Enabling Semi-Autonomous Manipulation on iRobot's PackBot

A Major Qualifying Project

Submitted to the Faculty of

Worcester Polytechnic Institute

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

By

Jessica Gwozdz

Nicholas Morin

Ransom Mowris

Date Submitted:

May 1, 2014

Sponsoring Organization:

iRobot Corp.

Project Advisor:

Professor Dmitry Berenson

Table of Contents

Table of Figures

Executive Summary

Human-robot teams have proved themselves to be useful in a wide variety of situations. One particularly useful application is in disaster zones and battlefields. Robots are able to reach areas that are inaccessible by humans and perform dangerous tasks by providing a mechanical proxy for their operators. Presently, search-and-rescue robots are just that – proxies. Their operation requires the complete attention of a human. In many dangerous situations where such robots are deployed, this distraction can put the robot's operator in danger. As a result, reducing operator workload is a major goal in search-and-rescue robot development. To do this, researchers endeavor to make robots more capable of completing tasks with little or no user input. Currently, autonomous navigation is a well-developed concept and some capabilities are commercially available. But autonomous manipulation is only beginning to be incorporated into field robots. We demonstrate how some of these capabilities could be integrated into iRobot's PackBot.

Our project aims to demonstrate intelligent motion-planning for the manipulator on iRobot's PackBot. Because controlling the arm is one of the more difficult aspects of teleoperating the robot, motion and grasp planning are a particularly useful part of the system to automate. To do this, we developed an interface between PackBot and The Open Robotics Automation Virtual Environment (OpenRAVE). With our work, we hope to provide a foundation for the development of autonomous manipulation behavior. We will demonstrate the capabilities of our system by enabling users to command the robot to grasp objects recognized by a sensor with a single click.

To accomplish this objective, we needed to complete several preliminary objectives. First, we modeled the robot within OpenRAVE. In order to do any motion-planning at all, OpenRAVE needed an accurate model of the robot. Next, we implemented an inverse kinematics solver for the PackBot's arm. To send trajectories to the robot, we developed a communication protocol and wrote a trajectory-executing controller that ran on the PackBot. After that was finished and tested, we integrated a PrimeSense 3D sensor with the PackBot by designing a secure hardware mount and writing a plug-in for OpenRAVE that received and interpreted data from the sensor. Using this 3D sensor and Point Cloud Library, we next incorporated object-recognition into our system, choosing to focus on recognizing cylinders (for grasping) and obstacles (for collision avoidance). Finally, we incorporated our system into a user interface that displayed settings for our software, camera feeds from the robot, and a view of the OpenRAVE simulator. Through this user interface, users could instruct the robot to attempt to grasp objects recognized by the sensor.

With our work, we demonstrate an approach to traded-control autonomy for iRobot's PackBot. Our system will require further work to incorporate into commercially available PackBots, but we believe we have demonstrated a compelling proof-of-concept. Our system is modular, expandable, and reasonably robust. With additional tuning and some improvements to our algorithms, our work could lead to a valuable addition to PackBot's operating system.

Introduction

iRobot's PackBot is the most successful defense-and-security robot in the world. PackBots have helped search-and-rescue efforts at Ground Zero after 9/11, disarmed improvised explosive devices in Iraq and Afghanistan, inspected suspicious packages at the scene of the Boston Bombing, and aided in clean-up efforts after the meltdown at Fukushima. PackBot keeps humans safely removed from dangerous situations by serving as a robotic proxy for its operator. Presently, PackBot is a fully teleoperated robot – a human controls its every motion using a laptop and a gamepad. Picking up objects with teleoperated control requires training, time, and attention. In many situations, it would be beneficial for the robot to act on its own, perhaps only supervised by a human. If some functionality could be automated, PackBot operator workload would be reduced and PackBots would become more capable assistants in a wider variety of situations. Reduced workload would also allow operators to dedicate more attention to their situation and surroundings rather than their PackBot. Because PackBots are often deployed in chaotic, dangerous environments, this could be very beneficial for the safety of PackBot operators.

Our project set out to make PackBot smarter by integrating an open source motion-planning and simulation library with PackBot's Aware2 operating system. This library, Open Robotics Automation Virtual Environment (OpenRAVE), was primarily developed at Carnegie Mellon University by Rosen Diankov and James Kuffner (Diankov & Kuffner, 2005). It provides motion-planning, grasp planning, and simulation functionality in an expandable platform. Its built-in libraries and expandability through plug-ins facilitate the process of implementing robot motion-planning and autonomous behavior for a wide variety of robots.

By integrating PackBot with OpenRAVE, we hope to provide a foundation for further autonomous manipulation research. We hope to demonstrate the capabilities, expandability, and ease-of-development of our OpenRAVE-PackBot system. To showcase our work, we integrate a 3D sensor, perform simple object and obstacle recognition, and enable click-to-grasp functionality within OpenRAVE. This serves as proof that OpenRAVE can reduce the workload and attention required to operate PackBot. If developed further, we believe this functionality could be implemented on commercial PackBots to lower PackBot's learning curve, reduce operator workload, and make PackBot a more capable field assistant.

Background

In this section, we will introduce the tools and research that made our project possible and inspired our work.

iRobot PackBot

Figure 1 - iRobot PackBot with Explosive Ordnance Disposal (EOD) Arm

iRobot's PackBot, shown above in **[**Figure 1**],** is the most successful military ground robot ever developed (Yamauchi, 2012). Over 4,500 PackBots have been deployed to explosive ordnance disposal teams, police forces, infantry regiments, and disaster relief/search-and-rescue organizations all around the world. It has saved hundreds of lives by inspecting and disarming improvised explosive devices in Iraq and Afghanistan (Yamauchi, 2013). PackBot is an extremely rugged robot. Its chassis is able to withstand up to 400Gs, or about a six foot drop. It is also waterproof up to 3 meters (Yamauchi, 2004).

Our PackBot was equipped with a 3-link arm – the Explosive Ordnance Disposal (EOD) manipulator. This 5-DOF arm is capable of lifting up to 30 pounds. Presently, except for functionality for driving the arm to a few predetermined "poses" in iRobot's user interface, the arm is entirely teleoperated. Operating the robot requires the careful attention of a human operator who uses an Xbox-like gamepad to control the robot over a radio connection.

PackBot's battle-proven hardware provides an intriguing platform for robotics research. In addition to such projects as PackBot Griffon, which enabled flight with a gas-powered motor and steerable parafoil (Yamauchi & Rudakevych, 2004), chemical/radiation sensors (Scott et al., 2003), automatic sniper-fire detection and location (Deligeorges et al., 2008), and a project to retrieve injured soldiers from the battlefield (Gilbert et al., 2006), iRobot has invested significantly into developing autonomous behavior. Most of these studies relate to autonomous navigation and robot-assisted driving (Yamauchi, 2005; Yamauchi, 2006; Yamauchi, 2012; Yamauchi & Massey, 2008). A similar project to ours was PackBot LABRADOR, which involved searching for, recognizing, and retrieving small objects in complex environments. But grasp-planning was beyond the scope of the project (Yamauchi et al., 2013). Besides a currently-ongoing project at iRobot to enable autonomous door-opening, we could not find any information on autonomous motion-planning or grasping research. We believe there is a need for this, and many researchers agree that autonomous or semi-autonomous grasping is a valuable feature for field robots (Kemp et al., 2007; Casper & Murphy, 2003; Burke et al., 2004).

OpenRAVE

The Open Robotics and Automation Virtual Environment (OpenRAVE) was developed by Rosen Diankov and James Kuffner at Carnegie Mellon University's Robotics Institute. The project's ongoing goal is to provide a standard platform for "3-D simulation, visualization,

planning, scripting and control" (Diankov & Kuffner, 2008). With this functionality, OpenRAVE facilitates the development of autonomous manipulation behavior. Its plugin functionality enables roboticists to focus on coding new functionality for robots without needing to "reinvent the wheel" by re-implementing motion planning, simulation, sensor integration, and control for every new project. With OpenRAVE, moving autonomous manipulation projects from simulation to real robots should take "only a few weeks rather than several months" (Diankov & Kuffner, 2008). Below, in **[**Figure 2**]**, are a few screenshots of the OpenRAVE simulator as it plans grasps.

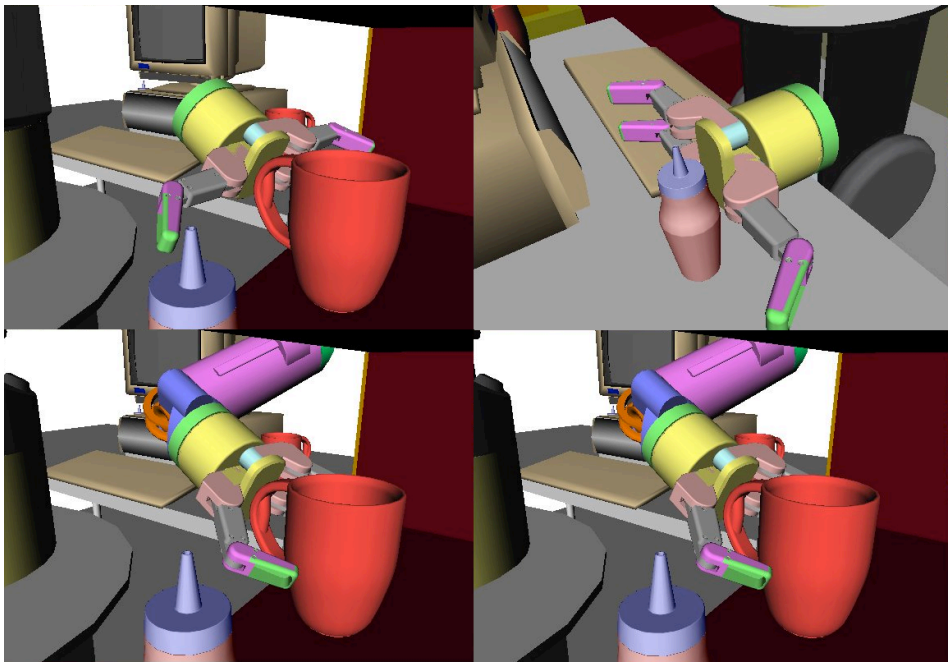


Figure 2 - A few views of grasp-planning in the OpenRAVE simulator

One of OpenRAVE's most useful features is its expandability through plug-ins and scripts. The diagram below, in [Figure 3**]** shows where plug-ins fit in the OpenRAVE architecture. The visual interface can be added onto, callbacks can be programmed to respond to

user interactions with the environment, additional sensors can be integrated, and new robots can be interfaced with OpenRAVE with custom robot controllers.

