking, web design, and desktop publishing: while technical researchers and systems architects enabled the development of these media, it is often the design-oriented (artists, film-makers, etc.) that can make the most of these outlets for expression, in terms of aesthetics and accessibility. We conjecture that the personal robotics industry will follow a similar route, in that once robot learning has been fully enabled by researchers such as ourselves, it will be the end users of robots who will be creating the best robot controllers, not the researchers.

Towards "human-guided" pathways into robotics, our aim is to realize interactive robot learning through developing an interface that can leverage **scalable policy-learning algorithms suited for long-term human-robot interaction (HRI)**. While differing in methods, these objectives for interactive robot learning fall under the broader scope of "socially-guided robotics" proposed by Breazeal and colleagues[9]. Assuming hardcoded routines for perception and motion control, as readily available in existing middleware packages[1] and perception libraries[2], we have focused on providing data to algorithms that learn decision-making policies $\pi : \hat{\boldsymbol{s}} \rightarrow \boldsymbol{a}$ that map perceived robot state ($\hat{\boldsymbol{s}}$) to robot actions ($\boldsymbol{a}$).

We claim that facilitating long-term data collection and human guidance is the primary challenge, more so than algorithm development, for interactive robot learning. Many approaches to robot policy development have been pursued, ranging from primarily hand-coded domain-specific algorithms[1] to general non-deterministic adaptive methods, such as learning with Partially Observable Markov Decision Processes[6]. While this space of algorithms and methods is being heavily explored, less attention has been paid to human-robot interfaces that will facilitate the longitudinal human-robot interaction and guidance to enable tractable interactive robot learning. Notable work in the latter area includes experiments on managing robot teams [3] and interface design[8].

Figure 2: A boy controlling a SmURV with a Wiimote at RoboBusiness '07 in Boston

Towards this end, we discuss our experiences working with the Nintendo Wii Remote (or Wiimote) as one possible human-robot interface, followed by the development of a larger system to facilitate the generation of data appropriate for the learning algorithms in question. Through various public demonstrations of the implemented system, we have informally observed that people find Wiimote interfaces "fun" and "engaging", usable within the space of a few minutes, and applicable to various robot platforms (e.g., iRobot Create, Sony Aibo, DLR robot hand). Additionally, we have been able to train simple robot

[1]Such as Player/Stage and Microsoft Robotics Studio.

[2]Such as Lowe's SIFT object recognition package and augmented reality tag-tracking libraries, such as ARTag.

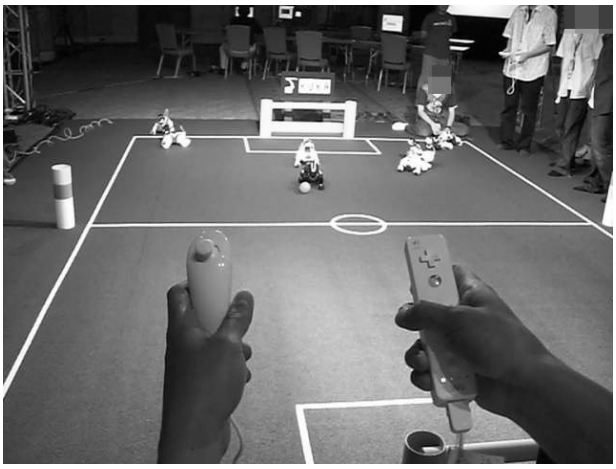

Figure 3: Nintendo Wii Remote and Nunchuk controlling a Sony Aibo playing robot soccer.

control policies using a Wiimote with our system.

2 The Nintendo Wii Remote

Released in December 2006, the Nintendo Wii Remote (or Wiimote, shown on the right in Fig. 3) is an inertial control interface designed for video games that is fundamentally different from traditional game control devices. The primary innovation of the Wiimote is its ability to localize itself within 2 rotational and 3 translational degrees of freedom. This localization is performed with a reasonable degree of accuracy, given that the wiimote uses 8 bits to represent up to 4 G's for each axis, which is well-complemented by the Wiimote's economical feasibility and compelling aesthetic. Rotational localization occurs with the help of three inertial sensors (accelerometers) that measure the direction of gravity along roll, pitch, and yaw axes. Translational localization is performed through triangulation against infrared light (IR) emitted by an external "sensor bar". The IR is sensed by the Wiimote through a built-in IR-sensitive chip. In addition, a Wiimote can receive input from 12 traditional gamepad buttons that can be used in complement with its localization.

