Affordance-Based Control of a Variable-Autonomy Telerobot

by Michael Fleder

Submitted to the Department of Electrical Engineering and Computer Science in Partial Fulfillment of the Requirements for the Degree of Master of Engineering in Electrical Engineering and Computer Science at the Massachusetts Institute of Technology
September 2012

Author: _____

Department of Electrical Engineering and Computer Science
September 4, 2012

Certified by: _____

Professor Seth Teller
Professor of Computer Science and Engineering
Thesis Supervisor

September 4, 2012

Accepted by: _____

Professor Dennis M. Freeman
Chairman
Masters of Engineering Thesis Committee

Affordance-Based Control of a Variable-Autonomy Telerobot

by Michael Fleder

Submitted to the Department of Electrical Engineering and Computer Science on September 4, 2012 in partial fulfillment of the requirements for the degree of Master of Engineering in Electrical Engineering and Computer Science

Abstract

Most robot platforms operate in one of two modes: full autonomy, usually in the lab; or low-level teleoperation, usually in the field. Full autonomy is currently realizable only in narrow domains of robotics—like mapping an environment. Tedious teleoperation/joystick control is typical in military applications, like complex manipulation and navigation with bomb-disposal robots.

This thesis describes a robot "surrogate" with an intermediate and variable level of autonomy. The robot surrogate accomplishes manipulation tasks by taking guidance and planning suggestions from a human "supervisor." The surrogate does not engage in high-level reasoning, but only in intermediate-level planning and low-level control. The human supervisor supplies the high-level reasoning and some intermediate control—leaving execution details for the surrogate.

The supervisor supplies world knowledge and planning suggestions by "drawing" on a 3D view of the world constructed from sensor data. The surrogate conveys its own model of the world to the supervisor, to enable mental-model sharing between supervisor and surrogate.

The contributions of this thesis include: (1) A novel partitioning of the manipulation task load between supervisor and surrogate, which side-steps problems in autonomous robotics by replacing them with problems in interfaces, perception, planning, control, and human-robot trust; and (2) The algorithms and software designed and built for mental model-sharing and supervisor-assisted manipulation. Using this system, we are able to command the PR2 to manipulate simple objects incorporating either a single revolute or prismatic joint.

Thesis Supervisor: Professor Seth Teller

Title: Professor of Computer Science and Engineering

Acknowledgements

Seth Teller for his help, advice, and support

I was supported by the Department of Defense (DoD) through the National Defense Science & Engineering Graduate Fellowship (NDSEG) program.

Sudeep Pillai and Jeremy Scott for their help in the non-algorithmic, version one segmentation tool.

Ron Wiken (MIT CSAIL) kindly built the “busy box” for the PR2.

Contents

1 Introduction and Vision

Many applications of robotics do not require full autonomy. The Mars rovers, for example, operate under significant human supervision, even though that supervision is complicated by the round-trip signal time to Mars (up to 30 minutes). The most advanced, perhaps, autonomous behavior ever deployed on Mars still involved high-level goal-selection by a human operator and help from the operator in case of failure [1]. The military-deployed iRobot packbot (Figure 1.1) [2], built for explosive ordnance disposal (EOD), provides almost no autonomous capabilities; the packbot requires the human operator to command the robot joint-by-joint, or to select from a limited menu of articulated poses (Figure 1.2).

Figure 1.1 iRobot Packbot

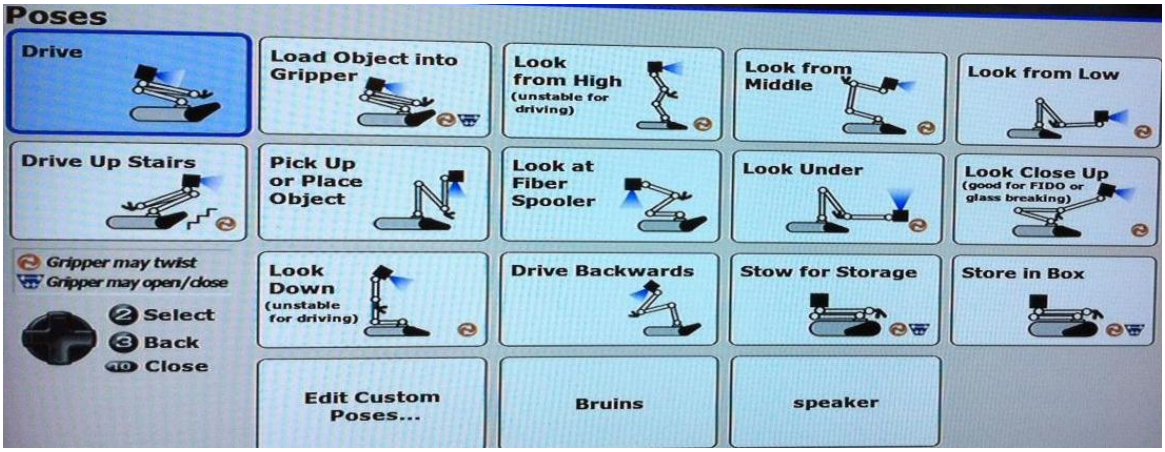


Figure 1.2 iRobot Packbot Pose-Selection Menu

Full autonomy is rarely used in applications where either mission failure or damage to the robot cannot be tolerated. In practice, the alternative to full autonomy is typically low-level teleoperation or extremely limited autonomy. This is unsatisfactory in applications like EOD, where bomb-disposal can be time-critical. We visited the Naval EOD Technology Division (NAVEODTECHDIV) in Charles County, Maryland on April 5th, 2012. At NAVEODTECHDIV, we saw a live demonstration of a highly-skilled EOD operator teleoperating a QinetiQ Talon EOD Robot [3] to open a cardboard box and remove a cylinder (Figure 1.3). This task, which would take an adult human a matter of seconds with his or her hands, took 14 minutes with the Talon under teleoperation. Were the cylinder in this example a bomb with a timer, 14 minutes could be unacceptable. We thus see an opportunity to speed up similar manipulation tasks by an order of magnitude or more.

Figure 1.3 Teleoperated Talon robot extracting a cylinder from a box at NAVEODTECHDIV

This thesis demonstrates a robot mobile manipulator capable of intermediate, variable autonomy. The mobile manipulator (surrogate) takes guidance from a human supervisor and varies its level of autonomy accordingly.

This thesis makes two contributions: (1) A novel partitioning of the manipulation task load between supervisor and surrogate. Using this partitioning, we side-step long-standing problems in AI and replace them with more tractable problems in interfaces, perception, planning, control, and human-robot trust. We focus our algorithmic development on the sharing of mental models between human operator and robot surrogate. (2) This thesis develops HRI and perception algorithms to allow (i) sharing of mental models between operator and surrogate and (ii) efficient execution of manipulation tasks. Using this system, we successfully command the PR2 robot to pull/push levers, lift a knob, and open boxes. These manipulation tasks are currently restricted to simple mechanisms incorporating a revolute or prismatic joint (Figure 1.4).

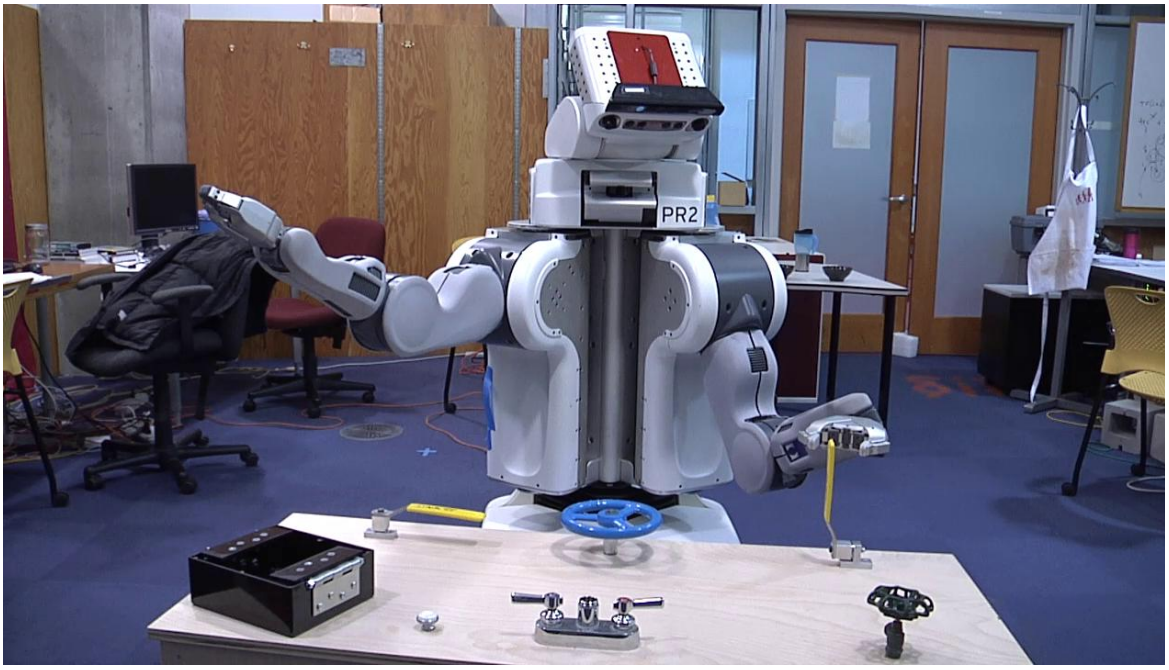


Figure 1.4 PR2 Preparing to grasp a lever

2 Background

2.1 Variable Autonomy and Human Interaction with Automation

Parasuraman et al. (2000) propose guidelines for automation design. They propose that automation can vary continuously from low to high (see Figure 2.1). However, even level 2 in their proposal can be difficult to implement in practice: "the computer offers a complete set of decision/action alternatives." A machine with the ability to suggest real, alternative action plans needs the ability to ground those plans in the real world – as in [4]. Grounding actions in the world is in itself a difficult, open research area and is discussed further in section 2.2. Thus, introducing even small amounts of autonomy into a system can be challenging.

Levels of Automation of Decision and Action Selection

High	10. The computer decides everything, acts autonomously, ignoring the human.
	9. informs the human only if it, the computer, decides to
	8. informs the human only if asked, or
	7. executes automatically, then necessarily informs the human and
	6. executes automatically, then necessarily informs the human, and
	5. executes that suggestion if the human approves, or
	4. suggests one alternative
	3. narrows the selection down to a few, or
	2. The computer offers a complete set of decision/action alternatives, or
Low	1. The computer offers no assistance: human must take all decisions and actions

Figure 2.1 Variations on automation [5]

Parasuraman also suggests the following classes of automatable system functions: (1) Information acquisition; (2) Information analysis; (3) Decision and action selection; and (4) Action implementation. These four classes are comparable to the categories in the human-information processing model in Figure 2.2. Parasuraman et al. discuss several issues with partial automation. For instance: "If the human operator is ever expected under abnormal circumstances to take over control, then … high levels of decision automation may not be

suitable," because the human will not be accustomed to manual control of an almost-always automated process. They describe another downside to automation: humans are "less aware of changes in environmental states when those changes are under the control of another agent" - an effect they call "over-trust" or "complacency."

