CONSTRUCTING MOBILE MANIPULATION BEHAVIORS USING EXPERT INTERFACES AND AUTONOMOUS ROBOT LEARNING

A Thesis
Presented to
The Academic Faculty

by

Hai Dai Nguyen

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Interactive Computing

Georgia Institute of Technology
December 2013

CONSTRUCTING MOBILE MANIPULATION BEHAVIORS USING EXPERT INTERFACES AND AUTONOMOUS ROBOT LEARNING

Approved by:

Prof. Charles C. Kemp, Advisor
Biomedical Engineering
Georgia Institute of Technology

Prof. James M. Rehg
Interactive Computing
Georgia Institute of Technology

Prof. Aaron Bobick
Interactive Computing
Georgia Institute of Technology

Prof. Andrea Thomaz
Interactive Computing
Georgia Institute of Technology

Dr. Kaijen Hsiao

Bosch Research and Technology Center

Date Approved: October 23, 2013

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

With current state-of-the-art approaches, development of a single mobile manipulation capability can be a labor-intensive process that presents an impediment to the creation of general purpose household robots. At the same time, we expect that involving a larger community of non-roboticists can accelerate the creation of new novel behaviors. We introduce the use of a software authoring environment called ROS Commander (ROSCo) allowing end-users to create, refine, and reuse robot behaviors with complexity similar to those currently created by roboticists. Akin to Photoshop, which provides end-users with interfaces for advanced computer vision algorithms, our environment provides interfaces to mobile manipulation algorithmic building blocks that can be combined and configured to suit the demands of new tasks and their variations.

As our system can be more demanding of users than alternatives such as using kinesthetic guidance or learning from demonstration, we performed a user study with 11 able-bodied participants and one person with quadriplegia to determine whether computer literate non-roboticists will be able to learn to use our tool. In our study, all participants were able to successfully construct functional behaviors after being trained. Furthermore, participants were able to produce behaviors that demonstrated a variety of creative manipulation strategies, showing the power of enabling end-users to author robot behaviors.

Additionally, we introduce how using autonomous robot learning, where the robot captures its own training data, can complement human authoring of behaviors by freeing users from the repetitive task of capturing data for learning. By taking advantage of the robot's embodiment, our method creates classifiers that predict using visual appearances 3D locations on home mechanisms where user constructed behaviors will succeed. With active learning, we show that such classifiers can be learned using a small number of examples. We also show that this learning system works with behaviors constructed by non-roboticists in our user study. As far as we know, this is the first instance of perception learning with behaviors not hand-crafted by roboticists.

CHAPTER I

INTRODUCTION

Robots are often portrayed in science fiction as maids or housekeepers–home agents that are able to perform a dizzying array of tasks such as cleaning, laundering, guarding, serving, and even food preparation. This concept of domestic robots as machines capable of human-like general purpose labor has inspired and attracted many roboticists and members of the general public alike. However, while modern robots, with their high degree-of-freedom manipulators, have been shown performing a wide variety of tasks, development of even just one of these capabilities is an involved process requiring the cooperation of many robotics experts. Furthermore, as many systems use hand-tuned sequences of controllers tailored to small subsets of objects, their capabilities often do not generalize to different environments; a system constructed to pull open drawer handles would not work on drawers equipped with magnetic push latches. Likewise, robots that use door handles would be defeated by doors equipped with push bars.

We make progress towards solving these issues in our work by showing that end-users can create robot behaviors with similar complexity to those currently crafted by roboticists using robot behavior authoring environments. By allowing behavior creation to be crowdsourced to end-users, we believe our approach can lead to systems that will scale to the complexity of real home environments as well as potentially new uses for robots unforeseen even by expert roboticists.

With ROS Commander (ROSCo), our authoring environment, end-users piece together behaviors represented as Hierarchical Finite State Machines (HFSMs) using modular building blocks that encapsulate commonly used algorithms in mobile manipulation. By using a modular representation, our system can piece together and parameterize blocks to create a wide variety of household behaviors that, as we later show, can reliably accomplish their goals. We show that our experiment participants, all computer literate non-roboticists, can employ this authoring system to construct household mobile manipulation behaviors. Additionally, we show that autonomous learning can function as a component of this behavior authoring system to enable the robot to learn perceptual classifiers without human intervention. This then enables the creation of behaviors with greater autonomy.

In the next sections, we first discuss the motivations for building general purpose robots following with discussions of challenges in doing so. Then we discuss our approach of using software authoring environments to construct robot capabilities and its advantages compared to alternatives. Finally, we conclude with an overview of future chapters.

1.1 Motivations for Building General Purpose Robots

Once viewed as an essential social status indicator, having full-time domestic services provided is unaffordable for most in the US and similar first-world countries. However, numerous postings for cleaning, laundry, organization, and various other tasks on sites such as TaskRabbit and Craigslist demonstrate continued existence of persistent demands for more limited and less costly services. Just as in manufacturing where usage of robots helped to contain labor costs, introducing general automation to domestic settings can make services more affordable, thus enabling a greater number of individuals access to home assistance. At the same time, robots, as generic stand-ins for human labor, can reduce the need for specialized home automation and thus clutter and wasted resources. As with machine manufactured goods, work provided by robots can also be more consistent in quality and more reliable.

Aside from household chores for able-bodied individuals, general purpose robots can function in assistive scenarios enabling older adults, those afflicted with motor degenerative diseases and the similarly motor impaired to live more independent lives. With populations in the developing world growing older, 22% of the global population is expected to be over 60 years old by the year 2050 [15], and with a shortage of nurses and caregivers, the ability to provide quality care is expected to be a major challenge in the near future. However, for many older adults, placement in a nursing home can mean loss of independence, isolation from friends and family members and exposure to opportunistic infections that can lead to decreased life expectancy. These impracticalities are also similar for those with neuro-degenerative diseases such as ALS, spinal cord injuries, and other conditions leading to motor impairments. Having general purpose robots that can assist caregivers in performing their duties, or function as surrogate bodies enabling individuals to perform activities of daily living (ADLs)–such as dressing, feeding, and hygiene tasks–can go far in addressing labor shortages while enabling individuals to live independently.

1.2 Challenges to Building General Purpose Robots

There have been a number of robots reported as being able to operate drawers [25], lamps, light switches, as well as load dishware in dishwashers [72], play musical instruments, fold clothes [50], make pancakes [26], and retrieve objects from flat surfaces [64] amongst other tasks [46, 33, 60]. However, even with recent advances, general purpose domestic robots are still out of reach. Development of even just one of these capabilities is an involved process requiring trained roboticists, of which there are few. Each new capability added can require different hand crafted manipulation as well as perceptual strategies. Additionally, many existing systems are only capable of performing a few tasks in experimental settings and are not robust enough to function in complex home environments.

Further exacerbating these problems, variations in the appearance and mechanical form of objects manipulated can require nontrivial re-engineering of carefully crafted systems. In many tasks, functional components on mechanisms, such as drawer and door handles, can be distinctive or even unique. For example, current systems described in the literature that are capable of door opening only operate on a few types of door handles common in office environments while ignoring those mechanisms, such as door knobs [98, 78, 68, 122] that are more common in homes. However, enabling these systems to operate on door knobs would potentially require roboticists to conceive of different manipulation and perceptual strategies created specifically for turning and detecting door knobs. In homes, even objects manufactured using identical processes can function differently depending on wear and use. Lighting condition and viewing perspective can likewise affect the appearance of objects drastically.

Thus, addressing this labor and knowledge intensive process that involves the use of expert roboticists for each new task and the variations on those tasks can be a crucial factor in allowing general purpose robots to function in household settings.

1.3 Approach Overview

In this thesis we propose a solution to address the lack of autonomous robot behaviors for different tasks and variations on those tasks introduced by the diversity of objects in common household environments. There are a number of related solutions to this problem including approaches as diverse as formal task planning [111], logic-based reasoning [52], learning from demonstration (LfD) [21], learning using unstructured data sources such as the Internet [26], and sharing capabilities using the "App Store" model [32].

We propose instead the usage of an end-user authoring environment called ROS Commander, or Robot Operating System Commander (ROSCo), to enable end-users and not just roboticists to create, modify, adapt, and deploy robot behaviors. With ROSCo, non-roboticists can build behaviors represented as a Hierarchical Finite State Machine (HFSM) by assembling reusable parameterizable building blocks. By using interoperable modules, our system allows the generation of more complex capabilities, incorporating tactile, and vision-based perception as well as arm motion, and navigation planners alongside feedback driven motions. As creating perceptual modules is a process that often requires labeling large quantities of data and can be a labor-intensive process, we also complement human effort with robots learning autonomously. This enables the robot to learn to perceive by interacting with the environment on its own instead of requiring constant supervision, enabling the creation of behaviors that have a higher degrees of autonomy with less involvement.

Since interactions between component modules, the robot, and the robot's environment are complex, our system uses an iterative construction process where users cyclically test behaviors with a real robot and environment, changing the behavior's parameters and structure as needed with each iteration. Within home environments where accurate simulations of objects are often not available, being able to observe outcomes during the authoring process is important in creating functional behaviors. Compared with more rigid frameworks that primarily use one type of method, such as learning [80, 115] or planning [73, 27] to generate robot capabilities, ours enable the construction of behaviors which integrate heterogeneous components that more closely approximate those used in practice [98, 138, 42].

With the current implementation of ROSCo, we primarily address a class of robot behaviors that operate mechanisms not often moved in homes such as refrigerators, doors, and drawers. As they are ubiquitous and important for operation in homes, being able to manipulate such mechanisms can be instrumental in advancing household robotics. Due to their approximately stationary nature and the wide-spread availability of off-the-shelf SLAM (Simultaneous Localization And Mapping) algorithms, pose localization can be greatly simplified by associating manipulable mechanisms to specific areas in maps made of the home. With ROSCo, we demonstrate a variety of behaviors operating this class of mechanisms including those that flip light switches, press rocker switches, open doors, pull drawers, and open refrigerators. We demonstrate that behaviors involving human robot interactions such as handing over objects are possible with our implementation.

ROSCo also allows behaviors to beh shared and deployed to new locations after construction. When deploying behaviors, users show the robot where to apply behaviors relative to fiducial markers (ARToolKit Tags) which allow the robot to quickly become operational in new environments. This behavior reuse enables the potential construction of large databases of behaviors to address the issue of needing qualitatively different behaviors for different tasks and variations on those tasks. Additionally, it enables other individuals with similar robots to download and use pre-constructed behaviors instead of having to construct their own.

In the following sections, we discuss a number of unknowns and difficulties in enabling end-users to author behaviors using software environments, then we discuss issues surrounding autonomous learning.

1.3.1 End-Users Constructing Behaviors with Reusable Modules

While there are similar systems to ROSCo for building behaviors to control generic robots [2], mobile robots [22], space manipulators [62], and video game characters [3, 13], interfaces for constructing behaviors have not been used in home mobile manipulation domains. In comparison to other domains, state-of-the-art mobile manipulation systems use a wide variety of different methods including behavior-based robotics, planning, learning-from-demonstration, force-based controls, as well as an equally diverse array of perception algorithms.

There are many unanswered questions in the setting of mobile manipulation in homes. For example, it is not clear if an authoring environment, no matter how well constructed, will be able to reliably generate functional behaviors. Furthermore, there is little known about which algorithms work best in such an ecosystem, how they will contribute to task success, and how such components can be structured to work together in a general manner. On the interaction end, the basic issue of whether users can learn to utilize these algorithms to accomplish tasks without needing a deep knowledge of robotics is also unresolved.

We attempt to address a number of these fundamental issues in the following chapters. First, we show a prototype system and a set of functional modules that can be used to create mobile manipulation behaviors for household mechanisms. Then we demonstrate with experiments that ROSCo–and perhaps other systems like it–can generate behaviors attaining success rates between 80% and 100%, levels that are comparable to current state-of-the-art manually programmed behaviors [98, 92].

In Chapter 3, we alleviate concerns about whether end-users will be able to use ROSCo to create behaviors without first having to devote large amounts of time studying robotics concepts. In a user study with 11 participants, non-roboticists recruited from Georgia Tech's campus, we show that after a short training period, all participants were able to construct functional mobile manipulation behaviors. Through a longitudinal study with Henry Evans, a person with quadriplegia, in his home, we show that ROSCo generated behaviors can be used during teleoperation to perform common household tasks, and that Henry can also use ROSCo to construct behaviors that can be used to automate his home and aid during teleoperation.

1.3.2 Augmenting Expert Skills with Autonomous Learning

While end-users often are able to use their domain knowledge to parameterize many modular building blocks in ROSCo, this becomes infeasible with the perceptual classifiers often needed for autonomous operation due to the large number of parameters involved. In our work, we will demonstrate that integrating a fully autonomous perception learning system, where the robot gathers its own data for training, enables the authoring environment to generate more autonomous and robust behaviors.

We introduce the concept of complementary behaviors and show that robots equipped with a pair of such behaviors are capable of learning autonomously. In this arrangement, two behaviors are complementary to each other if each reverses the effects that the other has on the world. For example, in order to learn how to open a drawer the robot should also have access to a complementary behavior that closes that drawer. By learning the perceptual modules of both behaviors at once, we enable the robot learn without relying on outside interventions during the process.

Without this complementary assumption, however, autonomous learning is usually of limited use for mobile manipulation systems as interaction with the world often means changing the physical state of objects in the world. These changes would then preclude the robot from learning in the same scenario again. Indeed, this is the limitation with prior robot learning projects. Previous work has opted to modify the environment to accommodate learning [115], create special robot playpens [135], learn with tasks where the environment returns itself to a nominal state [136, 79], or manually intervene in the learning process [104].

Our system learns to associate visual features at 3D locations on objects that are predictive of future success for a given pair of complementary behaviors. At each iteration of our procedure the robot picks a new candidate location at which to capture labeled data. By doing so, the resulting classifier learns not just to identify 3D locations such as those returned by an object recognition algorithm, but one that is tailor tuned to how the behavior interacts with the given target object to achieve task success. However, as each manipulation attempt is costly in terms of time and risk to the robot and environment, we use active learning to select potential contact points to minimize the number of data points needed to build functional classifiers. In the following chapters, we demonstrate our system learning first with manually programmed behaviors in Chapter 5 then with behaviors created by participants from a user study in Chapter 6.

1.4 Robots and Environments

For all our experiments we used a Willow Garage PR2 robot named GATSBII shown in Figure 1. The PR2 robot is a two-armed human-like mobile manipulator built by Willow Garage as a platform for research and development. As an autonomous mobile manipulator, the PR2 is also equipped with a suite of sensors including: two Hokuyo laser scanners at the base and on a tilting platform, two stereo pairs on the head, a high resolution 5-megapixels Prosilica camera, a pattern projector, and tactile sensors on its gripper tips. Our PR2 has been retrofitted with a Microsoft Kinect sensor on

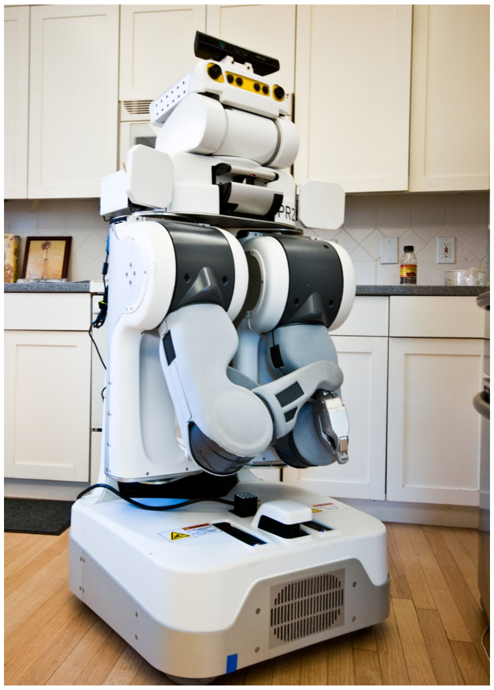


Figure 1: GATSBII, the PR2 used in this work. Photo courtesy of Jerome Choo.

the head. Although there are RFID antennas on the shoulders, they were not used in this work.

Additionally, our work was performed in the Georgia Tech Aware Home, the Healthcare Robotics Lab, and Henry Evan's home. The Aware Home is a three-story, 5040-square-foot home used as a test facility for household technologies. Unlike many other facilities used for this purpose, the Aware Home is a full-fledged house, not a simulated living space, and is complete with two kitchens, two bathrooms, two dining rooms, two offices, four bedrooms, and a basement.

1.5 Organization of this Thesis

We organize the rest of this thesis as follows.

In Chapter 2 we present ROS Commander (or ROSCo), a new authoring environment designed for mobile manipulation applications where users can create, modify, and deploy robot behaviors represented as Hierarchichal Finite State Machines (HFSMs). We present experiment findings showing that the behaviors generated by ROSCo, which integrates both mobility and manipulation, are robust and can be competitive with manually programmed robot capabilities.

Work in Chapter 3 answers a fundamental question about whether generally computer literate users can, in a short amount of time, learn to author mobile manipulation behaviors using interfaces like ROSCo. We show in a user study that all 11 participants were able to learn to create novel behaviors where the robot navigates to a location and interacts with the environment. Based on these pilot studies, we also identify issues related to creating authoring environments for mobile manipulation applications.

With Chapter 4, we investigate usage of ROSCo in an assistive mobile manipulation context with Henry Evans, an individual with quadriplegia, as our participant. We demonstrate ROSCo behaviors used in a shared autonomy teleoperation task where the user activates behaviors that perform frequently repeated and time consuming actions. We show that Henry can also use ROSCo to construct mobile manipulation behaviors in a similar manner as able-bodied users, and we also show an application where Henry uses ROSCo to create behaviors to compose a comedy skit.

In Chapter 5, we introduce how using autonomous robot learning can complement human authoring of behaviors by learning visual appearance image patches associated with successful execution of behaviors. We present an algorithm that takes in two manually crafted behaviors that can reverse each other's effects on the world and learn to perceive 3D points where those behaviors would be successful. We also show that integrating the same algorithm at runtime enables the robot to retry intelligently by learning to avoid past mistakes.

Chapter 6 demonstrates autonomous learning as a tool embedded in the ROSCo behavior authoring system. We demonstrate our system learning using not the author's manually coded behaviors, but using ROSCo behaviors created by user study participants in Chapter 3. Results demonstrate that the classifiers learned also reflect the particular nuances of each user constructed behavior.

Finally, we conclude in Chapter 7 with contributions made and discussions about future research directions with authoring environments for mobile manipulation and autonomous robotics learning as a complimentary tool.

CHAPTER II

ROS COMMANDER: AN EXPERT TOOL FOR CREATING BEHAVIORS ENCODED AS HFSMS

Creating general-purpose robots capable of performing a wide variety of tasks in home environments remains a significant challenge. First, the variety of tasks that people would want a general-purpose home robot to perform is potentially vast. As evidenced by the number of applications available for personal computers and smart phones, people can use general-purpose consumer technology in diverse and unexpected ways. Second, real homes exhibit large variation that can degrade robot performance. Even something as prosaic as drawer handles can vary widely in size, shape, materials, and appearance, all of which can impact a robot's ability to operate drawers.

How can home robots scale to handle such large task and situation variability? Proposed solutions to this challenge include approaches as diverse as formal task planning, logic-based reasoning, learning by demonstration (kinesthetic and by observation), learning using unstructured data sources such as the Internet, or sharing capabilities using the "App Store" model [32]. In contrast, our own work is inspired by the success of expert software tools that simplify the creation of complex artifacts (e.g., tools for photo editing and computer-aided design (CAD)), and by the observation that Hierarchical Finite State Machines (HFSMs) underlie many state-of-the-art demonstrations of robot capabilities.

In this chapter, we introduce a system called ROS (Robot Operating System) Commander (available at [7]) or ROSCo, which allows the rapid creation and usage of new behaviors structured as HFSMs. First, it allows expert users to construct robot behaviors via a graphical user interface, using a collection of generic, parameterized building blocks created specifically for mobile manipulation applications. Second, users who do not wish to construct behaviors can automate their environments through a separate interface that allows the association of environment parts with pre-constructed behaviors. In this chapter, we use our system to provide evidence for the following claims:

- HFSMs constructed from low-level building blocks, and without direct access to code, are

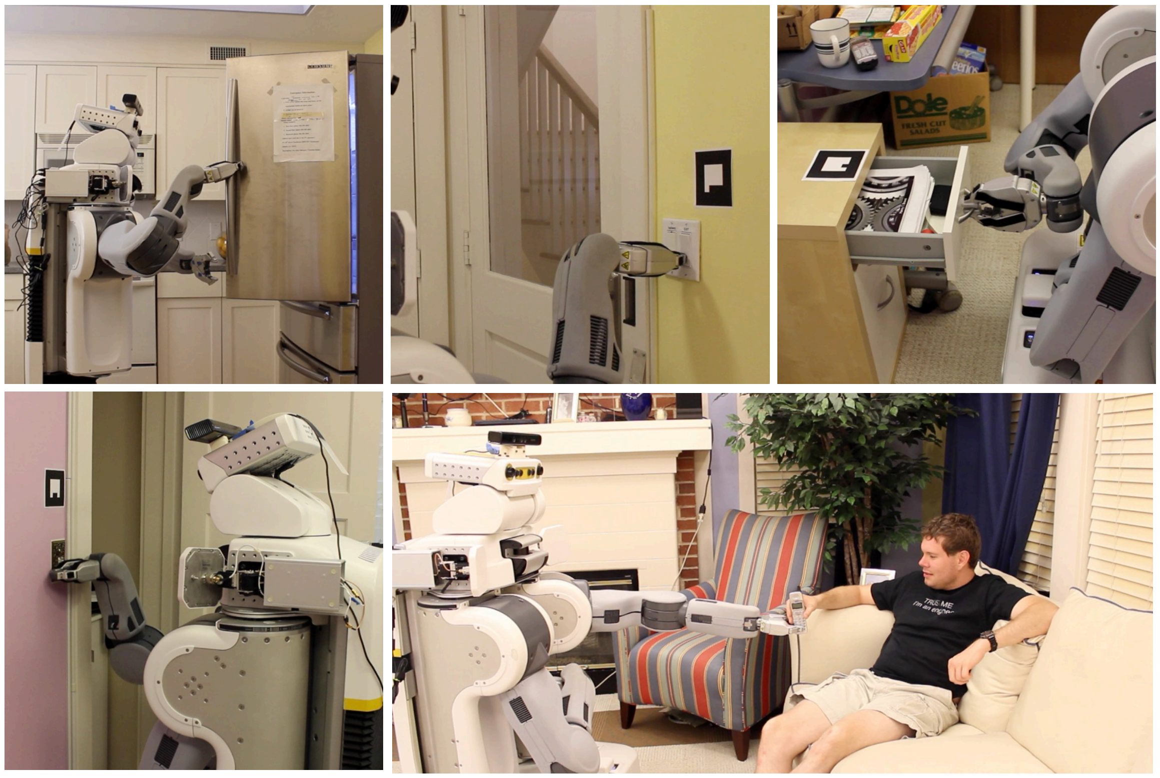


Figure 2: Examples of robot behaviors constructed using the ROS Commander system and encapsulated as HFSMs: opening a fridge, unlocking with a button, opening a drawer, flipping a light switch, handing off an object.

powerful enough to encapsulate many useful behaviors in home environments;

- By integrating perceptual (visual or tactile) cues, HFSMs can be executed with a high success rate and repeatability;
- HFSMs can be adapted to changes in the environment or new environments in many cases by using a combination of perceptual cues and simple adjustments from the user.

The interface to our system, shown in Figure 3, allows users to create HFSMs by selecting appropriate modules, providing them with parameters suitable for the task to create states, and connecting those states appropriately. The set of low-level building blocks includes modules for perception (e.g., detection of faces, fiducial markers, or tactile events), navigation, manipulation (e.g., arm motion planning, trajectory following, or Cartesian end-effector control) and others (e.g., torso movement or head movement).

In our application, we use an HFSM representation for a number of reasons. Compared to planners using classical STRIPS-like representations that can generate the equivalent of HFSMs at each invocation, static HFSMs, exhibiting less variance in executing paths, can be more predictable. This characteristic could also be the reason for the popularity of HFSMs in video and movie character design environments [3, 9]. Furthermore, as capabilities generated by users are potentially open-ended, behavior-based HFSMs have an advantage over planning-centric methods as they do not need physics-based models to predict the effects of different possible actions, such as with existing work in grasping [44], pushing [54], and making pancakes [84].

The work presented here is motivated by the long-term vision of a powerful yet accessible tool for building and adapting robot behaviors. As with packages such as Photoshop or Final Cut Pro, used for processing still images and videos, specialized tools can go far in empowering both roboticist and non-roboticist users in the creation of robust and versatile behaviors. End-users are the ultimate domain experts; a "Photoshop for Robotics" [88] tool could allow them to use their intimate environment knowledge for creating and customizing appropriate robot behaviors. Leveraging this potential could accelerate the deployment of mobile manipulators in unstructured home settings, and enable them to perform a wide variety of tasks.

We present experiments showing that by combining our set of generic modules in HFSM representations, our system can generate a wide variety of autonomous and robust mobile manipulation behaviors. We present trials where we execute each of 6 behaviors 10 times, attaining 80% to 100% success rates. Currently, this is comparable to state-of-the-art systems [98, 92]. Our behaviors include opening a refrigerator with two arms, turning on a light switch, unlocking a door with a push switch, opening a drawer, and handing objects over to a person.

2.1 Related Work

Enabling general purpose robots to perform a wide variety of tasks in complex human environments has been a long-standing challenge within robotics, with many proposed approaches. We will review recent proposed solutions that use primarily task-based planning or task learning, as well as those that use specialized interfaces.

Task-based planning methods attempt to move beyond step-by-step instructions by computing appropriate action sequences based on task level goals such as "Pick up the cup and place it in the dishwasher" along with detailed action representations. Such systems allow robots to perform different types or variations of tasks by relying on the flexibility of the planner and its available action set [59, 89, 97, 28, 38, 19]. Planning approaches for manipulation go as far back as the Handey system [89] for flexible pick-and-place tasks. More recently, the CRAM toolbox [27] provides a more heterogeneous collection of tools for knowledge-based representation and task-level reasoning. In comparison, ROSCo behaviors are created for cases outside the domains of typical task planners, where there are not good action or object models to plan with; this is a common situation for most home environments.

In contrast, learning approaches train generalized models with data from a wide variety of sources, in order to mitigate knowledge-engineering problems often encountered with task planning [101, 115, 79, 117, 37, 137]. A number of approaches [115, 79, 117] use a combination of reinforcement learning and Dynamic Movement Primitives [65] to learn demonstrated dynamic motions such as playing pool [115], playing with a ball-in-cup toy [79], and hitting a ball [117]. Cakmak and Thomaz [37] investigate guidelines for designing asking behaviors for improving robot skill learning from human kinesthetic demonstrations. Finally, Tenorth et al. [137] investigate using the web as an information source for robot learning. While the focus of ROSCo is not on learning, and is instead on integrating heterogeneous, generally-useful, generic modules, we believe that many of the existing ideas from skill learning could be integrated as smarter, more general ROSCo building blocks.