The Wiimote communicates with other devices using the Bluetooth wireless communication. There are certain events, such as button presses and releases, changes in the data from the accelerometers or the IR sensor, and changes in Wiimote extension devices, that cause the Wiimote to send a packet of updated state information to its connected device. The Nunchuk (shown on the left in Fig. 3) is one such extension. It physically connects to the Wiimote and adds a second set of 3 accelerometers, along with 2 trigger-style buttons and an analog joystick. This combined Wiimote/Nunchuck interface allows for two-handed user input. For more complex robot control, this may make the parallel coordination of navigation and manipulation easier, as it has been in my experience.

3 iRobot Create/Roomba control

Our initial work into Wiimote-based robot control began using a single Wiimote to control Brown's Small Universal Robotic Vehicle (SmURV), pictured in Fig. 2. The original SmURV platform used an iRobot Create as a mobility base for a 1.2 GHz Mini-ITX computer, but the platform has since been updated to use an Asus eeePC as the computing core while retaining the iRobot Create base. The total cost of an original SmURV (including a Create, computer parts, and a firewire camera) is $750 USD; an eeePC-based SmURV lowers the cost to about $550.

In terms of software, the SmURV runs a stripped-down Linux distribution from a flash memory card and is controlled directly by the Player robot middleware [2] through a serial interface. The client we built is a simple program that talks to the Wiimote through a Bluetooth USB dongle and converts Wiimote events into Player commands for the robot. The Wiimote control interface for the SmURV is as follows: the robot is engaged by holding down the trigger on the bottom of the Wiimote, and, once engaged, the robots forward/backward and rotational velocities are controlled by tilting the Wiimote up/down along its pitch axis and twisting left/right along its roll axis, respectively.

4 Sony Aibo Control

In addition to the SmURV, the other robot platform that we initially worked with was the Sony Aibo. We chose the Aibo for several reasons. First, although there is no simple Player interface for the Aibo, the

Figure 4: Young woman at AAAI '07 in Vancouver controlling an Aibo with the Wiimote and Nunchuck control system

Brown Robocup team had done a significant amount of Aibo development already, so there was a large codebase that could be shared. Second, Aibos are already very popular among researchers, such as those involved in Robocup, and are a common sight at many conferences. Third, the Aibo offers a lot more in terms of manipulation and maneuvers than the SmURV, being able to move sideways and execute a large variety of moves that are useful in soccer, including kicks, blocks, and rolls.

The challenge in using the Wiimote to control a soccer-playing Sony Aibo robot dog is how to map a small amount of user input into the large range of control parameters provided by the Aibo's 18 degrees of freedom (DOFs). Further, the Aibo must also be coordinated to perform both navigation/locomotion and manipulation functions in the course of playing soccer. We chose to simplify the control problem by utilizing a hand-coded motion controller, thus requiring only three real-valued DOFs in the head (along with a binary-valued jaw), three real-valued walking gait DOFs (forward, sideways, rotation), and various discrete-valued "soccer moves" (kicks, blocks, ball trapping and control) that involve ball manipulation of some form.

Previously, we had attempted using dual-analog gamepads to control these degrees of freedom. These gamepads consisted of two analog joysticks, a discrete directional pad, and various buttons. Following the model of standard first-person video games, robot locomotion and head movements were controlled by the analog joysticks. Each soccer move was associated with an individual button to trigger their execution.

This control interface, however, caused users to complain that it had a steep learning curve. We attribute this to several factors which we have addressed. First, we noticed that the analog sticks were often used in a "full-throttle" manner, effectively having little more benefit than a discrete directional pad. The real-world nature of the robotics domain appears to amplify this phenomenon, most likely the result of users' impatience to observe the effect of their actions. Second, users of the system found it difficult to navigate when both walking and moving the head are each controlled only by the thumbs.

The Wiimote/Nunchuck combination is able to provide an interface that, in our informal observations, is more usable for Aibo soccer. We decided to separate locomotion and manipulation-related functions by splitting their controls between the user's hands. Because the head is the Aibo's primary manipulator, the Wiimote is used exclusively to control the head and soccer moves of the robot. The orientation and directional pad of the Wiimote controlled the robot's head and soccer moves, respectively. On the directional pad, up was mapped to "kicking" with the chest, left/right mapped to left/right lunge blocks, and down executed a block with both forward limbs. Additionally, the Wiimote's "A" button was used to execute a trapping motion for acquiring the

ball between the robot's chin and chest and the floor. See Fig. 5.

Locomotion by forward, backward, and turning motions was controlled by the Nunchuck in a similar manner to that of our SmURV Wiimote controller. In addition, the analog stick is used for sideways strafing. This interface not only separates locomotion onto the Nunchuck and manipulation onto the Wiimote, but also separates head control (performed mostly by the wrist) and instantiation of soccer moves (performed mostly by the thumb). Further, Wiimote control relies only on a user's natural proprioception about the two wrists and one finger.