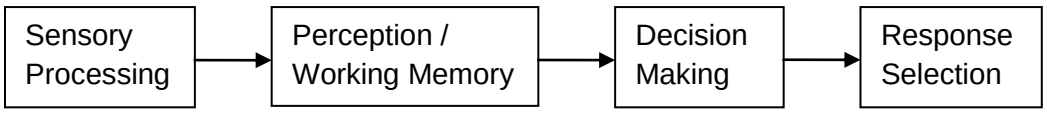


Figure 2.2 Simple Four-Stage Model of Human Information processing [5].

Work in "adjustable autonomy" systems tends to involve structured handoffs from the human user to the robot [6]. In contrast, the method described in this thesis leaves it to the human supervisor to decide when and how the system will operate autonomously; the supervisor is free to pause/cancel/modify autonomous operation at any time.

2.2 Commanding a Robot through Language and Gestures

Commanding a Robot with Natural Language:

There has been recent progress in commanding a robot with natural language [7] [8]. Tellex et al. (2011) demonstrate commanding a robotic forklift through natural language. The key challenge is grounding: how to map phrases like "Pick up the tire pallet off the truck and set it down" to objects and trajectories in the environment [8]. Tellex builds on Jackendoff [9] in categorizing spatial descriptions as one of four types: (i) Event/action sequence - like "move the tire pallet"; (ii) Objects - "the truck"; (iii) Places - "on the truck"; and (iv) Paths - "past the truck."

Tellex's system learns the meanings of verbs like "put" from training data that maps natural language commands to correct robot actions. The system does not incorporate other input channels, such as hand gestures, written gestures, or eye gaze.

Commanding a Robot with Gestures:

Teller et al. (2010) incorporate Tellex's work into a robotic forklift that can accept spoken commands and written gestures. For example, the supervisor holds a tablet that displays robot-point-of-view images. The supervisor can command the forklift to pick up certain objects by circling them (Figure 2.3). The supervisor is effectively "drawing on the world" to convey a desired action to the robot [10].

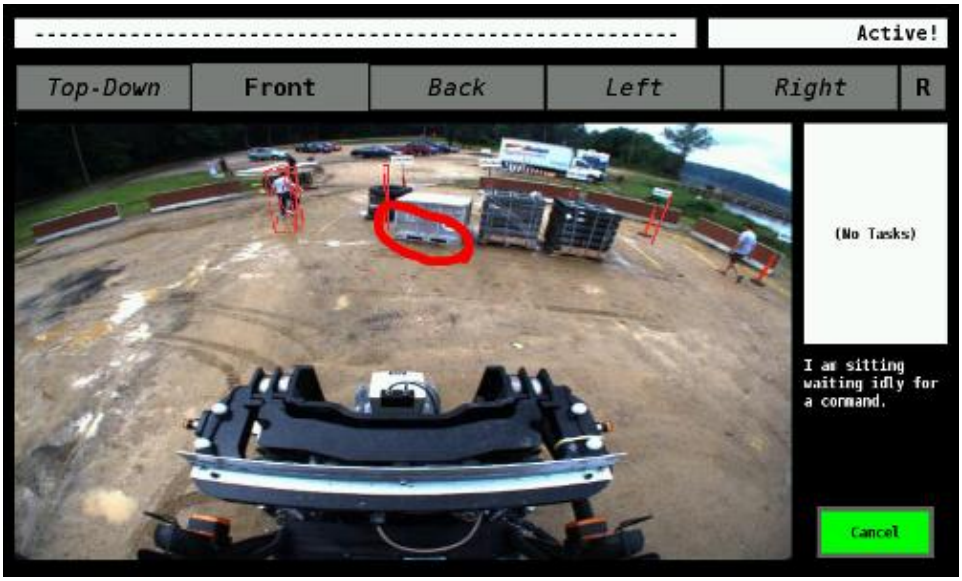


Figure 2.3 Human supervisor commands the robotic forklift to pick up the circled pallet [7].

2.3 Object Detection

Object detection using 2D data has been studied extensively. Torralba provides a nice survey of various object detection techniques in his MIT course 6.870, available online [11]. He describes

and provides references for face detection [12], human detection [13], and machine learning techniques in the context of object detection.

Object recognition using 3D data has also been tackled with several different methods. Besl and Jain [14] apply differential geometry in object recognition: they compute and use surface curvature to classify different pieces of their 3D data into one of eight surface types. RANSAC [15] may be perhaps the most common method for object recognition from 3D data. RANSAC proceeds by iteratively selecting random subsets of the 3D data, and then using that subset to estimate a transform aligning the subset with the 3D model. These iterations are repeated a fixed number of times, and the highest scoring transform is selected. Recent work from Papazov et al. (2012) uses a RANSAC-like sampling strategy but derives the number of trials needed to recognize a model based on a pre-defined success probability [16]. Papazov et al. use an offline, model-preprocessing phase in addition to an online object recognition algorithm, demonstrating recognition of several different models even with significant occlusion.

RANSAC variants have also been applied to object *detection* [17]. Schnabel et al. are able to detect planes, spheres, cylinders, cones, and tori more efficiently than previous RANSAC-based techniques. However, their approach requires the data to be amenable to computing a normal at each point — so their technique degrades if normal estimation is difficult. Schnabel details other prior work in detection of primitive shapes: he discusses both RANSAC and the Hough transform and states that inefficiency and high memory consumption are the major drawbacks to both RANSAC and Hough transforms.

3 Approach

We describe here the physical setup, software structure, and algorithms that comprise our system. We also include a detailed use case example.

3.1 System Overview

We first detail the physical setup of robot, operator, and workspace before describing the software system.

Physical Setup: We use the PR2 robot [18] with a Microsoft Kinect placed on the head. The PR2 is stationed in front of our manipulation workspace (Figure 3.1.1). The manipulation workspace, "the busy box," consists of levers, wheels, a box, and a sink faucet anchored to a wooden platform.

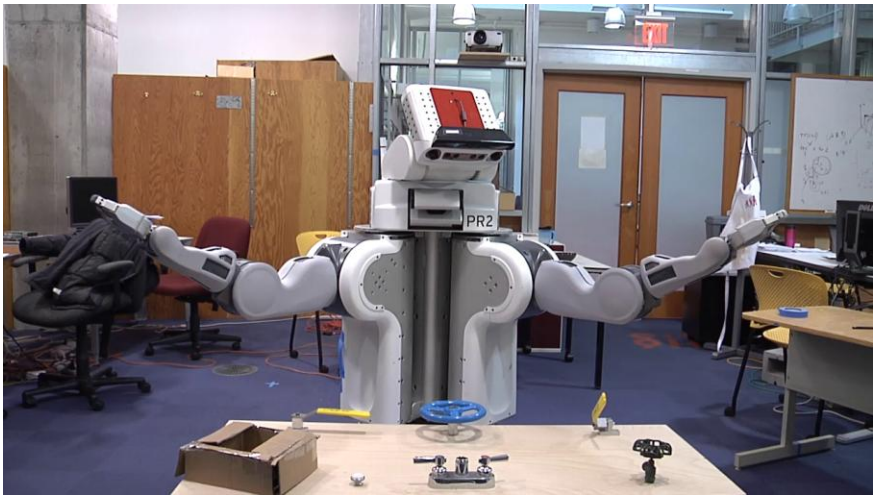


Figure 3.1.1 PR2 Stationed in front of the "Busy Box" workspace

The supervisor is stationed at a desktop running our GUI (Figure 3.1.2). The desktop and PR2 communicate over Ethernet using LCM [19] and ROS [20] inter-process communication. The operator terminal consists of two screens: (i) Our custom GUI (right screen) and (ii) ROS "rviz" visualization tool (left screen). Most of our development work is built into the right screen GUI.

The left screen rviz tool is primarily used for debugging. We discuss the GUI and software structure next.

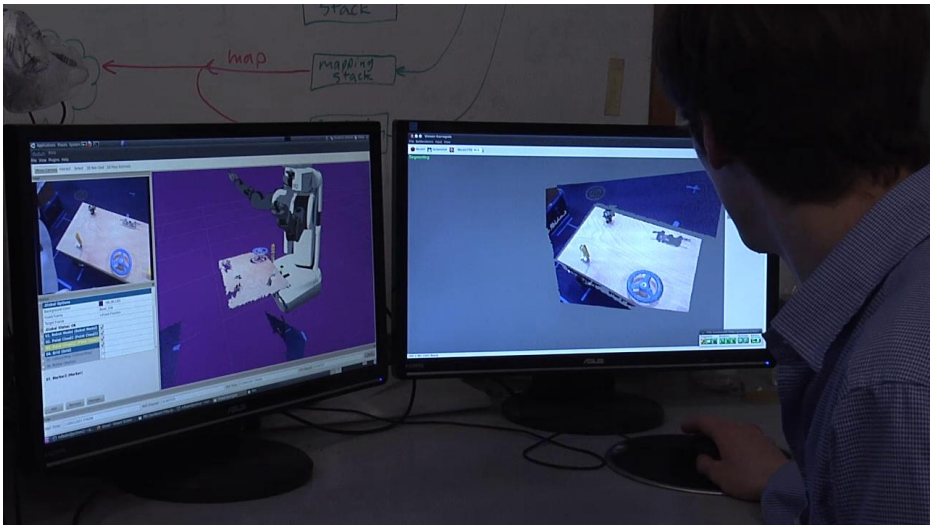


Figure 3.1.2 Operator computer running our GUI (right screen) and the ROS "rviz" visualization tool (left screen).

GUI: The GUI (Figure 3.1.2 at right) we developed is the only point of interaction between the supervisor and the surrogate. It is responsible for: (a) Rendering the 3D scene from live RGBD data; (b) Conveying the surrogate's world model to the supervisor; and (c) Allowing the supervisor to adjust the surrogate's world model and command the robot. The supervisor is able to adjust the surrogate's world model in either of the following ways: (a) Segment objects or refine the surrogate's guess at a segmentation; or (b) Adjust parameters determining revolute and prismatic joints: rotation axes, grasp points, initial direction for applying force. The next sub-section will describe how our software is distributed across machines.

Software Structure: Our software runs in two places: (i) The GUI on the desktop, which includes the interactive-segmentation, model-fitting, and trajectory-adjustment tools (right screen in

Figure 3.1.2); and (ii) The grasp and trajectory planning software, which runs on the PR2. We use both ROS and LCM messaging and have written custom message translators for converting between these two formats when necessary. Most of our software is platform-agnostic: only the planning software relies on the PR2 robot model. This is a key feature, since we intend to develop our system further on different robots for additional EOD tasks and for the DARPA Robotics Challenge.

The remainder of this chapter is organized as follows:

Section 3.2 describes a use case with opening a box.

Section 3.3 details the user-assisted segmentation mechanism.

Section 3.4 describes our model-fitting algorithm.

Section 3.5 describes the mechanisms by which the supervisor can adjust the model fitting.

Section 3.6 details tracking using iterative closest point (ICP).

Section 3.7 describes grasp selection.

Section 3.8 talks about planning/executing a grasp and trajectory.

