Technical Aspects of Roomba Pac-Man

Mark Moseley
Department of Computer Science
Brown University
115 Waterman St.
Providence, RI, 02912-1910
mmoseley@cs.brown.edu

***Abstract*— Robotics can be seen as a field of computer science still in its infancy. In recent years there has been a tremendous explosion in innovation, both in hardware and software. The dynamic nature of robotics as well as the necessity for physical hardware presents a unique challenge for those wishing to educate the next generation of roboticists. This paper presents the technical aspects of Brown University's Computer Science Robotics course. The goal is to describe our design choices and the overall software development necessary to establish the course structure.**

I. Introduction

Robotics is a rapidly growing field in academia and it is just beginning to gain a foothold in the commercial marketplace. While this growth and innovation is exciting it creates a problem for college-level educators. A robotics course needs physical robots and sensors for the students. The options for robotic platforms are very limited and often consist of significant trade-offs between price, performance, and availability. Additionally, its very common in robotics to get bogged down in developing the low level interface and drivers specific to the devices on the robot. However, in teaching robotics, the low level control should be abstracted away. Frequently, this is accomplished by closed-source middle-ware custom designed for the specific platform. Thus, a new API must be learned for different robotic platforms and changing or adding sensors is limited if not impossible.

There is one option available to avoid all the issues surrounding physical robots, simulation. However, simulation environments remove several key aspects of robotics. Most importantly, simulation removes the error and noise of physical sensors and the unpredictability inherent in the real world. Also, designing course material purely in simulation removes a great deal of the excitement and motivation for students.

This paper presents from a technical point-of-view our attempt at Brown University to overcome these challenges while providing interesting and motivating course material and projects. Quite a bit of development was necessary to establish a robust, intuitive, and extendible client environment for students. Support code for assignments was developed to allow students to concentrate on roboticsts and all projects were implemented to ensure that projects were appropriately difficult and to identify potential pit falls.

The course presents students with an accessible robotic platform and course material on par with that being used in robotics today. The course material and projects are designed around creating an autonomous robot to play a real world version of Pac-Man. While this may initially sound like a simple task, it is actually very complicated addressing numerous aspects of robotics. Furthermore, motivating and keeping student's interested is a common problem across education. The game-oriented approach of Pac-Man provides the motivation and excitement of competition to drive students.

The heart of the approach is the use of the Open Source Software (OSS) Player/Stage/Gazebo (PSG). PSG provides the dual purpose of serving as the middle-ware between the programmer and the physical robot hardware and providing the ability for 3D physics-based simulation. The open source nature of PSG gives the added advantage of being able to modify and make additions to the code. In development of this course the open source nature was taken full advantage of, as enhancements were made to numerous aspects of PSG. These enhancements have been submitted to the developers of PSG, some have already been incorporated into the project, while others should be in the near future.

Another critical aspect of the course is the robotics platform and sensors. The platform utilized is the iRobot Roomba vacuum. The Roomba is well suited for a robotics course. It is arguably both the most successful and accessible mobile robot ever developed. Furthermore, the Roomba is relatively cheap enabling a higher robot to student ratio and has an interface for external control. The sensors and processing on the Roomba are somewhat limited, but this is addressed through the addition of a web camera and a laptop. The laptop serves as the brain for the robot, hosting the PSG software and providing the means to connect the web camera.

Setup and installation of the hardware was an important aspect of development. In the interest of accessibility and ease of install across the multiple laptops used for the class, an Ubuntu Linux install script has been developed that creates the exact environment used for this class (including PSG modifications). Not only does the script automate the install process but it serves to enumerate the required libraries and install steps for use with other operating systems. This environment was also ported to Mac OS X which widens

Fig. 1. An early version of Roomba Pac-Man platform.

future possibilities for course development.

Overall the combination of PSG, the Roomba vacuum, and a web camera provides both robotic simulation and a mobile, programmable robot with abstracted device control and rich sensory input for relatively cheap. This structure enables the course to focus on critical and current robotic concept through both simulation and the real world.

II. Approach

A. Player/Stage/Gazebo

The course revolves around using the PSG software and the Roomba platform. PSG is used because it eases development and re-usability of code in robotics. The same code can be used for numerous types of robots and developers do not need to learn a new API every time a change in the physical hardware occurs or even between total switches in robotic platform. Player/Stage/Gazebo is actually composed of three separate programs. Player is an open source robot control interface, while Stage and Gazebo are simulation environments.

Player acts as middle-ware between hardware and the controller to abstract away manufacturer/model specific code. The current library of supported platforms and accessories is quite extensive. Essentially, Player presents generic devices to the controller, such as laser, position control, camera, etc. . These devices may exist either in the real world or simulation. The effect of this is that code is not tailored and dependent on the underlying hardware. The same code will run with different physical robots and sensors as long as the same generic devices are present. A simulation environment such as Stage or Gazebo can even be substituted in without requiring a change in code.