5 The Creation of a Robot Video Game

Although the Wiimote/Nunchuk control provides a novel and useful interface, by itself it is not very interesting. However, we are not interested in having only humans control robots; we want our learned AIs to control robots as well. More importantly, given some data about how a human controls a robot in a variety of situations, the robot should attempt to do what the human would do. A lot of research has been done in the area of algorithms that can do this [4] [5], but there is still not a lot of data for these

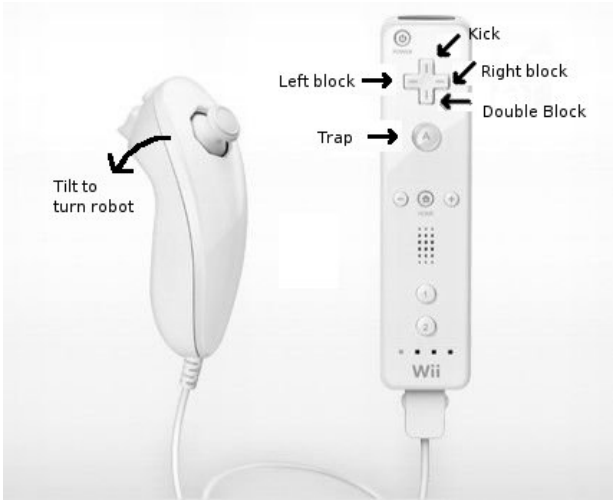

Figure 5: Wiimote control interface for the Aibo

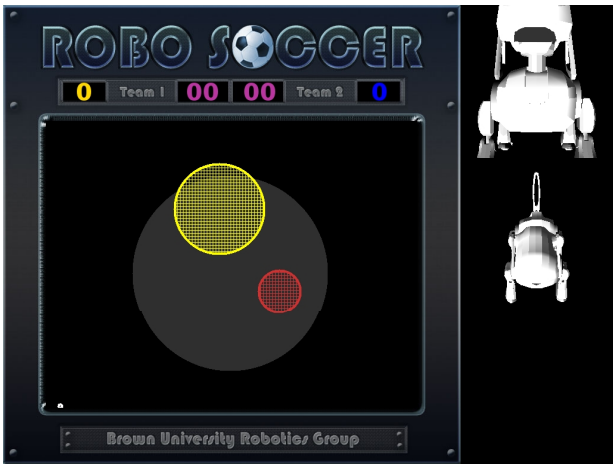

Figure 6: Visualization for the perception from a Sony Aibo.

algorithms to run on. It is towards this end that we set out to build a robot video game to harness the human computation at work while playing our game.

5.1 Rplay

The first iteration of the game was very simple, essentially a prototype. The front-end presented to the user was what has now become our iconic GUI (Fig. 6), designed by Daniel Byers. From there, the user could choose to control an Aibo using their preferred control method: mouse and keyboard, Wiimote, or game pad. The server was naively designed, running a total of five threads, one for each of the following: the GUI; receiving all robot outputs; sending all robot commands; receiving all user commands; and sending all users the appropriate robot's output. Although this functioned well with one or two robots, the lag with any more than that made effective control very difficult. Additionally, there was no way to run real-time learning on the server; all data had to be written to files and parsed later.

Despite its setbacks, this initial version was informally considered a success. After observing people using the system, it became clear that our system was engaging them. They adapted to the Wiimote control quite quickly and gave good feedback on the robot perception information being displayed in the

GUI. With this prototype behind us, it was time to move forward with the project by introducing both learning and multiple robot control.

6 RGame Architecture

The main downfall of the first iteration of this project, and what primarily drove us to the design and development of the current system, was its singular focus on the Aibo. Although at the beginning of the project the Aibo was our platform of choice, we realized that we would want to add in other robot platforms over time, starting with the SmURV. So when it came time to re-design the system, the primary goal was to make sure that not only would multiple robots be supported, but that their inclusion would be facilitated by our design decisions. Additionally, the growing scope of the project meant that the decisions we made would affect the potential usability of the project in the future. In the end, here is the list of desired features that led to our current design:

- support for multiple robotic platforms, with the support of future platforms to be easy to incorporate
- keep the same general client-server structure that we had already established
- new controllers and visualizations should be quickly integratable
- robot control and robot visualizations should be as independent as possible
- interactive learning should be incorporated
- it should be a compelling game

To accomplish the first goal, we decided to encapsulate all necessary information about a given robotic platform into a single idea, called (appropriately enough) the Platform. The Platform would present a uniform interface for robot communication and visualization to other parts of the program using polymorphism, while hiding the robot-specific implementation details in subclasses. In code, each Platform consists of a C++ header file defining the given robot's constants, like the robot's state and actuation space. To actually perform the communication and presentation, we created the RobotIO and UserIO superclasses.