3.2 System Use Case

Here we describe how the supervisor uses the system. Figures 3.2.1-3.2.9 show the timeline of events leading up to the robot grabbing and pulling open a box flap. As in the rest of the thesis, the words *surrogate* and *robot* are used interchangeably.

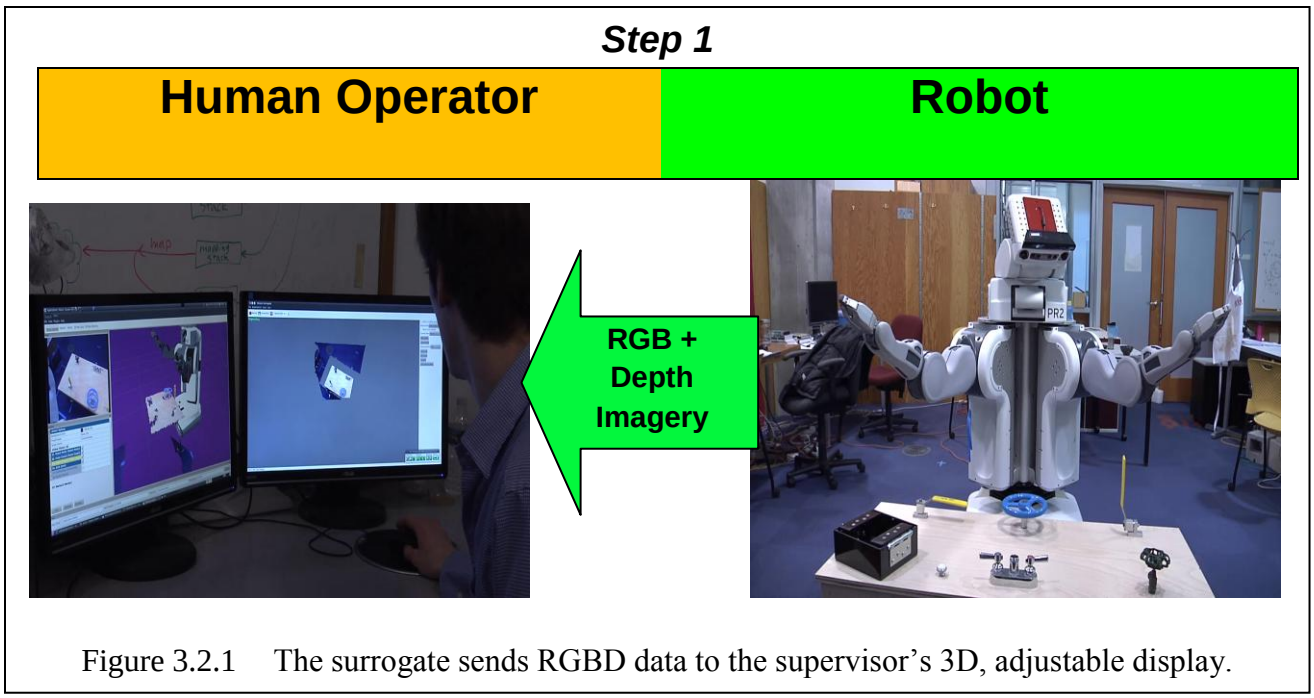


Figure 3.2.1 The surrogate sends RGBD data to the supervisor's 3D, adjustable display.

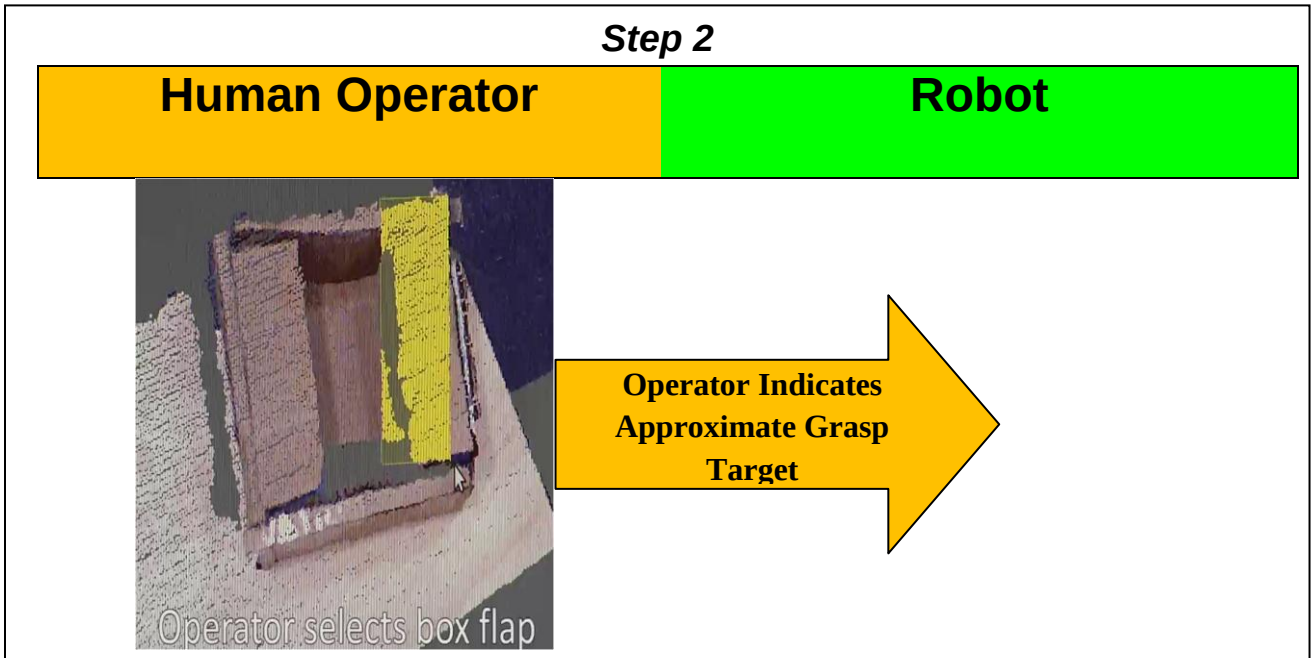


Figure 3.2.2 The supervisor indicates, roughly, the graspable region (yellow rectangle). The supervisor indicates this rough rectangle with a single mouse click-drag-release gesture. Upon completion of this gesture, point cloud updates freeze until tracking starts in Figure 3.2.5.

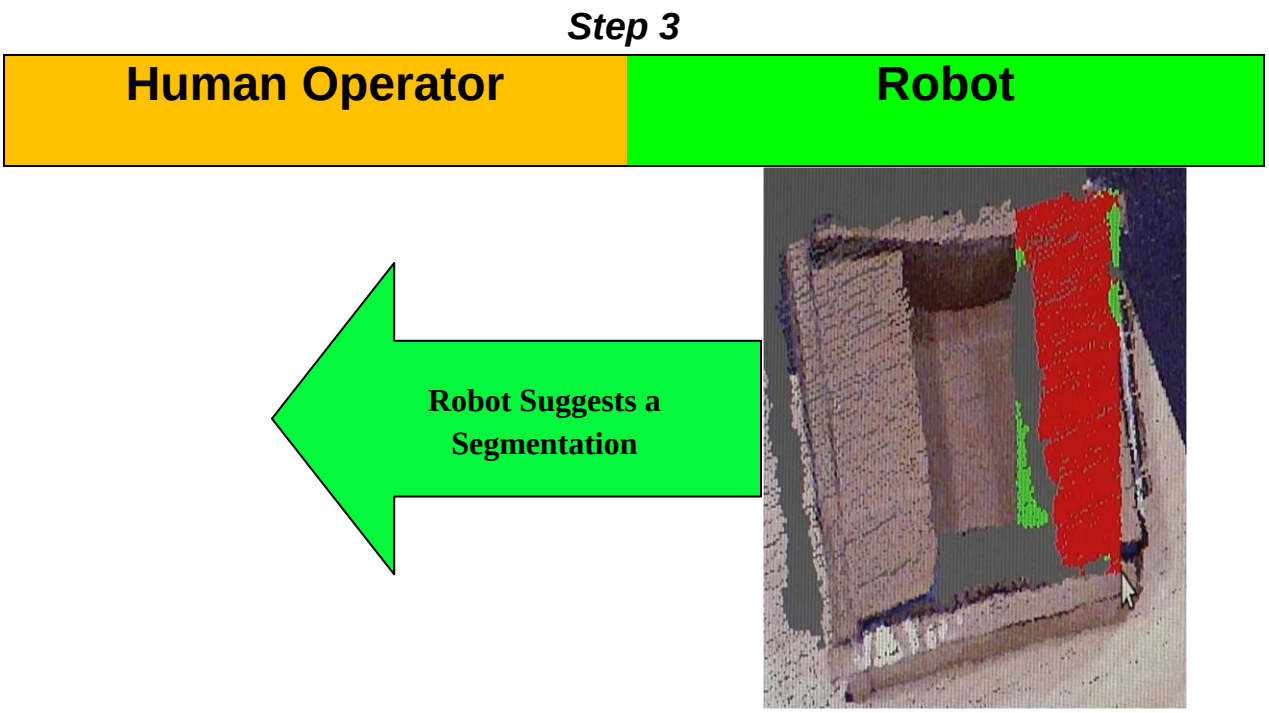


Figure 3.2.3 The surrogate suggests a segmentation for the selected region.

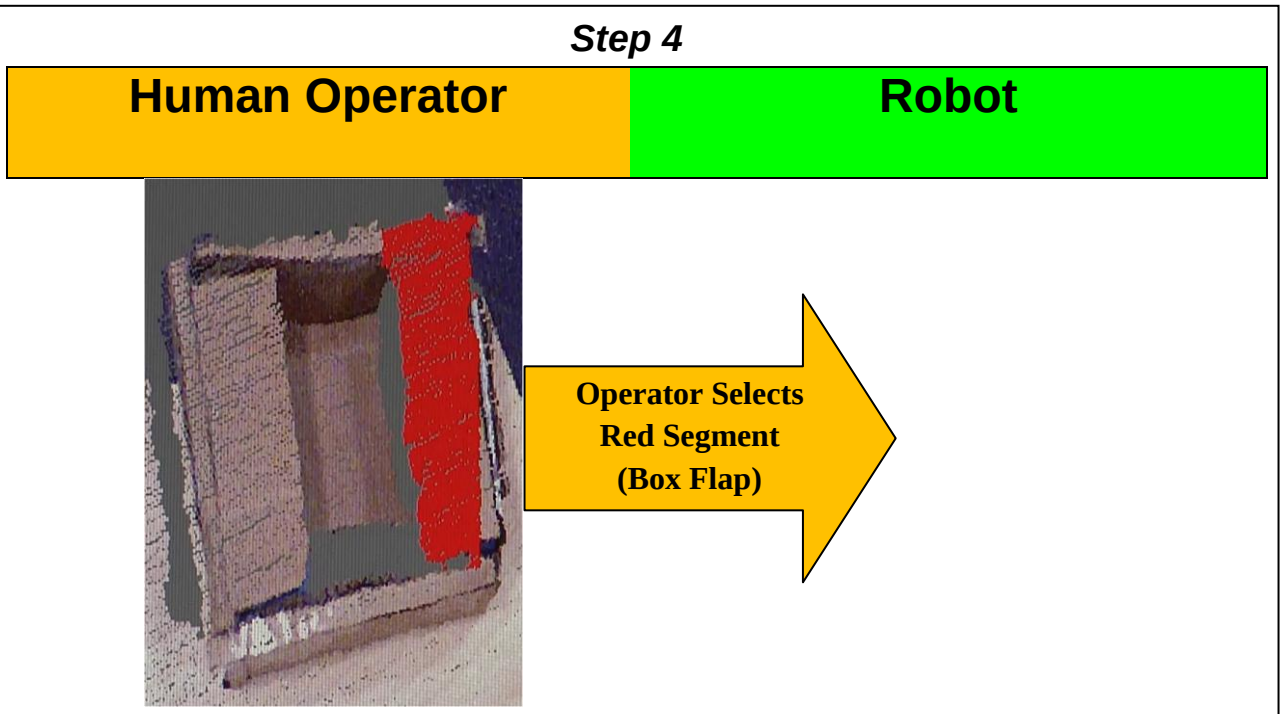


Figure 3.2.4 The operator selects the particular segment of interest. As detailed in the next section, the operator can adjust the segmentation if the algorithmically-suggested segments proved insufficient.