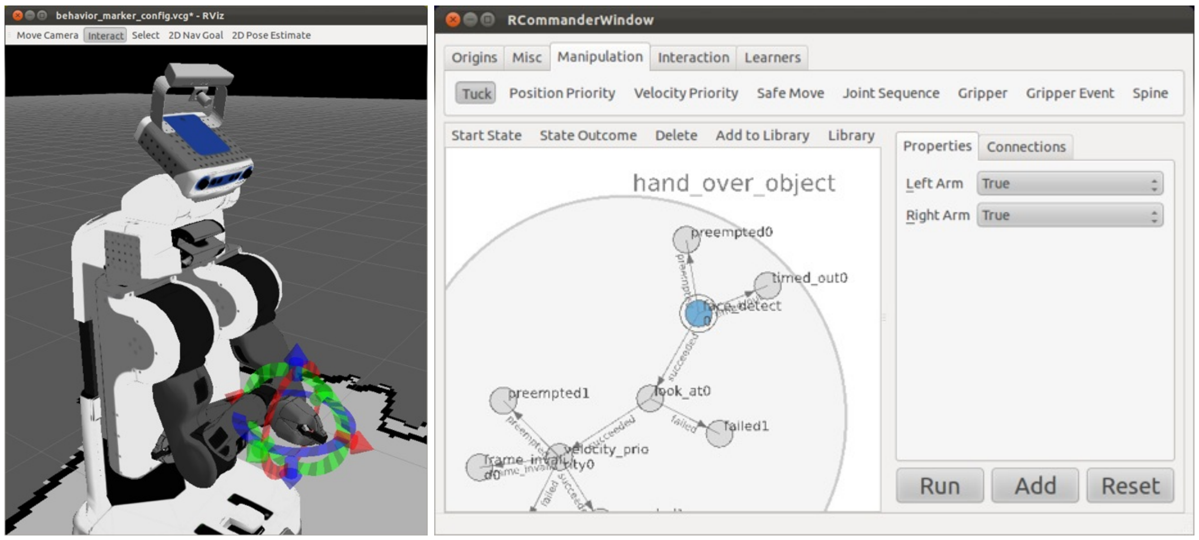
Perhaps more related to ROSCo are methods that use visual programming, where programming constructs are represented as visual symbols that can be manipulated (for an overview see [10, 34]). Visual representations are attractive as they can potentially make programs more easily understandable. However, such advantages can disappear when the visualization is too fine-grained, displaying too many irrelevant details. Block-based programming systems, which define a variety of procedural programming constructs and coordination primitives, such as RoboLab [120] (used as a basis for the popular Lego Mindstorms kit) and Microsoft Robotics Studio's Visual Programing Language [11], designed primarily to acclimate nonprogrammers to programming concepts, can suffer from this symbol explosion problem when used to construct larger programs. Flow-based methods, interfaces that use blocks to manipulate the flow of data through a graph, can be demonstrated by systems such as RobotFlow [6], Matlab's LabVIEW, and Ecto [1]. Visual programming is more appropriate in this domain as information processing graphs tend to not grow as large as graphs representing programs.

Visual programming has also found success in a different class of interfaces, designed specifically to accelerate the process of behavior creation in robots. Missionlab [22] was designed for creating new behaviors for reactive mobile robot navigation. The Robonaut 2 Command and Control Interface [62] is a more recent example, used for mobile manipulation with the Robonaut robotics platform. ABB's RobotStudio, created as a system for designing manipulator behavior in structured factory environments, uses tools that define end-effector paths based on CAD models of components. With Gostai Studio [2], the aim is more towards creating a tool that can be used for editing the structure of HFSMs, with states manually coded by users. While ROS Commander (ROSCo) similarly edits HFSMs, users provide parameters to individual states instead of having to write code.

In the video game and movie industries, state machines (FSMs), HFSMs, and behavior trees (which are more restrictive but believed to be more intuitive) [55] are used routinely for character behavior generation with great success. Methods used by Massive software [5] are responsible for generating the behavior of large crowds of virtual agents that have been used in many major films. The Kismet [4] state machine editor was used for designing many of the agents in the game Unreal. Xait [13], Havok [3], and Unity3D [9] are more recent commercial offerings that also enable authoring and customization of agent behavior. As game developers have also found FSMs to be a good trade-off between reasoning capability and simplicity, the overall structure of robot programs and video game agents can appear to be quite similar. However, ROSCo behaviors require different fundamental building blocks, as robots must, in addition to difficulties faced by artificial agents, face issues such as unstructured environments, noisy sensors, and faulty actuation.

2.2 ROS Commander

The goal of ROSCo is not just to provide a new, open-source interface for creating HFSMs. It is also to examine a broader research question: that of finding sets of general, parameterizable states that



Left Arm True
Right Arm True

Figure 3: The HFSM editor that we propose on the left side is paired with use of ROS's rviz with special interactive markers (such as 6-DOF rings-and-arrows controls around grippers) for posing and controlling the PR2 displayed. Combined, we believe that these interfaces will allow quick iterative development of novel parameterized abilities for the PR2.

can span a large variety of tasks that are useful in home environments. In contrast to task planning or learning-based approaches (who deal with new tasks through more capable planners, richer datasets, or smarter learning algorithms), ROS Commander depends on general building blocks and software tools guided by skilled human users to generate new varieties of behaviors.

ROSCo is both a user interface for building robot behaviors and also a tool for exploring the versatility and robustness of HFSMs constructed from relatively simple and accessible building blocks. In our framework, heterogeneous software modules can use wildly different perceptual, planning, and actuation spaces, and can thus encompass a wide range of approaches without having to reformulate them to fit into restrictive overarching architectures.

In this section, we introduce the user-facing component of ROS Commander and describe the general process of creating a new behavior, with a number of detailed examples. In the following sections we will explore the robustness of the resulting behaviors, as well as methods for deploying them into new environments.

2.2.1 Robot Behavior Creation: Basic Concepts

Our user interface for authoring behaviors is shown in Figure 3, as used on a PR2 robot. The ROSCo interface (on the right) is used for visually editing the current behavior's state machine and the parameters of its states. It is commonly used in conjunction with `Rviz` (shown on the left), a standard component of ROS used for visualizing sensor data and the state of the robot, as well as for controlling the robot through interactive markers [44].

Visually, the ROSCo interface for editing state machines is divided into three primary areas (right side of Figure 3). The large main panel on the bottom left displays nodes (states) and edges (transitions) in the HFSM currently being constructed. At the top is a tabbed grid with buttons for selecting first the category (such as Manipulation, Interaction, Learning, etc.) and then the individual desired building block (such as Speak, Move Head, etc.) for adding to the HFSM. Finally, on the right is a panel of properties that displays the parameters of the currently selected node and its connectivity, as well as controls for executing nodes and saving changes to them.

Typically, to build robot behaviors, users first place the robot in front of a person, object or mechanism that it will need to interact with, such as a door, drawer, or light switch. The next step is to select a tool that creates the desired state machine element in ROS Commander, specify that element's parameters, and add it to the current state machine. In addition, for tools where the robot's motions are more easily specified through demonstrations, users can demonstrate the movement using teleoperation tools in Rviz, and use the state of the robot at various points during the demonstration to set the parameters for the selected tool. This process of selecting a tool, specifying parameters for it, and testing the results on the physical robot is then repeated until completion of the behavior.

For example, to create a state that replays a joint trajectory, users first select the "Joint Trajectory" button in ROS Commander, then pose the robot as desired, clicking "Record" at key poses. After recording a set of joint space poses, they might adjust the timing between key poses or execute motions to check their performance on the robot. When satisfied, they can add the trajectory to the state machine as a state. Similarly, the same testing process can be applied to the state machine wholesale to check interactions between the various states and the environment.

For saving and running the final result of behavior creation, ROS Commander uses the SMACH (State MACHines) library [32]. ROS Commander state machines are compiled into Python SMACH state machines, which allows direct execution on the robot.

2.2.2 General Design Principles

A few important guiding principles were used in the design of ROSCo. First, that behaviors should be easy to modify and test while being constructed. In a similar manner to debugging in most programming environments, we have made it possible to step through the loaded behavior, executing one node at a time to observe its effects on the robot and on the objects manipulated. We believe that this ability is crucial in home robotics, as the environment can contain many unknowns. For example, it might not be apparent to users why a one-armed door opening behavior would not apply sufficient force for opening a magnetically sealed refrigerator door until execution time; by stepping through the behavior, the problem becomes apparent, and can be fixed through additions at the problematic phase of the behavior.

Second, as with image processing software such as Photoshop, tools should span varying levels of built-in intelligence to provide both greater autonomous assistance (when possible) and also the ability to accomplish a wider variety of arbitrary tasks in low-level, manual fashion. The tools provided by ROSCo range from those that create simplistic joint movement states, to those that use Cartesian controls, on up to those that provide autonomous motion planning for the robot's arms and base, with dynamically-updated obstacle maps. This approach makes it more likely that behaviors can be created for arbitrary tasks, albeit with less generality, even when capabilities with greater autonomy fail or are not available. For instance: when creating a behavior to hand over an object to a person, a more-intelligent-seeming behavior could use face detection to offer the object at an appropriate position relative to the detected face. However, if the detector is known to not work with face side views, it would still be possible to use a static joint trajectory motion as a backup strategy.

Third, we use the PR2 arm's physical compliance so that motions generated can be more tolerant errors. For example, by using compliance an arcing motion used for opening cabinet doors of one particular size can be expected to have a reasonable chance of opening cabinets with slightly larger or smaller doors.

2.2.3 Performance of Behavior Building Blocks

Modules in ROSCo encapsulate existing software packages in ROS that are often used in mobile manipulation settings. ROSCo then provides interfaces with appropriate visualizations for users to tune and parameterize the respective algorithms. As such, the performance characteristics of many component modules in ROSCo have been well studied. For example, the autonomous navigation module has been tested in empirical trials where the PR2 navigated for 26.2 miles [93] in a real office environment. With manipulation, the Jacobian transpose controller used for Cartesian task space control was compared to other schemes in Nakanishi, Mistry and Schaal [107].

Building upon such proven and frequently used components increases the likelihood that behaviors will execute with high reliability and that the modules themselves will be useful in diverse scenarios. Additionally, existing literature can serve as documentation to help to guide their usage in new applications.

2.2.4 Adding Perceptual Data

For many desired robot behaviors, robustness and generality require closing the loop by processing sensory data and extracting information needed to complete the task. In ROS Commander, states that process perceptual data either control execution of other states, or create 3D frames that serve as references for other states. These two modes greatly increase the range and flexibility of behaviors that can be created using ROSCo.

For example, defining a Cartesian movement with respect to a frame produced by the face detection module enables behaviors such as handing over an object in the robot's gripper to an appropriate pose for hand-off, relative to the person whose face was detected. When users execute the behavior, the desired motion of the robot's grippers is expressed with respect to the coordinate system defined by the face, allowing the behavior to generalize across changes in face positions with respect to the robot.

The AR ToolKit [74] module works similarly but also allows users to define custom reference frames, which we refer to as *task frames* (a concept introduced by Ballard in [23]). These frames are defined relative to detected AR Tag markers, and are used as frames for defining task-relative motions. For example, in a behavior where the robot navigates to a drawer and opens it, we store the position of the robot's base and also the gripper poses required for opening the drawer as poses relative to a frame defined at the center of the drawer (bottom left panel of Figure 5). The system supports using the task frame's last known position even when there is not a live estimate of the tag's position. This property enables mobile manipulation behaviors that use the strategy of first coarsely moving to a known map location, then adjusting arm trajectories using finer pose estimates once the pose of a known tag has been estimated (a strategy we have used successfully in previous work [67]).

In this work we have used AR Tags to illustrate the robustness and generality of robot behaviors once affixed to a perceptual cue in the environment. AR Tags and other fiducial markers are relatively non-intrusive, and can provide many benefits in the short term [108]. In the longer term, we envision them being gradually replaced by instances of state-of-the-art object and place recognition algorithms that do not require any modification of the user's environment.

Finally, for tactile perception where the output is the presence or absence of an event, ROSCo creates a state that acts as a container that can stop the execution of states or state machines inside it. For example, to generate a guarded (move-until-contact) motion, users can create an arm movement state inside a tactile perception (event detection) state. When this executes, the movement will be stopped as soon as the robot's tactile sensors make contact.

2.2.5 Example ROS Commander Behaviors in Detail

We now illustrate the anatomy of three typical ROSCo behaviors (Figure 4) to show how different states produced by ROSCo modules interact.

The first behavior navigates to a light switch and turns on the lights. It starts out by placing the robot in a canonical pose for navigation: first straightening the head, with the state move_head_ang1, and then tucking the arms, state tuck0. This is followed by a coarse navigation step that uses the robot's navigation stack (navigate0), which can autonomously navigate the robot to a chosen goal, planning paths that avoid obstacles along the way. While the navigation stack gets the robot close to the desired base pose, the resulting pose errors are too large for manipulation. Thus, we rely on a more primitive state (navigate_refined0) that refines the base position, moving directly to the goal.

After positioning the robot, this behavior puts the robot in a pose appropriate for manipulation by untucking the arms (tuck1), then lifting the PR2's upper body to the height needed by moving the spine (move_spine1). Finally, a joint trajectory command places the robot's arms in an appropriate start position for operating the light switch (joint_sequence0).

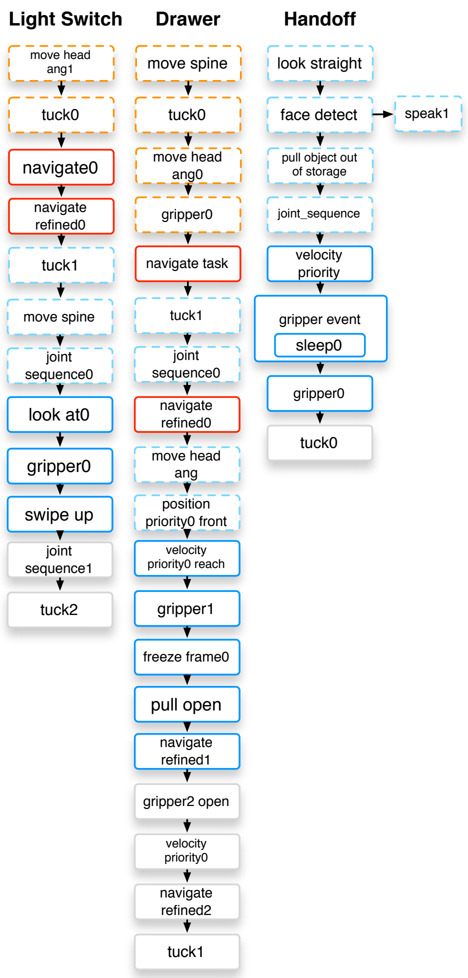


Figure 4: HFSMs for three prototypical behaviors with states colored based on which phase of the task they represent. Dotted orange boxes are states that put the robot into a pre-navigation pose. Red states are navigation states. Dotted blue states are pre-manipulation states. Blue states perform the manipulation action for that behavior. Finally, grey states place the robot back in its starting pose.

Now readied for manipulation, the behavior points the Kinect at the last known position of the AR Tag (look_at0) to get a better estimate of its pose. The detector for the AR Tag runs concurrently as an independent ROS process on the robot; pointing the Kinect at the tag increases the likelihood that the sensor obtains a good view, and thus a good estimate of the tag's position. This behavior then closes the gripper (gripper0), reaches forward, and swipes up, flipping the light switch (swipe_up) with a Cartesian movement that uses the task frame estimated by the AR ToolKit detector.

In the last stage, the robot puts itself back into a canonical pose with a joint trajectory movement that pulls back the gripper (joint_sequence1) and tucks its arms back in (tuck2).

We have observed that most mobile manipulation behaviors we have designed for the PR2 have a similar structure. Behaviors often start with placing the robot into canonical poses, then navigating. After getting close to the object to manipulate, the behaviors put the robot in a pre-manipulation pose conducive to that task, then perform the manipulation action. Finally, such behaviors often end with placing the robot back in its starting state. The drawer opening behavior shown in Figure 4 also has this canonical structure.

With the object hand-off behavior in Figure 4, we have a slightly different structure, with no navigation states present and no need for putting the robot into a canonical pose for navigation. The behavior starts out with the robot looking straight ahead, then trying to detect a face. If it does not see a face, the behavior transitions to a text-to-speech node to tell users that it cannot see their face and will not be handing the object over. In case the robot does see a face, the state machine later on transitions into a gripper_event node that encapsulates a sleep node, resulting in a state that sleeps for a specified amount of time (60 seconds) unless it detects an event on the robot's gripper, whereupon it stops the sleep and releases the object.

2.3 Deploying Robots in New Homes: Associating Behaviors with Locations

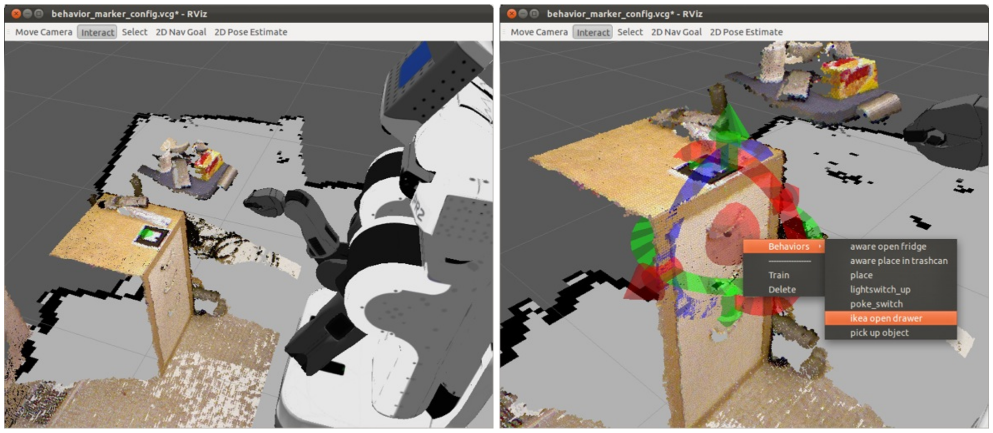


*Figure 5: Associating behaviors with new locations. Using as a reference point an AR ToolKit tag placed near the drawer (**Left**), the user can define a task frame centered on the handle (**Right**), and then start the appropriate behavior.*

Once a behavior is constructed, the user can save it onto the robot for reuse. We have already described how the addition of execution reference frames obtained from perception modules can make execution robust to variations in the exact pose of the robot relative to the manipulated mechanism.

The same approach can be used to generalize behaviors to different instances of a mechanism, increasing applicability to different locations inside a user's home, or even in a different home.

Once a user has a robot behavior available, either previously constructed by themselves or simply downloaded from a library, they can choose to apply it to various instances of a mechanism. For example, to use a pre-constructed drawer opening behavior, users would affix an AR tag close to the drawer, and, using a simple GUI, select the location of the task frame relative to the AR tag, thus defining how the pre-constructed behavior should be executed relative to the tag. In the case of drawer opening, this essentially consists of showing the robot the offset between the tag and the real drawer handle. The process is illustrated in Figure 5.

Given perceptual cues attached to relevant parts of the environment, a user can select a set of behaviors to be executed in the task frames provided near those cues. Typically, the first run of a newly deployed behavior can fail due to incorrect positioning of the reference frame or errors from various other sources. In this case, users would only need to modify the offset between the perceptual cue and the desired task frame using the visual interface, and try again. As mentioned, we envision state-of-the-art object recognition algorithms gradually replacing cues such as AR tags as they become more robust in unstructured environments.

This interface is inspired by the process of a user (employer) giving the new robot (employee) a tour of the house, pointing out relevant parts of the home and the tasks to be performed. It is not as simple for the user as a fully autonomous robot that can expertly operate in a completely new environment out-of-the-box. However, as autonomous systems that can handle thousands or even millions of different homes out-of-the-box still pose significant research questions, we believe this approach can be a practical solution for the deployment of robots into new homes on a much shorter time frame.

2.4 Robot Experiments

In this section, we evaluate the performance of several example behaviors created with ROSCo. Since we began development in the beginning of summer 2011, we have engaged potential users to better address their needs. Examples of use include a person with quadriplegia generating gestures for the PR2, and robot developers enabling a Turtlebot (small mobile robot) to pick up a block with

Table 1: Success rates from all trials.

Task	Successes	Failures
Drawer Opening	10	0
Refrigerator Opening	9	1
Unlocking a Door	9	1
Handing Over an Object	9	1
Turning On the Lights 1	10	0
Turning On the Lights 2	8	2
Turning On the Lights 3	9	1

Table 2: State types used by each behavior.

Node Type	Light S.	Drawer	Fridge	Hand-off	Total
Velocity Priority	1	3	4	1	9
Face Detect				1	1
Freeze Frame		1	1		2
Gripper	1	3	5	1	10
Gripper Event				1	1
Joint Sequence	2	1	2	2	7
Look At	1				1
Move Head	1	2	2	1	6
Move Spine	1	1	1		3
Navigate	1	1	1		3
Navigate Refined	1	3	2		6
Position Priority		1			1
Tuck	3	3	2	1	9

Figure 6: The Georgia Tech Aware Home

a custom arm. For the following experiments, however, the behaviors were created by an expert user (the primary developer of the system and first author).

We tested the behaviors constructed with our system through experiments on a PR2 robot: a mobile manipulator produced by Willow Garage with two compliant arms and an omnidirectional base, as well as a large suite of sensors. We used a head-mounted Kinect sensor for estimating the pose of AR ToolKit tags. Our experiments took place at the Georgia Tech Aware Home, a three-story, 5040-square-foot home used as a test facility for household technologies. Unlike many other facilities used for this purpose, the Aware Home is a full-fledged house, not a simulated living space, and is complete with two kitchens, two bathrooms, two dining rooms, two offices, four bedrooms, and a basement.

The ROS Commander-generated behaviors that we tested include two-armed refrigerator opening (as one is not strong enough), drawer opening, unlocking a door with a push button, turning on the lights with a light switch, and handing an object to a person (Figure 2). We selected these behaviors because we believe they are both representative of a large class of behaviors that would be useful in the home, and also relatively challenging, since they involve both navigation and manipulation. Each of these behaviors, with the exception of object hand-off, works in a similar manner to the mobile manipulation behaviors described in Section 2.2.5: there is a coarse, autonomous navigation phase with a goal defined by that behavior's task frame, followed by a refined navigation step, untucking of the arms, manipulation, and finally tucking again. To test giving an object in the gripper to a person, we positioned the robot in front of a Georgia Tech student seated on a couch in the living room and repeatedly handed him different objects. Success for object hand-off is defined as the behavior executing through its nominal path of states and the person receiving the object. We show a table summarizing the frequency which different types of states are used in Table 2 (with state names explained on our website [7]).

Each trial of a mobile manipulation behavior begins with the robot being driven manually to a random location in the space that contains the mechanism it needs to operate on. For the refrigerator and push button, this space includes the hallway, kitchen, dining, and living room, as those rooms are all one contiguous space. For the light switches and drawer, this space includes the room that the mechanism is in. After being driven to a random location, we select the behavior being tested on a web interface and record the results.

We performed two sets of trials. First, we tested the robustness of our behaviors by executing drawer opening, refrigerator opening, unlocking a door, and handing over an object each 10 times. Behaviors in this set of test attained a 90% to 100% success rate. Next, we tested the ability of our behaviors to generalize to different mechanisms by testing the light switch behavior 10 times each on three different switches in the Aware Home. With the switches, we had an 80% to 100% success rate. We present the results of these trials in Table 1.

Overall, we had five failures. In one trial of refrigerator opening, the robot's arms collided with the door while untucking, due to a bad localization estimate. In another trial, the robot failed to push on the correct part of the switch to unlock the door. We believe that this is due to small tracking errors in the Cartesian controllers; such errors could be mitigated by improved controllers, or by visually tracking and correcting the gripper pose. With the object hand-off behavior, the failure was due to the ROS process for tucking the arms freezing at the end of the behavior, which did not actually affect the robot's execution of the behavior.

In the second set of trials on light switches, we had two failures due to the navigation package's planner not finding a path, which happened because of incorrect sensor data reporting nonexistent obstacles at the robot's desired goal location. The last light switch failure was due to the gripper getting stuck while the robot attempted to slide it along the wall. Failures such as these could be mitigated by adding autonomous re-tries to the state machine.

These trials show the performance of basic state machines designed for each task, without any attempts at adding additional logic and branches for increasing robustness. Because each behavior is a hierarchical state machine, autonomous re-tries and even calls to states that ask for human-in-the-loop assistance to get robots un-stuck can be added to behaviors as needed.

2.4.1 Behaviors Showing the Generality of ROSCo

In its current form ROSCo is capable of generating a wide variety of behaviors. In the last section we showed that ROSCo can generate behaviors that accomplish their tasks reliably. We now demonstrate the generality of ROSCo by showing the diverse types of behaviors that we have been able to generate in Figure 7. We tested these behaviors in a number of different environments including: the Aware Home, Henry Evans's home, and Willow Garage's office. Henry Evans is a collaborator of ours with quadriplegia whom we will discuss more fully in later chapters. Willow Garage is the company that makes our robot, the PR2.

In the first row of Figure 7, the PR2 gives back rubs at the Aware Home with a behavior where the user leans back into its grippers which moves up and down rhythmically. Still at the Aware Home, the robot uses a hand fan placed into its gripper. Then it disposes of a piece of paper given by a person at Henry's home. In the fourth row, we show a grasping behavior with a tagged bottle at Willow Garage. In the last row, the PR2 places a bottle on a tagged table with another behavior at Henry's home.

With the back rub and hand fan behaviors, we show that there is potential for general purpose robots to emulate common household gadgets paralleling how smart phones have found use as leveling tools and flashlights. The grasping, trash disposal and bottle placement behaviors show that our framework is capable of operating with not only mechanisms but also objects. Using better object pose estimation and detection algorithms, it should be possible to employ ROSCo in object centric tasks such as tidying up a room.

We show in Figure 8 a door opening behavior with a door knob at Henry's home. As this particular knob is broken, needing to be rotated for a full turn to disengage, and door knobs are difficult to operate using a parallel gripper, our behavior had to re-grasp the knob mid-rotation to open it. In real home environments, idiosyncratic mechanisms such as this knob can be widespread and are difficult to anticipate. We believe that competing methods depending on accurate planning and analytical physical models of objects, such as this door, would likely fail in these scenarios. In these circumstances, having human guidance in the behavior construction process, such as with ROSCo, can be vital in generating functional robot capabilities.

2.5 Conclusion & Future Work

In this chapter we introduced ROS Commander (ROSCo), a system for building robot behaviors. A user can construct behaviors by combining parameterized states drawn from a set of building blocks. The capabilities of these building blocks cover aspects such as base navigation, arm navigation and manipulation, and head and torso movement. Behaviors constructed using ROSCo have the underlying structure of HFSMs, using the building blocks described above as individual states. The addition of perceptual cues to a behavior can greatly increase robustness: for example, by addressing variations in robot pose relative to the target at the start of execution. Examples from our system include modules creating states that generate reference frames for execution based on AR Tags in the environment, or modify execution based on tactile events.