Each Platform has its own RobotIO subclass that defines how to communicate with the robot in question. The RobotIO class is in charge of accepting a vector of desired actuation data for the robot, transforming it into a form the robot can understand, and then sending that data to the robot. Additionally, the RobotIO class is in charge of receiving the robot's raw sensor data, such as from the camera, and turning it into a vector of perception data. This polymorphism allows the server on the backend to be robot-agnostic during its communications: it makes the same method calls regardless of the specific kind of RobotIO object lying underneath.

The UserIO class works in the inverse fashion: it accepts a perception vector, presents it to the user in one of a variety of ways, receives the user's control input, and turns that into an actuation vector to be passed back to the robot. In this way, the two classes act like inverse functions, transforming the output from one class into the input for the other.

This made creation of both the user front-end client and the server back-end relatively simple, as they both use this polymorphism to make their tasks easier. The client uses the UserIO class to both read input from the user and generate actuation vectors to send back to the server, and display to the user the perception vectors that it receives from the server. The server uses the RobotIO class to send the user's actuation data to the robot and to receive the robot's perception data.

7 Interactive Robot Learning

One of our goals for our new design was to enable online robot learning, where the learning algorithm is able to process the user's control data as it comes in. This would enable training "on the fly", a very useful feature in any robot platform. We built this functionality into the server, with the option of using different kinds of algorithms or none at all. As long as the user decides to remain in control of the robot, they are putting more data into the algorithms bank. However, they can cede control at any time and let

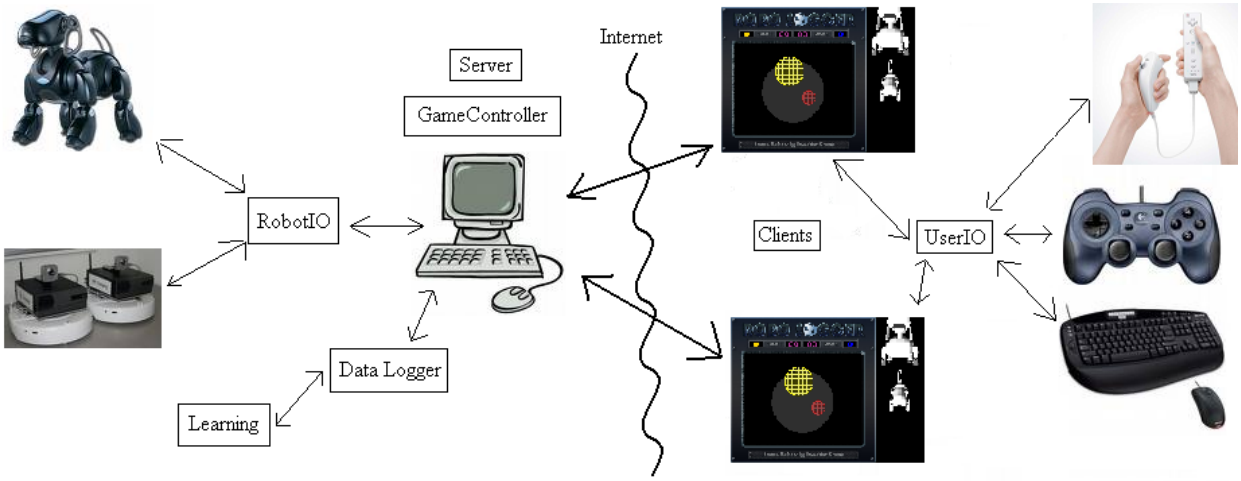

Figure 7: The RGame architecture.

the algorithm take over, at which point the algorithm will control the robot as best it can with the data it has incorporated. One advantage to this is it allows the user to teach the robot, observe any change (or lack there of) in its behavior, and then teach again, creating a teaching loop. This iterative process will hopefully lead to better control policies than would be produced otherwise.

8 Demos and Results

Having demoed the system at a large variety of venues, we have gotten a chance to observe many different people interacting with it. In the preliminary stages of the Wiimote control, prior to the existence of the GUI, we presented the system to other members of the Aibo community at the international Robocup competition[3] in Atlanta, GA in July 2007. During a break between matches, we ran a soccer game between our Wiimote-controlled Aibos and one of the other teams, encouraging members from all the teams to try out the system and give us feedback. Despite the pressure they were under to perform in the rest of the competition, we were able to convince quite a few Robocuppers to try our system. These new users were able to effectively control the Aibos after using the Wiimote for only a minute or two, and they seemed to be enjoying themselves. Although our team did not win the match, it was a very good demo for us.