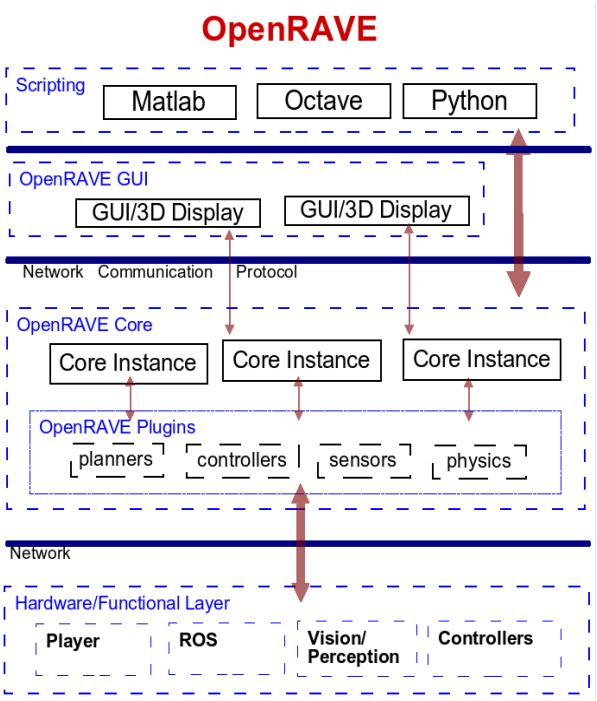


Figure 3 - OpenRAVE Architecture (Diankov & Kuffner, 2008)

Human-Robot Interaction with Disaster Relief Robots

A human-robot team can be an effective asset in the field. By combining the hardiness and capabilities of a robot with a human's perception and guidance, human-robot teams can be more capable than humans or autonomous robots working alone. The benefits of deploying search-and-rescue and explosive ordnance disposal robots are multifold and have been established through extensive research on robots in the field (Burke et al., 2004; Qian et al., 2006; Khatib et al., 1999; Yamauchi, 2004; Yamauchi, 2013). But there are several factors that can impede the performance of such a robot. A study analyzing human-robot interaction in disaster-relief teams operating at the World Trade Center site after September 11th elucidated several of these factors. This research is particularly relevant because PackBot was deployed at

Ground Zero. Researchers found that effective robot utilization was significantly hindered by several human and robotic factors (Casper & Murphy, 2003):

On the human side of the equation, sleep deprivation, lack of training, and a lack of situational awareness made operating the robots difficult. For example, a team from Foster-Miller worked for 56 hours without sleep while operating their robot at the site. The resultant cognitive errors resulted in misjudgments and mistakes while controlling robots. Faults and oversights in the design of robots and their interfaces compounded these mistakes. A lack of robot state awareness led operators to waste 54% of deployment time figuring out the arm configuration, error state, and the mobility state of their robots (Casper & Murphy, 2003). The robots' lack of mapping capabilities led to further confusion and lost robots. Communication dropouts further compounded the errors and significantly reduced operator confidence in the robots' abilities. Finally, the author argued that a lack of sensors decreased the utility of robots in this situation. The author noted that 3D information about the robot's surroundings would have been particularly helpful, citing as an example a robot that drove straight through a meter-long metal rod, severely impaling itself and compromising its mobility. The author noted that this incident would have been entirely avoidable with 3D perception.

The authors ended their paper with recommendations for increasing the effectiveness of disaster relief robots. Of their eleven recommendations, our project addresses six. We outline these six recommendations below:

1) *Human-computer interfaces and robot systems need to support people working without sleep and in an environment worse than the WTC disaster* – Increased autonomy leads to less reliance upon human operators. If the robot's

performance is less affected by its operator's deficiencies, the human-robot team will be more effective.

2) *Never allow a robot without proprioceptive sensors (sensors that provide measurement of movements relative to an initial frame of measurement) and image processing to be rated for unmanned search and rescue* – Casper observed that operators wasted a significant amount of time figuring out the orientation and configuration of their robots. He also suggests that low-resolution traditional cameras are inadequate for search-and-rescue situations and suggests some form of three-dimensional sensing. A lack of sensor information reduces the confidence and effectiveness of operators. Proprioceptive sensing has been incorporated into the autonomous navigation packages available for PackBot, but 3D image processing has not (Yamauchi, 2005). By incorporating a 3D sensor and displaying our robot in a simulator, operators will more easily understand the state of a PackBot controlled by OpenRAVE.

3) *Perform studies on ideal unmanned search and rescue specific user interfaces and robotic systems* – Casper observed that operators without any training were often called upon to control robots. Confusing interfaces led to steep learning curves and wasted operators' time. Frustration with interfaces made some operators abandon their robots and revert to traditional search-and-rescue methods that unnecessarily put humans in danger. Casper suggested that rescue workers would be more willing to use unfamiliar robots if their interfaces were more straightforward.

4) *More research is needed in perceptual user interfaces* – The robots observed for this study only transmitted video and audio to their operators. The positions of manipulators and treads were only represented within the camera stream. iRobot's Operator Control Unit's interface has made significant progress since this study's publication and incorporates many of Casper's suggestions. But we will demonstrate three additional improvements by displaying the robot's grasping capabilities, incorporating 3D sensor data, and representing the robot's camera's field of view within the simulator.

5) *Concentrate on researching how to extract and represent state of the robot and state of what has been seen* – Casper identified mapping and localization as the primary answers to this suggestion. But he also recommended that robot state cannot be overlooked. By this, he meant the configuration of the arm and the robot's capabilities. Since 2001, PackBot's user interface has been updated to display the arm's current configuration, but no indicators of robot capabilities have been added. Our project answers this requirement by displaying both the graspability of objects recognized by our 3D sensor and the camera's field of view.

6) *Investigate the user confidence in remote robots with intermittent communications* – Our project will respond to this by reducing the robot's dependency upon a network connection to its operator. By planning motions autonomously and only requiring user input in the form of grasp targets, intermittent communication will become less of a problem.

Throughout their paper, Casper and Murphy compare human-robot disaster relief teams to traditional human-canine partnerships. Human-robot teams were novel in 2001 but human-canine teams were a proven strategy in search-and-rescue long before 9/11. Although a robot might seem more capable when comparing straight statistics – the PackBot's ruggedness, floodlights, camera streams, and brawny arm and manipulator suggest it should be more effective than even the burliest German Shepherd – human-canine teams consistently provided more utility at Ground Zero. The authors suggested that this is due to completely different relationship dynamics. In a canine unit, the dog takes input from its handler but largely works independently, reporting back when necessary. In contrast, robots acted as the remote arms, ears, and eyes of their operators. If the operator became distracted, the robot would simply lie idle. To improve the effectiveness of human-robot teams, Casper and Murphy suggested conducting research that would turn the human-robot team into more of a collaborative relationship than an authoritarian one (2003).

Human-Robot Collaboration/Traded-Control Autonomy

Much has been written about semi-autonomous behavior in which control is traded between the robot and a human operator (Kortenkamp et al., 1997; Hayati & Venkataraman, 1989; Inagaki, 2003). Kortenkamp and his team defined a scale of control which ranged from complete teleoperation to team control (Kortenkamp et al., 1997). This scale is outlined below:

> **Teaming**: Robots and humans both have full autonomy, but work together as a team. The human commands the robot by defining goals.
>
> **Supervisory**: Robots work nearly autonomously, but the human watches and stops the robot when necessary. Commands are given to the robot as task sequences.

Traded: Robots perform most tasks autonomously, but the human takes control of the robot for more complex tasks. The human can command the robot through task sequences or through skills.

Guided: A human always guides the robot through a task but the robot has some autonomous capabilities. The authors offered obstacle avoidance and grasping as examples of these capabilities.

Teleoperation: The human controls the robot directly, controlling all motions.

The results of our project will be a guided-autonomy interface. With additional system design, our project could transition to supervisory control. This could be demonstrated by implementing commands such as "collect all of the objects in this area and bring them to this area," and incorporating the autonomous navigation capabilities developed by iRobot. The benefits of mixed-initiative control models such as these are multifold.

User interfaces were mentioned as an essential consideration in semi-autonomous control by several authors (Kortenkamp et al., 1997; Hayati & Venkataraman, 1989; Inagaki, 2003; Ferguson et al., 1996; Ferguson & Miller, 2007). The essential requirements of a traded-control interface were defined as: the ability to represent information on the machine's status, goals, beliefs, and intentions (Kortenkamp et al., 1997; Inagaki, 2003), non-rigid management of the exchange of control (Ferguson & Miller, 2007), clear representation of faults in autonomous control (Inagaki, 2003), and allowance for the user to change the robot's intentions and goals and alter the robot's beliefs of its perceived situation (Kortenkamp et al., 1997).