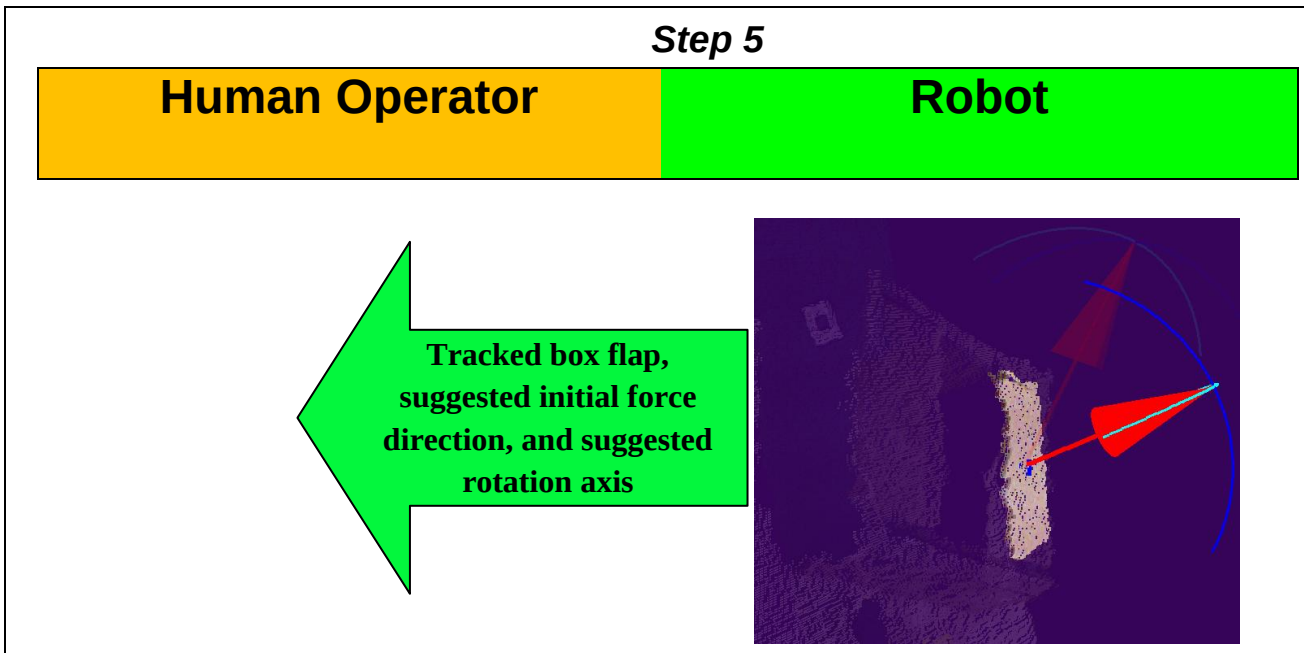


Figure 3.2.5 The surrogate begins tracking the segment of interest in live point clouds. The surrogate also suggests an initial direction for the application of force (highlighted vector) and rotation axis (de-selected vector).

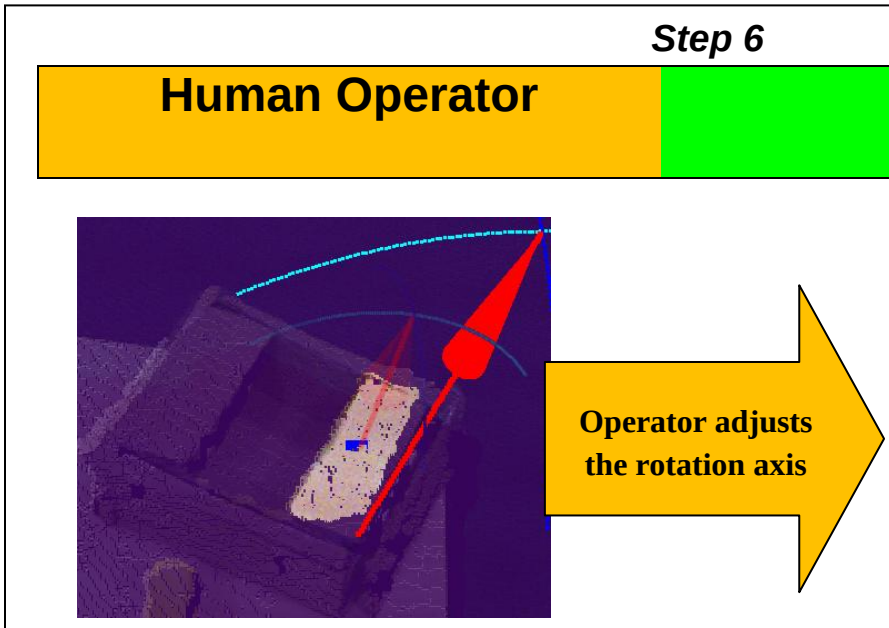


Figure 3.2.6 The operator adjusts the rotation axis (selected vector), which has 5 adjustable DOFs.

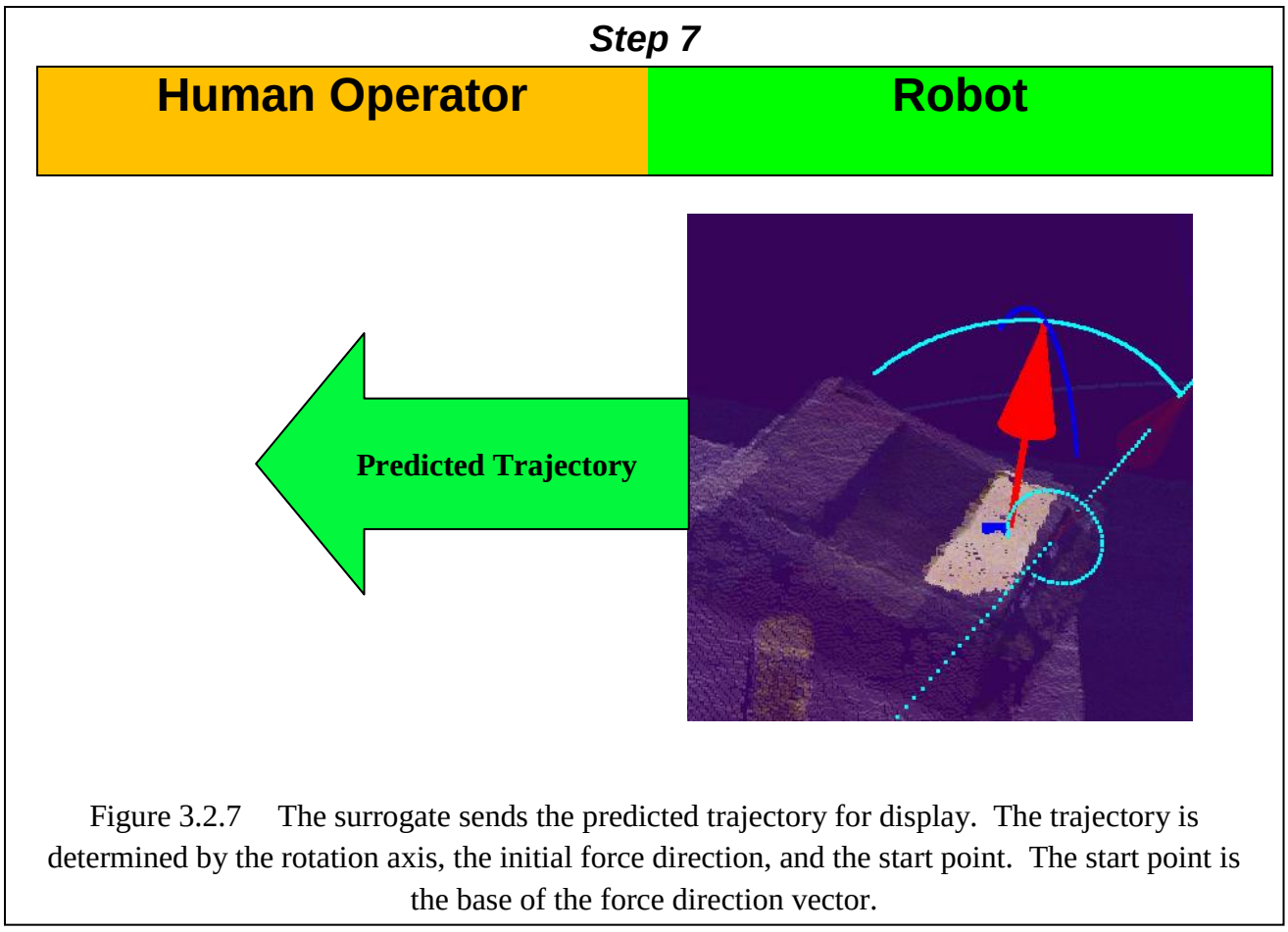


Figure 3.2.7 The surrogate sends the predicted trajectory for display. The trajectory is determined by the rotation axis, the initial force direction, and the start point. The start point is the base of the force direction vector.