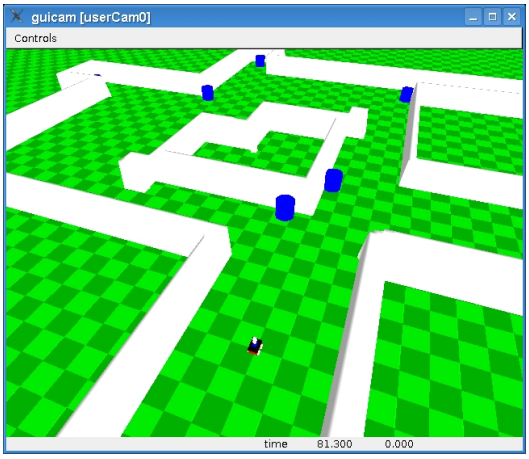


Fig. 2. A screen shot of the Gazebo simulation of the real world Roomba Pac-Man environment.

Player itself acts as a server between the robot and any program that sends commands or requests sensory information. Communication between a client and Player occurs over a TCP/IP network connection. The benefit of this approach is that a client is not tied to the development language of Player. In fact any language that uses TCP sockets can be used. Player itself supplies client-libraries for connecting to a player server in C, C++, Java, Tcl, and Python. However, the open source community has developed numerous other libraries including LISP, Ruby, Scheme, and more.

In the course structure which will be discussed later, simulation in Gazebo plays a key role. Gazebo is a high fidelity 3D physics-based simulation, allowing for a reasonable representation of the real world. Gazebo utilizes the Open Dynamics Engine (ODE) library for integrating physical dynamics and arbitrary kinematic structures. Furthermore, the sensors simulated by Gazebo are modeled after real devices which helps their behavior to match more closely with that exhibited in the real world.

1) Modifications: It was necessary to make several modifications to Player to enable full functionality between Player running on a laptop, the Roomba, and a web camera. The Roomba support in Player only enabled position control and the bump sensor. This functionality was extended to include support for the IR sensors, touch sensors, and control of the vacuum and LEDs. Other additions made throughout Player included bug fixes associated with the original Roomba support, modifications to the Video 4 Linux driver in Player, finishing support for the gripper proxy, and modifications to several of the utilities supplied with player.

2) Install and Setup: The configuration and installation process for PSG can be quite extensive. The majority of the difficulty arises from Gazebo and its dependency on several libraries and OpenGL support. To aid in this process an Ubuntu install script was developed. The script automates everything except for installing the video card drivers to enable hardware-accelerated OpenGL support (software support is added by the script). PSG was also ported to Mac OS X, which not only increases the accessibility of the software,

but opens new possibilities for future controllers.

B. Course Structure

1) Labs: The course consisted of guided discussions along with a series of labs and projects. The first three labs were aimed at getting students acquainted and comfortable with PSG, the Roomba, and the various sensors at their disposal. Lab 1 is an introduction to PSG, simulation, and simple reactive control . Lab 2 provides the first exposure to the use of a camera and object seeking as students must use a camera in Gazebo to navigate between two bright colored fiducials. Finally, Lab 3 serves as the introduction to the Roomba by extending Lab 2 to a physically embodied robot. Due to PSG's nature, the Lab 2 code will run and control the Roomba. However, students are quickly exposed to the discrepancies between simulated and real world sensors and the unpredictability of the real world.

2) Project 1: Subsumption: Lab 3 serves as the lead in for the first project. Project 1 consists of creating a purely reactive Roomba Pac-Man client that follows a Subsumption architecture. Clients must detect collisions with physical and virtual walls, avoid ghost Fiducials (green cylinders), attract to power-up fiducials (orange-green cylinders), collect food (orange beads), and wander when none of the previous objectives are available

3) Project 2: Monte-Carlo Localization: The goal of the second project is to implement Monte-Carlo Localization in simulation. The environment provided is a re-creation of the real world environment that Roomba Pac-Man will be conducted in. The world model includes the floor plan as well as a series of colored fiducials that will be placed in the same locations in the real world to aid in localization. The fiducials are necessary as we do not have laser range finders available for the Roombas, just web cameras. The goal of this project is for students to demonstrate the ability to localize using some combination of the bump sensor, laser range finder, and camera.

4) Project 3: Deliberative Roomba Pac-Man: Project 3 consists of creating a deliberative control policy for the Roomba Pac-Man task in simulation. This project utilizes the code developed in the first two projects, along with the development of a planning algorithm that must take into considering dynamic events such as avoiding ghosts. Students must overcome problems in perception and concentrate on robustness and accurate planning in order to maximize their score.

5) Final Project and Future Work: The final project serves a culmination of the previous projects. Students must combine everything they have learned and done in the class to create a competitive Roomba Pac-Man controller. The project is relatively open-ended with a strong emphasis on the students using what they have learned in the course as well as incorporating novel ideas into their design.

III. Implementation

A. Hardware

The aim for the course is an accessible and cheap robotics platform. It is often the case in robotics that price and accessibility are inversely proportional to performance, functionality and robustness. Fortunately, the iRobot Roomba platform combined with a standard laptop running Ubuntu Linux, PSG, and a web camera provides the ideal solution to this common problem. The Roomba as a well-established consumer product provides the robustness desired of a robotics platform along with a price-point aimed at the average consumer. While the Roomba has limited processing and sensory inputs, coupling the Roomba with a standard web camera and Dell Dimension laptop enables the desired functionality while keeping costs low. The Logitech Communicate STX web camera was chosen as the web camera because it is supported in Linux, runs at 640X480 at 30 frames per second, and it is available at several local electronics stores. The final standardized system consists of a Roomba at $150, a standard Dell Dimension laptop at $500, a Roo-Stick for connecting the Roomba to the Laptop at $25, and a Logitech Communicate STX web camera at $30. The final cost of the entire system falls under $700.