We have used ROS Commander to build and test a variety of behaviors applicable in home tasks. These behaviors include: opening a fridge using two arms, opening a drawer, unlocking a door, flipping a light switch, handing an object off to a person, back rubbing, operating a hand fan, disposing of trash, grasping a bottle, and placing an object. All of these behaviors were tested in a realistic home environment, and the switch flipping behavior was also tested on multiple, slightly different switches. Overall, the results showed that the proposed set of building blocks can be combined in an HFSM framework that is *general* (can be used for many different task), *robust* (behaviors are executed with a high success rate) and *versatile* (behaviors can generalize to different instances of the target).

More advances can potentially result from the use of smarter building blocks with greater autonomy, such as those that can detect task failure, or through the sharing of preconstructed behaviors on the Internet. Given a variety of different behaviors for a task, it is likely that at least one behavior would work for any particular instance, with the caveat of possibly requiring minor parameter changes that can be performed by the end-user. If this is true, widespread usage of such tools can potentially enable an explosion in robot competency with household mechanisms.

While the current system was designed for operation by trained end-users, the behaviors demonstrated in this paper were all constructed by a roboticist. In the next chapter, we investigate how non-roboticists interact with our system, and which changes are needed to enable the creation of useful robot capabilities by end-users.

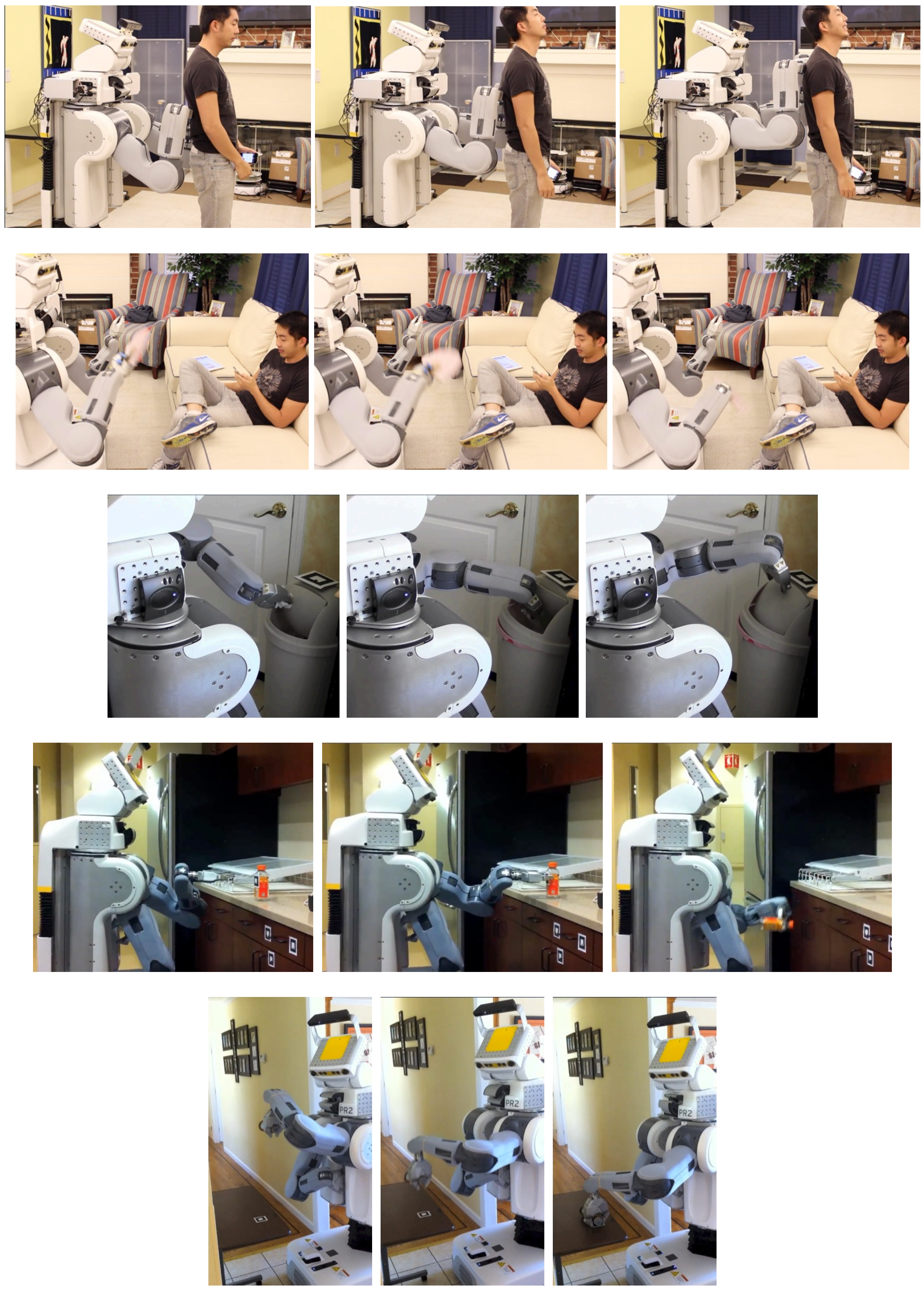


Figure 7: Behaviors created with ROSCo where the PR2 gives back rubs, uses a hand fan, disposes of a piece of paper, grasps an object, and places an object on a table.

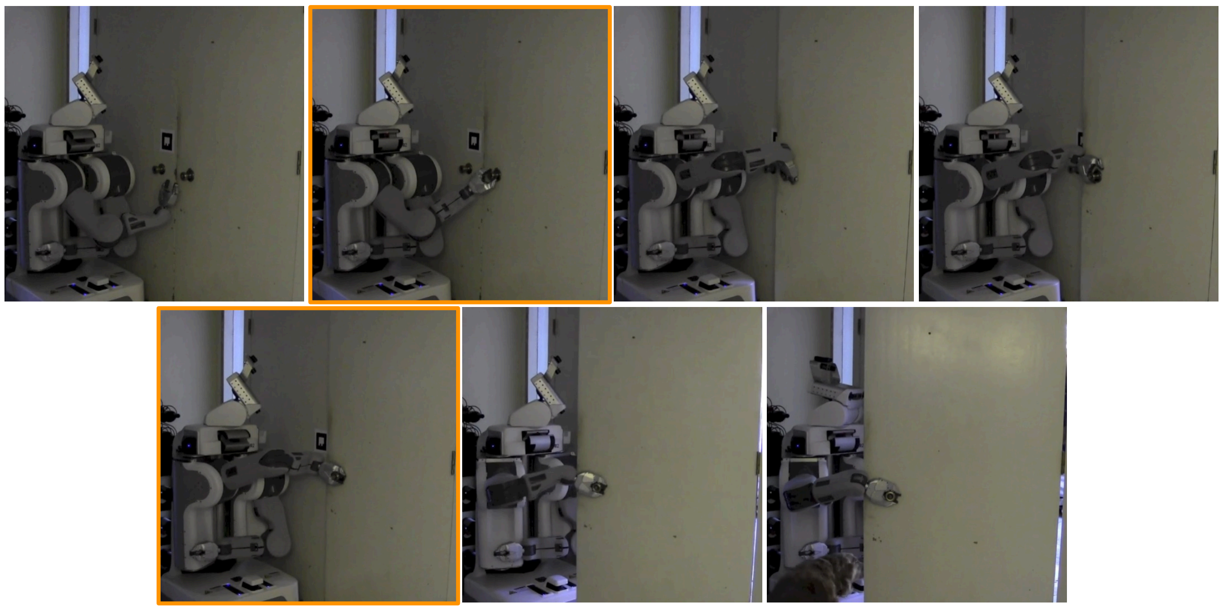


Figure 8: PR2 opening a door with a broken door knob. As door knobs are difficult to operate for a parallel gripper, and the knob needed to be rotated for a full turn to ensure that it disengages, our behavior had to re-grasps the knob mid-rotation to open it. We mark the beginning of each grasp with an orange outline.

CHAPTER III

ROSCO: INVOLVING NON-ROBOTICISTS

Freeing the creation of robot behaviors from limits imposed by slow development processes and limited skilled labor supplies can enable a boom in capability of general purpose robots. We propose to use more powerful behavior authoring tools that allow roboticists and non-roboticists alike to create robot capabilities. Involving non-roboticists has benefits beyond merely increasing the number of functions general purpose platforms are capable of performing. End-users often possess expert knowledge within their domain, potentially enabling robots to perform sophisticated manipulation techniques unknown to roboticists. Recently, researchers demonstrated t-shirt folding by a robot [50]. However, an end-user with work experience in clothing shops might instead construct behaviors that use the much faster Japanese folding method. Similar to the ever creative usage of 3D printers by hobbyists, imaginative non-roboticists have the potential to find new uses for home automation unimagined by today's researchers.

In the previous chapter, we showed that behaviors constructed using ROSCo can be robust and are able to run repeatedly with different mechanisms in a home. We also showed processes for deploying pre-constructed behaviors in new environments. Even so, demonstrations shown thus far use only behaviors created by roboticists familiar with the system. Consequently, it is unclear if non-roboticists, our target population, will be able to use tools such as ROSCo to create similarly complex and novel behaviors. While the proliferation of authoring interfaces in other domains such as photo, video, and 3D model editing demonstrates that users can understand the underlying principles of these mediums to generalize skills learned to other tasks, there is a possibility that robotics concepts, intrinsically, are difficult for users to understand and reason about even with extensive training.

In this chapter, we investigate whether the understanding of concepts unique to robotics will be a potential obstacle and explore challenges involved with behavior authoring interfaces built for mobile manipulation tasks through a user study with computer literate participants. Additionally, through a user-centered iterative design process we identify crucial interface factors to consider when building such interfaces for robotics.

To determine feasibility of non-roboticists using ROSCo, we conducted 16 trials, 5 pilot and 11 formal trials, using computer literate participants recruited from Georgia Tech's campus that do not have a background in robotics. Our study first trained participants to construct behaviors using a series of video tutorials (included as supplementary files to this document), and then tested to see if participants are then able to construct new behaviors using skills taught. In pilot trials, we iteratively tested and refined our study's procedures and system to fix potential deficiencies. This enabled us to make a number of changes to ROSCo's interface; the most important of which are changes for managing the key frame animations and mechanisms for selecting sets of joints to control.

Out of the 11 participants that completed our formal trials, all were able to construct behaviors asked of them in both training and testing sessions. Participants' success provides evidence that, for our population of computer literate users, there likely are not issues fundamental to mobile manipulation that prevent users from freely constructing new useful behaviors once trained. Consequently, our results also show that interfaces such as ROSCo can be viable mechanisms for involving non-roboticists in creating novel mobile manipulation capabilities.

Qualitatively, participants' activities during the study also support this conclusion. A minority of participants showed considerable aptitude while using our interface by moving through menu options and changing parameters quickly and confidently. Furthermore, in training and testing sessions, participants surprisingly produced behaviors that use a wide range of creative manipulation strategies.

For example, this was evident in training sessions where we asked participants to flip a light switch by following a tutorial video in which the robot activated that switch using the broad side of its gripper in a sweeping motion. Even though asked to replicate the behavior as shown, participants produced motions that used a variety of different manipulation strategies. One participant's behavior used the robot's wrist motors to flip the switch, a motion that resulted in gentler collisions between the gripper and wall. Another participant used the gripper's tip to make contact. Other participants used the robot's other arm as the arm they were asked to use occluded their view of the light switch. We believe these diverse outcomes supports the conclusion that behavior authoring environments can enable general purpose robots to move closer to being general purpose by enabling non-roboticists to adapt robots to circumstances unforeseen by their original programmers.

In the rest of this Chapter we will discuss first the specifics of our study in Section 3.2, its results in Section 3.4, then we will end with concluding remarks and comments on future directions in Section 3.5.

3.1 Study Objectives

We designed our study to fulfill two objectives. First, we want to verify that non-roboticists, after being trained, can use behavior authoring interfaces to create novel behaviors in the same manner as users in other domains such as photo editing. Second, we would like to understand the challenges in designing behavior authoring environments that allow users to interact with a robot, manage its interactions with the environment, and design manipulation oriented behaviors.

Additionally, as training participants in all aspects of ROSCo, from navigation to manipulation would require strenuously long trials and thus be impractical, we focused our study primarily on testing interactions with manipulation related building blocks.

3.1.1 Recruitment

We assessed ROSCo's usability with a population of computer literate non-roboticists. We recruited participants through flyers posted around Georgia Tech's campus, social media channels, and word-of-mouth. Prior to scheduling a session with us, we asked participants to fill out a questionnaire where we selected for participants who are computer literate, while excluding participants who are robotics researchers like ourselves.

In the questionnaire (shown in Appendix A.2), we defined computer literate participants as those who have used at least once an authoring interface in the following categories: photo-manipulation, computer-aided design, animation, video editing, sound editing, spreadsheet, page-layout, or programming. Using the questionnaire, we excluded participants who had experience with research robotics since we believe results that include roboticists have the potential to not generalize to other computer literate users.

3.2 Procedure

We organized our study into two sessions, a training session and a testing session. In the first session, we trained participants using a series of video tutorials (included as supplementary files to this document), and then in the second session, we assessed how participants perform without guidance. If participants are able to construct behaviors in the testing session successfully, we believe that it provides evidence for the possibility that non-roboticists will be able to use a tool such as ROSCo to create mobile manipulation behaviors without guidance from robotics researchers.

We kept our trials to a manageable length of time by focusing on testing user interactions with manipulation related building blocks. To do so, we trained participants to use groups of pre-constructed aggregated building blocks called behavior stubs. When used, these stubs attach to a subtree of the HFSM constructed by participants and perform the necessary set of preparatory actions so that manipulation can occur. We provided participants with a stub for each session where the task involves both mobility and manipulation. We will discuss details on the internals of these stubs in the following sections.

To ensure consistency of results between participants, we gave instructions using a script during the experiment. In the four-part training session, we first introduced robot safety concepts and acquainted participants with the system. Then we asked participants to follow along with a video giving instructions on how to construct a behavior where the robot waves and says hello, a task that teaches key frame animation concepts and ROSCo's general interface. Finally, we taught participants, using another video, how to create motions for interacting with the environment through a behavior that flips light switches. For this module, we had users start from a stub behavior.

After completing the training process, we scheduled a testing session on a separate date. During these sessions, we asked participants to make two behaviors, starting with an easier behavior first. In the initial task, we had participants create a behavior that turns off the lights by pressing a rocker switch button with the expectation that this would be simpler than flipping light switches, a task taught previously. We concluded by asking for the construction of a drawer opening behavior. For both of these sessions, we also had participants start with behavior stubs.

Subjects had access to three different types of guidance during the experiment. During training,

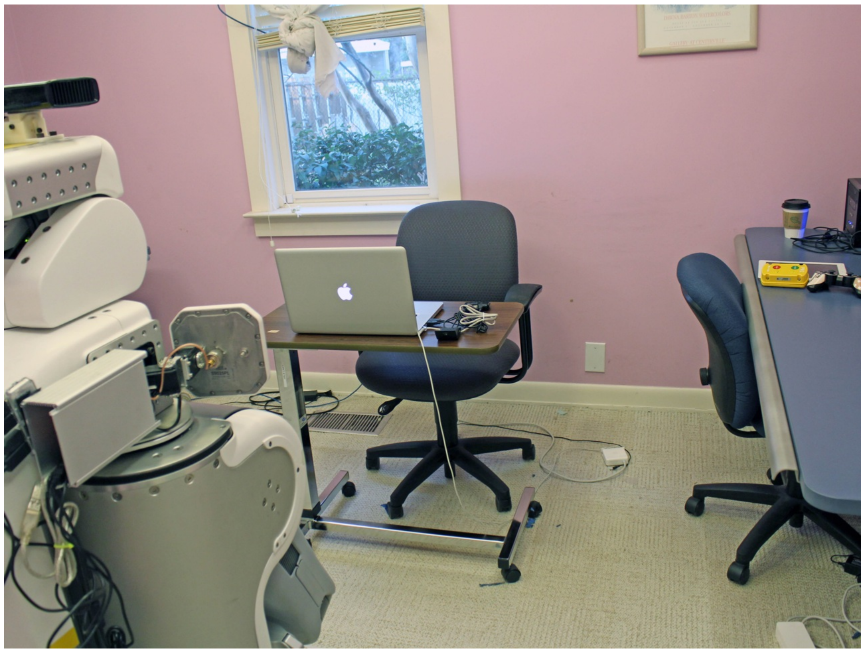


Figure 9: When the experiment starts, participants sit in front of the laptop and the experimenter sits adjacent with the robot's run-stop (yellow box in image). For all trials, we placed the robot in the same physical space as participants in a place where both the robot and the mechanism it operates would be visible.

participants could use training videos, an online manual that explains each command, and a "Help Desk" to verbally ask the experimenter questions if stuck. When participants elect to use the "Help Desk", we recorded these interactions as well as counted the number of questions asked. We limited the assistance given to answering only questions about how to operate the supplied software tools and not about specifics steps of how tasks can be performed. When the questions refer to program mechanics that has been explained by other instructional materials, we referred participants back to where that information can be found.

In the testing session participants had access to all videos and training materials presented in the training session, with the exception of the "Help Desk". The experimenters were only allowed to answer questions about the instructions given, and could provide no guidance otherwise.

3.2.1 Experimental Setup

We performed our study in an office setup at the Georgia Tech Aware Home, a three-story, 5040-square-foot home used as a test facility for household technologies. At the beginning of each session, we situated the robot a small distance away from the mechanism that it will be operating and the participant's desk such that the robot's actions could be easily observed. The experimenter also sat close by in the same room while attending to the robot's run-stop, a wireless switch that can disable all of the robot's motors. We show this setup in Figure 9.

3.2.2 Training Session

The training session is composed of four modules: safety, interactive manipulation, constructing a behavior to wave and say hello, and constructing a behavior that turns on light switches. Each module is accompanied by an instructional video where we encouraged participants to try operations that the video demonstrates step-by-step when applicable. Additionally, we encouraged our participants to ask questions from our "Help Desk" when they had a question or needed assistance completing the training session tasks.

In the first module, we showed Willow Garage's standard safety video for operating the PR2. By doing so we hope to mitigate the risk that participants can potentially harm themselves, the environment, or the robot through using the teleoperation interface provided by the interactive marker controls. During the experiment, we additionally have the experimenter observe participants and control the robot's remote run-stop.

In the second module, we acquainted participants with the RViz interactive manipulation interface. Our training video gave instructions on how to perform basic 3D movements such as translating, rotating, and zooming the 3D display using a mouse. After participants are better acquainted, our video gave instructions about how the robot can be teleoperated using interactive marker controls (Figure 10) and an overview of major ROSCo interface components. In this module, we started with the robot placed so that it faced the participant, with the experimenter next to the participant. We used this arrangement to ensure that participants will be able to see the robot's motions as they control it using the interactive marker interface.

In the third module, we asked participants to watch and follow along with an instructional video describing how to construct a behavior where the robot performs a gesture and then uses text-to-speech to say a greeting. In this task, we instructed participants on the basics of key frame animation, posing the robot, linking subcomponents together to create a behavior, and diagnosing issues by stepping through different states. Upon completion we asked that participants notify the

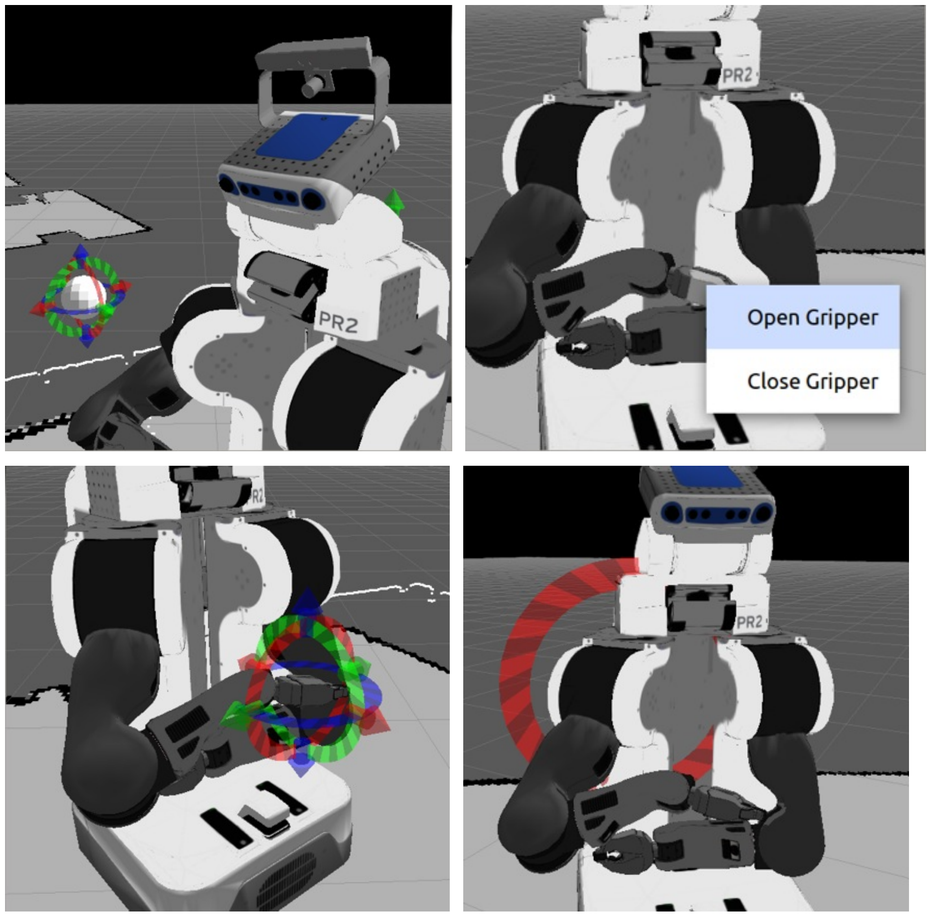


Figure 10: Interactive marker controls shown during the second module. ***Top Left****: Moving this marker causes the PR2's head to point at it.* ***Top Right****: Right-clicking on the grippers shows options for opening and closing them.* ***Bottom Left*** *Rings-and-arrows interface for positioning the gripper in 3D.* ***Bottom Right*** *Ring for controlling the arm's redundant degree-of-freedom.*

experimenter then demonstrate the behavior. In the event that the demonstration is not successful we allow participants to retry and note the number of failing attempts. After successfully demonstrating the behavior, we asked participants to fill out a NASA-TLX survey [61] for assessing perceived mental workload.

We introduced the ROSCo tuck, joint sequence and speak building blocks during this session (Table 3). The tuck block will tuck and untuck the robot's arms into and out of a default resting position. Activating the joint sequence block creates a splined trajectory using key frames defined in the arm's joint space. Finally, the speak block runs a text-to-speech engine.

In the fourth and final module of the training sessions, we introduce the creation of motions capable of interactions with the environment. To do so, we ask participants, using an instructional

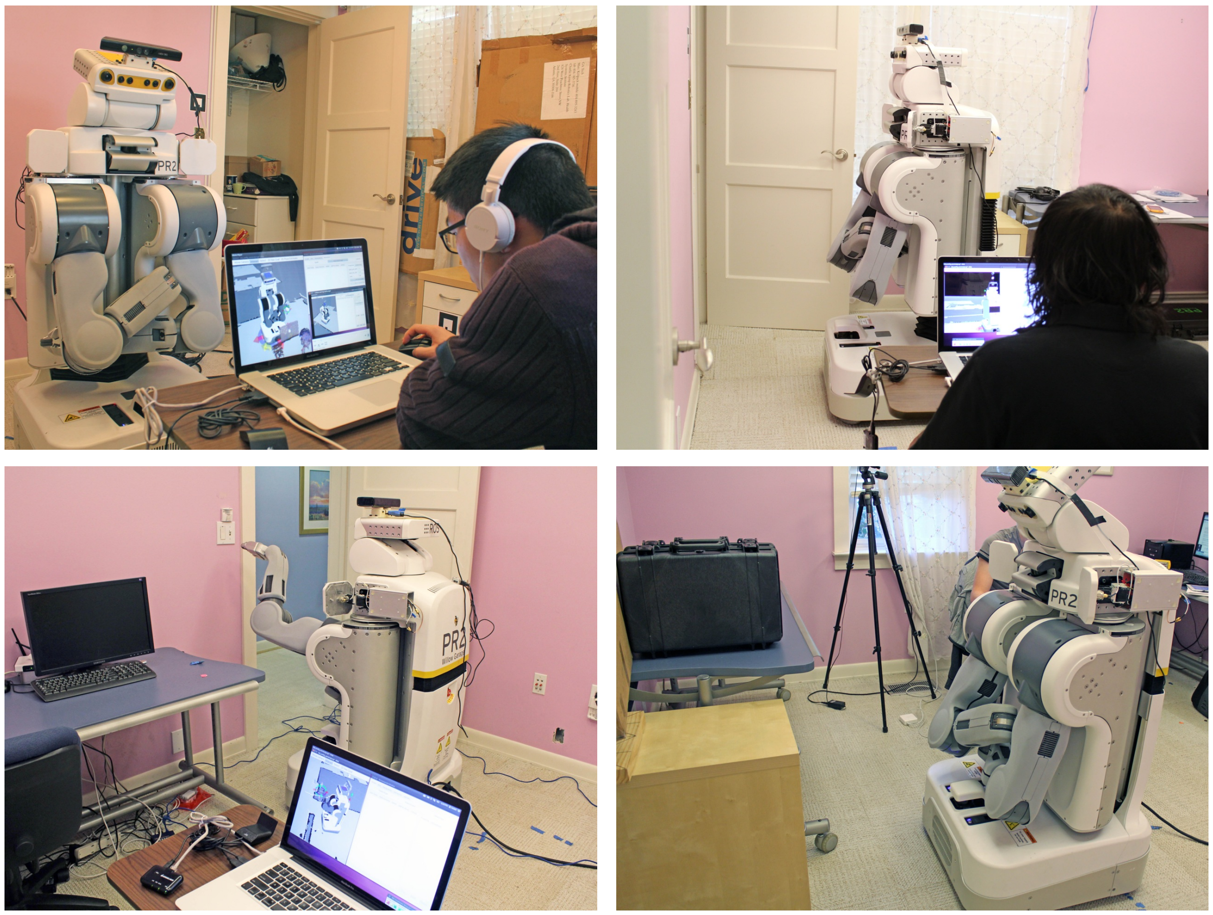


Figure 11: Positioning of robot and participant for each module. ***Top Left****: Subject constructs a waving behaviors.* ***Top Right****: Constructing a behavior to turn on the lights with a flip switch.* ***Bottom Left****: Participant constructs a behavior that pushes a rocker switch to turn off the lights.* ***Bottom Right****: Subject creates a behavior that pulls open a drawer.*

video, to construct a behavior that autonomously navigates to a light switch then turns it off using the broad side of robot's gripper. Our trials begin with the robot situated in front of a light switch but not with the switch within its workspace to force the use of autonomous navigation. We point the participant's desk in the robot's and switch's direction to ensure that the robot's actions can be observed both in the interface and in reality (Figure 11). To accomplish the task, participants would first create a 3D frame for the task by clicking on a representation of the detected tag, a green sphere, and then specify appropriate motions that interacts with the switch in that frame. After participants finish, we asked for a demonstration, recording failed attempts, then presented the NASA-TLX survey just as in the previous module.