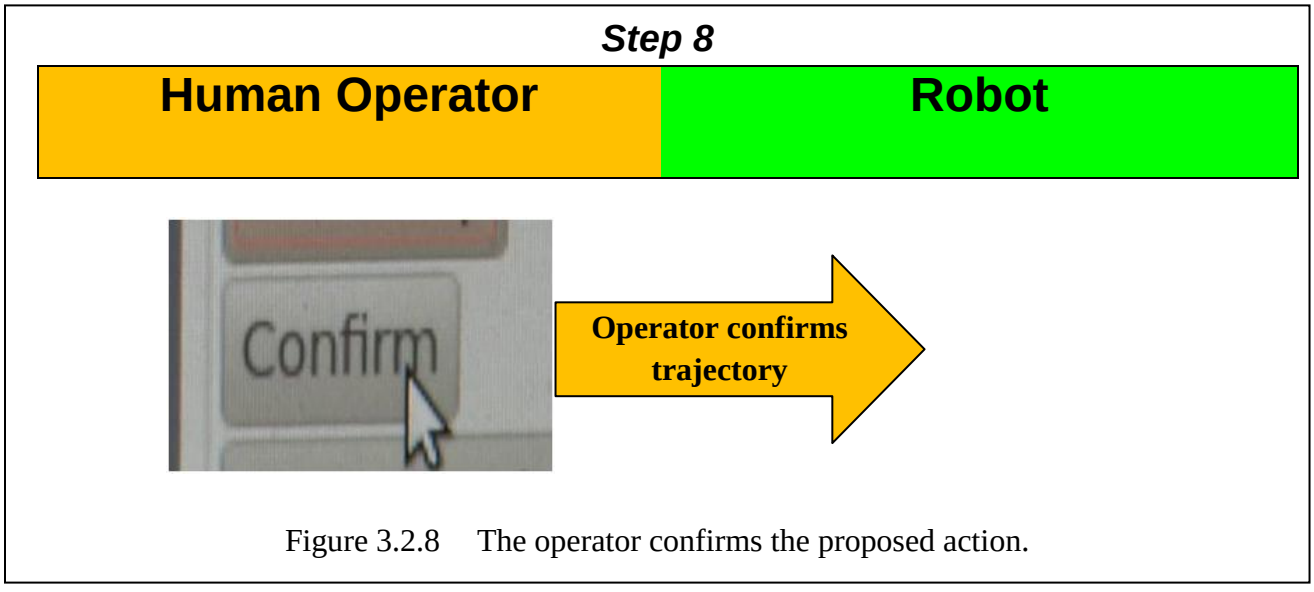


Figure 3.2.8 The operator confirms the proposed action.

Step 9

Robot

Figure 3.2.9 The surrogate grasps the box flap and opens the box.

An early video of the system is provided at: http://youtu.be/EMq9qn6_okc .

3.3 Segmentation

The segmentation module takes as input a 3D point cloud and outputs a partition of that point cloud into disjoint subsets; we call that partition a segmentation. The goal of segmentation is to partition the cloud according to the meaningful objects it contains. For example, if the point cloud contains 3D points for a wheel, a box, and a lever, then a perfect segmentation would

produce three sets: {points for the wheel}, {points for the box}, and {points for the lever}. In this section we describe the segmentation tool developed as part of the GUI.

Figure 3.3.1 shows a GUI snapshot at startup. Using the GUI, the supervisor can pan, rotate, and zoom the view with the mouse.

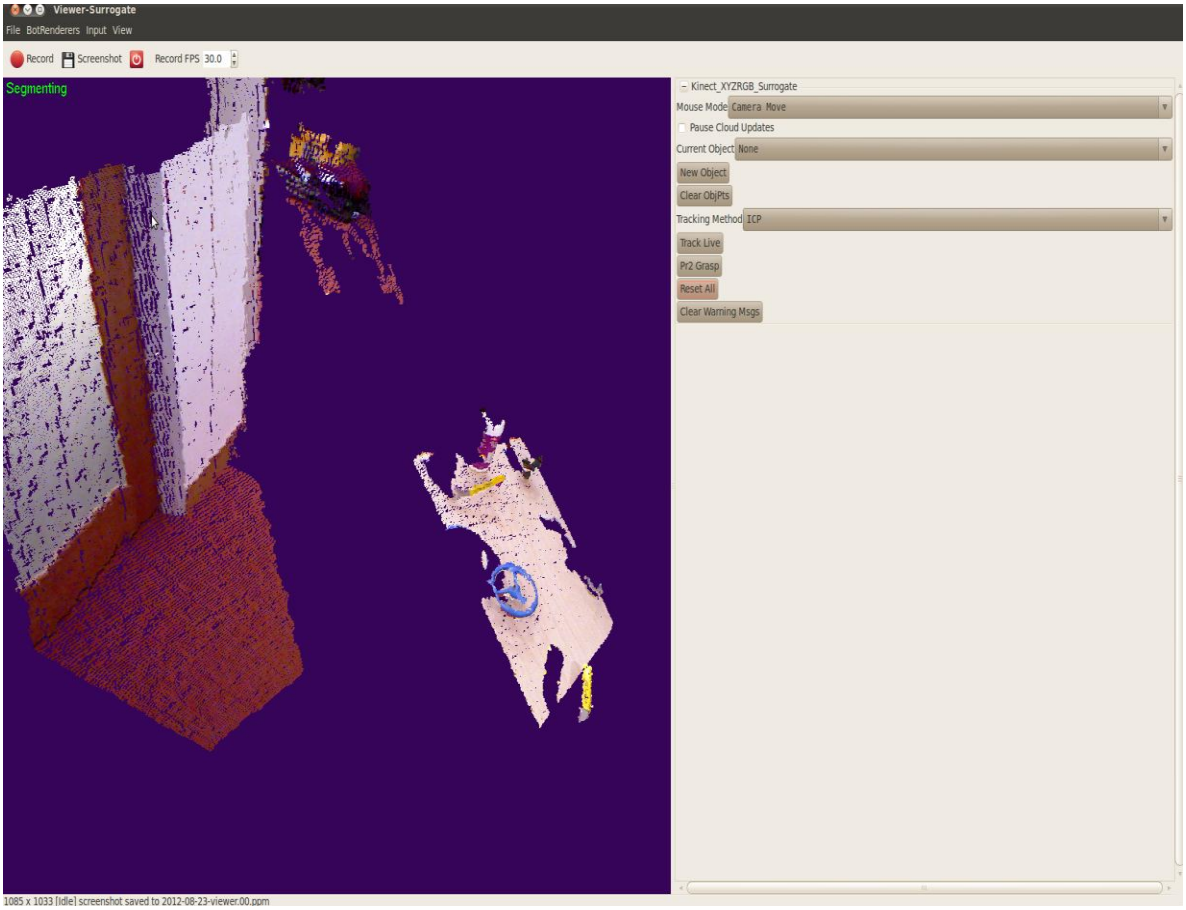


Figure 3.3.1 GUI Snapshot at system startup

At startup, the system is in *Segmentation Mode*, indicating that the operator is expected to select/segment an object for grasping and manipulation. It's informative to describe the segmentation tool in two parts: manual segmentation and assisted segmentation.

Version one of the segmentation tool can be seen in Figure 3.3.2. Figure 3.3.2 shows a user segmenting the blue wheel that was seen in Figure 3.3.1. The user can add points to a segment by holding *Shift* and selecting a rectangular region (left and center images). The user can also intersect a previous selection with a new selection (rightmost images).

Figure 3.3.2 User segments the wheel. In the left and center images, the user adds points to the segment. In the right-most images, the user removes points from the segment by intersecting with a new rectangular selection.

Version two of the segmentation tool provides algorithmically-suggested "auto-segments." In Figure 3.3.3, we see:

(i) The supervisor makes a rough rectangle-selection around the wheel (as in Figure 3.3.2);

(ii) The rough selection is automatically split into 2 segment suggestions: points corresponding to the wheel and points corresponding to the base; and

(iii) The supervisor uses the keyboard to toggle to the auto-segment for the wheel.

Version two of the segmentation tool is built on top of the manual version one. If the suggested segments are inaccurate, the supervisor has the option of ignoring the selections and applying additional manual select/remove/intersect gestures. The robot will still offer segmentation

suggestions as the supervisor refines the selected region, but the supervisor does not have to use the suggestions.

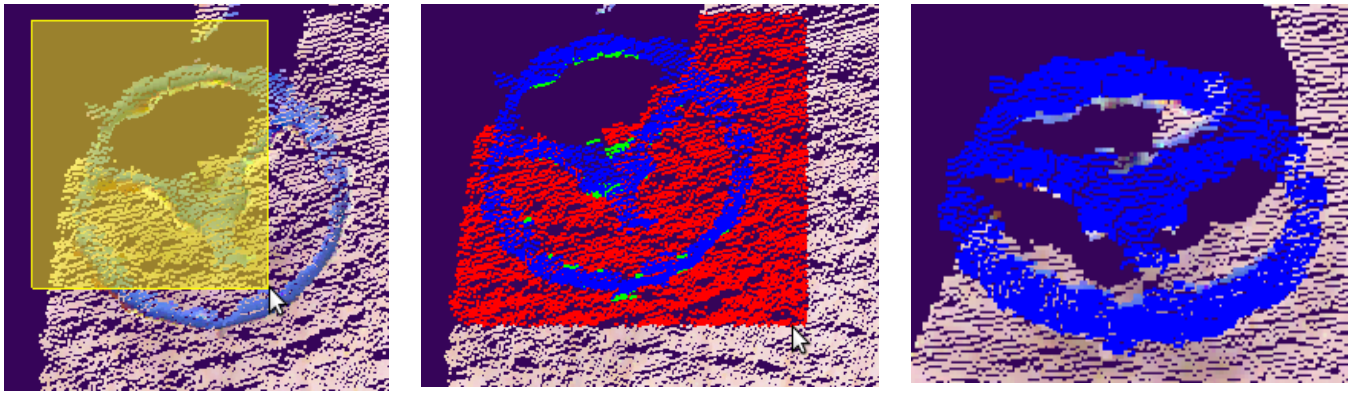


Figure 3.3.3: Segmentation Version 2: Robot-Assisted: (i) Supervisor makes rough selection; (ii) Robot makes segment suggestions; (iii) Supervisor selects desired segment.

Segmentation Algorithm: Segmentation suggestions are generated by applying RANSAC plane-fitting to the supervisor's selection. After extracting the largest candidate planes, we apply a clustering algorithm to the individual planes. This clustering step is necessary, because RANSAC may find a good plane-fit for points that lie in two co-planar clusters, but on two far-apart objects. In practice, these suggested segments work well for objects that have a somewhat flat, graspable surface. For instance, wheels, levers, and box flaps typically have graspable regions that lie within a plane. Our technique does not work well for spheres or other highly-curved surfaces, so when grasping objects of this type, the supervisor may need to resort to manual segmentation.

In practice, for a supervisor familiar with the GUI, it's quite fast (requiring less than a minute) to segment an object manually, and even faster (seconds) to use an auto-segment suggestion. Thus,

although we will devote more effort to algorithmic segmentation in future versions of the system, the current version is not an issue in practice.

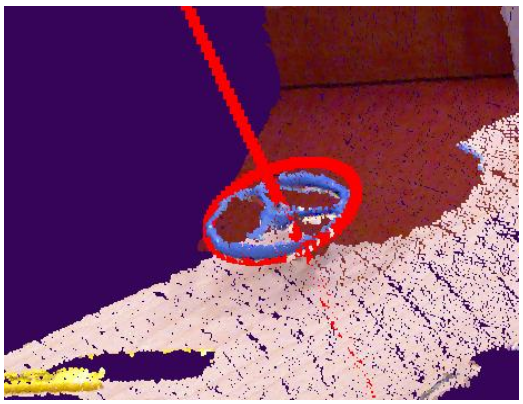
The assisted segmentation capability highlights the kind of variable autonomy that we are striving for: the surrogate autonomously proposes segmentation suggestions, but if the suggestions aren't accurate, the supervisor can resort to a less-autonomous mode, namely manual segmentation.