B. Extensions to PSG

1) Camera Proxy: One of the goals in developing the course was to be able to use standard off-the-shelf web cameras with Player. Luckily, Linux drivers providing the standardized Video4Linux API exist for nearly all currently available web cameras including the Logitech Communicate STX. Player's camera proxy retrieves frames from the camera utilizing the libfg frame grabber library. Libfg is basically a C-based wrapper around the Video4Linux API to enable high-level access to video devices.

According to Player's documentation, only one camera had been tested and proven to work with the camera proxy, but any Video4Linux camera "should" work. Unfortunately, this turned out to not be the case, as five different cameras with Linux drivers were tested and failed to work with Player. In all cases Player returned the error that it was unable to read frames from the video device. Player also required that several of the camera parameters, such as color depth, screen size, color-space, etc. be specified for the device by the client. While many cameras have support for different sets of parameters it is frequently the case that only a subset of the available parameters are supported. For several of the cameras tested the available parameters were not known, which made configuration and testing quite difficult.

It was decided that there were two possible sources of the problem. The first, could be a problem with the libfg library being used by Player. The second source could be in the parameter specification. It could be the case that the proper parameters were never found, or that there were parameters that had to be set that Player did not allow the client to configure.

In order to test the first hypothesize the latest version of libfg was retrieved. The latest libfg is significantly different

than that contained in Player. Libfg comes bundled with a small program called Camview which simply opens the video device and performs live streaming. Camview with the latest libfg was able to successfully stream all the Video4Linux web cameras tested. Unfortunately, the libfg within Player was too tightly integrated to be able to extract and test with Camview. Instead, the new libfg was integrated into the camera proxy in Player and all five cameras were retested. With the new libfg one of the five cameras worked, the Lego USB Camera. However, the Logitech Communicate STX still failed to work.

Documentation in Player on the camera proxy stated that it would be great if someone could figure out how to auto detect the camera parameters rather than require their specification. It was noticed that the libfg Camview utility did not require parameter specification. Through analyzing the source of Camview, the method for auto-detecting camera parameters was discovered and this was incorporated into Player. In the new behavior of the camera proxy, user specified parameters are optional and treated as overrides. The benefit of this approach is that those parameters that Player did not allow a user to specify are now guaranteed to be correct. Furthermore, the user does not have to worry about configuring camera parameters unless they have very specific requirements of the device, such as limited dimensions or frame rate.

The auto-detection of camera parameters was implemented on top of the new libfg integration, so the benefit of auto-detection on the original camera proxy is not known. With auto-detection and the new libfg library all Video4Linux cameras tested worked with Player. These changes have since been discussed with developers of Player Stage Gazebo and a patch file has been submitted which should be incorporated into the project in the near future.

2) Roomba Driver: Initially, Player came with support only for position control and reading the bump sensor values. However, the Roomba has several other inputs and outputs available. The structure of Player makes it relatively simple to add new devices to a robot. The robot driver simply has to register and define the desired interface and update the structure appropriately. Player was extended to enable the ability to read the six infrared sensors present on the robot through the IR proxy. The infrared sensors include the IR-Wall detector, a side regular wall detector, and four downward facing floor detectors. An opaque proxy was also added to allow button presses to be retrieved as well as LED colors to be set.

Finally, a gripper proxy was added to turn the vacuum on and off. The gripper proxy is primarily developed for robotic arms in three dimensional space. However, there is no ”vacuum proxy” within Player and it was felt that the gripper proxy provided the most intuitive match for the vacuum. The gripper development proved to be the most difficult aspect of the Roomba development. The gripper proxy was only partially implemented in Player and was not in a functional form. The developmental version of PSG had the complete implementation. However, being developmental, it contained

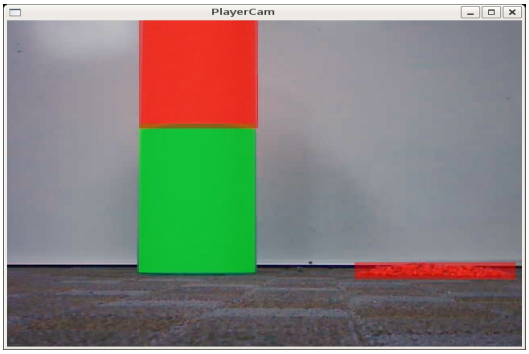


Fig. 3. A demonstration of playercam performing blobfinding with a fiducial and ’food’.

numerous other issues and could not be compiled. Therefore, work was undertaken to port the developmental gripper proxy code into the version of Player being used for the course. Gazebo also needed to be modified to reflect the gripper proxy changes. The Roomba vacuum only has two states, on and off, which are mapped to the gripper proxy open and close commands.