In this session we showed participants how to use the velocity priority and gripper block (Table

Table 3: Building Blocks Taught to Participants

Introduced During	Block Name	Usage
Wave	Tuck	Tucks and untucks the arms
Wave	Joint Sequence	Splined joint space trajectory from key frames
Wave	Speak	Text-to-speech
Light Switch	Velocity Priority	Cartesian end-effector control [107]
Light Switch	Gripper	Opens/closes the gripper
Stub	Direct Navigation	2D base navigation (odometry only)
Stub	Navigation	2D base navigation (obstacle aware) [93]
Stub	Move Head	Splined trajectory for the head

3). The first block allows users to define a series of 6D Cartesian waypoints for the end-effector to move through using a Jacobian transpose controller [107]. With the gripper block, users can control how wide the gripper opens.

As mobile manipulation behaviors such as our light switch task often has prototypical structures, we gave participants access to stub behaviors that facilitates the authoring process. Much like the use of templates for performing common tasks in design and word-processing programs, these stubs speed up development by defining default actions commonly used. Additionally, our stub helps to decrease the time each participant would need to perform the trial, reducing the likelihood of participants tiring out. In practice, we believe that such stubs will be used much like parts in a CAD program's parts library. In our case particularly, we provided participants with a chain of actions that tucks the arms in in preparation for navigation, navigates the robot to a fixed point in the user's defined task frame, untucks the arms, and then points the robot's head at the center of that frame. Usage of this stub is similar to using normal building blocks, participants would first define a reference frame to execute it in then click "Run" to execute it. Running the stub executes all its component actions in turn, placing the robot in front of the light switch.

This stub behavior contains HFSM states from the direct navigation, navigation, and move head block, tools that have not yet been introduced to users due to time constraints (Table 3). The direct navigation block uses the holonomic base to move the robot to a goal location with only odometry information. In contrast, the navigation block encompasses a large subsystem devoted to collision free navigation [93]. Finally, the move head block moves the head and robot's cameras so that they point to a particular location.

In case where participants do not successfully complete all training tasks, we record this as a failure. Otherwise, we invited participants to schedule a time for the second session.

3.2.3 Testing Session

Our goal with the testing session is to assess whether the combination of skills learned in the first session combined with training materials, and past examples can enable participants to construct new behaviors without experimenter assistance. In contrast to the previous session, there was no access to the "Help Desk" and the experimenter did not answer questions except to clarify instructions. This session consisted of three modules where participants first reviewed skills learned, constructed a behavior to press a rocker switch, then constructed a behavior to open a drawer. We administered these modules in a similar manner to those in the training session. Experimenters asked for demonstrations, recorded failed attempts, then asked participants to fill out a NASA-TLX survey at the end.

These sessions start with a review module as significant time–days, or even weeks–could have passed since training. For this module we prepared the room such that the robot was in front of the light switch and the participant's light switch behavior was loaded in ROSCo just as it was at the end of the training session. We then asked our participants to reacquaint themselves with the interface, executing the behavior and flipping the switch if necessary. As an exception to other modules in this session, the experimenter was allowed to answer "Help Desk" questions during this exercise. Additionally, all the training materials, videos and online manual, were also available. The review module ended once participants notify the experimenter that they are ready to proceed.

We began testing by asking participants to create a behavior that turns off the lights through pressing on the bottom portion of a rocker switch. In terms of states needed, this behavior can be created by taking the light switch behavior and deleting its swiping motion as the force resulting from reaching out to the switch alone is sufficient. We picked this task deliberately as it helps ease participants into constructing behaviors without assistance. In our the experiment, participants began with the same stub behavior as in the light switch task but have to reconstruct the other states independently. Similar to other modules, we began the trial with our robot situated close to the rocker switch and the stub behavior preloaded in ROSCo. The stub here is identical as the stub in the light switch task; when executed it attempts to safely bring the robot to the switch from its starting position.

After the rocker switch task, we asked participants to construct a behavior that opens the top drawer in a chest of drawers. Compared to previous behaviors, drawer opening involves additional steps and can be more complex. Instead of merely making contact with the mechanism, the robot additionally has to grip the drawer's handle as well as pull it in a linear trajectory. Due to this complexity, we placed drawer opening last in our testing sequence as users would have more experience with the system at this stage.

To test, as before, we moved the robot so that it faces the chest of drawers and started participants with a stub. This particular stub is similar to the previously used stubs in other behaviors. However, as the robot's base location, hence distance from the mechanism operated on, is a parameter of a navigation module embedded inside the stub which has not yet been introduced to participants, we provided a modified stub that manually displaces the base to be 15 cm further away from the mechanism. Doing so allows the robot room to pull the drawer out without its arm colliding into its body. In normal use, we expect that such modifications will not be needed as users would have more time to learn the tool and, consequently, know how to use the navigation module.

After participants successfully constructed and demonstrated the behavior, we ended the module by asking participants to complete an online questionnaire that asked them to rate the experience and to provide feedback and suggestions (shown in Appendix A.3).

3.2.4 Surveys

We administered 7 surveys in total for each participant. This includes 2 surveys before the experiment, one after, and 4 workload assessment surveys. We presented these workload surveys after each training task. We show all administered surveys in Appendix A.

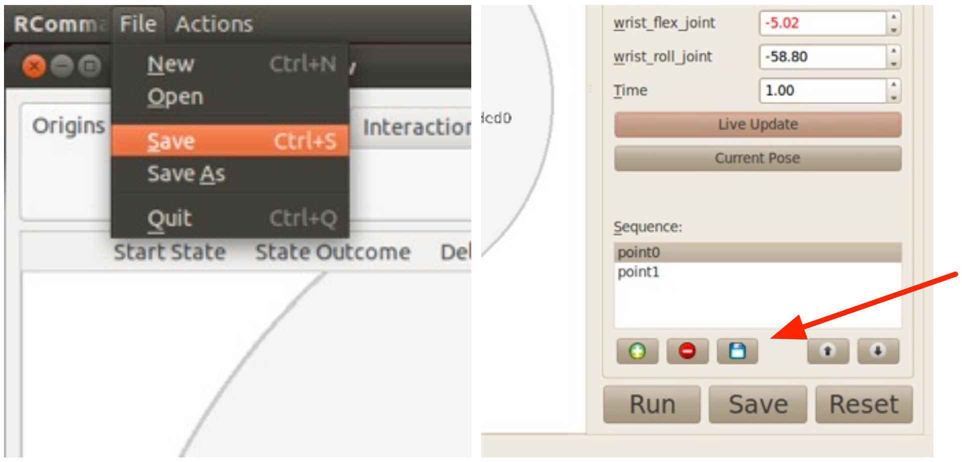
Prior the experiment, we sent participants a pre-experiment survey, as described earlier, that assess their proficiency with computers and research robotics experience. For computer experience, we accepted potential participants who have either had experience with a complex authoring software package (video, sound, image, spreadsheet editing) or programming. We excluded those who have had experience with mobile manipulators. We summarize the results of this survey and the pre-experiment survey in Table 4. As we recruited from a university setting, all participants met the criteria set by the screening survey.

After each module in the training and testing session, we administered NASA-TLX surveys [61] to assess participants' mental strain and workload after completing each task (Table 7). At the end of the study, we also asked questions to characterize participant's opinions of the interface in a post-experiment survey. This survey was a mix of questions that are based on a five-point Likert scale (questions 1 to 9) and those that are open-ended (questions 10-16). We list these Likert scale questions below:

1. I am satisfied with the speed that the robot was moving while using the interface.
2. I was worried that I might break the robot using the interface.
3. I was worried about my safety while using the interface.
4. I am satisfied with the time it took to complete the task using the interface.
5. I could effectively use the system to accomplish the task using the interface.
6. The interface was intuitive to use to complete the task.
7. It was enjoyable to use the interface.
8. Even after becoming acquainted with the interface, I found it challenging to use.
9. I found the interface fun to use.

Next, we list the open-ended questions.

1. Which part(s) of the interface did you find difficult to work with?
2. Which part(s) of the interface did you find easy to work with?
3. Which part(s) did you find confusing?
4. Which part(s) did you find intuitive?
5. Do you have suggestions for how we can improve this interface? If so, please list them below.



wrist_flex_joint -5.02
wrist_roll_joint -58.80
Time 1.00

Figure 12: Interface menu items with save-like functionality that users participants found confusing. ***Left****: File menu option to save state machine to disk.* ***Save Button****: Keeps changes made to a state, and ignores them otherwise.* ***Floppy Disk Icon****: Keeps changes made to a state, and ignores them otherwise.*

6. What types of tasks would you want to accomplish with this interface?

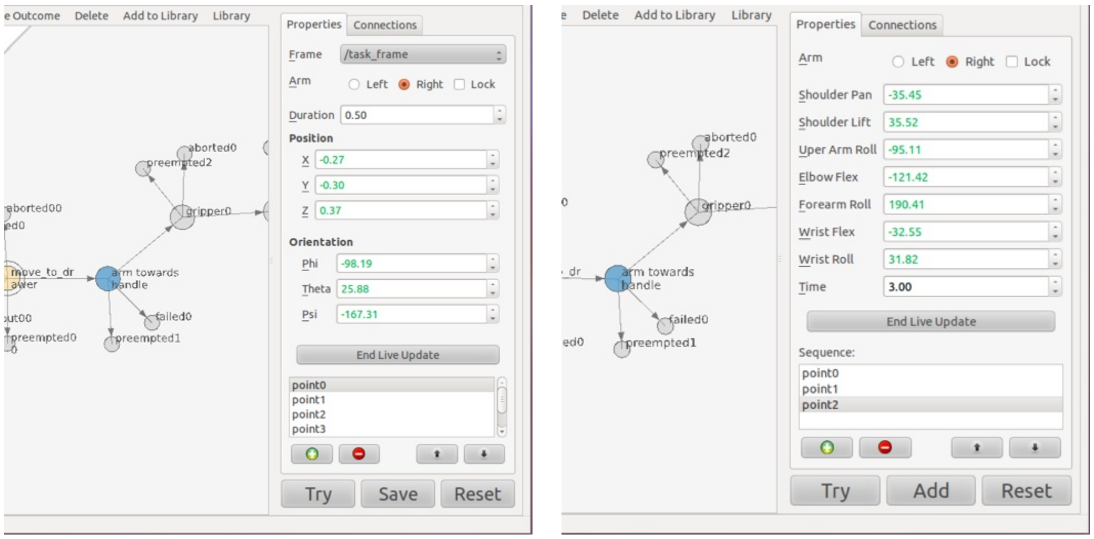
7. Do you have any other comments for us?

3.3 Pilot Runs

Prior to running our study, we ran a series of pilots where we tested 5 individuals, with 2 participants recruited via word of mouth, and 3 participants from the Georgia Tech Healthcare Robotics lab. From these pilot runs, we made a number of interface modifications to fix issues encountered by pilot participants. In this section, we discuss some of the more major modifications made.

3.3.1 Confusing Save Functionality

We discovered early on that pilot participants were often confused by three different interface elements that controlled saving changes made by users. As shown, the interface had a save option for recording all of the state machine to disk activated by a save option in the file menu. It had a large "Save" button underneath the state parameter area for committing changes made to a particular state. Additionally, the parameter area for tools that generate joint trajectories also had buttons with floppy disk images on them that commit the changes made to particular motion key frames. We show these various elements in Figure 12.



Frame: /task_frame
Arm: Left [✓ Right] Lock
Duration: 0.50
Position
X: -0.27
Y: -0.30
Z: 0.37
Orientation
Phi: -98.19
Theta: 25.88
Psi: -167.31
point0
point1
point2
point3

Arm: Left [✓ Right] Lock
Shoulder Pan: -35.45
Shoulder Lift: 35.52
Uper Arm Roll: -95.11
Elbow Flex: -121.42
Forearm Roll: 190.41
Wrist Flex: -32.55
Wrist Roll: 31.82
Time: 3.00
Sequence:
point0
point1
point2

Figure 13: Properties panel for a single state showing controls for managing key frames. Key frame specific controls shown at bottom of each panel. The top part (with fields in green) shows the content of each key frame element. ***Left****: Panel for editing a Cartesian trajectory using key frames that represent end-effector position (6 DoF poses).* ***Right****: A join trajectory editor with key frames that represent joint angles.*

The number of different elements with similar functionality made it difficult to clearly communicate the controls' various purposes to users. Pilot users also reported that it was hard to remember these differences during operation. As a result, participants often forgot to save changes made and became frustrated when the interface fails to record their changes. In response, we redesigned ROSCo to save whenever users make a change as opposed to only when users explicitly activate an interface control. This change eliminated many redundant save buttons and left only the one save option in ROSCo's file menu.

A significant lesson learned from this situation is that our interface should err on the side of saving recent changes rather than preserving past work through assuming that recent modifications might be harmful unless explicitly marked by users. Additionally, it also showed us that our users tend to group controls with similar names together despite those controls being in different locations.

3.3.2 Controls for Adding and Modifying Key Frames

A common operation in ROSCo involves creating, modifying, and managing key frames making interacting with recorded poses important to ROSCo's overall usability. Generally, key frame controls in ROSCo are made up of two components. The first displays parameters such as joint angles or Cartesian coordinates that describe the particular key frame selected (top of Figure 13). The second shows a list of key frames currently being edited and buttons for adding, deleting and re-ordering them (bottom of Figure 13). Additionally, our interface uses the top parameter fields to give feedback on how the current robot pose relate to the particular tool's parameters. This can be activated either using a "Current Pose" button that updates once per click or a "Live Update" button that updates continuously.

To create new key frames with these controls, in the nominal case, users would first move a robot body part to a new pose using on-screen controls, then click a button that has a green plus sign on it to create a key frame with that pose. However, we discovered from pilot testing that there were four different ordering of button presses that can plausibly result in the same desired outcome from users' perspective. However, only two of these four sequences produced the expected behavior. The remaining two orderings would merely create copies of the last frame added rather than a frame containing the robot's new pose. As users would not discover this error until they later try to execute the animation, this turned out to be extremely frustrating during pilot trials.

To improve usability, we removed the "Current Pose" button halving the possible number of button press sequences to two. If users clicked the plus button first, then the interface clones the key frame displayed. Else if users clicked "Live Update" then each press of the plus button would then create a new key frame recording the robot's current pose. In addition, the "Live Update" mode now turns the displayed numbers green, visually communicating whenever it is active to distinguish the two different modes.

3.3.3 Selecting the Correct Arm

Recording motions from the correct arm proved challenging with the system tested initially. Pilot participants would often forget to tell the interface that an arm different from the one selected in ROSCo is the arm that the interface should be recording from. This error normally resulted in motions that does not move the robot since the key frames recorded all came from the opposite and usually not manipulated arm. We show the radio buttons which controls arm selection at the top of Figure 13. The buttons also indicate which arm the displayed key frame will command and which arm's pose will be displayed when users click the "Live Update" button.

Table 4: Demographics of Participants

Variable	Values
Gender	Male(6), Female(5)
Age	19-34, (Median = 24.5, SD = 4.2)
Ethnicity	Asian (5), Caucasian (5), African American (1)
Education	High School (3), 4-year degree (5), Master's (2), PhD (1)
Photo-manipulation	Used more than once (10), Part of job (1).
CAD/Animation	No experience (1) Used once (3), More than once (5), Part of Job (2)
Programming Experience	Yes (6), No(5)

Table 5: Number of questions asked of the "Help Desk" during different sessions (mean ± standard deviation).

Task	Number of Questions
Greeting	1.5 ± 1.8
Switch	3.5 ± 3.0
Review	0.6 ± 0.8

We mitigated this with a new feature that automatically selects the arm being moved as the one that users will want to record from. This works by selecting the arm where joint angle differences between two successive poses are largest. However, in case that the arm being moved is not the one that users want to record from, we added a "Lock" check box to toggle this feature off.

3.4 Results

The 11 participants in our study were all able to finish the training session and testing session successfully. The minimum, average, and maximum number of days between participants performing the training and testing session is 2, 4, and 8 days respectively. In testing, participants successfully constructed behaviors that pressed rocker switches and opened drawers. Our results provide evidence that, despite not being familiar with robotics, participants are able to use ROSCo and perhaps

Table 6: Mean and standard deviation (±) of time taken to construct behaviors by study participants in minutes and seconds (MM:SS).

Task	Time to Completion
Greeting	$27:21 \pm 06:10$
Switch	$48:30 \pm 15:26$
Rocker	$24:19 \pm 06:21$
Drawer	$22:43 \pm 07:24$

Table 7: NASA-TLX Results: Mean and standard deviation (±) of user responses where ▼ and ▲ indicates that the measures are either statistically low or high using a two-tailed t-test on a mean of 10. This scale has a min of 0, and a max of 19.

Task	Mental	Physical	Temporal	Performance	Effort	Frustration
Greeting	10.0 ± 4.2	2.1 ± 1.4▼	6.9 ± 5.4▼	3.5 ± 1.9▼	8.8 ± 3.0▼	4.9 ± 3.0▼
Light Switch	13.5 ± 3.5▲	2.6 ± 1.8▼	8.1 ± 4.4	4.9 ± 2.8▼	12.7 ± 3.8	10.8 ± 5.0
Rocker Switch	9.5 ± 3.8	2.6 ± 2.1▼	6.0 ± 4.0▼	3.9 ± 2.3▼	10.7 ± 4.2	6.4 ± 4.7▼
Drawer	9.5 ± 3.4	3.0 ± 1.9▼	7.0 ± 4.5▼	4.2 ± 2.1▼	9.6 ± 4.5	6.1 ± 5.3▼

similar interfaces to create robot manipulation behaviors of similar complexity to those produced by roboticists.

In Table 5, we provide a summary how many time participants asked for assistance from the "Help Desk" during the testing session broken down by module.

We show the averages and standard deviations of times participants took to construct different behavior in Table 6. In this data, constructing the light switch behavior required the most time out of all tasks in our tests. This seems counter intuitive as our light switch behavior does not require as many states (Table 2) as other behaviors, we expected that participants needed less time not more.

Although our experiments are not equipped to determine the specific cause, there are a number of plausible explanations of why the light switch task caused participants to need more time. First, as this is the first time that we asked users to have the robot interacts with its environment, it could be due simply to learning effects. This theory is consistent as well with the number of questions asked being higher for the light switch task (shown in Table 5). Alternatively, or in addition to, the light switch task can be more challenging to complete since the flip switch is much smaller than either the push switch or drawer handle used by the robot in other tasks.

In Table 7, we provide a summary of our participants' self-reported workload surveys administered after each task. We ran two-tailed t-tests at the 95% confidence level for both extremes using a score of 10 as the mean for each measure. This tested whether the participant reported scores are significantly low or high, either below or above the mean, respectively. We denote responses that are significantly low or high with either a downward or upward pointing arrow in Table 7.

Overall, participants reported feeling low physical demand, low temporal demand, and able to

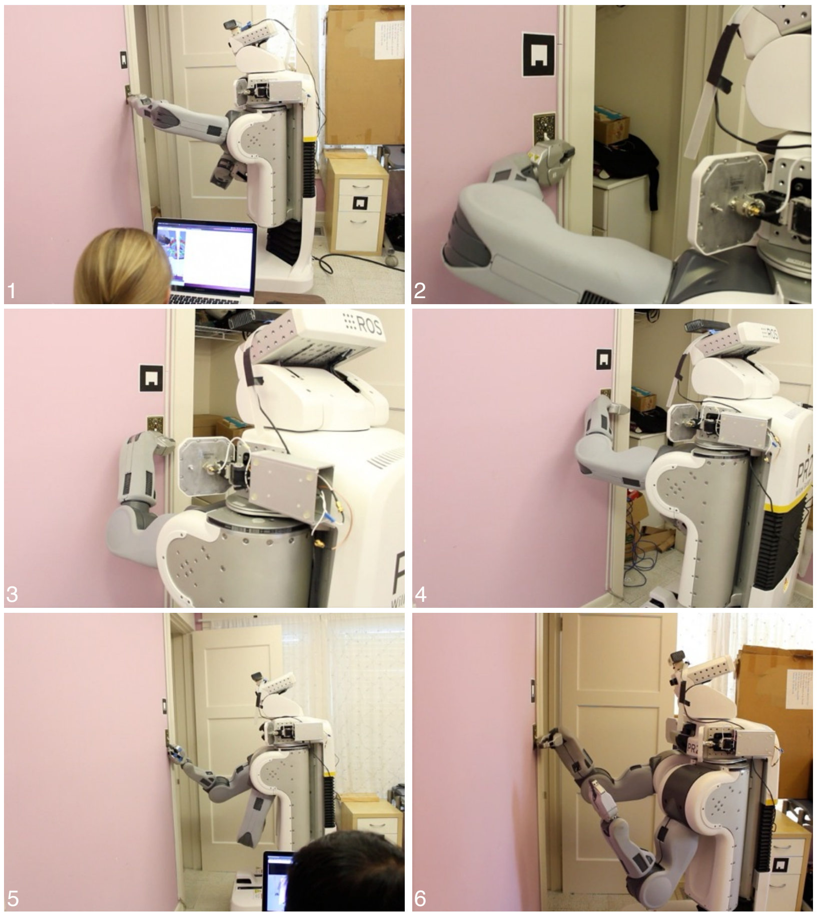


Figure 14: Subject's light switch behaviors varied widely despite following along with our video tutorials. (1) flicked the switch with the gripper's tip. (2) flicked with the side of the gripper. (3) used the side of gripper, but flicked the light switch primarily with the wrist motor, applying very little force compared to other behaviors. (4) used the gripper's side but with the gripper oriented horizontally (5) used the tip of the right gripper. (6) used the right gripper rotated horizontally.

accomplish the tasked they were asked to do (listed as performance) for all tasks. Effort and frustration levels were low for the greeting task, with frustration similarly low for the rocker switch and drawer task. As an exception, participants alternatively reported high demand during the light switch task. This workload data also showed a similar pattern to the time to completion table discussed previously with the light switch task being distinguished in mental, temporal, performance, effort and frustration measures.

3.4.1 Subject Constructed Behaviors

Subjects used a wide variety of different manipulation strategies in our study. The behaviors constructed vary in a number of different dimensions such as the speed at which they move the end-effector, whether the gripper slows down prior to contact, which part of the gripper is used to make contact, which joints are used to apply force, and how much force to apply during contact. A number of behaviors for operating the light and rocker switches even appear to apply less force compared to behaviors we constructed for the system in Chapter 2. Even though such behaviors can be less effective at accomplishing task objectives due to different trade-offs made, they appear potentially more suited in other manners. For example, where the rocker switch behavior we used pushed the switch forcefully enough to cause an audible thud, a participant created behavior was able to operate without creating noticeable collision noises. Unexpectedly, these variations are present even for behaviors where participants were instructed explicitly to follow our video tutorial closely.

We demonstrate participants' different variations of the light switch behavior in Figure 14. In image 1, even though our tutorial video depicted the robot swiping with the side of its gripper, the behavior constructed used the gripper's tip to make contact. The participant in image 2, in contrast, has the gripper oriented so that its flat side is against the wall. Despite our tutorial behavior using primarily the robot's shoulder joint to make a wide sweep upward for flipping the switch, the participant in image 3 used the wrist motor instead which resulted in a softer, and more gentle motion. However, some participants did construct behaviors that used the same strategy as shown in our tutorial (image 4). As using the left arm blocked participants' view of the gripper and light switch, a few elected to use the right gripper instead (shown in image 5 and 6).

We show participants' variations of the rocker switch behavior in Figure 15. Previously, our version of this behavior directed the robot to poke the switch with its gripper tips. In contrast, participants' versions utilized various parts of the gripper to make contact on either of the robot's arms. One of the more notable behaviors, shown in the 5th panel, used the wrist motor to create a motion that applies just enough force to press the switch.

Every drawer opening behavior, unlike other behaviors, used only the right arm. We show this in Figure 16. Behaviors constructed seem to differ primarily in how they solve the problem of the

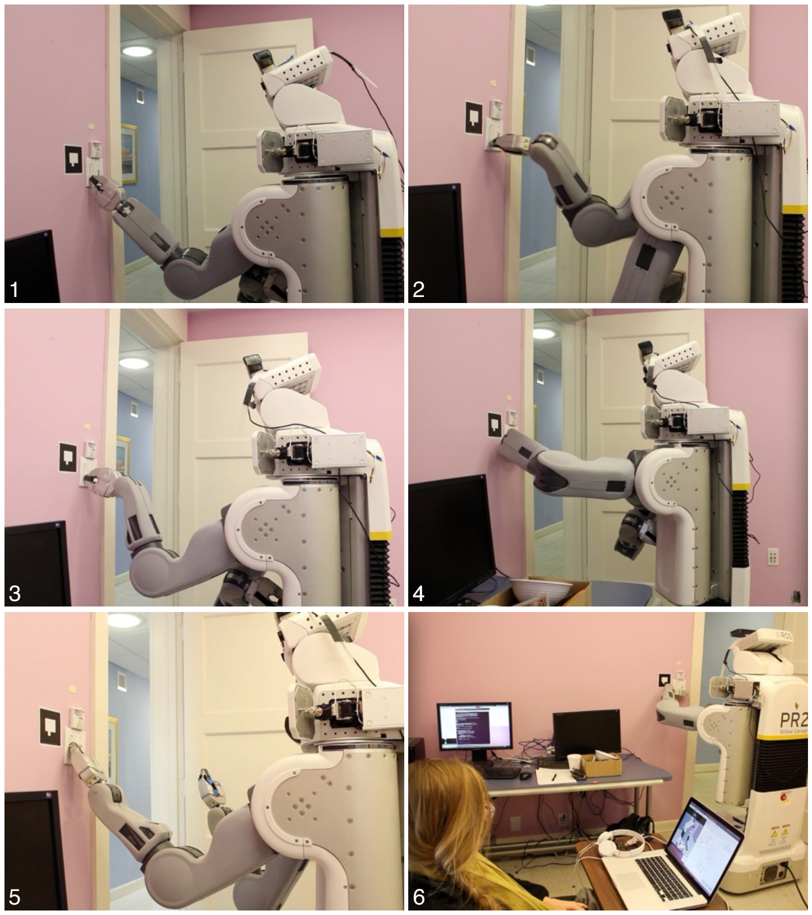


Figure 15: Similarly to the light switch task, participants used a variety of different gripper orientations and strategies. (1) pressed sideway. (2) used the right arm, and a poking strategy. (3) also poked but with the left arm. (4) pressed with the entire gripper. (5) like (3) in Figure 14 used the wrist motor to press more gently. (6) pressed with the side of the gripper, similar to (4).

robot's elbow colliding with its body while pulling the drawer open. Some behaviors moved the elbow away from the body, while others moved the elbow inward and to the opposing arm's side. One behavior even untucked the left arm completely to get it out of the way.