Inverse Kinematics

Inverse kinematics involves solving for the joint angles required to reach a desired end-effector position and orientation (Sicilliano & Khatib, 2008; Spong & Vidyasagar, 2008; Sciavicco & Sicilliano, 2000). Solving inverse kinematics solutions for a five degree-of-freedom arm such as PackBot's requires a more nuanced approach than 6DOF arm solutions. Although the arm can reach every position within its configuration space, it cannot attain each position with any orientation (Sicilliano & Khatib, 2008). Because of this, there is a trade-off between reaching a desired end-effector position and orientation and one must be weighted above the other.

Iterative IK, used in our project, involves computing a set of gripper poses, finding the one that brings the gripper closer to its desired pose, then iterates from that position. Once it reaches within some tolerance *damply squares* prevents guessing a position at a singularity (Wampler, 1986).

Grasp Planning

Many researchers have described methodology for grasp optimization for autonomous robots (Berenson et al., 2007; Markenscoff et al., 1990; Bicchi & Kumar, 2000). Their efforts primarily center around scoring grasps based on certain factors. Although much of the work in this field is beyond the scope of this project, two papers are of particular relevance. First, *Automatic grasp planning using shape primitives* describes an approach for optimal grasps for geometric solids like the cylinders our project endeavored to grasp (Miller et al., 2003). The second paper describes grasp verification without tactile sensor feedback (Jang et al., 2012). This

is relevant because the end effector on PackBot does not have tactile sensing.

Grasp Planning for Geometric Solids

The authors approached grasp planning by pre-determining a grasping strategy for several simple shapes. When grasping cylinders, their approach preferred to grasp from the side rather than the top (though they also specified a grasp strategy for the latter). When planning a grasp, their algorithm attempts to find a gripper approach that would be perpendicular to the central axis of the cylinder, or "in the plane containing both the approach direction and the central axis of the cylinder, in order to pinch it at both ends" (Miller et al., 2003). Although our PackBot's gripper is quite different from the one used in their research, we used a similar method in our grasp-planning algorithm.

Grasping without Tactile Sensing

In this paper, the authors describe a novel approach for detecting a valid grasp without tactile sensors. The researchers estimated the validity of a grasp by measuring differences in joint torque (Jang et al., 2012). The authors were able to reliably detect when the robot made contact with objects and when an object had slipped out of the robot's grasp. Although this work is very relevant to grasp-planning for PackBot, the lack of joint torque feedback makes it unusable for our efforts.

Objectives

Our project seeks to reduce the control requirements for PackBot by enabling users to command the robot to grasp objects by selecting an object. We identified six intermediate tasks that would be necessary to implement this functionality. This section will briefly outline each task. Project components are listed in descending priority.

1 - Accurately simulate PackBot

Before we could implement any motion-planning, we needed a realistic kinematic simulation of the robot. This would be done within OpenRAVE's simulator. This model would include information about the arm's kinematics which would be used for inverse kinematics solving and generating trajectories.

2 - Plan trajectories for PackBot's arm

Before we could drive the robot, we needed to implement a trajectory planner within OpenRAVE. Thankfully, OpenRAVE has functionality for generating a trajectory when supplied with a desired end-effector pose. So this task was significantly simplified. But we would still need to implement an inverse kinematics solver to generate our desired end-effector poses.

3 - Execute trajectories generated by OpenRAVE on PackBot

Once a trajectory is generated in OpenRAVE, it is necessary to send it to the robot for execution. To do this, we needed a communication protocol connecting OpenRAVE with PackBot. We also needed a program that would control the execution of trajectories from the robot and ensure accuracy in execution.

4 - Integrate a 3D sensor with OpenRAVE and the robot

Once we had a way to control the robot, we wanted to be able to sense the area around the robot. This would be useful for planning grasps and avoiding collisions with the environment around the robot. We chose to focus on three things: first, we needed to create a secure mount for our 3D sensor. Any slack in the sensor's position in relation to the robot would adversely affect the reliability of sensor readings. Next, we endeavored to recognize cylindrical objects. We planned to use these as grasping targets. Finally, we set out to import obstacles around the robot into OpenRAVE so they could be avoided in motion-planning.

5 - Grasp cylinders recognized by the 3D sensor

We planned to incorporate all of these elements into a single system that could plan a trajectory to a targeted object and grasp it. This would require calibration of data from the 3D sensor and verification of the accuracy of trajectory execution.

6 - Enable control through a user interface

Finally, we planned to incorporate all of these elements into a user interface that would enable users to select options within our system and define grasp targets.

Methodology

Accurately simulate PackBot within OpenRAVE

To control and display the robot within OpenRAVE, we needed an accurate model of the robot in a compatible file format. iRobot supplied us with a full SolidWorks assembly of the robot chassis and manipulator, but finding a method to convert this model into an OpenRAVE-accepted format was not straightforward. OpenRAVE uses an XML-based modeling format. Collada formats can be included from OpenRAVE XML. No straightforward SolidWorks-Collada export existed, we sought another method. We started by converting the SolidWorks assembly to Blender's format, then saving to Collada from there. But this was unsuccessful because OpenRAVE and Blender used different versions of the Collada format.

We discovered that SolidWorks could export VRML. The VRML files could be included from the OpenRAVE XML. These exported versions were not perfect. First of all, the coordinate frame used for robot commands in OpenRAVE and the coordinate frame of the SolidWorks model were different. We accounted for this by rotating the entire PackBot in the XML file. Our efforts were further complicated by a misrepresentation of joint angles. These were adjusted based off of the real joint angles found from the Solidworks assembly.

We quickly discovered that using the detailed SolidWorks-exported assembly for collision checking with OpenRAVE's physics engine was processor-intensive because the model was too complex. To simplify the calculations OpenRAVE needed to make, we created a collision model with much simpler geometry than the "true" PackBot chassis. Instead of a detailed VRML file, this base uses a simple box with the same dimensions as the chassis.

To model the current state of the robot, we needed to send joint data from the robot to OpenRAVE. To do this, we included functionality for the transmission of joint state values from the robot to OpenRAVE within our communication protocol. This communication protocol will be outlined in the trajectory execution section which follows.

Plan trajectories for PackBot's arm

To plan trajectories, we needed to first implement an inverse kinematics (IK) solver. Our system first uses Fast Library for Approximate Nearest Neighbor to search a database of gripper poses and their corresponding arm configurations for the nearest neighbor to the desired pose. Next, we improve upon this pose by using an iterative IK Jacobian solver. Once an IK solution is found, we pass this to OpenRAVE's default trajectory planner, BiRRT (bi-directional rapidly exploring random tree), to generate a path.

Execute trajectories generated by OpenRAVE on PackBot

To send trajectories to the robot for execution, we drafted a custom communication protocol between OpenRAVE and PackBot. We decided early on that commands would be sent as packets which contained the type of command being sent (*Send Trajectory, Disengage Brake and Execute Trajectory, Pause Trajectory/Engage Brake, Clear Trajectory, Open Gripper, Close Gripper)* and any relevant data needed for that control. These packets, containing the five joint values needed to model the arm, are time-stamped and transmitted at 10Hz. We learned early on that using time-stamped values was necessary. Without the timestamps, it was harder to plot the timing of actual and desired joint values. Without plots, we had a harder time verifying the robustness of our system. Trajectory commands, for example, contain 5 values: one for each degree of freedom of the arm. To execute commands on the robot, we wrote a Python module

that receives trajectories and stores these sets of joint angles in a queue and waits for the execution command.

A separate thread running on the robot checks the desired joint angles against the values reported by the PackBot operating system. Our PackBot originally sent these updates at 10Hz, but we increased this to 30Hz based on results from trajectory testing. Our trajectory follower uses proportional control to calculate joint velocities based on error. When the error threshold is low enough, the next set of desired joint values are processed.

A system diagram of our code can be found in **[**Appendix A - System Diagram**]**. This shows how a trajectory is planned, transmitted, and finally executed on the PackBot. A simplified model of our system is included below, in [Figure 4**].**

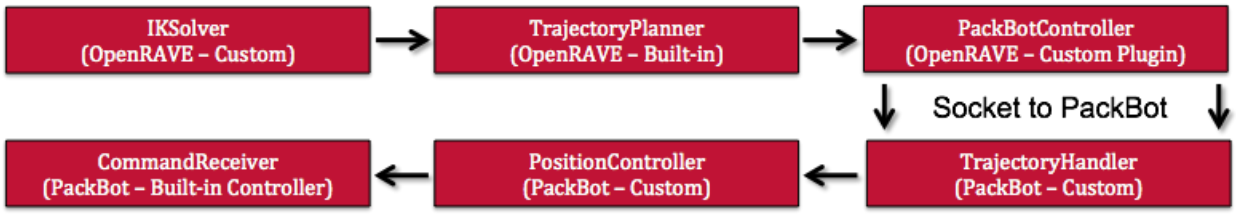


Figure 4 - System model displaying how IK solutions are turned into trajectories and executed on the robot

Integrate a 3D Sensor with OpenRAVE and the PackBot

Incorporating our PrimeSense-based ASUS Xtion sensor was a four-step process. First, we needed to design a secure mount to affix the sensor to the robot. Next, we needed to create a socket to bring data from the sensor into OpenRAVE. Finally, we interpreted data from the sensor. First, we wanted to recognize cylindrical objects to target them for grasping. Next, we incorporated obstacle detection, to inform OpenRAVE's motion-planning to avoid collisions with the environment.

i. Mounting the sensor

The first step in integrating our 3D sensor was designing secure a mount for it on the robot. The rapid prototyping capability of a 3D printer was invaluable in its design process. For less than ten dollars of plastic, we were able to prototype four different designs over the course of two weeks. We decided to mount the sensor on the camera head of the robot to enable users to move the sensor like the camera. We endeavored to make the mount as stable as possible. Stability is absolutely essential when using a 3D sensor because accurately importing sensor data to the simulator position requires knowing the sensor's exact position and orientation in relation to the robot. Our initial approach used the payload mount next to PackBot's camera. The final design used 3M Command Strips to adhere to the top of the PackBot's camera. We decided upon this design because its orientation kept it away from the arm and it was more stable when compared to our initial solution which used the standard payload mount.

ii. Importing Sensor Data into OpenRAVE

To handle the data from the sensor, we captured its point cloud using Open Natural Interaction (OpenNI). OpenNI is an open source framework for 3D sensors. It supports PrimeSense-based sensors like the Microsoft Xbox Kinect, Asus Xtion, and a wide variety of laser, infrared, and stereo vision sensors. By taking care of driver support and making depth data accessible to other applications, OpenNI significantly eases development requirements for 3D sensors. For additional functionality, we integrated Point Cloud Library (PCL). PCL was developed by researchers at Willow Garage with the goal of providing the common building blocks of 3D perception that are needed for more complex applications. The library contains algorithms that support filtering, feature estimation, surface reconstruction, registration, model fitting, and segmentation (Rusu, 2011).