Similar to the camera proxy changes, a patch file was created for the Roomba driver in Player and submitted to the PSG developers. These changes have already been incorporated into the PSG source code. They are available in the development version of Player and should be in the next official release.

3) Player Utilities: Player comes with several utilities to monitor sensors and control Robots. Changes were made to several of these utilities. PlayerJoy and PlayerV both required changes to enable control of the recently implemented gripper proxy. Playerjoy allows joystick-based tele-operation of a Robot and mappings were added to enable a Playstation-style controller to open and close the gripper proxy through the ”X” button. Playerv a program that allows both tele-operation as well as real-time viewing of sensory information was also modified to allow open and close control of the gripper proxy.

The most substantial utility changes involved modifications to the playercam utility. Playercam streams the frames from the camera proxy and overlays the blobfinder results (a color segmentation proxy) in the stream. The blobfinder uses the popular CMVision color segmentation library. The blobfinder performs color segmentation by reading color thresholds from a user-defined colors.txt file. This file specifies YUV ranges and the color that pixels within that range should be segmented into. Playercam’s initial behavior was to return an RGB value of the pixel corresponding to the location a user clicked in the image. Multiple clicks would then need to be converted by hand into YUV to determine a reasonable threshold. Playercam’s behavior was modified to allow the user to select a region of interest in the image and report the range of YUV values for that region. There are several ways to convert from RGB to YUV. Initially, the conversion algorithm used was not the same as that being done within the blobfinder and this created discrepancies in segmentation. The conversion was then changed to the

approximation used by the blobfinder which yielded much better results. This change was extremely helpful for camera calibration as proper ranges for segmentation could be determined quickly.

4) Bug Fixes: Also, during initial testing of Player and the Roomba two bugs were discovered. The first was a timeout that would occasionally occur. Extensive testing revealed that the timeout resulted from a bug in the Roomba communication code. An initial workaround was developed by limiting the length of the timeout and conducting a full restart of the communications. Player developers were made aware of the issue and several weeks later returned a simpler and cleaner fix, which was opening the Roomba serial device without the O_SYNC flag. The O_SYNC flag guarantees that multiple writes to a file are completed in a synchronized fashion. However, in the case of the Roomba and the serial device, commands are being streamed at a high rate which presumably caused synchronization issues. However, the developers stated they were unsure as to exactly why this fixed the problem.

The second bug involved boot up and shutdown sequences. During boot up, a series of commands are sent to turn on the Roomba, then activate the motors, and finally activate the sensors. However, these commands were sent too quickly and the motor activation packet was lost. This bug was solved by adding a small millisecond delay between boot up commands. The problem with the shutdown sequence was easily fixed, Player did not send the command to physically turn off the robot when Player was shutdown. Instead, the user would have to remove the Roomba's battery to shut it off.

C. Installation and Setup

Configuration and installation of PSG can be a rather challenging and time-consuming process made all the worse when multiple installations must be conducted The difficult installation also decreases the accessibility of PSG to students who may want to enable development on personal machines. To this end, it was decided to create an install script for the Ubuntu Linux platform. While the script automates the process it also provides a clear list of what libraries, packages, and steps must be executed in order to duplicate the environment used for the class.

Player itself requires a rather limited number of libraries. However, added functionality can be obtained through several optional libraries. Conversely, Gazebo requires quite a few libraries and packages, with no optional packages. The required libraries and packages, are the mesa OpenGL development libraries, ODE compiled from source, the gtk+2.0 development library for the PSG GUIs, several image libraries, gcc and g++, x11, xml, glib, gdal, and python. Optional libraries include the computer vision libraries, libcv and libxmu, the science library gsl, libgeos, fire wire camera support from libavc1394-dev, and the boost development libraries. The complete list of libraries and the installation process can be found in the install script included in the appendix.

It is also highly recommended that proper video card drivers are installed to enable hardware-based graphics acceleration Video card driver installation requires modifying and recompiling the kernel and the installation process can vary greatly between different manufacturers and models. Therefore, it could not be included in the install script.

The install script also applies patches for player and gazebo to incorporate all the changes that were made for the course.[1][2] This includes the camera proxy, Roomba driver, and player utilities changes. Though the install script was developed in Ubuntu, it utilizes apt-get for all the library installation. Apt-get is a relatively universal library installation utility throughout Linux, which means that the install script will work on other variants of Linux besides Ubuntu as long as appropriate libraries have been compiled and registered with apt-get for that version of Linux.

PSG was also ported to Mac OS X. The reasons for this are two-fold. First, the dual core Macbook laptop proved to be by far the most powerful laptop available to course staff. Second, having OS X ability for the PSG structure opens up new avenues for future hardware for the course, such as Mac Minis.