These participant constructed behaviors show that, even in constrained settings such as this user study, users produced a variety of creative behaviors with interesting characteristics that can be viable alternatives to those produced by roboticists. Behaviors produced are qualitatively similar to those constructed by the authors in previous chapters and some were even less forceful, producing little audible sound when the robot makes contact. As home manipulation involves everyday objects that users likely have extensive experience with, thus useful predictive models of, and in domains where roboticists might not be familiar, involving end-users can be advantageous in creating novel,

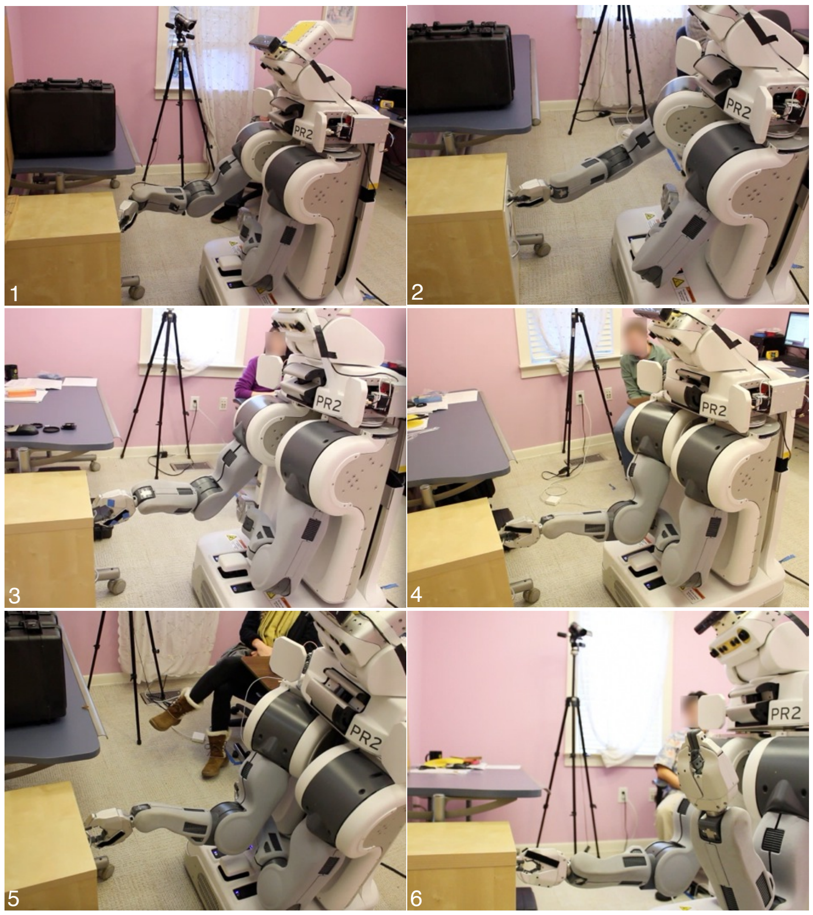


Figure 16: Participants were less varied in their approach to manipulating the drawer, perhaps due to having to grasp the drawer's handle. Behaviors differed on where they placed the elbow. (1), (2), and (3) moved the elbow out; (4) and (5) moved it to the other side; (6) moved the left arm out of the way.

creative, and more effective robot behaviors.

3.4.2 Post-Experiment Survey Results

We ran two-tailed t-tests with Likert responses in our post-experiment survey at the 95% confidence level to determine if the scores reported are non-neutral, i.e. significantly below or above a mean of 3. Of the responses that are statistically significant, participants reported being satisfied with the speed of the robot (question 1), not worried about their safety (question 3), satisfied with the time taken to complete the task (question 4), and their ability to use the system (question 5). Furthermore participants reported believing that the interface was intuitive (question 6), and enjoyable (question 7) but challenging (question 8). Just as we expected, participants reported that the interface itself

Table 8: Post Task Questionnaire Results: Mean and standard deviations (±) of scores. We show using ▼ and ▲ that the Likert measures, on a five-point scale, are either statistically low or high using a two-tailed t-test using a mean of 3.

Question	Score
Satisfied With Speed	4.0 ± 1.0▲
Concern About Harming Robot	2.7 ± 1.3
Worried About My Safety	1.5 ± 0.7▼
Satisfied With Time	3.5 ± 0.7▲
Able to Use System	4.3 ± 0.6▲
Interface was Intuitive	4.1 ± 0.8▲
Interface was Enjoyable	4.5 ± 0.7▲
Interface was Challenging	4.0 ± 1.0▲
Interface was Fun	4.1 ± 1.1▲
Challenging After Learning	2.1 ± 1.3▼

was challenging to use (question 8) but also reported that the interface was not challenging after learning (question 10). We show the responses in Figure 17 and Table 8.

With open-ended questions, we received many diverse answers. We coded the responses by first breaking each response into snippets based on the interface element which it concerns then grouped all these phrases across participants together again based on the interface element. We summarize the results of these groupings in the following section.

3.4.2.1 Parts of the Interface That are Difficult or Unintuitive

When asked about which part of the interface was difficult or unintuitive, participants pointed out a variety of issues in: recording motions (11 times), movement controls (5 times), using point-clouds for fine motion control (4 times), distinguishing between different controls (4 times), and moving in a 3D view (2 times).

Generally, participants mentioned components having to do with the process of recording motions most often (11 times). These difficulties include not knowing what the "Live Update" mode does at first, not knowing what the name "Velocity Priority" means (mentioned twice), forgetting to select the task frame to record in, accidentally moving the task frame, and controlling the velocity of motions recorded.

In terms of teleoperating the robot using interactive marker controls in RViz, as the controls uses a Jacobian transpose controller [107], users were frustrated when the robot's arms were close to singularities causing it to respond minimally to user input. Additionally, participants also mentioned that circles for controlling the extra 7th degree of freedom of the robot's arms were unintuitive. Altogether, these difficulties with movement controls were mentioned 5 times.

Next, we observed that most participants decided early on to look directly at the physical mechanism being manipulated and the physical robot rather than the 3D visualizations. Accordingly, many participants reported having issues with the visualizations in our survey. Two participants reported that they were able to be more accurate by looking at the physical scene rather than the static point-cloud on-screen. One participant mentioned that it was hard manipulating the robot using on-screen controls and knowing what the robot is doing. Another participant wished that the physical robot was transparent as it occluded the scene. Issues with maintaining situational awareness such as these were mentioned 4 times.

Another sizable cluster of issues centered around having too many controls on screen at the same time. One participant mentioned that it would be great to have a larger screen. Another said that having all the spheres and arrows used for various controls were confusing. Overall, problems with visual clutter were mentioned 4 times.

Besides these larger issues, participants mentioned frustration with the zoom and pan functions in RViz twice. There was also one mention of the process of connecting states to one another in ROSCo being confusing.

3.4.2.2 Parts of the Interface that are Intuitive

When asked which parts of the interface seem intuitive and easy to use, participants mentioned components relating to ROSCo (9 times), interactive marker controls (5 times), and the task frame controls (1 time). With regards to ROSCo, participants commented on the intuitiveness of the graph visualization (4 times), the ease of connecting states (3 times), the try button (2 times), and the "Live Update" mechanism (1 time). The interactive marker controls using directional arrows were also favored by many participants. One participant enthusiastically noted the effectiveness of the head movement controls.

3.4.2.3 Different Tasks That the System Might Be Used For

Similarly, participants gave many diverse responses when asked what types of tasks they would want to accomplish with this type of system. As participants do not have extensive experience with robotics, tasks given ranged from simple operations that are possible with ROS Commander such as opening a door to more complex operations such as cooperatively playing a game or making a sandwich. To code these responses, we scanned for explicit mentions of tasks then grouped them according to similarity, creating a group for each collection of tasks with more than two items.

From this process, we created 4 groups of tasks: household cleaning activities, assistance with simple meals, entertainment, and a miscellaneous group for tasks that did not fit other categories.

The biggest category of tasks involved cleaning activities of some sort: vacuuming, transferring clothing from washers to dryers, sorting laundry, cleaning out litter boxes, mopping, loading dishwashers, ironing, cleaning tables, putting back objects and scrubbing toilets. The next category contained tasks that involves simple preparation of food such as fetching water, boiling water, taking food out of refrigerators, wrapping a sandwich, making coffee, and roasting meat. There were a few entertainment related tasks including playing a game, giving massages, and opening the door during parties. Additionally, there were a number of tasks mentioned that did not fit into any of the above categories including walking the dog, and object fetching. For fetching, the remote control was mentioned most frequently as is consistent with past findings. Finally, one participant also mentioned the task of "helping get out of the house in the morning" as a larger organizing task that can include activities such as fetching food out of a refrigerator, perhaps wrapping sandwiches, and putting back towels thrown on a bed

Theoretically, many of these tasks are possible using ROSCo's Hierarchical Finite State Machine (HFSM) representation and have been demonstrated with systems using finite state machines such as fetching items from a refrigerator [32], and making coffee [121]. However, with more interactive tasks such as walking a dog or playing a game it is unclear how best or whether such tasks can adequately be represented with HFSMs. Additionally, there are still many limitations in ROSCo due to the current set of components within the system. For example, to fetch items from a refrigerator, depending on the acceptable level of robustness, might require being able to recognize specific objects requested by users [32], manipulate in while in clutter using methods that prioritize whole-body contact [69] or more advanced occlusion reasoning [53].

3.5 Concluding Remarks

Results of this study support our claim that non-roboticists, after a training period, can create robot behaviors with behavior authoring interfaces such as ROSCo that are of comparable complexity to those created by experienced roboticists. Due to simplifications made possible by ROSCo's Hierarchical Finite State Machine representation and parameterization of robot base position and arm motions with respect to user-defined task frames, the behaviors created were able to incorporate both mobility and manipulation components. Additionally, other users can very plausibly use behaviors created by our participants, who are not roboticists, for home automation.

From these results, we believe that authoring interfaces are a feasible means of allowing users willing to watch video tutorials to create new robot manipulation capabilities. Even in the restricted settings of our study with limited time and constrained goals, participants demonstrated their creativity by producing with many diverse strategies. Encouragingly, we have observed that behaviors generated by users, such as the light switch flipping behavior that twists just the robot's wrist, can also be, in some respect, better than the behavior we ourselves created in the previous chapter.

In the course of this work we have also discovered that having intuitive mechanisms for creating, editing, and refining key frames are vital to the success of systems such as ROSCo that depends on users manipulating motion trajectories. While work in similar fields such as learning by demonstration [21, 18] overcome problems with errors in demonstrated trajectories primarily through relying on having more demonstrations [16], a complementary approach where users are given the facilities to correct errors in examples has the potential to help users achieve the desired behaviors. In future work, we believe methods that visualize demonstrated trajectories in conjunction with editing mechanisms such as scrubbing, cut, copy, paste, split, fade, and join–borrowing concepts from software for video editing–can provide a more powerful mechanism for users to express their manipulation intentions.

From the success of our system, we believe that this intersection of mobile manipulation, domain specific visual programming interfaces, and learning from teleoperation will be fertile ground for future exploration.

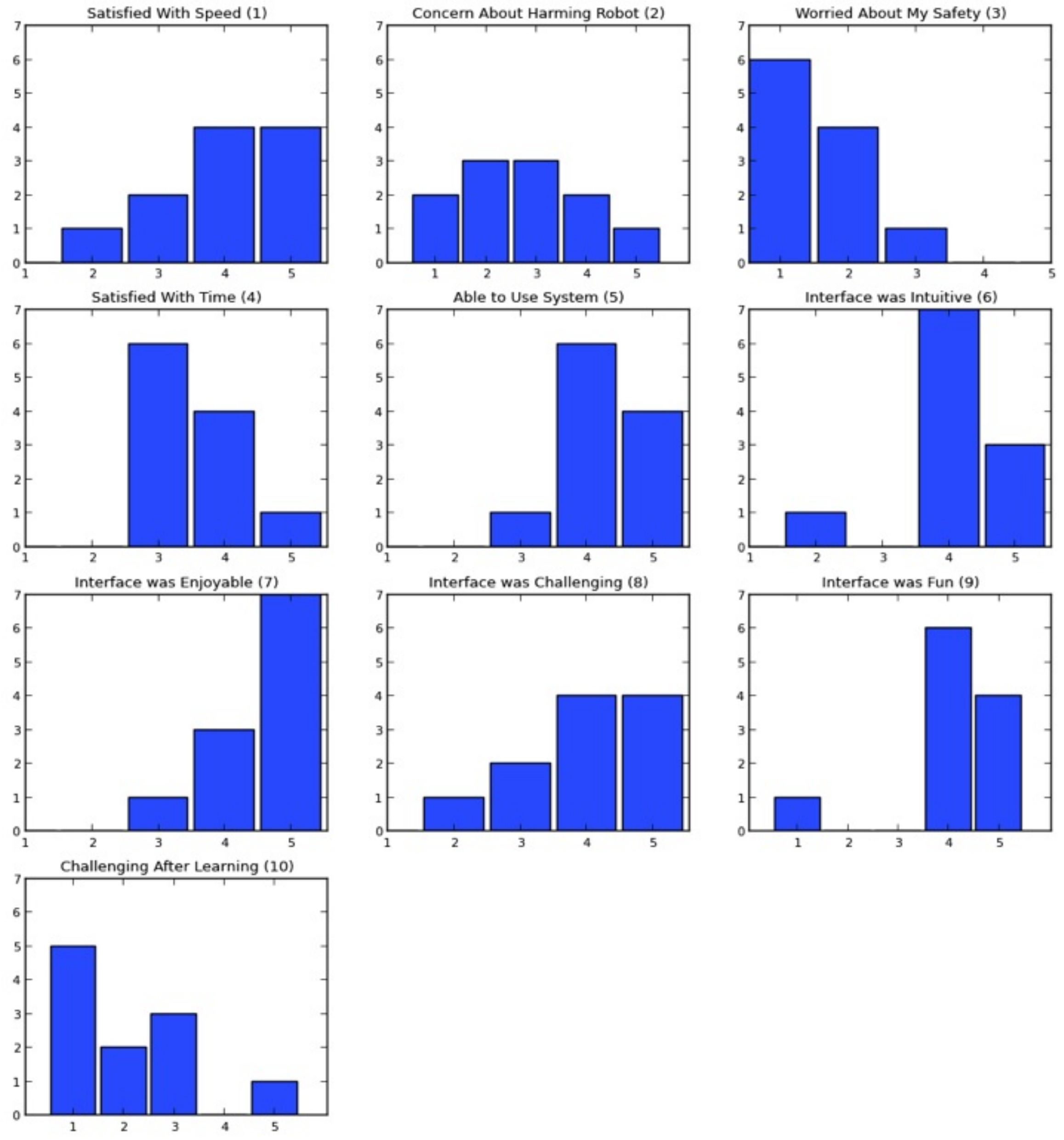


Figure 17: Participant's responses to questions administered after completing study (corresponding questions in Section 3.2.4). A score of 5 and 1 represents strong agreement and strong disagreement, respectively. Scores of 3 are neutral.

CHAPTER IV

ROSCO: USAGE BY A MOTOR IMPAIRED EXTREME USER

Assistive mobile manipulators (AMMs), such as the PR2, are unique amongst competing solutions such as stationary arms, wheelchair mounted arms and robotic exoskeletons, as AMMs are independently mobile, do not have to be attached to users, and can potentially operate without supervision. Being independent systems, i.e., not needing users to be physically attached or present, operation of AMMs place few constraints on users' physical capabilities and conditions, requiring only the basic ability to operate computing devices.

In this chapter we describe an exploratory case study where we use ROSCo behaviors in various assistive mobile manipulation scenarios to help enable Henry Evans, an individual with quadriplegia, to regain his independence and expressive abilities. Our broader research goals in working with Henry is to find roles that ROSCo can fulfill in an assistive setting, and tasks that robots can perform in this environment. While attempting these operations, we also hope to identify possible challenges with our system when operating in a home with motor impaired individuals.

Severely motor impaired as a result of a brain stem stroke, Henry is interested in mobile manipulators as potential surrogate bodies and contacted us to volunteer as a test participant and collaborator when he saw a PR2 robot demonstration of ours on a CNN news segment. The work we present is part of a larger effort kick-started by Henry's initial contact with us called the "Robots for Humanity" project [41]. The goal of this larger project is to empower people with severe motor impairments to interact with the physical and social world, thereby enhancing their quality of life and independence through the use of an AMM.

To identify important tasks to perform in assistive settings, we employed a user-centered and participatory design process where Henry and Jane, his wife and caregiver, offer feedback and ideas during interviews. We organized tasks into three separate categories: those that involve manipulation near or on the body, manipulation of the physical environment, and social interaction. From these categories, we discovered three specific roles that ROSCo could fulfill. First, we investigated using ROSCo as a social tool for Henry to create stand-up comedy routines that use the PR2 as a proxy for his body. Second, we used ROSCo behaviors as part of a human-in-the-loop, or shared autonomy teleoperation system where the human and robot share responsibility and control when interacting with the environment. In these scenarios, Henry deployed pre-constructed behaviors in his home and then used them during teleoperation. Third, we investigated whether motor impaired individuals such as Henry can use ROSCo to create novel behaviors that manipulate the environment, just as in the last chapter with able-bodied participants, potentially enabling him to use our system to automate common tasks around the home.

Although there are many alternatives for controlling robot manipulators [115, 86, 106], our interface is designed to be accessible through pointing and clicking with a conventional mouse. This choice is driven mainly by Henry's ability, and the prevalence of assistive technologies allowing the motor-impaired to use mouse input.

By working with Henry and Jane, our investigation focused on extreme users–those with outlier characteristics such as advanced age and physical impairments–to better design more universally accessible interfaces [134]. Due to having sensitive needs, users like Henry and Jane can help amplify deficiencies in our system and interfaces, potentially helping to design more robust and useful systems for more typical users [123]. As a result of his injury, Henry only has limited control of computer cursor movements through his head tracker and limited ability to click buttons on a mouse. As Jane is Henry's full-time caregiver, consideration of her needs is also paramount to the system's success since a mechanism that needlessly encumbers or inconveniences Jane will be much less likely to succeed.

Our contributions in this chapter are as follows. First, we demonstrate one of the first instances of a person with quadriplegia performing open-ended tasks using an AMM through a head-tracker. Second, we show that by using an authoring interface, after a training process, Henry was able to construct a novel mobile manipulation behavior that interacts with the environment. In a separate trial, he was also able to create gesture behaviors suitable for social interaction. Finally, we provide evidence that, even though motor impaired, Henry was able to create new behaviors after a training process. Additionally, the time taken by Henry to create behaviors in our training and testing scenarios is comparable to that of able-bodied users performing similar tasks.

In the following sections, we first discuss our interface's evolution through trials with Henry using it to create social gestures. With Section 4.2, we describe experiments using ROSCo behaviors in a shared autonomy teleoperation scenario. We then show experiments where Henry authors mobile manipulation behaviors that interacts with the physical environment in Section 4.3.

4.1 Performing Comedy Skits

Henry initially suggested using the PR2 to perform stand-up comedy skits. Despite our years of experience with AMMs, this was not a task that we had previously considered. Gestural expressions can play an important role in stand-up comedy and in social interactions in general. Gestures can also be diverse, distinctive, and personal. As such, we wanted to empower people to create their own gestures, reuse those gestures, and share and adapt gestures created by other users in their communities. Although built as a tool for creating general robot behaviors, this is a first application where we use ROSCo solely for authoring communicative gestures. As a mode of interaction, ROSCo is well suited to Henry's impairments as it uses on-screen controls instead of requiring him to physically pose the robot, as is typically done in many learning from demonstration methods.

We tested our interface over two sessions where Henry attempted to create comedy skits primarily using components that create splined trajectories and speech from text. Each test began with a demonstration, followed by an exploration period in which Henry tried out the new methods and mechanisms introduced.

After each session, we applied lessons learned to iterate on the interface. Our sessions have highlighted the importance of quick access to behaviors previously constructed during use, larger interface elements, robustness of elements to accidental clicks, and dual arm coordination. Furthermore, results from our pilot trial with Henry where he constructed creative and unexpected gestures also re-emphasize the importance of involving end-users in creating robot behaviors.

4.1.1 Experimental Setup

We carried out our tests for creating comedy skits with Henry in a room reserved for presentations at Willow Garage, the manufacturer of our PR2 robot, located in Menlo Park, California. The interface used in these tests is the same as was tested with able-bodied users in the previous chapter. In the same manner, we placed the robot in front of Henry while he interacted with it to provide real world feedback. When controlling ROSCo, Henry used a head tracker in conjunction with a computer mouse that he was able to left-click by squeezing his hand. For text input and right-clicking, Henry used “onboard”, an on-screen keyboard program for Linux.

From the library of modules in ROSCo, we trained Henry to use the arm joint trajectory, gripper, head trajectory, spine, and speech modules. The arm trajectory module allows recording sets of individual joint coordinates that can then be splined together during runtime to generate smooth gestures; the gripper module controls the gripper’s opening; states generated by the head trajectory module allows control over the pan and tilt motors, potentially useful for gestural communications; moving the spine helps control the how tall the robot appears. Vital to comedy skits, the text-to-speech module generate speech during performances.

For controlling the robot, our interface used 3D rings-and-arrows controls introduced in RViz as part of Willow Garage’s shared autonomy teleoperation interface [44] just as in the previous chapter. Rings-and-arrows displayed on the grippers, for controlling where the robot’s grippers should be, and rings around the shoulder joint, which control the arm’s redundant degree of freedom, allow users to pose the robot and capture motion key frames to create joint level trajectories. Controlling the head can be accomplished through moving a 3D sphere acting as a target at which the robot points its head.

More autonomous capabilities are also available through right-click menus. Users can command the robot to tuck and untuck its arms with context menus on the robot’s arms. Likewise, the grippers can be opened or closed completely through right-click menus. As accurately clicking items on these menus is difficult for Henry using his head tracker, we enlarged the menu options for easier selection.

4.1.2 Iterative Design Changes Through User Testing

We tested creating comedy skits with Henry over two separate sessions, each lasting one hour. We began each session by verbally giving Henry an overview of the task that we would like him to attempt then showed the interface elements that he will need for that task. In the first session, we introduced Henry to basic concepts for creating trajectories expressed as joint angles and the text-to-speech module then asked him to create a basic behavior where the robot waves and speaks a phrase. We then made changes based on observations of his usage patterns and tested the same task a second time. We discuss these changes below.

4.1.2.1 Robustness to Unintentional Clicks

Due to having limited control of his hands and the experience of strong emotions such as laughter causing whole body spasms, Henry often clicked on different interface elements unintentionally. As designed, this was problematic in ROSCo.

Originally, the basic interaction sequence for creating a new state starts with users selecting the module by clicking on that module's button, providing the resulting dialog box with appropriate parameters, then clicking "Add" to add that as a new state in the state machine. However, providing parameters can take time for users in such cases as there can be many to specify even with short trajectories. During this process, if users clicked on another state, represented as a circle, in the state machine visualization area, it would deselect the current state then select the new state but would also clear any unsaved changes made to the current state.

This failure mode proved to be frustrating for Henry. Our fix includes having the interface remember parameters of states that are in the process of being constructed to safeguard them from being lost in the event that users switch selection, either accidentally or intentionally, to another element on screen.

4.1.2.2 Faster and More Powerful Key-framing

Due to mouse control requiring considerable concentration for Henry, we made design choices to reduce the number of steps used for common operations. One such operation involves creating new key frames for representing motions. To create new key frames in our initial prototype, users would first pose the robot, click a button called "Current Pose" to update the properties display with the robot's new configuration, then click a plus button to create a new key frame from the updated information.

During tests, this three-stepped procedure that must be repeated for each key frame quickly proved to be cumbersome for Henry. We reduced the number of operations by reducing the need to click the "Current Pose" button for each new key frame. We added a new button that would turn on a mode that constantly updates the properties display, similar to the "Current Pose" button's

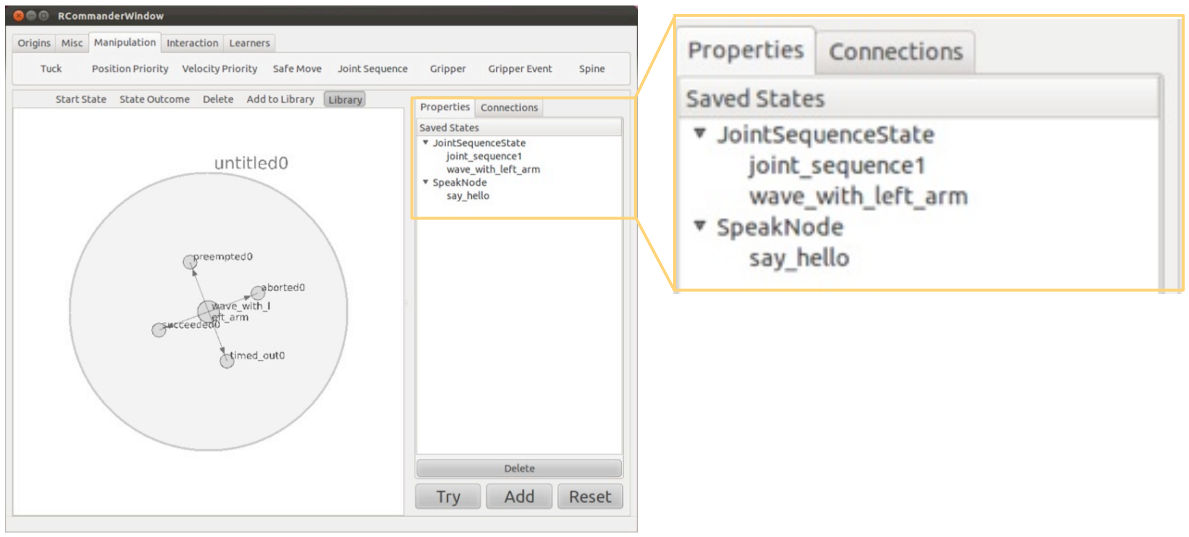


Figure 18: Users can save parameterized states in the states library for later use, similar to using libraries of pre-made parts in CAD software. Saved states here are shown by their names (e.g. joint_sequence1, wave_with_left_arm, and say_hello) and grouped according to the type of state that they are (e.g. JointSequenceState and SpeakNode).

functionality but continuously. Once this mode is activated, to add new key frames users would first pose the robot then click the plus button to add it to the current motion. This change reduced the number of steps needed to two, and helped to provide users with feedback on how different values such as the joint angles vary as the robot moves. This newly added feedback also aids users while debugging behaviors as the display also highlights in different colors joint angles that are too close to the robot's joint-limits or violate velocity constraints.

Also in our initial prototype, each sequence of key frames created could only be associated with one particular robot arm, either the left or right arm, selectable through a pair of radio buttons. These buttons, in testing, were often mistaken as controls that applied to just one key frame instead of applying to the entire sequence. After discussions with Henry, we discovered that his assumption was due partially to a number of gestures that he wanted to create needing both arms to move at the same time resulting in trying to force individual key frames to be associated with an arm different from the arm selected for the sequence. In response, we relaxed the restriction that the entire sequence has to be associated with only one arm, enabling one sequence to command both robot arms, an ability that we later used for bimanual tasks such as refrigerator opening.