Using these two libraries, we developed two additional OpenRAVE plugins to parse data from the sensor. One plugin imported the point cloud data into OpenRAVE. The second plugin handled cylinder detection. These plugins are detailed in the following subsections.

iii. Cylinder Recognition

Next, we implemented simple object detection to enable grasping of objects in the environment. We decided to detect cylinders because of the straightforwardness of grasping cylindrical objects with PackBot's gripper. To do this, we wrote an OpenRAVE plugin. PCL allows the user to grab a point cloud from OpenNI. A PCL clustering algorithm is used to return several different point clouds. Clustering entails grouping neighboring points from the full point cloud (Rusu, 2011). Several point clouds representing different regions in space allow for multiple cylinders to be recognized from the same scene. A PCL segmentation algorithm is then applied to each point cloud returned from the previous step. The segmentation algorithm segments a cylinder from the rest of the cloud. The function returns the x,y,z axis direction, and radius, as well as a set of points belonging to the cylinder. The cylinder point cloud is rotated based on the transformation matrix found from the cylinder axis direction. From this, the cylinder height can be found by finding the extremal points of the cloud. The height is found by finding the mid z-point between the minimum and maximum z-points. The xyz position of the cylinder is computed by finding the midpoint and transforming the point back using the cylinder axis direction. The OpenRAVE plugin imports the x,y,z position, x,y,z rotation, radius and height of all cylinders in the environment.

In order to convert the cylinder information into an actual cylinder, the euler angles of the cylinder axis direction are first found. A transformation matrix is then applied which relates the pose of the scene in PCL to the pose of the scene in OpenRAVE. Another transformation is

applied which moves the cylinder depending on the position of the 3D sensor in the OpenRAVE environment. This information is then used to add a kinematic body to the OpenRAVE environment.

iv. Obstacle Detection

A second plugin was created in order to implement obstacle detection. This plugin imports unprocessed point-cloud data from the Xtion using the openNI grabber function in PCL. A coordinate transformation on each point is then performed. The transform is obtained from current position and orientation of the sensor as determined by OpenRAVE. A box occupancy grid is created using a multi-dimensional array. If a box is occupied a check is run to see if any of the cylinders collide with vectors along the box's vertices. If cylinders collide with these bodies, the box is not added to the list of occupied boxes. Boxes are then added to the OpenRAVE environment as filled voxels. These objects are considered in motion-planning by OpenRAVE's collision-avoidance algorithm.

Grasp cylinders recognized by the 3D sensor

Our approach for grasping cylinders was inspired by *Automatic Grasp Planning Using Shape Primitives* which was introduced in the background section (Miller et al., 2003). Our algorithm generates grasps that approach the target cylinder at an orientation perpendicular to the vertical axis of the cylinder. To do this, the grasping script used the IK solver to generate a list of poses that approach the cylinder.For each solution, the script then verified that the target was within a graspable area of the gripper. This was done by checking for collisions between the target and two small vectors located in the center of the gripper as shown below on the left side of **[**Figure 5**].** If both of these vectors were in collision with the target, then the solution was kept. For each solution left, the dot product of the unit z-vector coming out of the gripper and the unit

axis vector of the cylinder was taken. These can be shown on the right side of [Figure 5] where the grey line represents the axis of the cylinder and the blue represents the z-axis of the gripper. The dot product was taken because a grasp with an associated dot product of one meant that the gripper would be perpendicular to the cylinder. The script then ordered all valid solutions thus far by their associated dot product. From here, the script then used OpenRAVE's trajectory planner to verify that a path to the desired solution was viable based on the obstacles in the environment. As soon as a valid trajectory was found it was stored and the target cylinder was considered graspable.

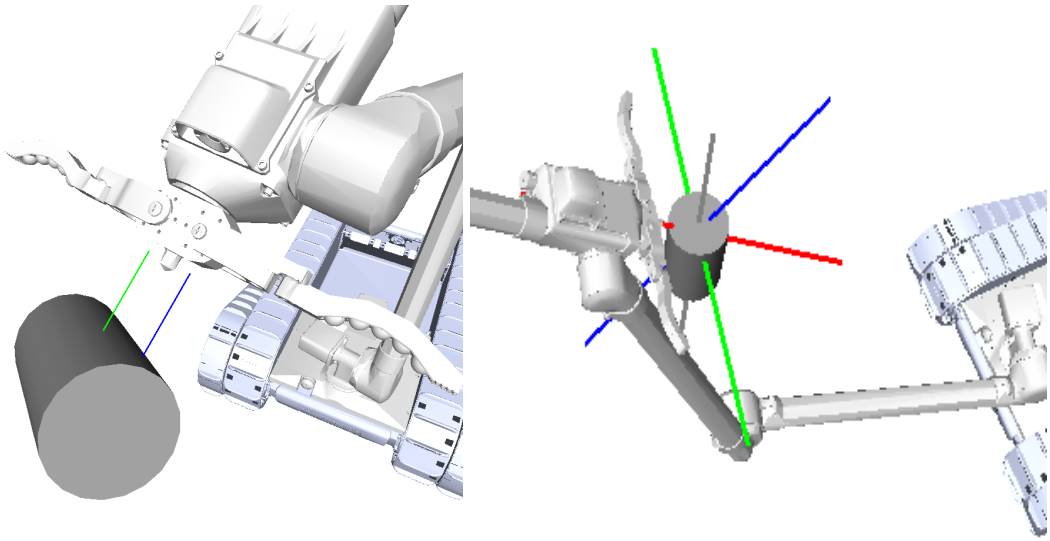


Figure 5 - A visualization of our grasp optimization algorithm

Enable control of our system in a user interface

Once grasping functionality had been added to the PackBotIKSolver plugin, displaying whether objects could be grasped proved to be straightforward. OpenRAVE allows developers to color objects in the environment. For each cylinder in the environment that wasn't colliding with

the PackBot itself, our OpenRAVE script searched for a valid grasp. If no grasp was found, the object was colored red. If there was at least one possible grasp, the object was colored green.

OpenRAVE allows for click callbacks to be added to its user interface. Our project used this feature to allow users to click on cylinders that were loaded into the environment. If the cylinder the user clicked on had a valid grasp, the user was shown the expected trajectory to execute the desired grasp in the simulator. To execute the grasp on the robot, users clicked an "Execute" button in our options UI which is shown below in [Figure 6].

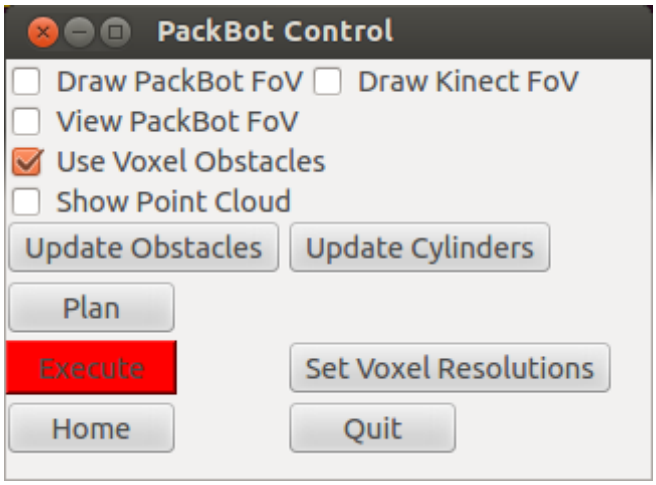


Figure 6 - Configuration interface for our system

In order to save time while loading obstacle data into the OpenRAVE environment, we decided to use a fairly large default voxel size. This not only sped up the obstacle importation process, but also increased the speed at which trajectories were generated. We realized that this would not be ideal in situations that require higher resolution obstacle detection, so we added the ability to specify the resolution of the x, y, and z directions in the voxel grid.

Since the user will be operating the robot remotely, we thought a useful feature to add to the user interface would be the ability to see the areas of the environment that the PackBot's camera and the Xtion could see. In other words, their field of views. To do this we added

functionality for drawing the corresponding fields of view in the OpenRAVE interface. We then added checkboxes to the user interface that allowed the user to turn these features on and off. With the display of the robot's and Xtion's field of view, we hoped that operators would more easily understand the video coming from the camera.

Results and Discussion

Simulation

Because we were able to import Solidworks files into OpenRAVE as VRML files, the model was very accurate. After some fine-tuning of the joint limits, the model precisely captured the capabilities of PackBot's arm. A screenshot of the model within OpenRAVE is included below in **[**Figure 7**]**. The model was less than ideal in its representation of the Asus Xtion. The only way to determine the location of the sensor was by measuring the location of the robot's camera head. But the joint affixing PackBot's camera head has to the arm is not perfect. Some of the errors in cylinder recognition and obstacle detection can be attributed to joint slop and the resulting error in the model's representation of the sensor's location. This could be improved upon by adding calibration functionality for the sensor. We suggest adding a calibration target to the chassis of the robot in a location that is easily seen by the Xtion.