1) Mac OS X Installation: The port to OS X proved a very difficult task Gazebo, while Player was substantially simpler. All the required libraries were installed using the fink installer, which behaves similar to apt-get in a Linux environment. However, significant issues arose with OpenGL when attempting to install Gazebo. Macs do not have the OpenGL extensions (glx.h). While the Mesa OpenGL libraries do provide the OpenGL extensions, it proved extremely difficult to set up the environments and paths such that the default OpenGL libraries on the Mac would be ignored in favor of the Mesa libraries. There also turned out to be a discrepancy between several of the OpenGL API function calls in the types of the variables being passed in. It turns out that the Mac version of the Mesa OpenGL libraries implement several API calls with integers instead of the GLint type that Gazebo expects. Fortunately, the GLint and integer types are really the same and it was possible to just change the variable types in the Gazebo source. The complete Mac OS X installation has completely functional Gazebo code that runs extremely fast. Player is also functional, however it is lacking camera support. The camera proxy in PSG only provides drivers for video4Linux enabled cameras and Linux firewire cameras. No code exists for reading in frames from a camera in Mac OSX which uses a completely different API.

D. Support Code

Several classes were developed as support code for the projects.[3]. The first class developed was a fake bumper proxy. For the simulation based projects, the sensors available should match those present on the physical Roomba platform. Unfortunately, Gazebo does not currently have

[1] http://cs.brown.edu/people/mmoseley/cs148/player.patch
[2] http://cs.brown.edu/people/mmoseley/cs148/gazebo.patch
[3] http://cs.brown.edu/people/mmoseley/cs148/supportCode.tgz

Fig. 4. An example of a particle filter output using the draw world class. A green square is the robot's true location and light blue is the value determined by localization.

support for bump sensors and the changes and modifications to add bumpers to the simulation would have been quite extensive. Therefore, a fake bumper proxy class was created with the same interface as the bumper proxy in Player. The fake bumper actually utilizes Player's laser proxy. Three laser readings are checked, straight ahead, left at 45 degrees, and right at 45 degrees. If any of the three readings fall below .5 meters then the appropriate bumper values are set to true.

Another support class developed is the truth proxy class. Earlier version of Player contained a device called a truth proxy. This proxy worked only in simulation and returned the true coordinates of the robot within the simulated environment. The proxy goes against the nature and structure of Player and thus, it was removed. However, when testing core concepts in simulation, such as localization, being able to compare estimated location to true location is an essential debugging tool.

The removal of the truth proxy from Player was justified, and it was determined that it should not be reintroduced into the Player source. Instead, an external truth proxy class was developed that connects directly to the Gazebo simulation engine to retrieve the truth information for the robot. Gazebo provides an interface library for retrieving truth information about objects in the world which made the process relatively simple. The current incarnation of the truth proxy returns the two dimensional (x,y) and orientation (theta) of the robot and exposes the same interface to the client as Player-based proxies do.

The final support class developed is a DrawWorld class. The goal of this code is to provide students with a simple means of representing and accessing a map of the world within their code as well as providing a simple method for visualization. Player does contain a map proxy that can read in a map and create an occupancy grid. However, it does not have a method for discerning fiducials within the map and it cannot be easily integrated with a client-designed Monte-Carlo Localization, only Player's built in MCL proxy.

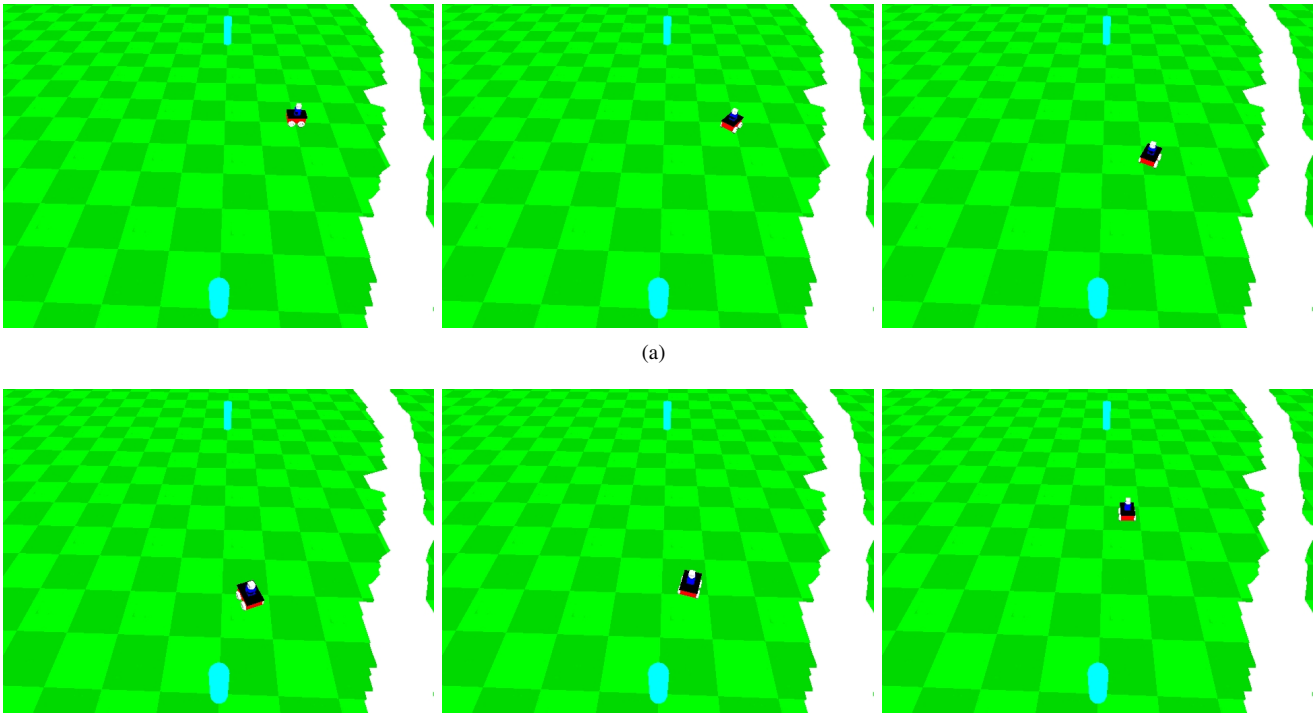
The draw world class allows clients to load a map of the world that exists in png image format. Simple functions are supplied for retrieving the image dimensions, setting and retrieving pixel values in the image and saving the image out into several image formats. These processes are necessary for the deliberative controller projects. Localization and path planning require some internal map of the world including walls and obstacles. Another function is also included in the class to aid in debugging and visualization of the particle filters for the deliberative projects. the function takes a vector of x,y points as input and draws red X's at those locations in the image. The purpose of all this image and visualization support code is to enable students to concentrate on the core robotics problems. In the case of localization, the draw world class also provides a bit of direction on how to approach and debug a complex problem that can seem overwhelming to students new to robotics.