3.4 Model Fitting

We use the term "model fitting" somewhat synonymously with "object detection." We define model fitting as taking two inputs: (i) a 3D point cloud and (ii) a particular *type* of object we are looking for (e.g. a wheel, but not a particular wheel); the desired output is either (a) the location and orientation of the object instance (e.g. a particular wheel) within the input cloud or (ii) null if no such object is present.

Given the ease and low cost of obtaining colored point cloud data (e.g. from the Microsoft Kinect), it seems wasteful to throw out color information and use only depth information, or to throw out depth information and use only image data. The model-fitting algorithm described below uses both color and depth information. To our knowledge, object detection of parametric shapes (e.g. circles of unknown radius, position, and orientation) has not been tackled using the techniques described in this section.

Figure 3.4.1 shows the result of our model-fitting method applied to three different wheels. The goal of our model-fitting algorithm is not to obtain a perfect match for a wheel, lever, knob, etc. But rather, we wish to obtain just a reasonable initial guess. As detailed in section 3.5, the user can adjust parameters of the model fit: rotation axis, grasp point, initial direction of rotation. In fact, if model fitting fails completely, the user can still use the GUI to specify the parameters for a revolute and prismatic joint manually. Thus, the model-fitting method need only seed those parameters with a good initial guess, and the supervisor can fine-tune if necessary.



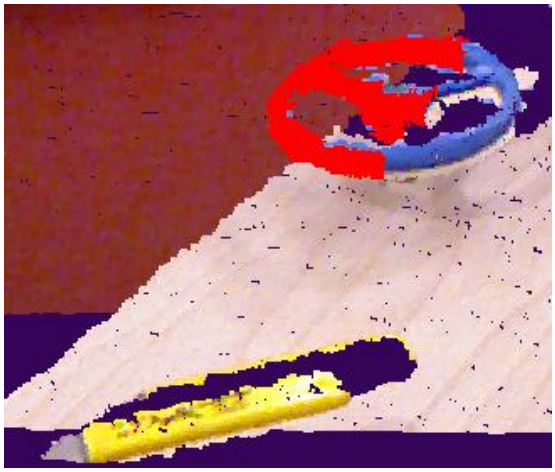
(i) Valve Wheel

(ii) Bicycle Wheel

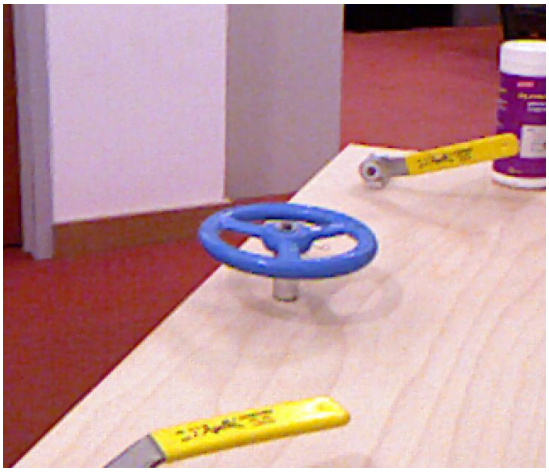
(iii) Steering Wheel

Figure 3.4.1: Parametric Model-fitting applied to three different wheels

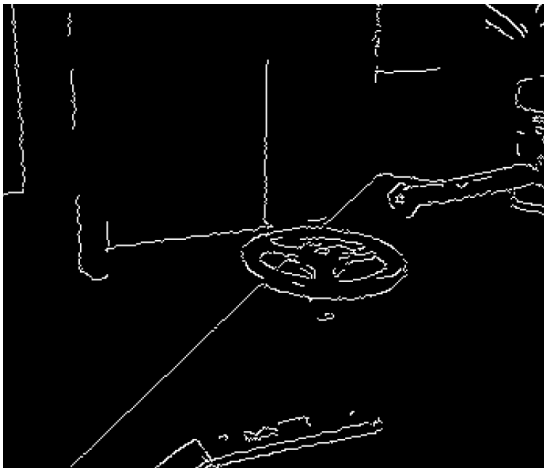
Figures 3.4.2 shows the model-fitting pipeline for a wheel. Following Figure 3.4.2 we describe the pipeline.



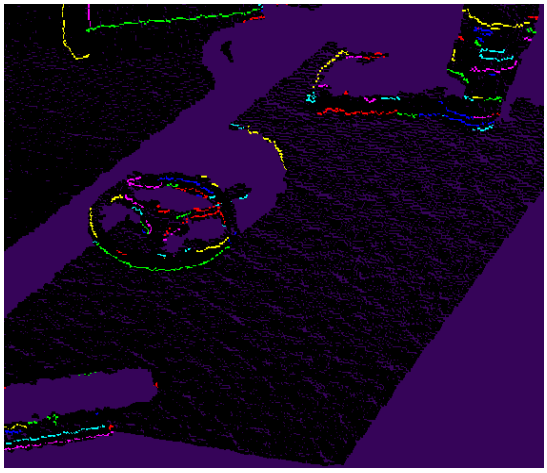
(1) 3D partial segmentation (red) of the wheel



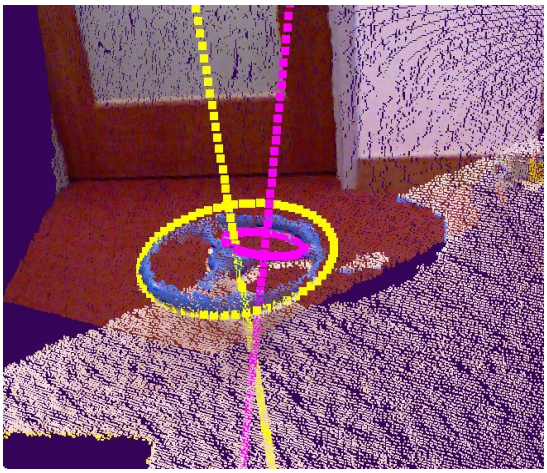
(2) 2D RGB image



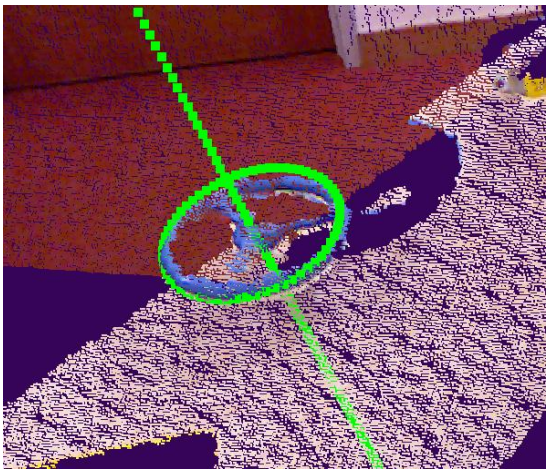
(3) 2D Canny edges [21]



(4) Canny edges with depth



(5) Candidate circles



(6) Final Selected Model
(same as yellow circle in step 5)

Figure 3.4.2: Model-fitting Pipeline.

In Figure 3.4.2, we see:

Step 1: The supervisor's segment selection – the output from segmentation in section 3.3. Note that the supervisor did not take the time to fully segment the wheel. Step 5 discusses how the model-fitting is robust to such partial-segmentation, or even possible occlusion.

Step 2: The 2D RGB image from the Kinect.

Step 3: The result (2D image) of running Canny Edge Detection [21] on the RGB image.

Step 4: The projection of the Canny edges into R^3 using the depth information. We do not project pixels for which we do not have depth information.

Step 5: Circles that we fit to the 3D Canny edges using principle components analysis (PCA) and modified least-squares circle fitting [22]. Before applying circle fitting for an edge, we

(i) Run PCA on the 3D edge

(ii) Project the 3D edge into the dominant plane determined by PCA; and

(iii) Apply a change of basis and then run modified least squares circle fitting [22].

This 3-step circle fitting process does not require that the input edges collectively or individually cover the entire wheel. That is, we can estimate the radius and 3D center of a circle even from just one small circle arc. In fact, in the process illustrated by Figure 3.4.2, only edges from the portion of the wheel that were contained in the segmentation (Figure 3.4.2 image 1) are

considered. This is a useful feature since we may have occlusions, or as in the Figure, the user may not take the time to segment the wheel completely.

Step 6: The subset of circles that have diameters within 50% of the selection diameter (the maximum distance between any two points in the input segment). This filter is designed to remove spurious circles with widely-varying diameters, which can be seen in the previous stage. If only one circle remains after this filtering, we return that circle as the selected model. If multiple candidate circles remain, we choose one arbitrarily. In a version of the system currently in development, we choose among remaining candidates by taking the circle with the rotation axis closest to the normal of the best-fit-plane for the entire segment. ▫

The resulting model-fit may be imperfect or misaligned, as in the final image of Figure 3.4.2. This can happen if the 3D Canny edge that generates the fit does not lie completely within the plane normal to the wheel's rotation axis. For example, in Figure 3.4.2, the resulting model-fit comes from an edge that traces along the outermost surface of the wheel; this edge is circular but with an orientation slightly different from the wheel's orientation. Consequently, the computed rotation axis is slightly skewed.

The technique of transforming 2D Canny edges into space curves allows us to extract meaningful 3D geometric information in an intuitive manner. If we are looking for a wheel or torus, then we expect to extract 3D edges with approximately the right curvature. This technique is more computationally tractable than running a RANSAC-like fitting method. For circles oriented in R^3, we need to find 6 parameters: the 3D circle center, the two-angle roll/pitch of the circle, and

the radius. Schnabel [17] is able to achieve circle fitting in 3D using point cloud normals as an additional input to RANSAC. Using a naïve version of RANSAC to find all 6 parameters can be computationally prohibitive; perhaps that's why the Point Cloud Library [23] has not provided an implementation for RANSAC fitting with circles in 3D or tori [24].

Our model fitting for tori is accomplished by approximating a torus as a circle oriented in 3D. In future versions of the system, we intend to apply the same 2D → 3D feature mapping techniques to generate candidate model fits for levers, knobs, and boxes. Our system already has the ability to find 3D line segments using the same Canny-to-3D-Space Curve mapping – from which we intend to fit lever/box models.

3.5 Model Adjustment

Whether model fitting succeeds or fails, the supervisor has the option of adjusting the rotation axis, grasp point, and initial force vector. Figure 3.5.1 shows: (i) Adjustable 5-DOF rotation axis (highlighted); (ii) Adjustable 5-DOF initial force direction (deselected), which starts at the grasp point; and (iii) Trajectory determined by i and ii (light blue circle). The resulting trajectory is circular, begins at the grasp point, and starts with an initial movement in the direction of the force vector.