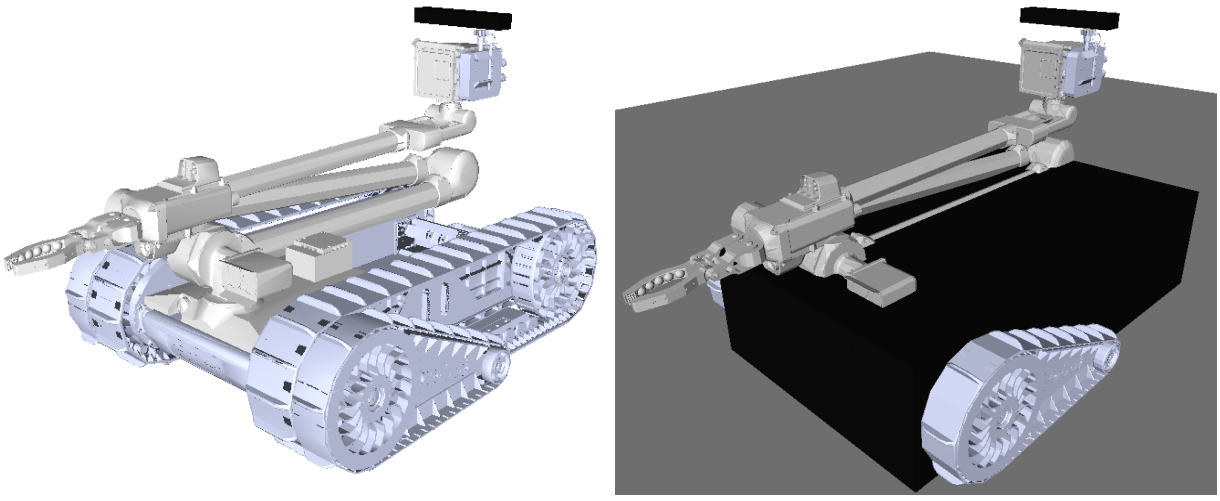


Figure 7 - Visual and collision models of PackBot within OpenRAVE

One change that became necessary during the project was a simplification of the collision model within OpenRAVE. The collision model, shown above on the right side of **[**Figure 7**]** was changed to include simply a rectangle for the base. The visual and collision model for the arm remained the complex model. This change allowed for faster self collision checking for the base of the PackBot during trajectory planning, while still keeping accurate self collision checking for the arm.

IK Solver

With tuning, the iterative IK solver turned out to be fairly good at finding solutions within a reasonable amount of time. The results of our testing can be seen in the table at the top of the next page. The database size is the number of poses and arm configuration stored in the database. Lambda indicates the weighting given to position versus orientation (a higher lambda correlates to more emphasis on position). Success was defined by whether the gripper could reach within .05 meters of the desired position. As the table indicates, we significantly improved upon our system by starting the Jacobian-based solver from multiple starting configurations.

Database Size	Lambda	Jacobian	Success Rate
2,000	.95	Multi	3%
2,000,000	.90	Single	16%
2,000,000	.95	Single	27%
2,000,000	.95	Multi	34%
2,000,000	.99	Single	50%
3,000,000	.95	Multi with optimization	84%
3,000,000	.99	Multi with optimization	97%

While we tested trajectory planning, we noticed that OpenRAVE would often generate unreasonable trajectories. The paths were often very indirect, nonsensically driving the arm or gripper in circles before approaching the target. We used OpenRAVE's built-in trajectory generation functionality which is reliable for reaching the desired pose but could use some improvement in the paths it takes. We started to look into trajectory smoothing but unfortunately ran out of time and chose to prioritize other aspects of the project.

Trajectory Execution

By the end of the fourth week of our project, we were able to demonstrate trajectory-following on the robot. An OpenRAVE plugin sent a series of joint positions to the robot which were executed at 10Hz (the default refresh rate of the PackBot OS's robot controller). At this point, we were still using the built in positional controller on the PackBot -- we drove the arm by

sending desired joint angles to iRobot's robot controller and it determined and set joint velocities accordingly. This worked well enough to demonstrate basic trajectory following. Unfortunately these were not accurate enough during the length of execution to use as the final product.

The inaccuracy of our system at this stage is shown below in **[**Figure 8**].** The discrepancy between desired and actual positions was due to many factors. These included a positional update rate of only 10Hz from PackBot and no joint position checking during trajectory execution to ensure the arm made it to each pose in the trajectory. The low update rate was insufficient for accurate trajectory following because 10Hz allows for a lot of time between joint velocity updates. This means that the arm's joints can more easily overshoot their desired angles, adding error during execution. The lack of positional checking during trajectory execution meant there was no guarantee that the PackBot's arm would reasonably follow the expected trajectory.

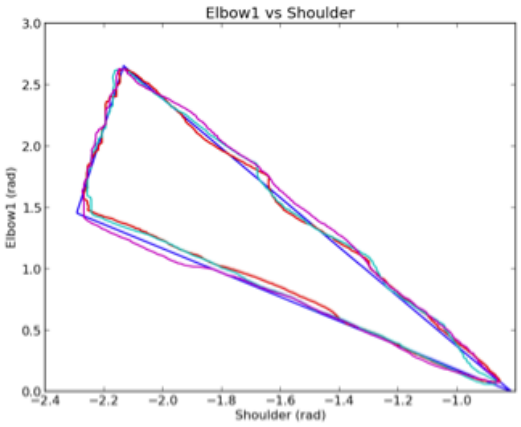


Figure 8 - Trajectory execution testing data

In order to correct for the low update rate, iRobot sent us a software update for PackBot that allowed us to change the update rate. We chose to set the rate to the maximum update rate that was recommended by iRobot, 30Hz. This would allow the trajectory following code to execute more often and accurately correct for any errors in the execution of a trajectory.

Unfortunately, this update made the onboard positional controller no longer functional, meaning we needed to implement our own controller to define joint velocities instead.

Since PackBot was only able to accept velocity commands after the update, we needed to implement a velocity controller that would run on the robot. We decided to start out with just a proportional controller that took the difference between the desired and the actual angles in order to set the velocities of each joint. These joint velocities were then sent to iRobot's PackBot controller via localhost. The proportional controller proved to work well enough for the goals of our project, so we did not need to implement a more complicated PID controller. A graph of the accuracy of our system at this point is included below, in **[**Figure 9**].**

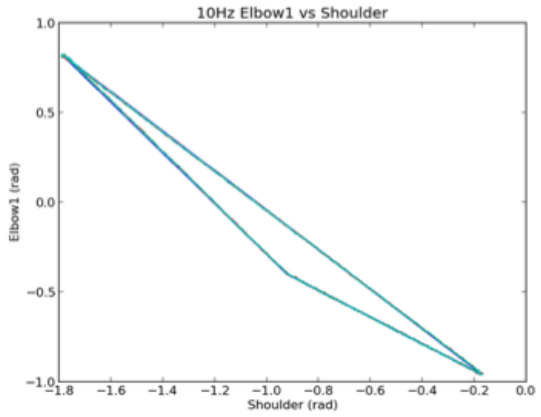


Figure 9 - Trajectory-following test data after implementing proportional control

In order to guarantee that the PackBot's arm was following our desired trajectories, we needed to make sure that it reached each of the poses in the trajectory. To ensure the PackBot is able to reach a particular pose during a trajectory, it repeatedly makes calls to its velocity controller for the particular pose until the arm's joints are within specified tolerances. These tolerances can be found in the table at the top of the next page, along with their associated error

rate. The maximum error is calculated as the error as a portion of that joint's full range. The total error is the maximum, compounded error.

Joint	Tolerance (rad)	Maximum Error (%)	Total Error (%)
Turret	0.05	0.796	0.796
Shoulder	0.05	0.796	1.60
Elbow1	0.075	1.19	2.81
Elbow2	0.075	1.19	4.04
Wrist	0.05	0.796	4.87

Sensing

The first step in adding a sensor to the robot was designing a suitable mount. This mount can be seen below on the left side of [Figure 10**]**. It was intended to fit in the expansion bay on the camera head of PackBot. Once it was mounted, we realized that the sensor was very likely to collide with the arm. Although OpenRAVE could account for it in motion-planning, its position had the potential to constrain arm motions. So we chose to mount the sensor on top of the camera instead, where it had no chance of colliding with the arm. This final iteration is included below, on the right hand side of **[**Figure 10**]**. A SolidWorks drawing for the mount is included in **[**Appendix B – Sensor Mount SolidWorks Drawing**]**.