4.1.2.3 Faster Access to Motion Sequences

As timing and delivery are important components of comedy, we found that it was not adequate to have simple sequences of text-to-speech and gesture states replayed using ROSCo's HFSM mechanic. This issue occurred early on when we observed that even though Henry constructed chains of states to represent his skit, he would select and command individual states to run during his demonstrations to us instead of running the complete sequence. To allow easier random access of states, we added a mechanism called the state library that allows for storing, running, and replicating frequently used states.

Shown in Figure 18, the state library is designed to appear similar to other tools in ROSCo except the properties panel shows a list of states stored instead of widgets displaying parameters. For comedy skits, we intended for Henry to store frequently used states in this library and select from them as needed then click "Run" to execute them during his performance. In other uses, the states library would function similarly to libraries of parts in computer-aided design (CAD) software, allowing users to save time constructing new behaviors by using pre-constructed states.

4.1.3 Task Results

During the second session, with the above modifications, Henry was able to construct a behavior where the robot waved and said a greeting using text-to-speech. He also used the text-to-speech tools to make jokes and the new dual-arm key framing ability to create a gesture where the robot raised one of its arms and crossed the other arm in front.

In addition to enabling Henry to construct gestures, the robustness to accidental clicks and key frame animation changes made the interface more usable for participants that tested it later on within the user study described in Chapter 3. The new two-armed key frame animation ability also enabled construction of the refrigerator opening behavior shown in Chapter 2.

4.2 Shared Autonomy Teleoperation

For assistive contexts, we used ROSCo behaviors as part of a human-in-the-loop, or shared autonomy teleoperation system [102, 48, 132] where the robot and user share responsibility and control.

With shared autonomy schemes, the teloperation system is structured such that the human teleoperator can effectively use superior task understanding and knowledge to monitor and guide the robot's autonomous systems. Those systems can also in turn provide guidance, reduce the complexity of controlling high degree of freedom systems, and enable lower latency responses to environmental stimuli. In our system, teleoperators trigger pre-constructed behaviors to perform complex but repetitive operations, a scheme known as behavior-based teleoperation [132].

Involving a human teleoperator can drastically improve the effectiveness and applicability of mobile manipulators in unstructured environments. As a potential "surrogate body" for the motor-impaired, there can be many cases where users like Henry would want to perform new tasks that the robot is not preprogrammed for or existing tasks in novel situations where autonomous capabilities fail either partially or completely. For example, researchers have encountered instances where household curtains blown by the robot's fans appear as obstacles causing its path finding algorithm to fail to find a path [44]. In another scenario, when the robot attempts to open a door, that action might fail as the door could be stuck or locked, causes that might be unknown to autonomous procedures [66]. By using teleoperation, augmenting the system with human judgement, robots can potentially perform many more tasks more robustly than they otherwise would be able.

Even though teleoperation enables robots to accomplish more tasks, performing simple operations by directly controlling complex mobile manipulators can be challenging and time consuming. Our system addresses this by building upon previous work by Ciocarlie et al. [44] that uses a tiered Human-in-the-Loop (HitL) framework for teleoperation. At the highest level of autonomy, users provide low-dimensional input intermittently to guide the robot in tasks such as segmentation of point clouds, autonomous navigation, and grasping. However, where the robot's autonomous modules fail or are inapplicable, such as during open-ended operation or more complex tasks, the framework allows users to fall back to less autonomous direct teleoperation tools. This tiered arrangement enables the system to accomplish more complex tasks robustly and also speeds up regular teleoperation using autonomous components.

Our work augments the system's existing capabilities with autonomous and semi-autonomous ROSCo behaviors to further help decrease the users' cognitive load while teleoperating, and reduce the time required to execute tasks. By using autonomous behaviors our system is also robust to factors such as time delay during teleoperation [132]. We tested our system with our pilot user, Henry Evans, in a number of different tasks. In the first test, he fetched a bottle from a refrigerator then placed it down on a table. We also had Henry open a drawer then, with separate systems, fetch a towel and bring it back to wipe his face. Besides performing multi-step tasks, we tested deployment and execution of pre-constructed ROSCo behaviors separately including turning off the lights, fetching an object from another person, placing a napkin in a trash can, and operating a door (both opening and closing).

Beyond having Henry use our system in his home, our research goal in this section is to discover the challenges associated with using behavior-based manipulation in a motor-impaired person's home. We pursue this goal by attempting to perform a wide range of tasks. To our knowledge, this is the first instance of a mobile manipulator being used in a real home to perform the tasks tested.

In the next section, we will describe our system setup for testing with Henry. In Section 4.2.2, we review our trials with Henry in his home. We will then end this section with some concluding remarks.

4.2.1 System Overview

Our interface is a modified version of Willow Garage's tiered human-in-the-loop (HilT) teleoperation package [44]. We describe here the subset of tools made available during teleoperation and those that we introduced that enable users to activate ROSCo behaviors.

Shown in Figure 19, the interface we expose is similar to that used in [44]. At the top left corner is a live video view through the PR2's head-mounted Kinect sensor, a depth sensing camera. Below it is ROSCo's web interface listing behaviors that the robot can execute in hierarchical menus. The right-hand side shows the same RViz teleoperation interface as was used in Section 4.1.1. In the previous section, we introduced controls for the end-effector position, spine, gripper and head. We now introduce several RViz interface elements not used when creating gestures for comedy skits.

One new set of elements enables users to activate the built-in autonomous navigation module and performs fine base movement controls. To use autonomous navigation, users would drag a representation of the robot's model within RViz to a new position and orientation. This sends a goal to the PR2's navigation stack [93] to plan a collision-free path, if possible, through a previously

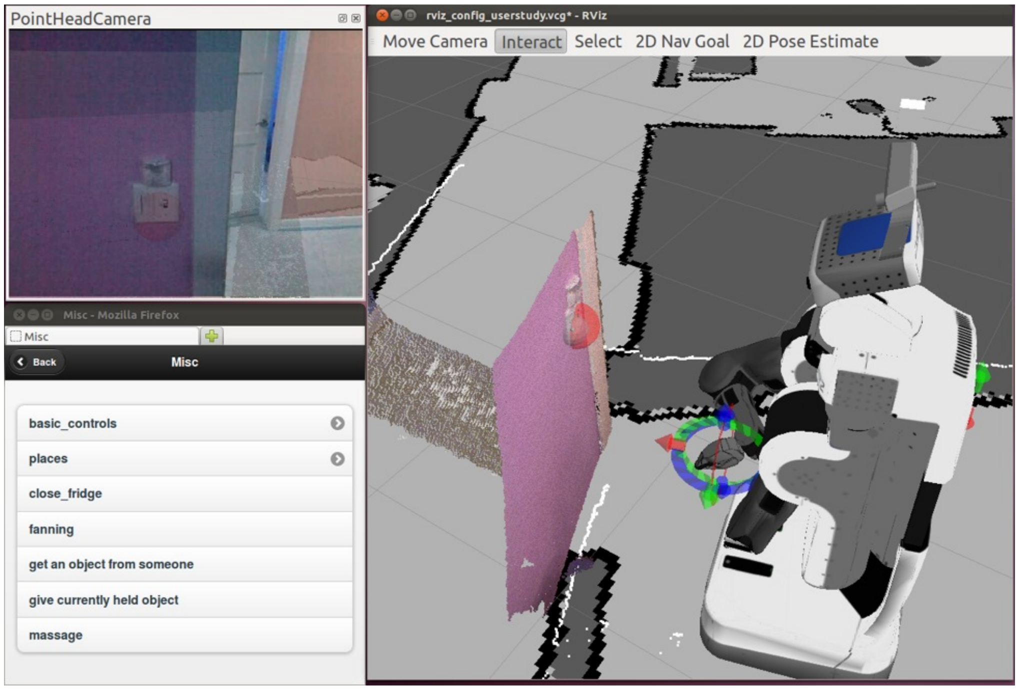


Figure 19: Shared-autonomy teleoperation interface that Henry used during our runs. ***Right****: Rviz display where the robot can be controlled through on-screen controls.* ***Top Left****: video from a camera on the robot's head.* ***Bottom Left****: Web interface for accessing ROSCo behaviors.*

constructed 2D map and navigates to the goal while avoiding obstacles. As interacting with objects often requires the robot to be close to those objects, usually closer than the autonomous navigation module obstacle avoidance behavior allows, control rings and arrows around the robot's base let users exert direct control when autonomous navigation refuses to approach objects.

Another set of controls helps teleoperators gain situational awareness through manipulating the robot's sensors. Right-clicking on the robot's head exposes menu options allowing users to ask for a 3D point cloud snapshot. When displayed, this snapshot lets users examine the scene being manipulated from virtually rendered viewpoints that can be more natural to the task than viewpoints provided through the live video view.

We additionally added controls to interact with ROSCo's behaviors during teleoperation. We visualized detected ARTags as green spheres that allow Henry to create new task frames, represented as rings and arrows, and associate them with pre-constructed ROSCo behaviors, described in Chapter 2. These task frames represent a rigid position and orientation offset from the tag to the

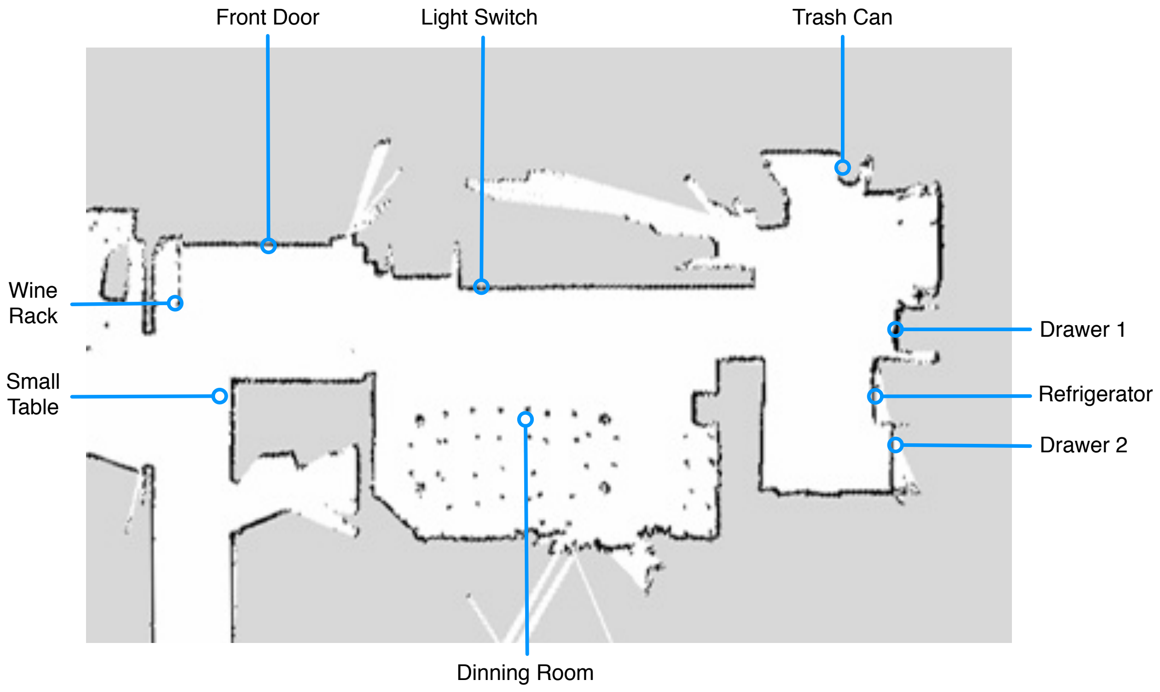


Figure 20: A map of Henry's home made using the slam_gmapping package with the robot's base laser scanner. This cropped map shows locations where events in this experiment took place. At center is the dining room with chair and table legs appearing as dots. The kitchen is on the right side of the dining room.

functional component of the mechanism operated such as door or drawer handles. After being associated with a frame, behaviors can be executed and stopped through the displayed web-interface.

These interface elements work together allowing mixing of teleoperation and autonomous operation, potentially improving the performance of both. For instance, behaviors can sometime miss the task-relevant component of the mechanism manipulated, for instance, failing to grasp a handle or make contact with a button, causing the task to fail. In such cases, however, the teleoperator can adjust that deployed behavior's task frame through its rings-and-arrows controls, then test the behavior again.

4.2.2 Demonstration Results

Our trials with Henry teleoperating the PR2 took place at his home in the San Francisco Bay Area. The runs we describe here occurred over three different visits where we traveled from our lab in Atlanta, Georgia to Henry's home. For 2D navigation, we used maps made of his home by Willow Garage employees as part of the "Robots for Humanity" project. Figure 20 shows locations where events in our tests took place. At the center of the map is the dining room with chair and table legs appearing as dots. The kitchen is then to the right of it. The small passage way between the kitchen and dinning room is slightly larger than the PR2's foot print.

During each run, an experimenter stands next to Henry to give instructions and suggestions during task execution. In many cases, however, Henry elected to perform tasks using methods that he prefers or perform sub-tasks that he was more interested in.

4.2.2.1 Fetching an Object Inside a Refrigerator

As one of the goals of our project is to enable motor-impaired individuals to live independently, we determined through interviews with Henry and Jane that being able to fetch edible items, such as a cup of yogurt, from a refrigerator can be of help in daily activities. Currently, if Jane is away, Henry would have to wait many hours for her or another person to return home for assistance. With this demonstration, we show progress towards motor-impaired users independently fetching objects out of refrigerators through a teleoperation system that uses ROSCo behaviors to execute subtasks.

Before starting our trial, we attached ARTags to Henry's kitchen refrigerator, and a small table in his living room, shown in Figure 20. In addition, we modified an existing refrigerator opening behavior, designed previously for a larger refrigerator door that opens to the left, to work with Henry's refrigerator, which is smaller and opens to the right. This modification involved recording new trajectories for the pulling action in the behavior. We also constructed a complementary refrigerator door closing behavior. Having these behaviors available simulates a situation where users who do not have time or knowledge to create their own robot behavior can access or download behaviors constructed by other users stored in large databases.

Henry began the trial by deploying the above refrigerator opening behavior. He attempted placing the behavior's task frame at a number of locations initially that missed the refrigerator handle but was able to later target a location that worked. While opening, the behavior pulled using both of the robot's arms in order to apply enough force for opening the refrigerator's door. This is a difficult task as just one of the PR2's arms did not apply enough force. From previous work we have shown that refrigerator doors can require upwards of 40 N [70] to open due to magnetic forces designed to keep their doors shut.

After opening, the behavior held open the refrigerator door with the robot's right arm allowing the left arm to be free for manipulation and grasping. Next, Henry captured a snapshot of the scene inside the refrigerator, then grasped a sauce bottle placed in a cluttered shelf on the door using direct teleoperation through rings-and-arrows controls displayed on the left gripper. He then closed the fridge using a ROSCo preprogrammed door closing behavior.

With the bottle in the robot's left gripper, Henry used autonomous navigation tools to move it to a small table in his living room (location shown in Figure 20). However, as the robot moved closer to the front door of the house, noisy laser detections near a wine rack close to the door caused it to not find a path. In the process of replanning and turning around, its base became entangled in Henry's door mat. As it failed next to fragile wine bottles, we intervened to free the robot and moved it past Henry's doorway closer to the goal. After this intervention, Henry deployed a pre-constructed object placement behavior. This deployment placed a task frame, which specifies where to place in this behavior, on the table's surface. Finally, Henry activated the placement behavior, placing the sauce bottle on the table.

4.2.2.2 Grasping a Towel

Another scenario we investigated involved Henry fetching a towel from a drawer using the PR2 to wipe his face. For the overall "Robots for Humanity" effort, this task provides an opportunity to integrate separate functional components developed by different research groups. It also demonstrates the utility of ROSCo behaviors in a larger shared autonomy context.

Prior to the performing the task, we placed an ARTag on the front face of Henry's towel drawer and deployed a drawer opening behavior, placing its task frame in the middle of the drawer's handle. We situated Henry and the robot in his dining room, which is adjacent and connected to the kitchen, where the towel drawer is located (Figure 20).

Henry started the task by commanding the robot to autonomously navigate from the dining room to the towel drawer. He then opened it by calling on the drawer opening behavior using ROSCo's web interface. This behavior started with grasping the drawer handle, then pulling with the PR2's arm and base. In this case, the base is needed as the drawer is custom constructed and requires large forces to open. After opening the drawer, Henry grasped a towel inside it using direct teleoperation

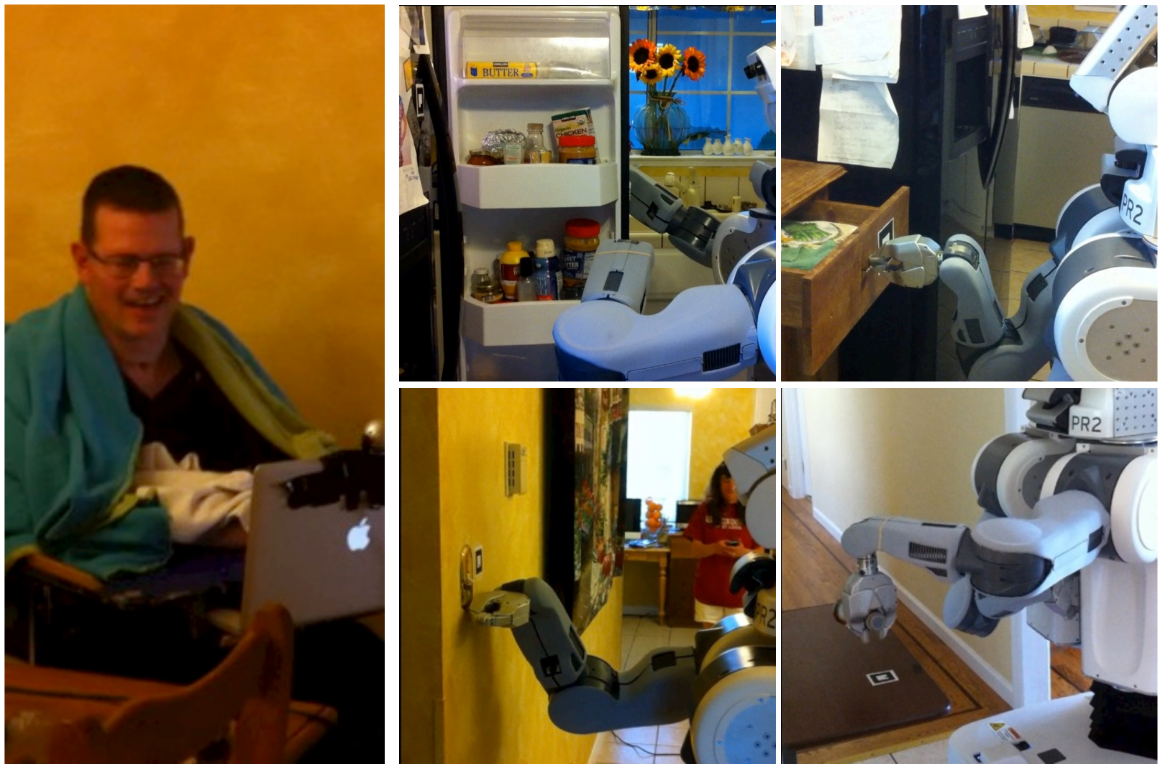


Figure 21: Behaviors executed by Henry Evans (leftmost image) at this home: opening a refrigerator, opening a drawer, turning on the lights, and placing an object.

with rings-and-arrows on the robot's gripper. Next, he closed the drawer by pushing on it with the robot's forearm using direct teleoperation. Then, Henry tucked in the PR2's arms by calling on an autonomous behavior. This prevents the arms from protruding outside the robot's mobile base during navigation.

Afterwards, Henry commanded the robot to autonomously move next to his wheelchair in the dining room. At this point, we had Henry switch to a different interface and system that allows the robot to manipulate around his body [41]. With these tools, Henry was able to command the robot to bring the towel close to his face and wiped by rubbing his face against the robot-held towel.

4.2.2.3 Standalone Behaviors

In addition to testing full aggregate tasks such as the above, we have also tested a number of standalone ROSCo behaviors at Henry's home. For each trial, we first described to Henry what the behavior does then instructed him to deploy and execute it.

Henry primarily had difficulty with positioning the task frame for each behavior in our 3D interface. As the robot's calibration is not precise in some parts of its workspace, placing the task frame according to the Kinect point cloud does not always result in successful execution.

In these cases, we instructed Henry to iteratively adjust the task frame's location based on where the robot physically made contact when the behavior was executed. However, as Henry has to be able to see the physical robot and mechanism interacting, which was not always possible due to where he was situated, we gave guidance by observing the behavior's execution then communicating to him where the robot made contact.

We tested behaviors from Chapter 2 that were previously executed in the Aware Home, a home-like testing environment, including: flipping a light switch, pulling a drawer, opening a refrigerator, and handing off an object. We used nearly identical behaviors at Henry's home with some minor differences. The drawer opening behavior at Henry's needed the base to also move backwards to pull as his custom constructed drawer is not mounted on slide rails. Additionally, our refrigerator closing behavior had to be adjusted to accommodate his smaller door. While we are not yet in a position to make strong claims for our behaviors' ability to generalize based on this qualitative data, our experiences suggest that many behaviors constructed using ROS Commander can be used in different homes with minor changes only to one or two states. Of behaviors previously tested, we used the light switch and object hand-off behavior without modifications. From our tests at his home, we were able to enable Henry to successfully open a refrigerator, open a drawer, turn on a light switch, and place an object on a small table. We show the robot operating in Figure 21.

In addition to testing behaviors previously tested at the Aware Home, we took advantage of the unique opportunity to work at Henry's home to create and test new behaviors suited to his living situation. These included: retrieving an object from a person, placing an object in the trash can, and operating the front door.

The object retrieval behavior uses face detection to find a face, then raises the robot's arm so that the gripper is a certain distance from that face. Next, the robot asks for an object to be placed in its gripper using text-to-speech and waits for an acceleration or tactile sensor signal to close the gripper. After either a fixed amount of time or detecting that an object is in its gripper, the robot closes its gripper, grasping the object handed to it. During testing, Henry used this behavior to successfully fetch a piece of junk mail from Jane, his wife.

Complementing object retrieval, we also created a behavior that will transport an object to a trash can and drop the item inside. This behavior, designed to pair with the object retrieval behavior, assumes that the robot has an object in its gripper. It navigates to a location defined in a user defined task frame, raises its arm then opens the gripper to drop the object carried. We attached an ARTag to the top of Henry's trash can and then instructed him to position the behavior's task frame at its opening. After multiple attempts of positioning the task frame, Henry was able to throw away a used napkin on the fourth attempt. In the initial failure cases, the robot was able to drop the napkin while its gripper was inside the trash can but the napkin was pushed out by the trash can's lid.

We additionally made a behavior to open his front door and tested it by simulating a situation where Henry attempts to greet another person at his door step. However, we faced some challenges in constructing a behavior that operates his particular door. The front double-door had a door knob that rotates continuously, not stopping when reaching its limits, and resets the state of the door's latch when it has been rotated too much. Additionally, the PR2's parallel jaw gripper with its flat, non-compliant finger tips is also ill-suited to grasp round, smooth, and metallic doorknobs. After a few attempts, our final solution grasped the door knob from the side to twist it, then let go and re-grasped the knob from a different angle so that the robot's arm could pull the door open. After opening the door, the behavior holds on to the door knob so that if needed users can trigger a door closing behavior that assumes the robot has already grasped a door knob.

For testing, we attached an ARTag to the door adjacent to the door to be opened and instructed Henry to place the behavior's task frame on the target door's door knob. After 4 tries, Henry was able to position the task frame such that the robot grasped the door knob and was able to open then close the front door on command. In most of the failure cases, the robot missed the door knob by a small amount or slipped off the knob after grasping it.

4.2.3 Concluding Remarks

From these runs we have discovered many complexities inherent for robots operating in homes. Curtains and carpets can appear as obstacles; doors can have knobs that are difficult to grasp; fragile furniture such as wine racks can need special considerations. Additionally, lighting conditions can often be extreme with certain regions dimly lit and others in bright sunlight. While discovering

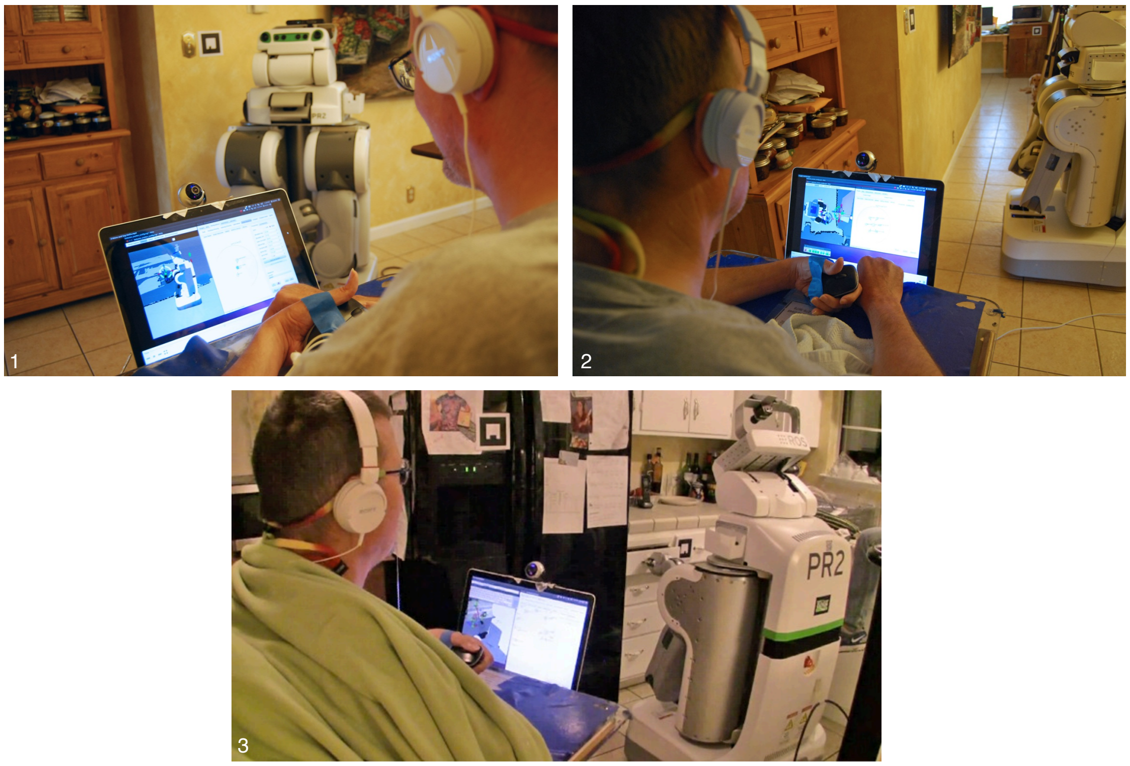


Figure 22: Henry constructing a (1) waving, (2) light switch and (3) drawer behavior.

these difficulties is informative, it is encouraging that we were able to use our shared autonomy teleoperation system to accomplish tasks that would otherwise not have been possible.

4.3 Authoring Behaviors that Manipulate the Environment

With the exception of creating gestures for comedy skits, Henry has primarily been a consumer of pre-constructed ROSCo behaviors. In this section we test in a more structured experiment to see whether Henry can create behaviors that interact with the environment without expert assistance.