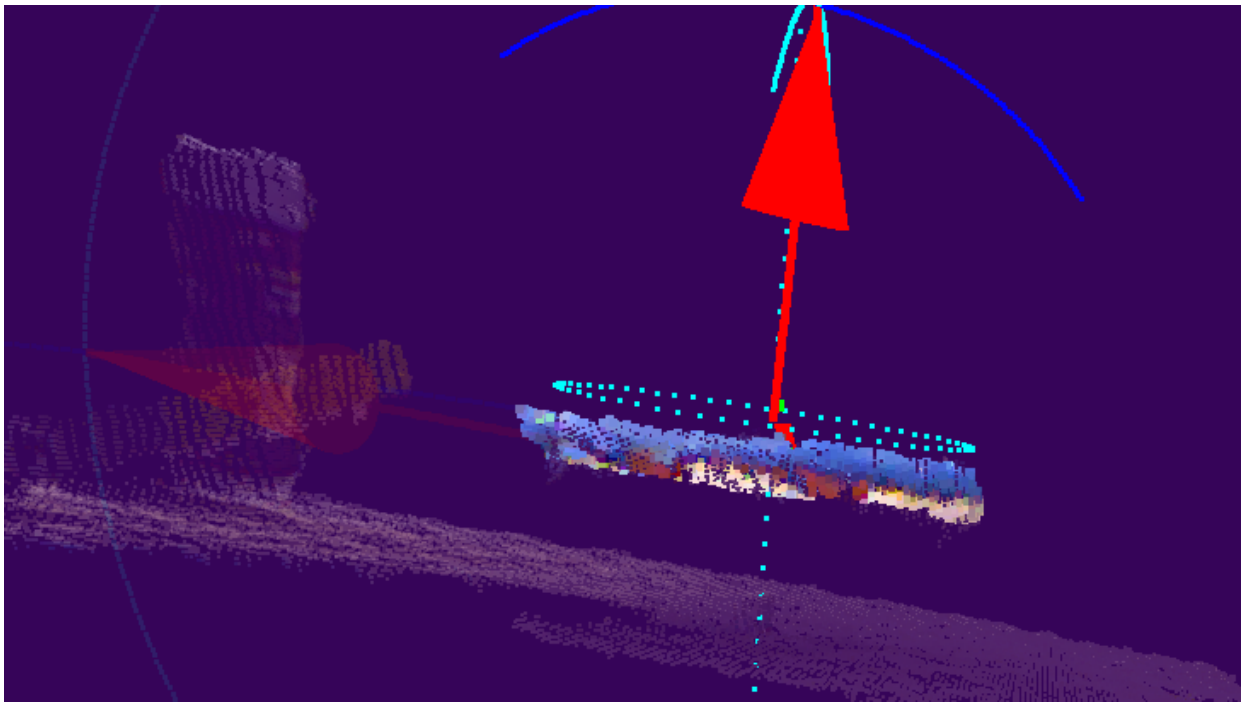


Figure 3.5.1: Model Adjustment: (i) 5-DOF rotation axis (highlighted); (ii) 5-DOF Initial force vector (de-selected vector); (iii) Circular trajectory for the gripper determined by i and ii (light blue circle).

3.6 Tracking

After segmentation is complete and model-fitting finishes (or fails), the system begins tracking the segmentation selection from live point cloud data. We use iterative-closest point (ICP) for tracking [25]. Since ICP is computationally expensive, we down-sample the point clouds by a factor of four before running ICP. ICP works well for small object displacements between point cloud updates. In our applications, we expect the workspace for manipulation to be mostly static, and therefore, we expect at most small displacements of the manipuland between updates. In future versions of the system, we may also attempt tracking using the 3D Canny edges we computed in section 3.4.

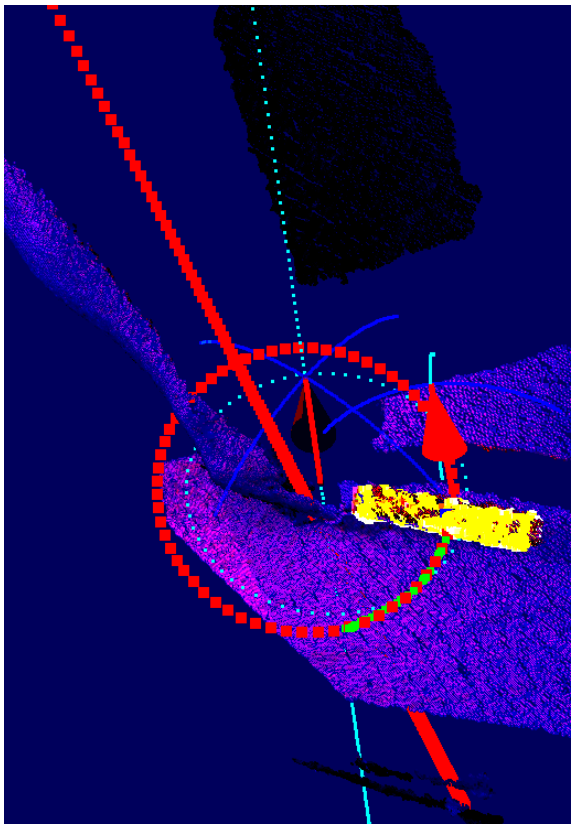


Figure 3.6.1 Axis classification and fitting from motion. The input to the classification/fitting is the 3D trace of the centroid for the tracked object segment (green points). The output is a joint fit that is one of {prismatic, revolute, unknown} and is shown as the red circle and red axis. The supervisor's initial confirmed trajectory is shown as the light blue circle and axis. The red vector is the adjustable 5-DOF force vector (see previous section) that the supervisor can modify to adjust the force direction. The arm and hand of the person moving the lever can be seen on the left.

In addition to tracking the segment selection in live point clouds; we use the object's trajectory (if it moves) to estimate more accurately the axis type and location of the manipuland: {revolute, prismatic, or unknown}. Figure 3.6.1 shows the supervisor's confirmed axis location choice for the lever (light blue axis and circle); this is an output from the proceeding sections (before the lever is moved). The Figure also shows the system's inferred axis-type and location estimate (red circle and axis) computed from motion tracking. Finally, the Figure shows the 5-DOF force vector (red) that the supervisor can adjust. The inferred axis shows a much better fit than the supervisor's input. This system ability to infer the axis location (and type) means that the

supervisor need only provide a decent initial estimate of the axis parameters – the supervisor need not fine-tune parameters extensively. Note that, here, the lever was moved by a 3rd party – that is, neither the surrogate nor the supervisor moved the lever. We do not expect this scenario to arise in practice, but we use it to test this preliminary work in updating object affordance models from motion. Figure 3.6.2 shows the axis classification and location result for a prismatic joint.

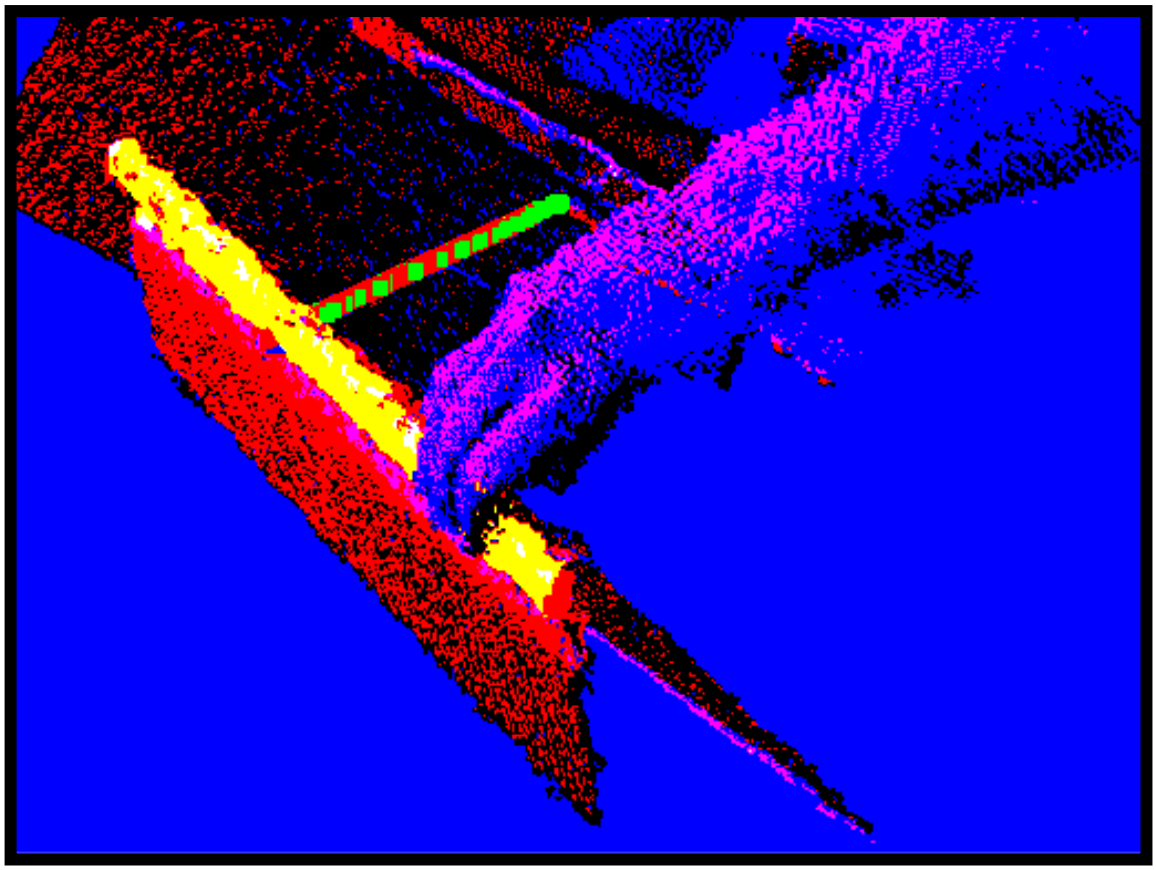


Figure 3.6.2 Axis classification and axis location estimation from motion. The top of the drawer is being tracked (highlighted in yellow). The trace of the centroid for the tracked segment is shown as green points (one for each iteration of tracking). The inferred prismatic axis is shown as the red line segment, which starts at the drawer's closed position and ends at the drawer's current (open) position.

3.7 Grasp Selection

After the supervisor confirms or adjusts the relevant model/axis parameters, she is provided with the option of selecting the approach-direction for grasping the manipuland. As in Figure 3.7.1, the supervisor is greeted with a display showing the possible approach vectors for the end-effector. Currently, this grasp-selection GUI is located in a 2nd window and is built using ROS modules; in future version of the system, we will merge this grasp display into our current framework. In the spirit of variable autonomy, the supervisor has the option of either (a) letting the surrogate automatically pick and use a grasp or (b) toggling through possible grasps.