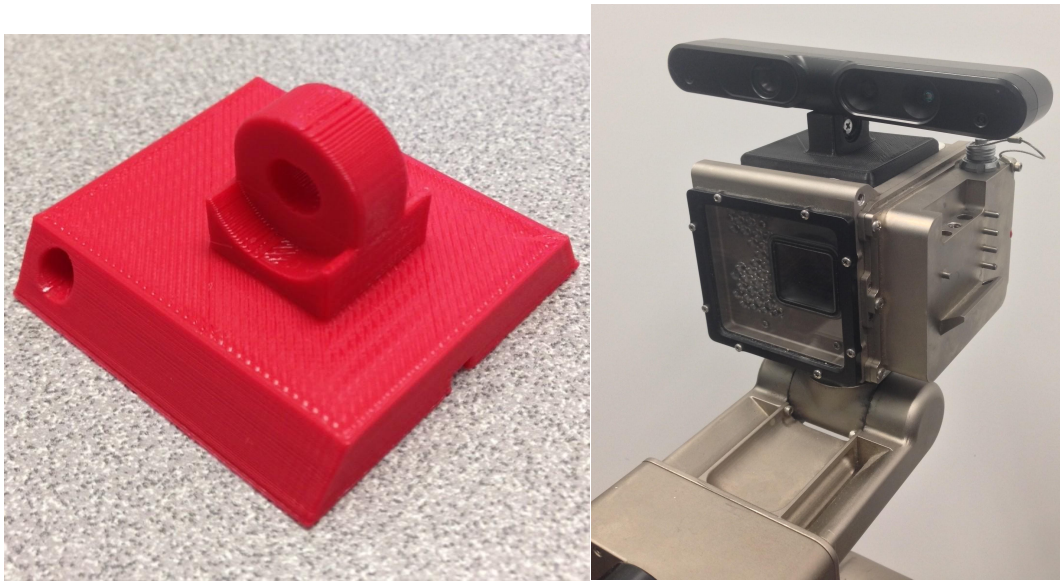


Figure 10 - Initial and final sensor mounts

By week nine of the project, we had already drafted an OpenRAVE plug-in that opened a socket between the Xtion depth sensor and the OpenRAVE interface. A representation of our progress at this stage can be seen below in **[**Figure 11**].** The figure shows the work we still needed to do, as PackBot's model was not the right size in relation to data from the sensor. Initially, we tried compiling the OpenNI library as an OpenRAVE plug-in, but this proved difficult. Instead, we opened a socket between openNI and OpenRAVE that allowed openNI to send its point cloud data along. This was later scrapped when we started using PCL. When PCL was used we simply used the OpenNI grabber functionality included in PCL. Though our sensor had a documented range of .8-3.5 meters, it sometimes returned points outside of this range. Since these were not accurate readings, these points were discarded.

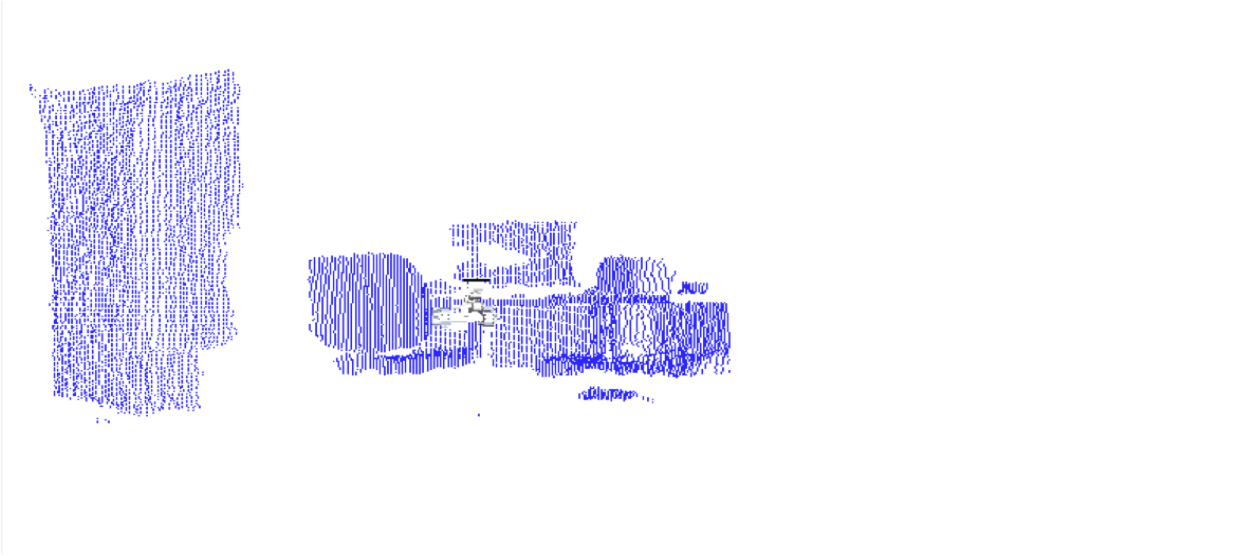


Figure 11 - Point cloud data within OpenRAVE before system calibration

The pictures in **[**Figure 12**]** below represent how a scene is interpreted by our sensor and imported into OpenRAVE. The first image is a camera representation of a scene. The next picture displays a point cloud representation within OpenRAVE of the same scene. The third picture is the same scene after it has been converted into voxels. This is how obstacles in the scene are represented for our trajectory planner.

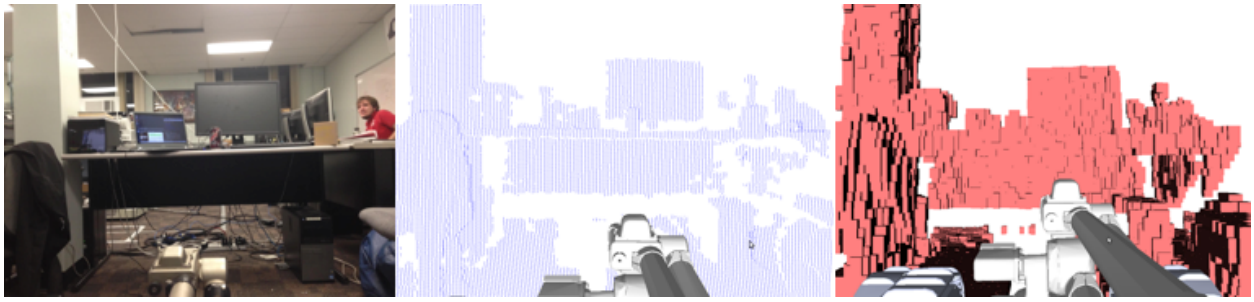


Figure 12 - A complex scene represented by a picture, unprocessed point cloud data imported into OpenRAVE, and voxels within OpenRAVE

One issue that was encountered was missing data. This can happen for several reasons. First of all, we noticed that reflective surfaces can sometimes result in erroneous sensor readings. Because the Xtion relies upon reflected infrared light, this issue is inherent to the sensor. Missing data can also be attributed to the position of obstacles in relation to one another. An obstacle might obstruct the sensor's line-of-sight to an item in the environment that is necessary for grasp planning. This obstacle will be left out of the simulation therefore causing errors. We recommend improving this by enabling multiple sensor snapshots to be taken of the same scene from multiple angles.

Grasping

Planning a trajectory to grasp a target involved incorporating all previous aspects of our project. This meant that it was fairly difficult to tune and perfect, as there were many pieces that needed to work together. Although we were able to successfully grasp objects during some attempts, there were many details that prevented us from achieving 100% accuracy.

One of the most glaring issues was that the robot often missed the can. This is due to sensor miscalibration and joint slop which resulted in inaccuracy in OpenRAVE's location of the sensor. As mentioned in a previous section, we recommend improving this by adding a calibration target for the sensor to the robot's chassis or arm.

Grasping also highlighted how computationally expensive much of our system was. Importing cylinder took several seconds and calculating grasps took five to ten seconds per cylinder. Optimizing the system for computational efficiency was beyond the scope of our project but will be a necessary task if our work is to be incorporated into PackBot's operating system.

User Interface

The results of our interface design can be seen below. Coloring the cylinders was straightforward but took a few seconds because computing the graspability of each cylinder was processor-intensive. A view of the OpenRAVE simulator including colored cylinders is below on the left side of [Figure 13]. This could have been mitigated by calculating the graspability of objects as they were imported, but this would simply slow down computation at another step of our process. Drawing fields of view was also straightforward, and our results can be seen below on the right side of [Figure 13]. Initially, we simply used green shading, but this was hard to understand. So we decided to trace the vertices of the shaded region with blue lines.

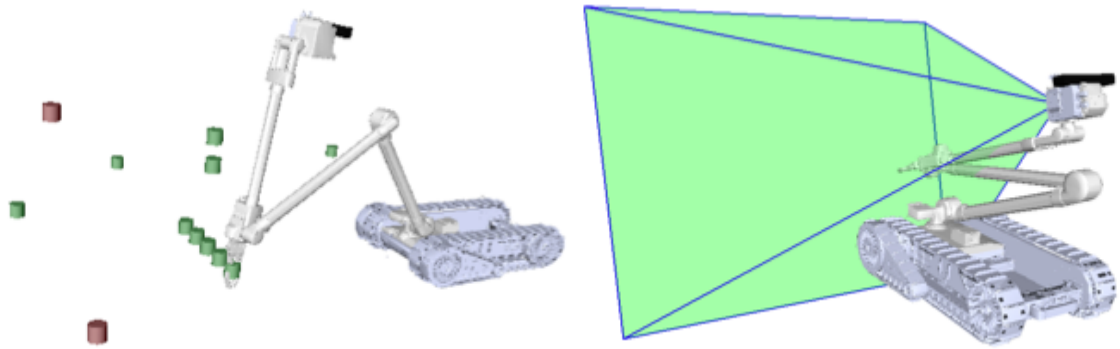


Figure 13 – Left: A representation of graspability in the OpenRAVE viewer. Green cylinders are graspable and red are not. Right: A visualization of the camera's field of view

The options in the PackBot control window which is shown below in [Figure 14] were selected to improve usability. The field of views are explained above. “Use Voxel Obstacles” enables or disables obstacle avoidance with the voxels when generating trajectories. “Show Point Cloud” displays the raw data from the Kinect. “Update” allows users to redraw the UI according to the numbers input into the text boxes at the right. “Execute” sends a planned trajectory to the robot and is colored green when a trajectory is available. “Quit” cleanly exits OpenRAVE. “Set

Voxel Resolutions" opens three text boxes to configure the resolutions of voxels for each dimension.