E. Implementation of Labs and Projects

All of the course labs and projects were also implemented well ahead of the class and are available for future iterations of the class in a CVS repository. While this was an arduous task in the limited time available it turned out to be a great benefit. The foremost reason for course staff implementation of the projects was to enable the staff to be able to easily answer student questions and provide help as student's implemented the projects. It turns out that this indeed provided the expected benefits. The source code was frequently used for reference when presented with questions, and it was utilized in the classroom to provide demos of expected results.

The implementation of the labs and projects also had the unexpected benefit of identifying potential pit falls. One prime example discovered are the set and replace rules that exist in Player. In Player every devices streams its sensory information to Player at some rate. Player queues the messages, interprets them one at a time, and then queues the messages for transmission over TCP/IP to the client. By default, it is up to the client to read the messages off the queue. If the client does not read the messages fast enough, the queue becomes backed up and the messages retrieved are no longer up to date. Furthermore, the queue runs out of space for new messages and the new sensory information is dropped. This default behavior can have devastating effects on a controller if students do not take precautions to prevent it. However, the set and replace rules provided by Player allow for this default behavior to be changed. The set rule changes the read behavior conducted by the client so that instead of just retrieving the latest sensory information, all messages are retrieved from the queue with one read. The replace rule builds upon this by making it so that messages from the same device are replaced in the queue. In other words, the client will always receive the most up-to-date sensor information.

Implementing the projects also revealed several potential difficulties in conducting vision based localization. One of



(a)

(b)

Fig. 5. An example of learned behavior by the robot. In this sequence of images, the robot has been taught to drive back and forth between fiducials

the most significant revolved around the coordinate space that Player returns odometry values in. Odometry values are returned in reference to the coordinate space in which the robot started from. Understanding how to transform the odometric coordinate system into that of particles in a localization particle filter proved a difficult task for the course staff. To help students avoid this problem, the problem was brought up by staff in class discussion and several possible solutions were presented.

Finally, it was discovered that vision based localization can suffer greatly from issues such as occlusion, two fiducials appearing as one, and approaching a fiducial so closely that the top and bottom are no longer seen. All these cases can lead to severe errors in estimation of bearing and distance to the fiducials. Solutions for these problems were developed by the staff during the course of implementation. Then, through the discussion oriented structure of the class these potential issues were brought up by the staff and both students and staff presented possible solutions.

F. Mixed initiative Q-learning from Demonstration

Work has also begun on developing a mixed-initiative machine learning control structure for the Roomba Pacman task. The underlying idea is that rather than have a roboticist sit down and program a control policy, a human will be able to tele-operate the robot to perform the desired task and have the robot learn the behavior. So far, we have developed an application within the PSG framework. The robot acts as the student, and watches as a teacher, either human-controlled or a traditionally programmed controller perform the task. While running, the user can switch between learning mode, where the teacher demonstrates and the client learns, and autonomous mode, where the robot exhibits the behavior it has learned. If the client is unsure about what actions to take within a state, the user can simply return control to the teacher to demonstrate the desired behavior.

The underlying learning method utilized is QLearning where the robot basically learns state action mappings. Our QLearning uses the work from (Szepesvari and W.D. Smart 2004). To update each value in the QTable we apply the following Q Function $Q[S][A] = Q[S][A] + \alpha(r(S) + max(Q[S][A'])$ where S is the current state, A is the action executed, $r(s)$ is the reward given to the "student" for being within a certain state, andα is the learning rate. At every iteration the value give for a state-ation pair is updated by multipling the reward and the Q-value of the best action one can take from a given state by the learning rate. Intuitively, the robot is determining what the benefit is for the state it is in and what benefit is achieved from all the actions that can be taken from the state.

To demonstrate its learned behavior the student determines its current state. It then retrieves the possible actions for that state from the Qlearning table and probabilistically determines which action to take. This probabilistic approach takes into account the randomness that may be inherent in the decision making of the teacher.