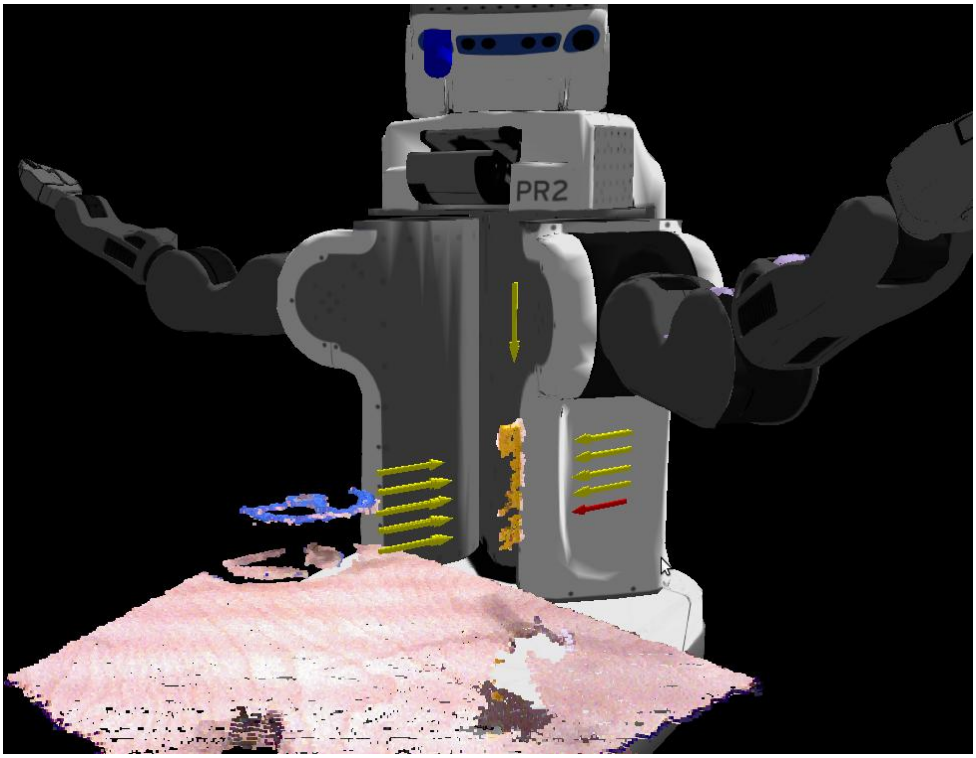


Figure 3.7.1: Grasp Approach Selection: Candidate approach vectors are shown in yellow. The supervisor's current selection is the red arrow. Here, the supervisor might want to switch to a grasp at the top of the lever to get more torque.

We use the ROS pr2_gripper_grasp_planner_cluster package [26] which is based on work by Kaijen Hsiao et al. [27]. This grasp planner takes as input an unstructured point cloud, which in

our case is the supervisor's segment selection. The grasp planner outputs a list of candidate grasps, with estimated grasp "qualities" between 0 (low quality) and 1 (high quality). Figure 3.7.1 visualizes the approach vector for a few grasps.

This grasping module highlights again the variable autonomy aspect of our system. The surrogate can autonomously pick and execute a grasp. Or, the supervisor can intervene and decide which of several possible grasps he would like the surrogate to use.

3.8 Planning and Execution

We use ROS's Arm_Navigation stack for planning and executing trajectories with an RRT-based planner [28]. We use this module off-the-shelf and have not made modifications. We use the arm_navigation stack both for planning arm movements to a grasp pose and for subsequent arm motions.

4 Results

Here we compare our affordance-based system to joint-by-joint teleoperation. The physical setup for using our system is as described at the start of section 3: A supervisor is stationed at a desktop running our GUI (Figure 3.1.2), and the PR2 is situated in front of the busy box (Figure 3.1.1). For teleoperation, a supervisor uses joint-by-joint control software based on a ROS PR2 teleoperation software package [29]. In both scenarios, the operator can see the busy box workspace, and the PR2, only by looking at the monitor.

In our trials, each of 3 supervisors receives brief instruction on how to use both systems. We time and record each supervisor as they guide the PR2 through 3 manipulation tasks: opening a box flap, pushing a lever, and lifting (straight up) a baking soda box. The first 2 manipulands can be seen in Figure 1.4. Each supervisor executes each task, first using the teleop controller and then using our affordance-based system. Before each trial, the PR2's arms are brought back to the fixed initial state – shown in Figure 3.1.1. The results of these trials are recorded in tables 4.1 to 4.3.

Supervisor	*Teleoperation* seconds)	*Affordance-based* system (seconds)
1 (author)	230	47
2	191	77
3	221	143

Table 4.1 Time to Open Box Flap

Supervisor	*Teleoperation* (seconds)	*Affordance-based* system (seconds)
1 (author)	250	51
2	693	68
3	229	64

Table 4.2 Time to Push Lever

Supervisor	*Teleoperation* (seconds)	*Affordance-based* system (seconds)
1 (author)	273	55
2	212	70
3	874	65

Table 4.3 Time to Lift Baking Soda Box

Across all supervisors and trials, the affordance-based system is consistently faster. We see an average speedup improvement (averaged across all supervisors) of 2.97x with the box flap, 6.22x with the lever, and 6.14x with the soda box lift.

5 Contributions

We have developed a mobile manipulator capable of taking guidance from a human supervisor and varying its level of autonomy accordingly. Our first application is explosive ordnance disposal. Our system is designed so that the human supervisor can grant and retract autonomy to/from the surrogate at any time. After the surrogate reflects to the supervisor its understanding of the salient aspects of the workspace, including available affordances and planned actions, the operator grants permission to the robot to execute those actions.

Our contributions include:

(1) The algorithms and software designed for mental-model sharing and supervisor-assisted manipulation with a robot surrogate.

(2) Using this system, we successfully command the PR2 robot to pull/push levers, lift a knob, and open boxes. Currently, the system is able to manipulate revolute and prismatic pairs. We will be extending our techniques to more general kinematic chains in future versions of the system.

(3) Our HRI software is platform agnostic, as mentioned in section 3. Only the planning software contains a model of the PR2. A nice consequence of this platform-agnostic feature is that we can and will further develop this system for use in both EOD, with a PackBot, and in the DARPA robotics challenge, with a humanoid robot.

5 References

[1] W. Kim, I. Nesnas, M. Bajracharya, R. Madison, A. Ansar, R. Steele, J. Biesiadecki and K. Ali, "Targeted Driving Using Visual Tracking on Mars: from Research to Flight," *Journal of Field Robotics,* 2008.

[2] "iRobot 510 PackBot," [Online]. Available: http://www.irobot.com/us/robots/defense/packbot.aspx. [Accessed 8 August 2012].

[3] "Talon," [Online]. Available: http://www.qinetiq-na.com/products/unmanned-systems/talon/. [Accessed 22 08 2012].

[4] A. Huang, S. Tellex, A. Bachrach, T. Kollar, D. Roy and N. Roy, "Natural Language Command of an Autonomous Micro-Air Vehicle," in *Proceedings of the International Conference on Intelligent Proxys and Systems (IROS)*, 2010.

[5] R. Parasuraman, T. B. Sheridan and C. D. Wickens, "A Model for Types and Levels of Human Interaction with Automation," in *IEEE Trans. on SMC*, 2000.

[6] J. Crandall and M. Goodrich, "Experiments in adjustable autonomy," in *IEEE International Conference on Systems, Man, and Cybernetic*, 2001.

[7] S. Teller, M. Walter, M. Antone, A. Correa, R. Davis, L. Fletcher, E. Frazzoli, J. Glass, J. How, A. Huang, J. h. Jeon, S. Karaman, B. Luders, N. Roy and T. Sainath, "A Voice-Commandable Robotic Forklift Working Alongside Humans in Minimally-Prepared Outdoor Environments," in *IEEE International Conference on Robotics and Automation*, Anchorage AK, 2010.

[8] S. Tellex, T. Kollar, S. Dickerson, M. Walter, A. Banerjee, S. Teller and N. Roy, "Understanding Natural Language Commands for Robotic Navigation and Mobile Manipulation," in *Conference on Artificial Intelligence (AAAI)*, San Francisco, CA, 2011.

[9] R. S. Jackendoff, Semantics & Cognition, MIT Press, 1983, pp. 161-187.

[10] A. Correa, M. Walter, L. Fletcher, J. Glass, S. Teller and R. Davis, "Multimodal Interaction with an Autonomous Forklift," in *International Conference on Human-Robot Interaction*, Osaka, Japan, 2010.

[11] A. Torralba, "6.870 Grounding Object Recognition : Fall 2011," 2011. [Online]. Available: http://people.csail.mit.edu/torralba/courses/6.870_2011f/lectures/lecture4.ppt. [Accessed 30 08 2012].

[12] P. Viola and M. Jones, "Rapid Object Detection Using a Boosted Cascade of Simple Features," in *CVPR*, 2001.

[13] N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection," in *CVPR*, 2005.

[14] P. Besl and R. Jain, "Invariant Surface Characteristics for 3D Object Recognition in Range Images," in *Computer Vision, graphics, and image processing*, 1986.

[15] M. Fischler and R. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Communications of the ACM,* vol. 24, no. 6, pp. 381-395, 1981.

[16] C. Papazov, S. Haddadin, S. Parusel, K. Krieger and D. Burschka, "Rigid 3D geometry matching for grasping of known objects in cluttered scenes," *International Journal of Robotics Research,* vol. 31, no. 4, pp. 538-553, 2012.

[17] R. Schnabel, R. Wahl and R. Klein, "Efficient RANSAC for Point-Cloud Shape Detection," *Computer Graphics Forum,* vol. 26, no. 2, pp. 214-226, 2007.

[18] "PR2 Specs," [Online]. Available: http://www.willowgarage.com/pages/pr2/specs. [Accessed 12 08 2012].

[19] A. Huang, E. Olson and D. Moore, "LCM: Lightweight Communications and Marshalling," in *Int. Conf. on Intelligent Robots and Systems (IROS)*, Taipei, Taiwan, 2010.

[20] M. Quigley, B. Berkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler and A. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

[21] J. Canny, "A Computational Approach To Edge Detection," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 8, no. 6, p. 679–698, 1986.

[22] D. Umbach and K. Jones, "A Few Methods for Fitting," *IEEE Transactions on Instrumentation and Measurement,* pp. 1881-1885 , 2003.

[23] "Point Cloud Library," [Online]. Available: http://pointclouds.org/. [Accessed 12 08 2012].

[24] "Point Cloud Library 1.7.0 : model_types.h," [Online]. Available: http://docs.pointclouds.org/trunk/model__types_8h_source.html. [Accessed 12 08 2012].

[25] Z. Zhang, "Iterative Point Matching for Registration of Free-Form Curves," *IRA Rapports de Recherche,* Vols. Programme 4: Robotique, Imageet Vision, no. 1658, 1992.

[26] "pr2_gripper_grasp_planner_cluster," Willow Garage, [Online]. Available: http://www.ros.org/wiki/pr2_gripper_grasp_planner_cluster. [Accessed 12 08 2012].

[27] K. Hsiao, S. Chitta, M. Ciocarlie and E. G. Jones, "Contact-reactive grasping of objects with partial shape information," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010.

[28] E. G. Jones, "arm_navigation," ROS, [Online]. Available: http://www.ros.org/wiki/arm_navigation. [Accessed 25 08 2012].

[29] G. Jones, "pr2_teleop_general," ROS.org, [Online]. Available: http://www.ros.org/wiki/pr2_teleop_general. [Accessed 25 08 2012].

[30] T. Sheridan, Teleproxyics, automation, and human supervisory control, MIT Press, 1992.

[31] B. K. Horn, Robot Vision, The MIT Press, 1986.