We ran a similar demo at AAAI in Vancouver, Canada in August 2007. This time, instead of presenting to members of the Aibo community, we were showing to the artificial intelligence community at large. Again, many of those who tried the system had little difficulty in directing the robot in the desired fashion, and there were certainly smiles on their faces during that time. However, we would argue that the system passed perhaps a more important test in being a compelling interface: two children, ages eight to ten, were entertained enough to play with the two Aibos for a full thirty minutes until the batteries finally ran out.

Since the inclusion of the learning algorithms, we have run a few teaching tests using the system. Following what we expect our users to do, we controlled an Aibo using a Wiimote and nunchuk, only looking at the GUI. We were able to successfully train the robot to follow and approach either the orange ball or the yellow goal in separate trials, as well as being able to score a goal. Although these preliminary results are informal, it shows that this is a step in the right direction. As algorithm development continues, we can plug in the new algorithms and compare their

[3] http://robocup.org/

results.

In addition to these demos, we have tested our architecture's goal of being able to integrate new robots easily. A fellow robotics researcher from Harvard, Mark Woodward, brought his robot Harvey to try in our system. In a total of five hours with four people working on it, we were able to get Harvey fully working with RGame: we could control him with a wiimote; we could visualize his perception data; and we could learn. We consider this to be a good showing of how modular this system is and how quickly new platforms can be added.

9 Conclusion

The architecture we have proposed for a robot-learning-based video game has allowed us to support all the robot platforms we currently use, as well as quickly incorporate other platforms. It lets users remotely control any of these robotic platforms, seeing the world the way the robot sees it, all while playing soccer. It is able to utilize the work that others have done in robot learning to let users train a robot while they play and then let the trained AI take over immediately. It is designed to and capable of incorporating new platforms quickly, as well as new control types and visualizations. RGame was meant to be a growing platform, and we hope that it will be adopted and supported by other researchers in the future.

10 Acknowledgements

I would like to acknowledge my girlfriend for allowing my late-night coding habits, my family for being proud of me despite not quite understanding what I do, Dan Grollman for getting me into robotics all those years ago, Dan Byers for making the stuff we do look good, the folks in RLAB for their feedback and interest, Jesse Butterfield for work in helping design and build the system, and Chad Jenkins for seeing potential in me all those years ago and helping me chart what has turned out to be an amazingly successful and enjoyable course of study.

References

[1] Aaron Ladd Edsinger. *Robot manipulation in human environments*. PhD thesis, Cambridge, MA, USA, 2007. Adviser-Brooks,, Rodney A.

[2] R.; Gerkey, B.; Vaughan and A. Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the 11th International Conference on Advanced Robotics (ICAR 2003)*, pages 317–323, Coimbra, Portugal, 2003.

[3] Michael A. Goodrich, Timothy W. McLain, Jeffrey D. Anderson, Jisang Sun, and Jacob W. Crandall. Managing autonomy in robot teams: observations from four experiments. In *HRI '07: Proceedings of the ACM/IEEE international conference on Human-robot interaction*, pages 25–32, New York, NY, USA, 2007. ACM.

[4] Daniel H Grollman and Odest Chadwicke Jenkins. Dogged learning for robots. In *International Conference on Robotics and Automation*, pages 2483 – 2488, Rome, Italy, April 2007.

[5] Daniel H Grollman and Odest Chadwicke Jenkins. Learning multi-objective control policies from demonstration. In *IROS workshop on Robotics Challenges for Machine Learning*, Nice, France, September 2008.

[6] Leslie P Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. Technical report, Providence, RI, USA, 1996.

[7] Matthew M. Loper, Nathan P. Koenig, Sonia H. Chernova, Chris V. Jones, and Odest C. Jenkins. Mobile human-robot teaming with environmental tolerance. In *HRI '09: Proceedings of the 4th ACM/IEEE international conference on Human robot interaction*, pages 157–164, New York, NY, USA, 2009. ACM.

[8] Tetsuo Sawaragi and Yukio Horiguchi. Ecological interface enabling human-embodied cognition in mobile robot teleoperation. *Intelligence*, 11(3):20–32, 2000.

[9] A.L. Thomaz, G. Hoffman, and C. Breazeal. Reinforcement learning with human teachers: Understanding how people want to teach robots. pages 352–357, Sept. 2006.