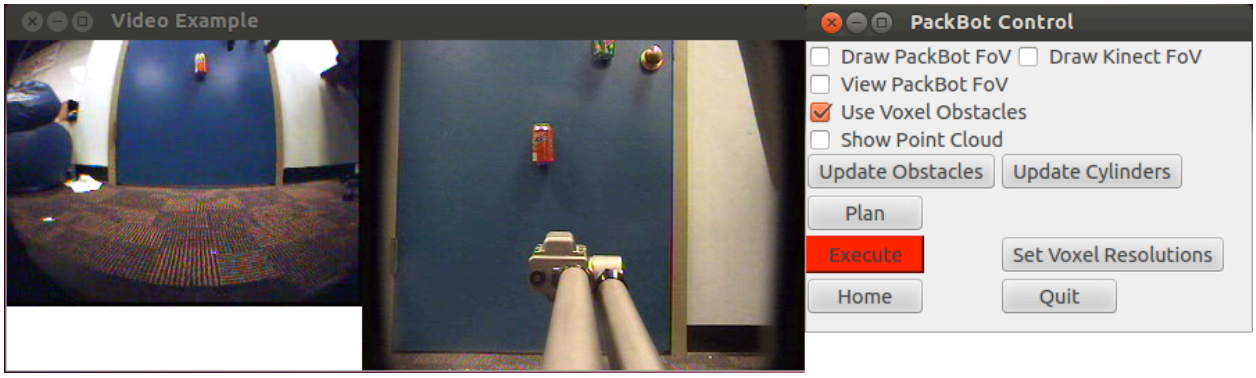


Figure 14 - Camera streams from the robot alongside our configuration window

Conclusions

With our project, we have demonstrated how OpenRAVE can control a robot, facilitate the development of autonomous behavior, and incorporate sensor data into a useful system. Our work is not intended to be incorporated into the PackBots iRobot sells to customers. If iRobot wishes to incorporate our efforts into PackBot's operating system, significant work is still required. But our project should serve as a demonstration of how OpenRAVE can improve usability and reduce the effort required to control PackBot.

Limitations and Recommendations for Future Work

If we had more time, there are three areas where we would like to improve our system: object recognition, user interface, and trajectory generation.

Object Recognition

We would suggest working to improve cylinder detection. Our current cylinder detection algorithm has a few flaws. False-positive cylinders are recognized, the locations and orientations of cylinders are occasionally incorrect and cylinders are sometimes not recognized. The segmentation of the cylinder might be improved on if we could have an additional sensor mounted in a different location on the robot. This could also be done by capturing multiple sets of point cloud data from the same sensor. The direction coordinates could be better defined by writing our own algorithm which parses the point clouds obtained from cylinder segmentation.

We would also suggest improving object recognition to support more shapes. Our recognition of cylinders was an effective demo, but has limited applications in the field. It would be helpful to recognize more complex shapes than simple geometric solids like cylinders. Performing object detection through surface matching or spin images, instead of our feature-extraction and segmentation-based approach would be one way to improve our system (Johnson & Hebert, 1998; Johnson, 1999). If the sensor could import more complex shapes into OpenRAVE's environment, this feature could be very useful in the field. Thanks to OpenRAVE's robust code base and grasp planners, grasping complex shapes should be relatively straightforward.

User Interface

If our project is to be incorporated into commercially-available PackBots, our user interface will need improvements. Clicking to grasp objects was useful as a demonstration of our system, but this functionality would require a different interface for field robots. Using a mouse for input is not a suitable interface for field robots. So our code includes a callback for grasping specific items which could be called by any piece of code. Touch input or voice commands could

be useful ways to call this functionality. If users could command the robot by saying something like “pick up the object in front of you and move it 500 feet northwest,” our work could become a very useful aspect of PackBot.

Trajectory Generation

OpenRAVE is always able to generate a path that reaches the desired pose. But many of the trajectories generated by OpenRAVE took unreasonable paths to reach this pose. We suggest improving OpenRAVE’s trajectory planner or implementing a custom planner. We began looking into this, but decided to prioritize other aspects of our system.

PackBot Upgrades

We also identified three key upgrades to the PackBot that would enhance the functionality of our system.

Manipulator

First, the PackBot needs a manipulator that is more suited for autonomous grasping. The current manipulator does not have force feedback, so teleoperated grasping requires visual feedback from the gripper camera. During autonomous grasping, the lack of feedback when the gripper makes contact with its target could lead to failed or damaging grasps as the PackBot’s manipulator can squeeze with quite a bit of force. With a force-sensing gripper, OpenRAVE’s OpenGRASP plug-in would be able to account for feedback from the gripper when executing a grasp. Torque feedback from the gripper could be a viable alternative to true tactile sensing (Jang et al., 2012).

Embedded Computer

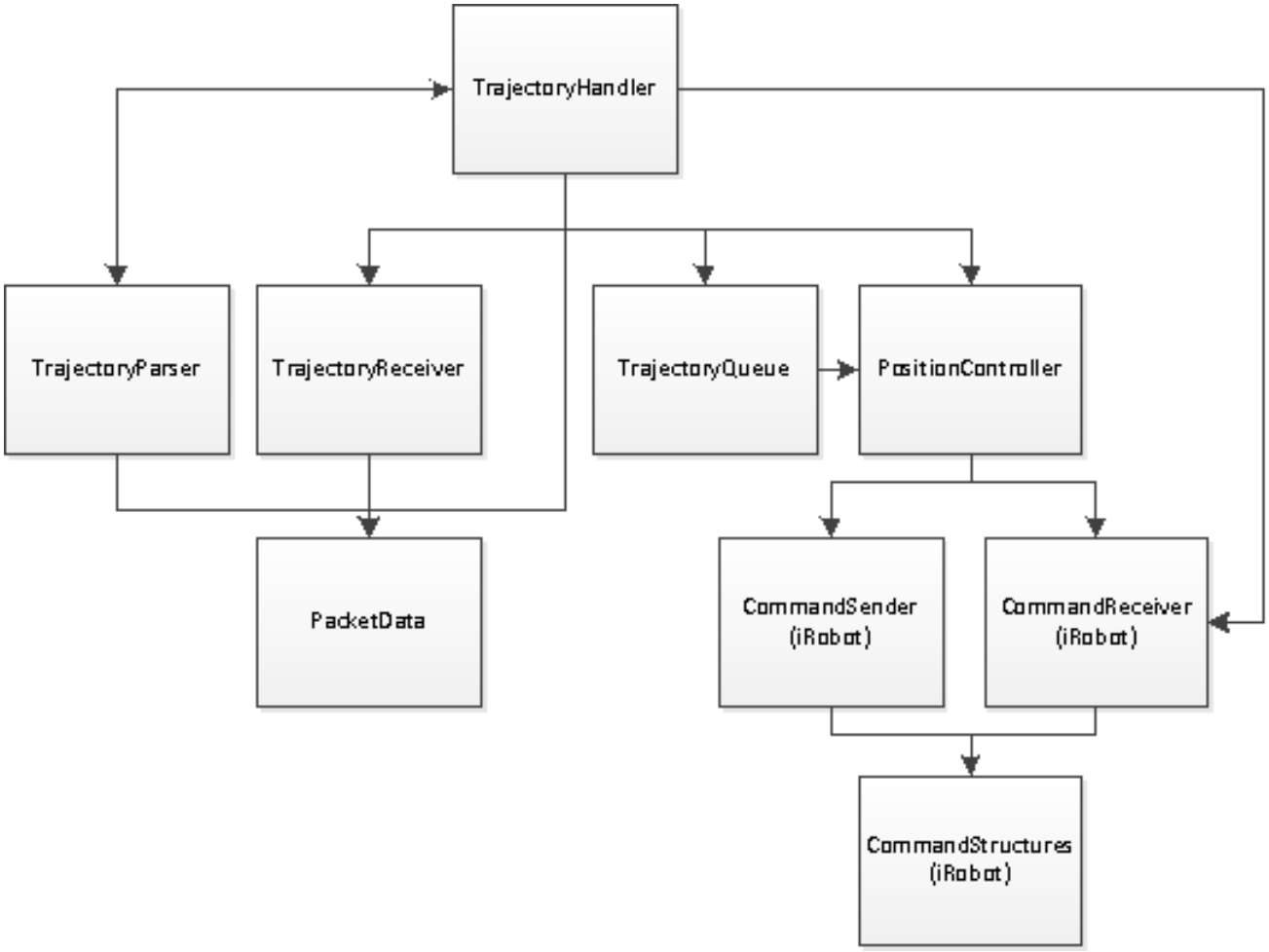
To implement autonomy in field situations, the PackBot's internal computer will need an upgrade. Because robots in the field might not always be within communication range of a more capable computer, OpenRAVE should run on the robot itself. Our PackBot's Pentium 3 processor and limited hard disk space seriously hindered our efforts to get OpenRAVE running on the robot. Adding more sensors and complex software puts strain on its computer. Processor, hard disk, and memory upgrades will be necessary if iRobot wants to integrate the sensors and software required for onboard motion-planning,