Attention has been paid to creating a very robust and extensible code structure. The number of states and actions used can easily be modified, and the table representing the learned behavior can be saved and loaded. Furthermore, the

table representing learned behavior has been designed so that it will work with whatever dimensionality of state and action space desired. The table is also template-based allowing changes to the internal representing of a table entry.

This is a very different approach to control policy programming in robotics. It is our hope that the QLearning-based demonstration learning we have begun may become part of the curriculum for future iterations of the course.

IV. Discussion

Overall the course went very well. Feedback from students was encouraged throughout the course and it is evident that many students found the course interesting and motivating. The combination of simulation and real world gave students a well rounded look at robotics and aided in prototyping and testing. The development of this course has also resulted in a very solid and well structured codebase that is easily reproducible and modifiable. Furthermore, thanks to PSG, the structure developed is easily expandable to new and different platforms as robotics continues to innovate and grow.

PSG itself has also received enhancements that have already begun to be incorporated into the official source to benefit roboticists throughout the world. Roombas in PSG now have full support for reading of all their sensors and control of the vacuum. PSG also now works with a much wider range of web cameras and color segmentation is a much less tedious process.

Unfortunately, despite efforts to prevent students from getting bogged down in hardware, configuration and general non-robotics issues, it still inevitably happened. Common problems involved improper use of Player, and crashes in Gazebo due to zooming out too far or running for too long. Most of these problems can be significantly reduced in future years just by enumerating these problems and their solutions to future students.

There were also significant problems with vision in the real world. Cameras are extremely sensitive to changes in light. The effect is so significant that due to the windows present in the Roomba Pac-Man area it was not possible to run during the day. Even at night, careful attention had to be paid to fiducial placement so that significant shadows were avoided. Another problem with lighting is that the Logitech cameras contain hardware-based automatic white balancing. This means that the camera would auto adjust the brightness when the overall brightness of the image changed, making similar colors appear significantly different. Unfortunately, the automatic white balance is hardware-based and no immediate solution exists to fix the problem. Instead, it must be compensated for by relaxing the color segmentation thresholds.

Other vision problems were also present due to the specific environment chosen for Pac-Man. Glass reflections often resulted in phantom objects being seen. There are also several railings present that as far as the Roomba is concerned act as walls. However, fiducials are seen by the cameras through the railings, resulting in significant localization issues. There are only two apparent options for fixing these problems in the future. Either conducting Roomba Pac-Man in a different environment, or making modifications to the environment such as placing sheets over the railings and on windows.

Another major issue with the hardware involved the connection between the Roomba and the laptop. The Roomba's default connection is serial, so a special device called a Roo-Stick was purchased that converts the serial connection to USB for connection to the laptops. However, these connectors have significant heating issues and are quite fragile. Through the course of the semester, nine of the Roo-Sticks stopped functioning from what appears to be overheating issues. Some of the sticks were clearly left connected for too long. However, other brand new sticks overheated in under 30 minutes. In the future, alternatives to the Roo-Stick should be explored in depth or discussion should be opened with RoboDynamics, the manufacturer of the Roo-Stick, to resolve the problem.

Also, the Roomba sensors are not quite as accurate as desired. Odometry returned from the Roomba is extremely noisy and can vary greatly between different Roombas. Also, the infrared wall detector present on the Roomba can easily be completely blocked by sensors or devices placed on the Roomba. During demos the Roombas would frequently drive right past infrared walls, not because the student's failed to detect them, but because the signal failed to be perceived by the sensor. Unfortunately, there are no obvious solutions to these problems at this time.

V. Future Improvements

There are quite a few possibilities for future improvements to the course structure, the hardware, and PSG. Major concentration should be paid to solving the difficulties of vision based localization in the real world. One possible solution would be to increase the number of fiducials and their colors. However, careful consideration needs to be paid to ensure that new fiducials do not result in the possibility of merging or occlusion which create their own problems with localization. Also, two many different colored fiducials could simplify the problem too much. For instance, if every fiducial was uniquely colored, then localization is trivial as students could just dead reckon off the fiducial they see.

Another possible solution, rather than increase the number of fiducials is to add a second camera. Primitively, a second camera can be used to simply double the field of view of the camera, allowing it to see more fiducials at one time. However, a second camera also allows for the possibility of stereo vision. While stereo vision would be too difficult for students to be expected to implement, PSG does contain a stereo vision proxy that may perform all the necessary computation. More research should be done into the PSG implementation of stereo vision. Two cameras do have the drawback of increased computation. Several students had complaints about performance of the laptops, so a second camera may have too much of a negative impact on performance.

Finally, a less awkward hardware solution would be nice to have in the future. Currently, the laptops are too large to be adequately mounted on the Roomba, a smaller more compact controller capable of running PSG would be ideal. Possible choices are the Gumstix micro-controller or the Mac Mini. Both these approaches would require some form of power solution, and the battery of the Roomba will most likely not be strong enough to support a Mac Mini. PSG on the Mac OS X also cannot currently read from a web camera. A mac camera driver would need to be developed and incorporated into the camera proxy in Player.