To do so, we ran the same tests with Henry as we did with able-bodied users in Chapter 3. Due to time constraints and limitations in Henry's stamina we elected to run only a reduced version of our study from the previous chapter. Even so, results of this experiment can give us clues on differences between difficulties faced by able-bodied users and Henry when using ROSCo to create new behaviors.

4.3.1 Experimental Procedures

Just as with other trials in this chapter, we ran our experiment in Henry's home, picking mechanisms and locations that preserved the spirit of the original study intent as well as possible. We used the same procedure as in the previous chapter and the same script for administering the experiment. We had two sessions: training and testing. In the training session, we informed Henry that he could ask any questions when stuck from our "Help Desk". With the testing session, we gave Henry access to the same training materials, both videos and reference manuals, but no "Help Desk" assistance.

However, we skipped sections that are not relevant with Henry. In the previous chapter, the training session consisted of four modules: safety, interactive manipulation, constructing a behavior to wave and say hello, and constructing a behavior that flips light switches. Through his involvement in our efforts, Henry is familiar with the PR2 and its 3D teleoperation interface, more so than any of our previous experimental participants, enabling us to omit training modules on safety and interactive manipulation, which teaches users now to operate the 3D camera and teleoperation controls.

We started our experiment with the third module and asked Henry to watch and follow along with a tutorial video. The video describes how to construct a behavior that performs a gesture then activates text-to-speech to utter a greeting. Even though Henry has performed similar tasks while constructing comedy routines, this served as a review session and helped to teach interface features that have been introduced since he used it last. In this trial, we placed Henry in the dining room with the robot facing him, mimicking the setup used in the previous experiment.

In the next module, we asked Henry to construct a behavior to flip a light switch with the guidance of an instructional video. The trial began with Henry facing the side of the robot while the robot faces the light switch in his living room. We also started Henry with a stub behavior, that when correctly initialized, autonomously navigates the robot to the task frame he defines for the light switch.

After these two training sessions, we omitted the review session that was in the original experiment as the testing session was performed on the same day. We proceeded to asking Henry to construct a behavior to open the drawer in his kitchen, the same drawer used in the previous section.

In this case, we placed the robot in front of the drawer and Henry facing it from the side. We also started Henry with a behavior stub. The experiment then concluded after Henry demonstrated the complete behavior to us. As he was fatigued at this point and needed to rest, we sent a link to our post-experiment online questionnaire and had Henry complete it when he was able. We show in Figure 22 Henry creating behaviors using ROSCo during our experiment.

4.3.2 Results and Remarks

Despite the obvious challenges, Henry was able to complete all our sessions by successfully demonstrating fully functional behaviors. Surprisingly, he was able to finish the sessions in time competitive with that of able bodied participants. Henry took 37 minutes 23 seconds, 57 minutes 51 seconds, and 38 minutes and asking 4, 2, and 3 questions to create behaviors for the waving, light switch, and drawer task respectively. As a point of reference, the maximum times taken by able-bodied participants for similar tasks are 37 minutes 3 seconds, 1 hour 9 minutes, and 56 minutes 12 seconds.

In the questionnaire, Henry commented that the “circles” or state representation was confusing and that “it [the interface] has a steep learning curve.” However, he also said, via email, that “it’s a very precise/powerful application [its like AutoCad v PowerPoint - harder to learn but much more powerful]” and that “I like the RCommander more each time i use it.” Such reports and comments provide encouraging results that interfaces such as ROSCo has the potential to empower motor-impaired individuals to customize and create robot behaviors for their own needs in the same manner as able bodied individuals. It also illustrates that although authoring interfaces such as ROSCo can be complex, the gains in terms of flexibility and power can help offset their downsides.

4.4 Conclusion

In this chapter we introduced usage of ROSCo in an assistive setting through an exploratory case study where we first observed Henry Evans, a motor impaired individual, use ROSCo to construct state machines for comedy skits. From this experience, we identified interface factors needed for Henry to perform this task such as being robust to receiving accidental mouse clicks, reducing the number of operations needed to create individual key frames in motions, and making motion sequences easily accessible during performances.

We also performed tasks identified by other researchers in the “Robots for Humanity” project as potentially worthwhile for robots to provide assistance with. Our behavior opened a refrigerator so that a bottle could be retrieved from it using teleoperation. In a task where the robot fetched a towel so that Henry could wipe his face, our system was used to open the drawer. Additionally, our behaviors were also able to operate a number of different mechanisms in Henry's home. Through performing these tasks, we showed the potential for ROSCo behaviors to function as helpers that autonomously execute subtasks during complex teleoperation procedures.

From our shared autonomy teleoperation runs, we also discovered that there are numerous, varied and difficult to categorize challenges with robots operating in a home environment. Curtains, carpets and noisy obstacle perception can cause serious problems with the current navigation module. For manipulation, difficult-to-grasp door knobs, hand-built drawers, and cluttered refrigerators can foil autonomous manipulation attempts. At the same time, we show that it is also possible to mitigate many of these issues by having a human teleoperator in the loop and trained user to adapt the pre-behaviors.

Tests similar to our user study in Chapter 3 with Henry are suggestive that it might also be possible for motor impaired users to author robot behaviors. The time taken to construct behaviors, issues encountered, and questions asked in these tests does not appear to be unique nor attributable to Henry's motor impairment.

For behavior authoring systems such as ROSCo, in future work, we believe that further development following an interactive process of empirical testing with a user in the loop to identify and address issues, can produce robust systems enabling individuals with motor impairments to perform tasks independently. Of particular focus, infusing normally autonomous components such as navigation or grasping with the ability to fail gracefully then ask for user assistance (e.g. [126]) has the potential to address many of the challenges that our system have encountered.

CHAPTER V

AUTONOMOUSLY LEARNING TO VISUALLY DETECT WHERE MANIPULATION WILL SUCCEED

Informing robot manipulation with computer vision continues to be a challenging problem in human environments such as homes. With homes, two types of challenges are particularly notable. First, the robot must handle wide variation in the appearance of task-relevant components of the world that can affect its ability to perform tasks successfully. Lighting can vary from home to home and from hour to hour due to indoor lighting and windows. In addition, important components of household mechanisms used during manipulation, such as drawer handles and switches, can be distinctive or even unique. The perspective from which a mobile robot observes the component will also vary.

Second, the relationship between the appearance of task-relevant components and the success or failure of a manipulation behavior is complex. For example, the mechanics of a specific device may require that the robot act at a distinct location, such as a particular drawer that needs to be pushed in the center to be closed, or a convoluted handle that the robot's gripper can only grasp at particular locations. The robot itself may also change over time and thus alter the relationship between visual appearance and a manipulation behavior, as parts of its body settle, deform, and wear.

One potential solution to these two problems is for robots to learn how specific objects respond to manipulation attempts using a behavior, and to continue to learn as they perform tasks. By using data generated through their actions without human intervention, robots can autonomously learn direct mappings from visual features to the input parameters for behaviors, enabling robust execution despite errors in calibration, pose variation, sensor noise, unexpected environmental interactions, and other factors. By continuing to learn over time, robots can also adapt to changes in the environment, the objects, and their bodies.

In previous chapters, we have explored collaborating with expert users to create robot behaviors. While visual perception, with its versatility and potential as a high bandwidth information channel, is a fundamental component of many robot systems, creating vision systems that operate robustly

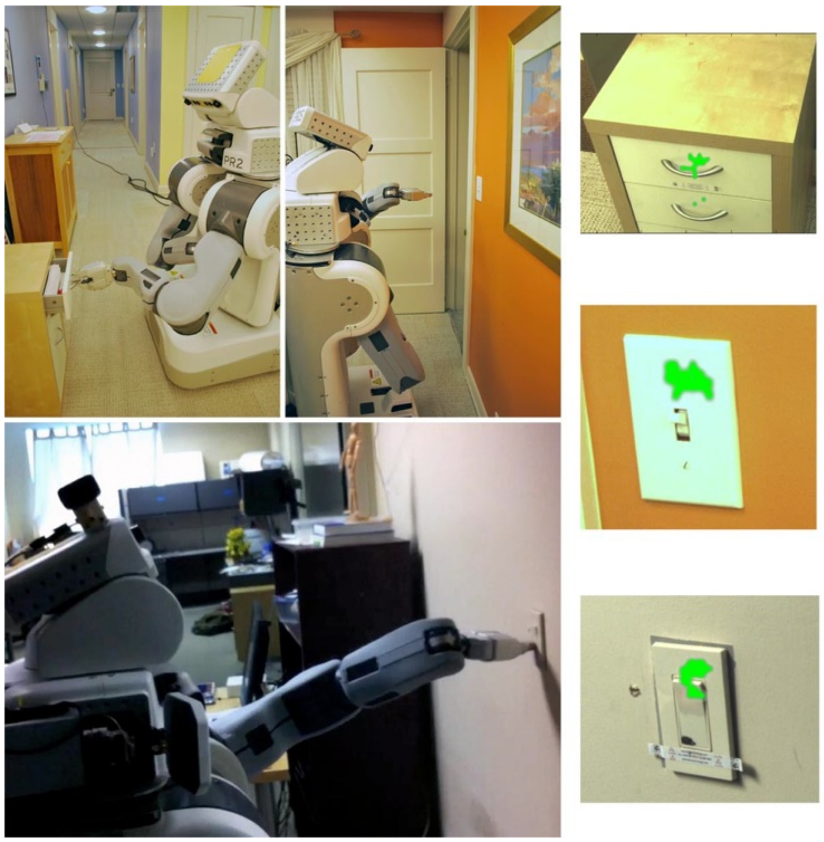


Figure 23: ***Left:*** *Willow Garage PR2 operating a drawer, light switch and rocker switch using learned detector that detects regions where manipulation will succeed.* ***Right:*** *Results from learned detectors during execution.*

in real world settings can be challenging. For approaches using machine learning, this often means finding creative methods to gather and label data for training robust detectors. Complementary to other tools in ROSCo, the system we present here aims to reduce the amount of user effort required for collecting and labeling data for constructing robust visual detectors. In this chapter, we learn initially with our robot using hand-programmed behaviors. In the chapter following, we will then replace these behaviors with ROSCo behaviors constructed by users in Chapter 3.

Our work advances robot learning in three ways. First, our research addresses challenges associated with learning in scenarios that involve both mobility and manipulation. While our system does not intentionally vary the robot's base position prior to manipulation, standard navigation methods can result in significant pose variation that jeopardizes task success. We show that this issue can be resolved by directly using the robot's mobility during learning to account for this and other sources of pose variation. Notably, our methods result in predictions that implicitly account for complexities that can arise from this variation, such as alterations in the mechanism's appearance due to the robot's viewing angle, the arm having difficulty reaching a location due to the base position, or a grasp not functioning reliably due to the gripper's angle of approach.

Second, we show that autonomously learning to visually predict where a behavior will be successful can be tractable, requiring no more than a few hours to learn in real-world scenarios. By using active learning, the robots in our tests learned each visual function after fewer than 150 interactions with each device, even though the robot started from scratch and only used data it collected. The learned visual functions enabled the robots to successfully operate the devices and also have intuitive interpretations.

Third, our methods autonomously learn visual mappings for devices that have an approximately binary state, such as a light switch being up or down or a drawer being open or closed. This presents a challenge, since the robot's actions change the state of the world, which deters the robot from trying the same action again. For example, it would be difficult to learn to open a drawer if, once it is open, the robot is unable to close it. Our system addresses this difficulty by simultaneously training pairs of behaviors and alternating between them as necessary. We also formalize the ideal relationship between these pairs of behaviors and name them complementary behaviors.

We evaluated our system using an implementation on a Willow Garage PR2 robot [12] at the Aware Home, which is a free-standing house at the Georgia Institute of Technology constructed to test new technologies. First, the robot learned to operate 6 devices. After learning, we tested the robot's performance in trials with each of the 6 devices for a total of 110 trials. In all trials, the robot autonomously operated the device successfully after at most two attempts. If the first attempt failed, the robot detected the failure and then retrained using this new negative example prior to trying a second time. We tested opening and closing drawers, turning on and off light switches, and turning on and off rocker switches. Figure 23 shows example output from the resulting trained classifiers, which classify image feature vectors as being associated with success or failure of a behavior.

5.1 Related Work

In this section, we discuss related work in robot learning, robot manipulation, and active learning. Our work also builds on our earlier workshop publication [110].

5.1.1 Robot Learning

Different robot learning methods such as imitation learning, interactive learning and developmental learning [91, 118] can be grouped by how they approach the issue of gathering data. We focus on work in which the robot learns with little human input.

5.1.1.1 Autonomously Learning Robot Skills and Behaviors

Learning from demonstration typically relies on substantial human involvement to acquire training data from sources such as teleoperation, shadowing, placing sensors on the demonstrator, and external observations [20]. Robot reinforcement learning often involves substantial practice by a real robot, during which the real robot acquires training data. However, implementations in this area often focus on acquiring and refining a skill, rather than applying a known skill in a new context, and typically use specially engineered environments and human intervention during training [65, 80, 115].

Many developmental learning systems [90] use data from the robot's autonomous interactions with the environment. Much of the work to date has investigated sensorimotor coordination skills such as gaze control [30, 31, 36], reaching [100, 35], pointing [94], and poking [99]. Our work focuses on complex multi-step behaviors that have direct applications in domestic settings.

5.1.1.2 Learning to Visually Detect Grasps

Grasping, being a fundamental skill for mobile manipulators, has received significant attention in robot learning. Similar to our work, many have investigated associating visual features with 3D locations on objects to localize grasp frames. One of the earliest investigations in grasp learning is by Dunn and Segen [56] that matched objects using visual features and learned grasps through trial and error. Instead of learning one grasping classifier, Zhang and Rossler's system [142] learned separate classifiers for grasp position and grasp orientation. Saxena et al.'s [127] method learned a classifier using a data set of simulated grasps and was successful in grasping objects in uncluttered environments. The authors in [103] views the same problem as one of learning object affordances and proposed a method for estimating grasp densities in images of objects on uncluttered backgrounds. Researchers in [57] learned a mapping from 3D edge features using active and semi-supervised learning. For most systems discussed, the authors gathered data manually with the exception of [127] where a simulator was used. In contrast, we present a system that can function with a variety of different behaviors, without the need for custom simulators, and in settings where hand-labeled data are not available.

5.1.1.3 Autonomously Learning to Perceive

In contrast to motor learning, most work in learning for perception relies on data captured manually [119], captured in simulation [78, 127], or downloaded from the web [14, 40]. Although large data sets can be collected from these sources, data generated can be biased [140] and may not match what the robot will encounter. Likewise, the relationship between these data and the robot's actions may not be clear. For example, a good location for a person to grasp or a location that a person believes would be good for grasping may not be appropriate for a particular robot. Accurate simulation of physical objects can also be hard to obtain [17].

The system that we present uses data generated from self-experience, (similar to [125, 57, 115], and [80]). This has the advantage of training data that is well-matched to the particular robot and its task. However, obtaining labeled examples can be difficult, since the robot needs to act in the real-world and human labeling can be labor intensive and have errors, ambiguity, and inconsistencies [24]. We address this issue in our work by combining active learning, which reduces the number of examples needed, with autonomous learning methods that eliminate the need for human labeling beyond an initialization process.

Past work in learning for perceptual categorization, a process where agents learn through interaction with the world to divide sensory information into distinct groupings, has used data from the robot's experience. However, most systems were designed to classify simple geometric objects such as cylinders and rectangles using cross-modal information [83, 45, 43].

A relatively small subset of work investigates more complex objects found in human environments. For example, Stober et al. demonstrated an approach for extracting spatial and geometric information from raw sensorimotor data [133]. Kraft et al. [82] presented a system that gradually learns object representations and associates them with object-specific grasps. Katz and Brock in [75] showed a method with which a robot determines the structure of articulated objects through experimentation. And, Hoof et al. [141] presented a system that selects maximally informative actions to segment tabletop scenes.

Paolini et al. [114]'s system uses a generative approach to estimate task success by estimating the poses of grasped objects with sensor readings then combining it with a model of task success given the estimated pose. In contrast, our approach uses discriminative modeling mapping straight from sensor readings to a label correlated with expected success. While generative modeling often resulted in a more interpretable model, discriminative modeling allows our system to be more agnostic of the particulars of behaviors used.

Previous work by the authors of [136] is notable for its similarity to our approach. They presented a system that uses an uncertainty sampling scheme to actively learn the appearance of doorbell buttons. In contrast, our approach uses a different active learning algorithm, works with a mobile manipulator operating in situ devices, and handles persistent change to the state of the world.

5.1.2 Task-Relevant Feature Detection

In a parallel thread to robot learning, there has been recognition in the mobile manipulation community of the importance of exploiting task structure to reduce the complexity of operating in the real-world [76]. Work in articulated object perception [75], tool tip detection [77], door handle detection [78], behavior-based grasping [67], use of a sink [112], and corner detection for towel folding [92] suggests that low-dimensional task-specific perception can be highly-effective and that recovery of complex representations of the state of objects prior to manipulation is often unnecessary. These particular examples have used hand-coded and hand-trained feature detectors. With our approach, robots autonomously learn, after a human-aided initialization period, to classify visual features as being relevant to the success of a specific behavior or not.

5.1.3 Active Learning and Curiosity Driven Learning

With many robot learning scenarios, unlabeled data can be readily acquired but labeling the data is costly, a issue that can be addressed by approaches such as active [131] and semi-supervised learning. In our work, use active learning to pick data to be labeled based a data point's value in improving the learner's model. With many active learning algorithms, at each iterative learning step the learner is given an option to select a data point to be labeled out of a set of unlabeled data points. For one class of proposed approaches, the learner picks the data point whose label it is most uncertain about [87, 49, 130]. With disagreement-based methods, learner ensembles select the data point they most disagree on [47]. More computationally demanding methods, however, attempt to explicitly minimize future expected error or variance [124, 29, 85]. There are also proposals to combine semi-supervised and active learning to exploit structure in unlabeled data [96, 105, 143]. Although there have been several large scale studies of active learning methods on different data sets showing its superiority over randomly picking data points for labeling [81, 128, 130], the best active learning algorithm to use in each circumstance has been application specific. In our work, we use a heuristic that picks the data point closest to the decision boundary of a support vector machine (SVM) for labeling, a method that has been shown to perform well in a variety of applications [71, 129, 139].

5.2 Approach

Our approach enables a mobile manipulator to autonomously learn a function that takes a 2D RGB image and a registered 3D point cloud as input and returns a 3D location at which a manipulation behavior is likely to succeed. To do so, it requires a pair of manipulation behaviors, verification functions that detect when each behavior has been successful, and an initial hint as to where one of the behaviors will be successful.

Each behavior must have input parameters that correspond with a 3D location that specifies where the behavior will act. During training, our system executes each behavior multiple times using different 3D locations around the device being manipulated and records whether or not the behavior succeeded at each location. For each 3D location, the system creates an image feature vector using an area of the registered 2D RGB image associated with the 3D location. These image feature vectors are labeled with whether or not the behavior succeeded or failed at their associated 3D locations. In other words, the collected data set consists of positive and negative examples of image feature vectors that were or were not associated with the success of the behavior. With a classifier trained from this data set, the robot can then predict if the associated behavior will succeed at a 3D location based on the image feature vector associated with the location.

To avoid user intervention during training, our procedure trains two behaviors at the same time, switching to the other behavior when the current behavior succeeds. This enables our method to operate devices that can be approximated as having two binary states, such as a drawer being open or closed. Using a pair of behaviors allows the robot to change the device back and forth between these two states, so that training can continue autonomously. For example, instead of training a drawer opening behavior in isolation, our process flips to training a drawer closing behavior when opening succeeds and vice versa until the classifier converges. We also formalize the relationship between the two behaviors and define them as complementary behaviors.

Using self-generated data takes considerable time, since each labeled image feature vector requires that the robot execute the behavior at a 3D location and observe the results. To avoid needing an intractable number of trials, our method uses active learning to execute the behavior at an informative 3D location at each iteration. Specifically, our procedure trains a support vector machine (SVM) after each trial using the current labeled data. It then uses a heuristic proposed by Shohn and Cohn [129] to select the unlabeled image feature vector that is closest to the current SVM's decision boundary to be labeled next. It then executes the behavior at the 3D location associated with this image feature vector.

Our training procedure has two phases. The first is an initialization phase where the user selects the behavior pair to train, gives a seed 3D location, and positions the robot's mobile base for training. The next phase is an autonomous phase where the SVM active learning procedure runs until the learner converges. After convergence, each behavior has a classifier that predicts 3D locations where it will succeed.

During runtime, if the behavior's verification function detects a failed attempt, our procedure appends this negative example to the data set, retrains the classifier, and tries again using the output of this new classifier (Section 5.2.3.3).

In the following sections, we discuss the requirements of our learning procedure 5.2.1, properties of complementary behaviors (Section 5.2.2), our training procedure in detail (Section 5.2.3), and classification infrastructure (Section 5.2.4).

5.2.1 Requirements

Our methods make the following assumptions:

1. The robot can execute a set of behaviors, $\{B_1, ...B_n\}$, where each behavior, B_i, requires a 3D location, p_{3D}, in the robot's frame of reference as initial input. We have previously demonstrated that behaviors of this form can perform a variety of useful mobile manipulation tasks when provided with a 3D location designated with a laser pointer [109].

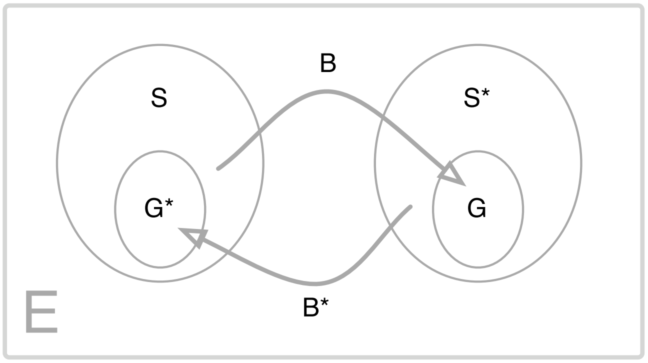
2. The robot has a way of reliably detecting whether or not a behavior it has executed was successful or not. Specifically, a verification function, V, returns whether or not a behavior succeeded. For this work, V takes the form $V(I(b), I(a))$, where $I(x)$ is the array of robot sensor readings when the state of the world is x. The states b and a are the states before and after the robot executes a behavior.

3. For each behavior, B, there is a complementary behavior, B^*. If B successfully executes, then successful execution of B^* will return the world to a state that allows B to execute again. We discuss the implications of this requirement in the next section (Section 5.2.2).

4. Upon successful execution, a behavior returns a 3D location near where its complementary behavior can successfully execute.

5.2.2 Complementary Behaviors

In order to train without human intervention our procedure uses a complementary pair of behaviors during its data gathering process. We introduce the notion of a complementary robot behavior B^* to a behavior B as being a behavior that is capable of "reversing" the state of the world, so that behavior B can be used again. For example, if behavior B's function is to turn off the lights using a light switch, its complement, B^*, would turn the lights back on using that light switch. If a behavior opens a door, then its complement would close the door.



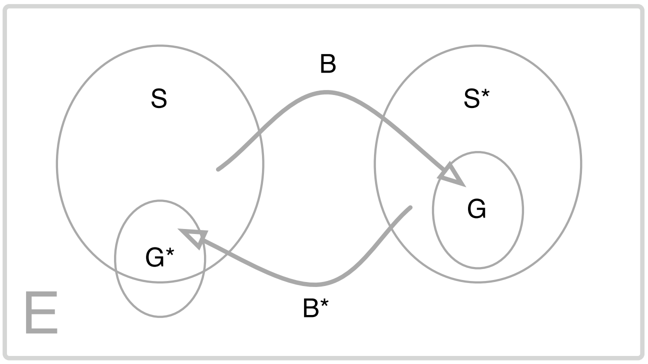


Figure 24: Relationships between set S, G, S^*, G^*, B, and B^*. ***Top:*** *An example set of complementary behaviors where $G^* \subseteq S$ and $G \subseteq S^*$. In this case, the effect of B is reversible using B^*.* ***Bottom:*** *An example set of behaviors that are not complements with $G^* \nsubseteq S$, so B^* can produce states that are not in S.*

We formalize our notion of complementary behaviors by defining the relationship between ideal complementary behaviors. We first define a hypothetical state space E that contains the states of everything in the world, including the robot's state. We then represent execution of behavior B given an initial state of the world $i \in E$ as $B(i)$, where B is an operator that takes the initial state of the world i as input and returns the resulting state of the world $r \in E$. Furthermore, when B is applied to a state $s \in S$, where $S \subseteq E$ is a set of starting states, it returns $g \in G$, where $G \subseteq E$ is a set of goal states. We define

$$G = \{g | V(I(i), I(g)) = success \wedge g = B(i) \wedge i \in E\} \tag{1}$$

and

$$S = \{s | g \in G \wedge g = B(s) \wedge s \in E\}. \tag{2}$$

Intuitively, if the state of the world, s, is a start state, $s \in S$, then the behavior B will be successful and the resulting state of the world, $g = B(s)$, will be a goal state, $g \in G$.

We now define a complement B^* of behavior B to have a set of start states, S^*, and a set of goal states, G^*, such that $G^* \subseteq S$ and $G \subseteq S^*$ (see Figure 24). This guarantees that applying B's complement, B^*, after successfully applying B will result in a state of the world that allows B to once again be applied successfully. More formally, it guarantees that $B^*(B(i)) \in S$ when $i \in S$, and that $B(B^*(i)) \in S^*$ when $i \in S^*$.

5.2.3 Autonomous Training

5.2.3.1 Initialization

Our initialization procedure is motivated by the scenario in which a user would take the robot on a home tour and point out 3D locations using a green laser pointer [109] and specify behaviors applicable to those locations. After this tour, the robot would later autonomously navigate back and learn to robustly perform the behaviors.