Sensing

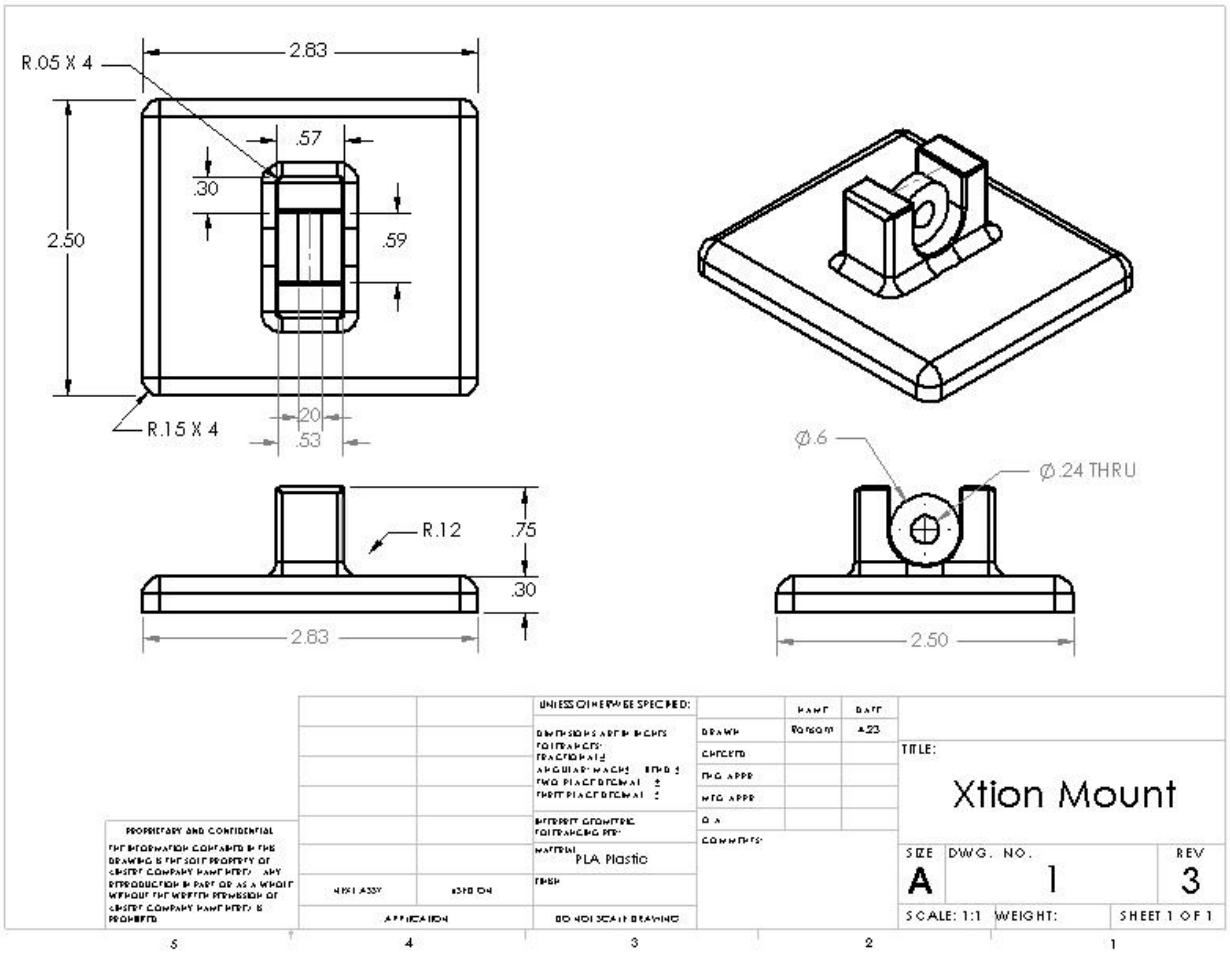
Though our 3D-printed mount firmly affixed our depth sensor to the robot in a lab setting, field PackBots would require a more rugged configuration. We recommend mounting the 3D camera alongside the PackBot's default camera, as we did, because of its maneuverability and the high vantage points it enables. 3D data from PackBot Wayfarer's sensor head could be imported into OpenRAVE to eliminate the need for more hardware development (Yamauchi, 2005).

Appendices

Appendix A - System Diagram



Appendix B – Sensor Mount SolidWorks Drawing



Works Cited

Berenson, Dmitry, Rosen Diankov, Koichi Nishiwaki, Satoshi Kagami, James Kuffner, "Grasp planning in complex scenes." *Humanoid Robots, 2007 7th IEEE-RAS International Conference on*. IEEE, 2007.

Burke, Jennifer L., Robin R. Murphy, Michael D. Coolvert, Dawn L. Riddle, "Moonlight in Miami: Field study of human-robot interaction in the context of an urban search and rescue disaster response training exercise." *Human–Computer Interaction* 19.1-2 (2004): 85-116.

Casper, J., R.R. Murphy, "Human-robot interactions during the robot-assisted urban search and rescue response at the World Trade Center," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol.33, no.3, pp.367,385, June 2003 doi: 10.1109/TSMCB.2003.811794

Deligeorges, Socrates, David Anderson, Casandra A. Browning, Howard Cohen, David Freeman, Tyler Gore, Christian Karl, Sarah Kelsall, David Mountain, Marianne Nourzad, Yirong Pu, Matt Sandifer, Shuwan Xue, Leah Ziph-Schatzberg, Allyn Hubbard, "The development of a biomimetic acoustic direction finding system for use on multiple platforms." *SPIE Defense and Security Symposium*. International Society for Optics and Photonics, 2008.

Diankov, Rosen, Siddhartha Srinivasa, David I. Ferguson, James Kuffner, "Manipulation planning with caging grasps." *Humanoid Robots, 2008. Humanoids 2008. 8th IEEE-RAS International Conference on*. IEEE, 2008.

Diankov, Rosen, and James Kuffner. "OpenRAVE: A planning architecture for autonomous robotics." *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34* (2008): 79.

Ferguson, George, and James Allen. "Mixed-initiative systems for collaborative problem solving." AI magazine 28.2 (2007): 23.

Ferguson, George, James F. Allen, and Bradford W. Miller. "TRAINS-95: Towards a Mixed-Initiative Planning Assistant." AIPS. 1996.

Ferrari, Carlo, and John Canny. "Planning optimal grasps." *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*. IEEE, 1992.

Hayati, S., and S. T. Venkataraman. "Design and implementation of a robot control system with traded and shared control capability." *Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on*. IEEE, 1989.

Inagaki, Toshiyuki. "Adaptive automation: Sharing and trading of control."*Handbook of cognitive task design* 8 (2003): 147-169.

Jang, Junwon, Kyungrock Kim, Kicheol Park, Hoseong Kwak, Kyungsik Roh, "A method to detect object grasping without tactile sensing on a humanoid robot," *Control, Automation and Systems (ICCAS), 2012 12th International Conference on* , vol., no., pp.1419,1422, 17-21 Oct. 2012

Kemp, Charles C., Aaron Edsinger, and Eduardo Torres-Jara. "Challenges for robot manipulation in human environments." *IEEE Robotics and Automation Magazine* 14.1 (2007): 20.

Kortenkamp, David, Peter R. Bonasso, Dan Ryan, and Debbie Schrenkenghost, "Traded control with autonomous robots as mixed initiative interaction." *AAAI Symposium on Mixed Initiative Interaction*. 1997.

Markenscoff, Xanthippi, and Christos H. Yapadimitriou. "Optimum grip of a polygon." *Dept. Comput. Sci., Stanford Univ., CA, Rep. STAN-CS-87-1153*(1987).

Rusu, R.B.; Cousins, S., "3D is here: Point Cloud Library (PCL)," *Robotics and Automation (ICRA), 2011 IEEE International Conference on* , vol., no., pp.1,4, 9-13 May 2011 doi: 10.1109/ICRA.2011.5980567

Org, OpenNI. "Introducing OpenNI." *OpenNI Organization.[Online] http://www.openni.org/images/stories/pdf/OpenNI_UserGuide.pdf*.

Qian, Shan-hua, Shi-rong Ge, Yong-sheng Wang, Yong Wang, Chang-qing Liu, "Research status of the disaster rescue robot and its applications to the mine rescue [J]." *Robot* 28.3 (2006): 350-354.

Sciavicco, Lorenzo, and Bruno Siciliano. *Modelling and control of robot manipulators*. Springer, 2000.

Scott, Andrew J., Jose R. Mabesa, Chantelle Hughes, David J. Gorsich, Gregory W. Auner, "Equipping small robotic platforms with highly sensitive more accurate nuclear, biological, and chemical (NBC) detection systems."*AeroSense 2003*. International Society for Optics and Photonics, 2003.

Siciliano, Bruno, and Oussama Khatib, eds. *Springer handbook of robotics*. Springer, 2008.

Spong, Mark W., and Mathukumalli Vidyasagar. *Robot dynamics and control*. John Wiley & Sons, 2008.

Wampler, Charles W. "Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods." Systems, Man and Cybernetics, IEEE Transactions on 16.1 (1986): 93-101.

Yamauchi, Brian M. "PackBot: a versatile platform for military robotics." *Defense and Security*. International Society for Optics and Photonics, 2004.

Yamauchi, Brian, Mark Moseley, and Jonathan Brookshire. "LABRADOR: a learning autonomous behavior-based robot for adaptive detection and object retrieval." *IS&T/SPIE Electronic Imaging*. International Society for Optics and Photonics, 2013.

Yamauchi, Brian, and Pavlo Rudakevych. "Griffon: a man-portable hybrid UGV/UAV." *Industrial Robot: An International Journal* 31.5 (2004): 443-450.

Yamauchi, Brian, and Kent Massey. *Stingray: High-speed teleoperation of UGVs in urban terrain using driver-assist behaviors and immersive telepresence*. IROBOT CORP BEDFORD MA, 2008.

Yamauchi, Brian. "Autonomous urban reconnaissance using man-portable UGVs." *Defense and Security Symposium*. International Society for Optics and Photonics, 2006.

Yamauchi, Brian. "Fast online learning of control regime transitions for adaptive robotic mobility." *SPIE Defense, Security, and Sensing*. International Society for Optics and Photonics, 2012.

Yamauchi, Brian. "The Wayfarer modular navigation payload for intelligent robot infrastructure." *Defense and Security*. International Society for Optics and Photonics, 2005.