It is also our hope that the work begun in Qlearning mixed initiative demonstration learning may be incorporated into the class in the future. Demonstration learning would provide students with an alternate view on how to approach programming in robotics. We feel that demonstration learning has alot of potential and may become an integral part of robotics in the future.

VI. Conclusion

Overall, the robotics course was very successful. A significant robust and re-producible robotics platform and code base has been developed. The Player Stage Gazebo Roomba driver, camera proxy, and utilities have also been enhanced. Some of these changes have already been incorporated into the project, and the rest should be added in the near future. Thus, the effort and work put into development of this course will reach far beyond the confines of Brown University and hopefully benefit people throughout the world. Finally and most importantly, students seemed to enjoy the course and gain a very good understanding of robotics.

APPENDIX

```
#!/bin/bash

# Install script for psg on ubuntu 6.0.6


# Set all necessary environment variables
echo "export PKG_CONFIG_PATH=${HOME}/local/lib/pkgconfig:$PKG_CONFIG_PATH" >> \
     ~/.bashrc
echo "export PYTHONPATH=\
      /usr/lib/python2.4/site-packages:/usr/local/lib/python2.4/site-packages:\
      ${HOME}/local/lib/python2.4/site-packages:$PYTHONPATH" >> ~/.bashrc
echo "export PATH=${HOME}/local/bin:$PATH" >> ~/.bashrc
echo "export CPATH=/usr/local/include:${HOME}/local/include:$CPATH" >> ~/.bashrc
echo "export LIBRARY_PATH=${HOME}/local/lib:$LIBRARY_PATH" >> ~/.bashrc
echo "export LD_LIBRARY_PATH=${HOME}/local/lib:$LD_LIBRARY_PATH" >> ~/.bashrc


source ~/.bashrc
mkdir psg
cd psg



# Required installs
#================================

#The usual development libraries
sudo apt-get -y install gcc g++ make

#GTK
sudo apt-get -y install libgtk2.0-dev




# GUI and graphics libraries



# Mesa and Open GL
sudo apt-get -y install libglu1-mesa-dev
sudo apt-get -y install mesa-common-dev libglut3-dev

# Some more GUI stuff
sudo apt-get -y install libxml2-dev libx11-dev libltdl3-dev



sudo apt-get -y install libglib2.0-dev libgdal-dev gdal-bin
#Gdal image stuff
sudo apt-get -y install libpq-dev libungif4-dev libhdf4g-dev \
                cfitsio-dev libjasper-dev libtiff4-dev \
libxerces27-dev netcdfg-dev
```

```
#Python libraries
sudo apt-get -y install python2.4-dev
sudo apt-get -y install python-wxgtk2.6 python-wxtools wx2.6-i18n

#ODE Library (special flag needed, so no package)

wget http://archive.ubuntu.com/ubuntu/pool/universe/o/ode/ode_0.5.orig.tar.gz
tar -xvzf ode_0.5.orig.tar.gz
cd ode-0.5
echo "OPCODE_DIRECTORY=${HOME}/psg/ode-0.5/OPCODE" >> config/user-settings
make configure
make ode-lib
cp -r include/ode/ ../../local/include/
cp lib/libode.a ../../local/lib/
chmod -R 777 ../../local/include/ode/
chmod 777 ../../local/lib/libode.a


cd ..


# Python/Java binding library
sudo apt-get -y install swig



#=================================


# Optional stuff for additional psg libraries
#=========================================

# Computer vision libraries
sudo apt-get -y install libcv-dev

sudo apt-get -y install libxmu-dev


# Advanced science libaries
sudo apt-get -y install libgsl0-dev


# Geo drivers
sudo apt-get -y install libgeos-dev

#1394 camera drivers
sudo apt-get -y install libavc1394-dev


#boost drivers (faster c algorithms)
sudo apt-get -y install libboost-dev
sudo apt-get -y install libboost-signals-dev
sudo apt-get -y install libboost-thread-dev


sudo apt-get -y install lib3ds-dev
```

```
sudo apt-get -y install proj

#============================================




# Player and Gazebo Installation
# ============================================

cd ~/psg

# download player
wget http://superb-east.dl.sourceforge.net/sourceforge/playerstage/player-2.0.2.tar.bz2
chmod 775 player*

# unzip/untar player
bunzip2 player-2.0.2.tar.bz2
tar -xvf player-2.0.2.tar

# download the cs148 patch
wget http://cs.brown.edu/people/bcd/patch.patch

#apply the patch file
patch -p0 <patch.patch




#download gazebo
wget http://superb-east.dl.sourceforge.net/sourceforge/playerstage/gazebo-0.6.0.tar.gz
tar -xvzf gazebo-0.6.0.tar.gz

# download the cs148 patch
wget http://cs.brown.edu/people/bcd/gazebo.patch

patch -p0 <gazebo.patch


# Configure player
cd player-2.0.2
./configure --prefix="${HOME}"/local --disable-p2os

# Install player
make install

# Configure gazebo
cd ~/psg/gazebo-0.6.0
./configure --prefix="${HOME}"/local

#Install gazebo
make install
```

```
cd ~/psg/


# ==========================================
```