For this system, we have implemented an initialization procedure that starts with the user navigating the robot to be in front of the device to be operated using a gamepad interface. Then using a green laser pointer [109], the user designates an initial 3D location to begin exploring by holding the laser point steady at the intended target. The robot samples 3D points around this designated

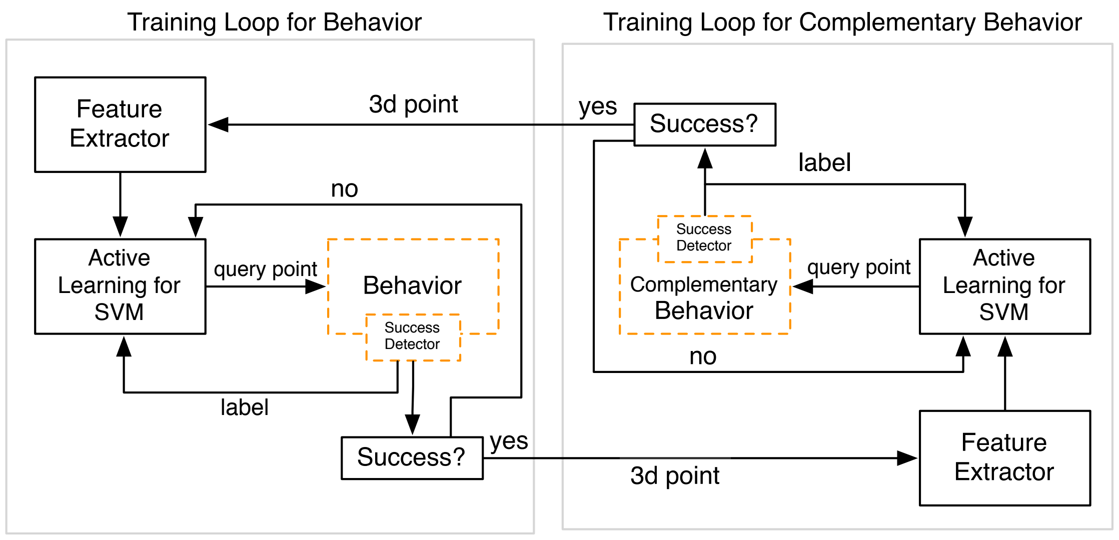


Figure 25: Illustration of the classifier training procedure where the system trains the complementary behavior upon success of the first behavior and vice versa. Dashed orange boxes on the two behaviors and success detectors highlight that these modules are provided as input to our system.

location (using a spherical Gaussian with a variance of 4 cm) and executes the behavior pair with respect to them. After each execution of a behavior at a 3D location, the behavior's verification function returns a label of either success or failure. The sampling process continues until the procedure gathers data points from at least one successful and one failed trial. These two data points are then used to train SVM classifiers that guide the data gathering process with the active learning heuristic [129].

After this initialization, the robot stores a 2D mobile base pose with respect to a global map, the user provided 3D location, an SVM trained using two labeled data points, and labels indicating which pair of behaviors is applicable at the specified location. We illustrate this procedure in Figure 26. In addition, the user navigates the robot to eight different poses in the room, referred to as practice poses, each at least a half meter away from the device. The robot also stores the 2D mobile base poses associated with these eight practice poses.

5.2.3.2 Training Procedure

Our training procedure is designed to emulate conditions that the robot would encounter when performing the task. After receiving a command, the robot navigates to the device so that it can execute the commanded behavior. Navigation and localization errors result in variations that can

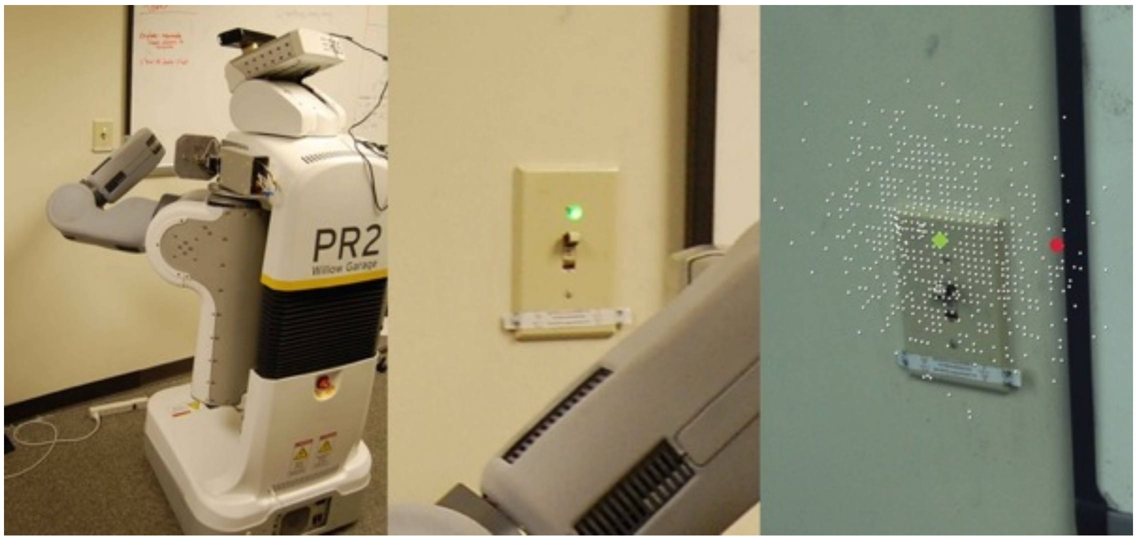


*Figure 26: Illustration of the initialization procedure for a pair of behaviors that flip light switches. **Left** Position robot in front of the switch. **Middle** Illuminate an initial 3D location as input to the behavior using a laser pointer. **Right** A 3D location associated with success (green) and a 3D location associated with failure (red) after initialization. Candidate 3D locations to explore are shown in white.*

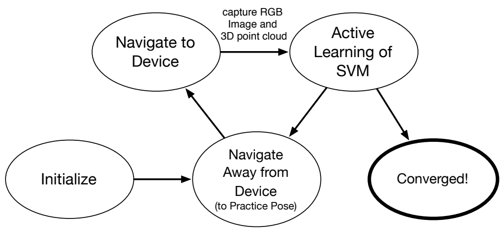

Navigate to Device
capture RGB Image and 3D point cloud
Active Learning of SVM
Initialize
Navigate Away from Device (to Practice Pose)
Converged!

Figure 27: Overview of the training procedure: initialization of the classifier; specification of practice poses in the environment by the user; and a loop that navigates the robot to each practice pose and back to the device until the robot gathers enough training data.

substantially reduce the performance of a behavior, such as variation in the robot's point of view. We illustrate task variation due to navigation in Figure 28. Our training method samples from this source of task variation by commanding the robot to navigate to one of eight practice poses in the room and then commanding it to navigate back to the device (see Figure 27).

After navigating to the device, our procedure begins an active learning phase (see Figure 25). We summarize this phase in Algorithm 1. The process starts with the robot capturing an RGB image and a registered 3D point cloud. The robot then computes image feature vectors for 3D points randomly sampled from the point cloud around the device (*extract_features*). It then iteratively selects image feature vectors (*svm_pick*) that it labels by executing the behavior at the associated 3D

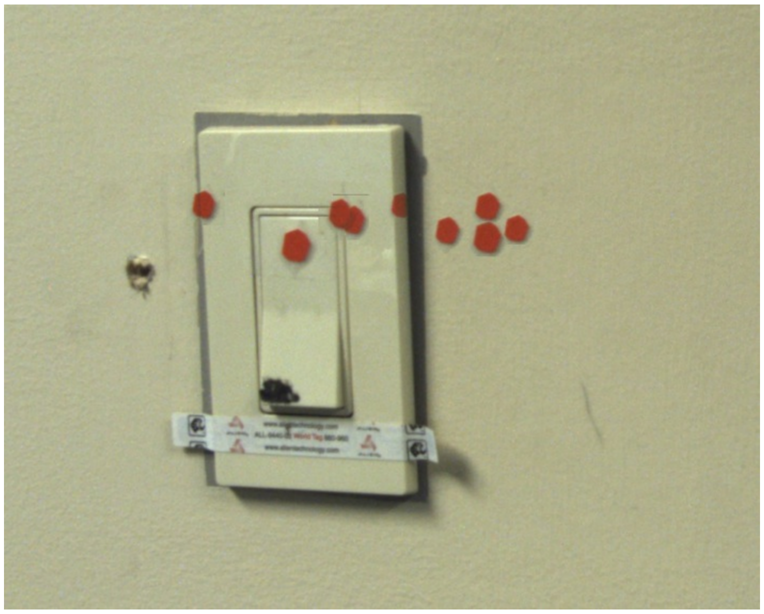


Figure 28: This figure shows a visualization of task variation due to the robot's mobility. We affixed a red dot at the center of a rocker switch. The robot attempted to navigate to the same pose and take the same picture of the switch 10 times. This image superimposes the red dot from 9 images onto the first image to illustrate the wide variation due to navigation. One of the 10 dots is obscured by 2 others. The switch plate shown has a width of 7.0 cm. If the robot were to use its localization estimate to press this switch, most of the attempts would result in a failure.

location and using the verification function (*execute_behavior*). After each execution, the process retrains the SVM classifier with a data set that incorporates the newly acquired labeled example (*add_instance_and_retrain_svm*). In order to make our implementation efficient, the robot does not collect a new RGB image or registered 3D point cloud between these executions. If an execution succeeds (*If(success)*), however, the robot begins this same process with the complementary behavior, which first captures a new RGB image and registered 3D point cloud. When executing the complementary behavior, the system does so until it succeeds with no limits on the number of retries.

The procedure stops after gathering a maximum of six labeled image feature vectors or the learner converges (*stop_criteria*). We imposed this conservative maximum limit, determined heuristically, because image feature vectors gathered from the same view are correlated, which can confuse the learning heuristic and result in the training process stopping prematurely. However, if we pick a number that is too small, the robot's base would move more often lengthening the training time.

Algorithm 1: practice($point^{3D}$, behavior, comp_behavior, stop_criteria)

```
instances, $candidates^{3D}$ = extract_features($point^{3D}$);
while True do
    instance, $candidate^{3D}$ = svm_pick(behavior, instances, $candidates^{3D}$);
    if stop_criteria(behavior) or svm_converged(behavior, instances) then
        break;
    end
    success, $candidate^{3D*}$ = execute_behavior(behavior, $candidate^{3D}$);
    add_instance_and_retrain_svm(instance, success);
    $instances = instances \setminus instance$;
    $candidates^{3D} = candidates^{3D} \setminus candidate^{3D}$;
    if success then
        practice($candidate^{3D*}$, comp_behavior, None, stop_criteria=stop_on_first_success);
    end
end
```

This process continues until *svm_converge* is satisfied for each of the eight practice poses. Once it is satisfied for a particular practice pose, the robot no longer navigates to the pose. We define convergence for a practice pose to occur when after driving up to the device from the practice pose, none of the initially computed image feature vectors are closer to the decision boundary than the current support vectors.

5.2.3.3 Behavior Execution Procedure

The training process above produces a classifier that can reliably detect locations where the associated behavior will succeed. To use this classifier, our robot navigates to the device using the 2D map pose stored during initialization, classifies 3D points in the view that it sees, finds the mode of the positive classified points using kernel density estimation [8], selects the 3D point in the point cloud closest to this mode, and executes the associated behavior using the resulting 3D location.

If the behavior fails to execute using this 3D location, our procedure adds the associated image feature vector as a negative example to the data set and retrains the classifier. This new example changes the classifier's decision boundary. The robot then selects a new 3D location using the retrained classifier with the originally computed image feature vectors. This continues until the behavior is successful. It then adds the image feature vector associated with this success to the data set as a positive example and retrains the SVM. In contrast to systems where the execution process is independent of data gathering and training, the robot has the opportunity to retrain its classifier when it detects errors made during execution, giving the possibility of lifelong training.

5.2.4 Classification

Our methods use standard support vector machine (SVM) classifiers trained with fully-labeled data. Our current implementation uses batch learning and retrains these classifiers many times. Our datasets tended to be small with fewer than 200 examples, which made this feasible, but online learning with appropriate classifiers might achieve comparable performance with improved efficiency.

We denote the data for our classification problem as $D = \{(x_1, y_1), \ldots (x_N, y_N)\}$ where $x_i \in \mathbb{R}^M$ is the feature vector that represents the 2D appearance of 3D point i and $y_i \in \{1, -1\}$ is the label that represents success, 1, or failure, -1, of the behavior when it was executed with 3D point i as input. Our goal is to produce a classifier that predicts y_j for future instances of x_j encountered by the robot.

As functional structures on many household devices are often small compared to nonfunctional components, such as the size of a switch relative to the plate or wall, there is typically an unbalanced data set problem, since there can be many more negative than positive examples. In unbalanced data sets the SVM can return trivial solutions that misclassify all the positive samples, since the misclassification cost term in the SVM objective is defined over all samples. To prevent this issue, we use an SVM formulation that separates the costs of misclassifying the negative class from the cost of misclassifying the positive class [39],

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C^+ \sum_{y_i=1} \xi_i + C^- \sum_{y_i=-1} \xi_i \\ s.t. \quad & y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + b) \geq 1 - \xi_i \\ & \xi_i \geq 0, i = 1, \ldots, l, \end{aligned} \tag{3}$$

where $\mathbf{w}$ and b are SVM parameters, ξ_i counts the margin violations for misclassified points (in the case of nonseparable data), and $\phi()$ is the radial basis kernel function we use (discussed in Section 5.3.1).

This formulation separates the SVM misclassification cost scalar C into C^+ and C^- which are, respectively, costs due to negative and positive misclassifications. For our system, we set C^- to be 1, and C^+ to be the number of negative examples over the number of positive examples. This scaling keeps the percentage of misclassified positive and negative examples similar in our skewed data set, where there might be many more negative than positive examples. Without this adjustment, we found that training often returned trivial classifiers that classified any input vector as negative.

5.2.4.1 Active Learning Heuristic

Our training process iteratively builds a data set that it uses to train the classifier. Before each trial, the system selects the image feature vector to label. To select the feature vector, the system uses a heuristic developed in [129] that selects the feature vector closest to the decision boundary of the existing SVM, under the condition that it is closer to the boundary than the SVM's support vectors. The procedure converges when no feature vectors remain that are closer to the decision boundary than the support vectors.

At each iteration i of our procedure, we define the previous iteration's data set as D_{i-1}, the current set of support vectors as $X_i^{sv} = \{x_1^{sv}, \ldots, x_P^{sv}\}$, the unlabeled image feature vectors as $X_i^q = \{x_1^q, \ldots, x_M^q\}$, and the SVM distance function, which measures distance to the decision boundary, as $d(\mathbf{x_i}) = \left|\mathbf{w}^T \phi(\mathbf{x}_i) + b\right|$. The system selects the unlabeled image feature vector that is closest to the decision boundary as specified by the following expression:

$$\underset{\mathbf{x}_i^q : \forall \mathbf{x}_j^{sv} \; d(\mathbf{x}_i^q) < d(\mathbf{x}_j^{sv})}{\operatorname{argmin}} \qquad d(\mathbf{x}_i^q) \tag{4}$$

5.2.4.2 Features

The feature generation procedure, which is illustrated in Figure 29, takes as input a 3D point cloud, a registered high resolution RGB image, and a reference 3D point. The system first selects random 3D points from the point cloud, without replacement, around the reference 3D point according to a Gaussian distribution $\mathcal{N}(\bar{p}, \Sigma)$, where $\Sigma = diag(v_x, v_y, v_z)$ with v_x, v_y, and v_z being, respectively, variances in the x, y, and z direction. The Gaussian mean $\bar{\mathbf{p}}$ is set to the 3D reference point. This Gaussian search prior enables the system to save computational effort and focus its attention on the device that the robot is supposed to manipulate.

After randomly selecting a set of 3D points, the system projects each 3D point $\mathbf{p}_i^c$ into the high resolution RGB image, $proj(\mathbf{p}_i^c)$. For each projected 3D point, it collects square image patches

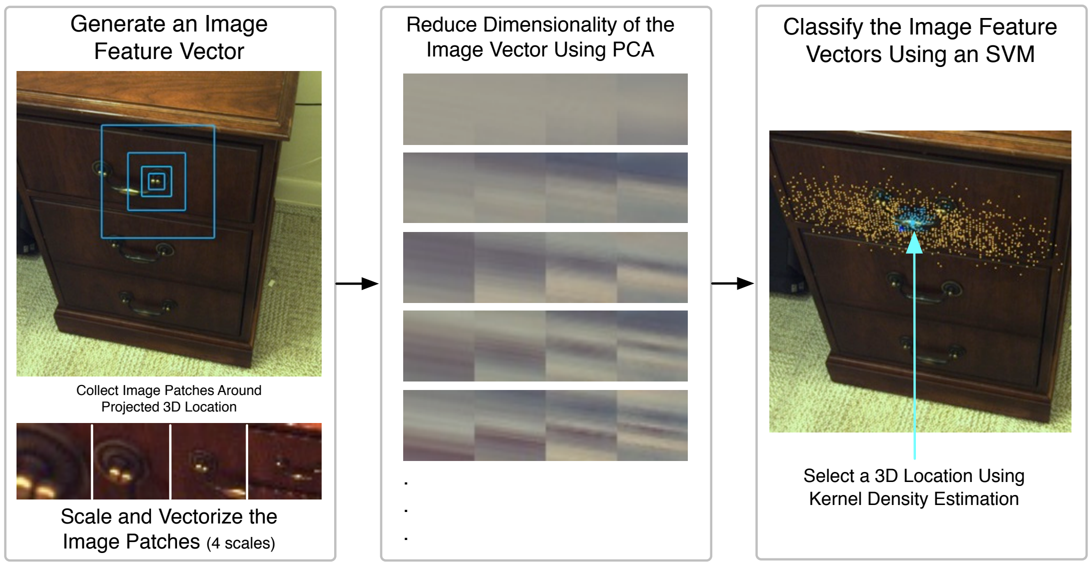

Generate an Image Feature Vector
Collect Image Patches Around Projected 3D Location
Scale and Vectorize the Image Patches (4 scales)
Reduce Dimensionality of the Image Vector Using PCA
Classify the Image Feature Vectors Using an SVM
Select a 3D Location Using Kernel Density Estimation

Figure 29: To select a 3D location at which the behavior is likely to be successful, the system first generates image feature vectors for a set of 3D locations. It does so by vectorizing and then reducing the dimensionality of scaled image patches centered around the 2D projection of each 3D location. Then it uses an autonomously trained SVM to classify each of these image feature vectors as predicting success (blue) or failure (orange) of the behavior. Finally, it selects a specific 3D location using kernel density estimation.

of successively increasing size centered at the projected 2D point in the RGB image, scales these patches to have the same height and width, vectorizes them, and concatenates them into an image feature vector. The system then uses Principle Components Analysis (PCA) to reduce the dimensionality of these image feature vectors. We discuss the specifics of these steps in Section 5.3.1.

5.3 Implementation

5.3.1 Learner Parameters

Our system runs on a PR2 robot [12]. Starting with a 3D point cloud and registered RGB image, our process randomly selects 3D points from the point cloud as described in Section 5.2.4.2. For each selected 3D point, the system collects image patches at 4 scales centered around the point's 2D projection in the RGB image. The raw image patches have widths of 41, 81, 161, and 321 pixels. They are then scaled down to be 31x31 pixel image patches, vectorized, and concatenated into an 11,532 element image feature vector for each 3D point. Computer vision researchers have used similar representations. We selected these particular sizes by hand while developing the system. Computational limitations, the resolution of the robot's camera images, and the appearance of the mechanisms influenced our selection, but we would not expect system performance to be sensitive to these sizes. The vectors are then reduced to 50 element vectors by projecting them onto PCA basis vectors that are calculated for each action using the 11,532 element image feature vectors computed from the first 3D point cloud and RGB image captured during initialization.

To classify these 50 dimensional image feature vectors, we use SVMs with radial basis function kernels. We set the hyperparameters of this kernel using an artificially labeled data set. To create the data set we took 10 different 3D point clouds and RGB images of a light switch from different views and geometrically registered them. After hand-labeling one 3D point cloud and RGB image, we geometrically propagated labels to the other 9. To find the kernel hyperparameters, we split the labeled image feature vectors from this data set into a training set and a test set. We then performed a grid search [39] for the set of hyperparameters that best generalized to unseen data in the test set.

5.3.2 Behaviors

To evaluate our system, we implemented three pairs of complementary behaviors that operate light switches, rocker switches and drawers. These tasks are sensitive to the location at which an action is performed. For example, light switches are small targets that require high precision and accuracy for the PR2 to operate with its finger tips. As illustrated in Figure 28, we have found that a PR2 will rarely succeed at flipping a light switch if it simply navigates to a pre-recorded location and moves the arm through a pre-recorded motion without visual feedback.

5.3.3 Light Switch Behaviors

Our light switch behavior's strategy is to reach forward to the specified 3D location, stop on contact detected with gripper tip tactile sensors, then slide along the contacted surface in the direction of the switch. A successful 3D location needs to place the robot's finger so that its width will make contact with the switch and far enough above or below the switch so that the finger will move the switch down or up. Figure 30 shows the sequence of actions taken by this behavior.

The behavior starts with the robot closing its gripper (Close Gripper), moving the gripper to a pre/manipulation location (Move to Start Location), reaching to the given 3D location (Reach), flipping the switch by sliding along the flat surface (Flip Switch), moving the gripper back (Move Gripper Back), then moving back to the initial location (Move to Start Location).

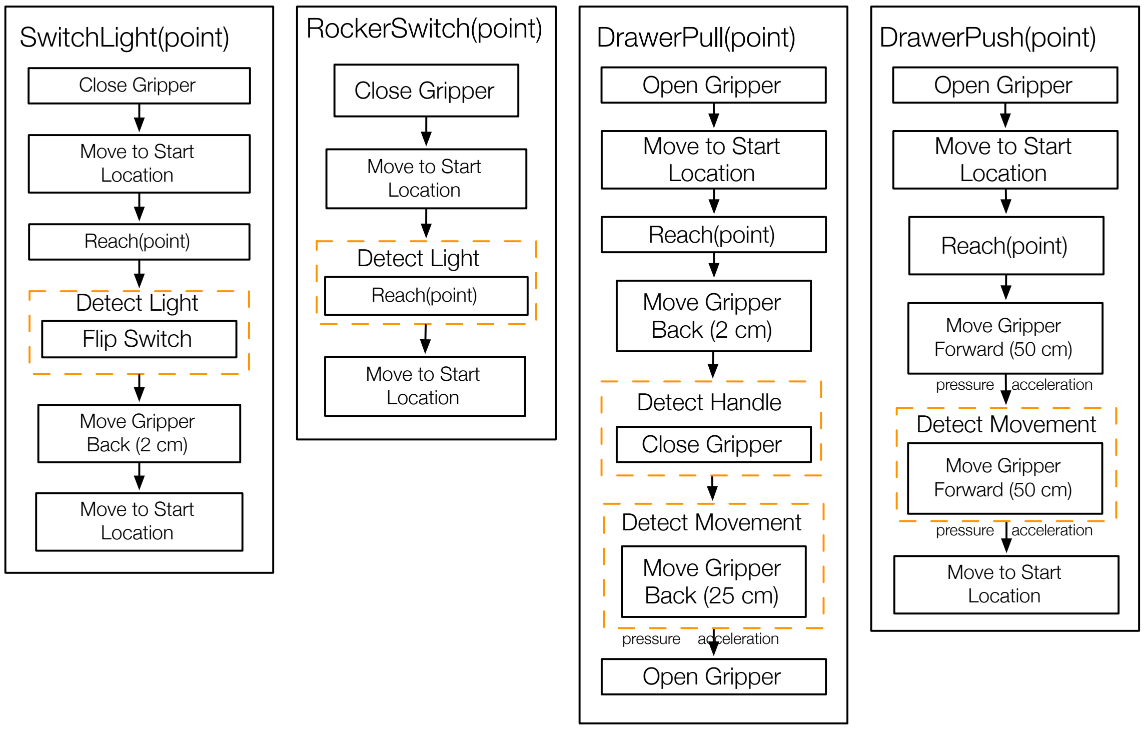


Figure 30: Sequence of actions performed by each of the eight behaviors used in this work for operating a light switch, rocker switch and drawer. Dotted orange boxes indicate procedures for detecting success or failure in a given behavior.

There are a few steps in this behavior where the robot detects tactile events. When reaching, the robot stops when it detects contact using pressure sensors on its finger tips. Next, the sliding movement stops after detecting a spike in acceleration with the accelerometer embedded in the robot's gripper. In the context of this task, this spike in acceleration typically corresponds with the light switch flipping.

To detect success, our behavior measures the difference between the average intensity of an image captured before sliding along the surface and an image captured after. A large difference indicates that the lighting intensity changed.

The complementary behavior is identical except for a change in the direction of flipping. After executing, the behavior and complementary behavior return the 3D location input with a predefined offset ($\pm$ 8 cm).

5.3.4 Rocker Switch Behaviors

Our rocker switch behavior consists solely of a reaching out step similar to the light switch behavior above, since the force applied from contact during the reach procedure is enough to activate the switch. A successful 3D location will result in the robot's fingers pushing in the top or bottom of the rocker switch.

This behavior uses the same image differencing method to detect success as the light switch behavior. It calculates the difference between images captured before and after the robot reaches forward. After executing, the behavior and complementary behavior return the 3D location with a predefined offset ($\pm$ 5 cm).

5.3.5 Drawer Behaviors

Pulling open and pushing closed a drawer require different behaviors and success detection methods. Our pulling behavior reaches to the drawer handle location, detects contact, moves back slightly, grasps with the reactive grasper from [63], and pulls. When pulling, failure is detected if the grasp fails or the robot fails to pull for at least 10 cm while in contact with the handle. A successful 3D location will result in the robot's gripper grasping the handle well enough to pull it back by at least 10 cm. When pushing, failure is detected if the gripper does not remain in contact with the surface for at least 10 cm. This classifies events where the robot pushes against a closed drawer or an immovable part of the environment as failures. After executing, the behavior and complementary behavior return the 3D location the tip of the gripper was in immediately after pulling or pushing.

5.3.6 Robot Controllers Used

We now discuss the various controllers used for sub-behaviors shown in Figure 30. For the arms, we had the option to use either a joint or Cartesian controller. The joint controller takes as input a set of 7 joint angles for each arm and creates splines for each joint individually to move the arm smoothly to the given goal. Built on Nakanishi et al's [107] acceleration Jacobian pseudo-inverse control scheme, the Cartesian trajectory controller (provided by the ROS package robot_-mechanism_controllers) attempts to keep the robot's end-effector as close as possible to a given 6 degree-of-freedom (DoF) goal. As the arms possess 7 DoF, this controller allows the remaining one degree-of-freedom to be specified using a posture goal consisting of a set of 7 joint angles. Finally, the Cartesian controller allows for much more compliant motions since it is less concerned about following exact joint level goals.

At the beginning and end of most behaviors discussed, we issue a "Move to Start Location" command that internally calls the Cartesian controller to move to a preset 3D point then uses the joint controller to fix the arm stiffly in place. For the close and open gripper command, we use the pr2_gripper_action package in ROS to move to a fully opened or fully closed position while limiting the maximum effort. To implement "Reach", we send a series of Cartesian goals to the arm in a straight line starting from the end-effector's current position and ending at the given goal with the option to stop when detecting readings from either the pressure of acceleration sensors. Finally, for modules that move the gripper backward or forward for a given distance, we again use the Cartesian controller but instead of linearly interpolating to the goal, we just send the final goal point. These motions also have the option of stopping based on tactile events detected by the robot's grippers.

5.4 Evaluation

We evaluated our system using six separate devices. We first tested on a rocker switch using the PR2 robot named GATSBII in our lab, the Healthcare Robotics Lab (HRL). For the remaining five devices we performed tests in the Georgia Tech Aware Home, a residential lab on campus used as a test bed for new technologies.

In each environment, we began our evaluation by creating an occupancy grid map of the area with the PR2's built-in navigation package [93]. Then, after initialization (Section 5.2.3.1), we ran the autonomous training system (Section 5.2.3.2) until convergence. The experimenter provided 8 practice poses, 4 for each behavior in the pair. Here, we picked 4 for each behavior as it allowed us to exhaustively pick locations from which the robot would travel to the mechanism in most rooms. We have not observed the training process to be particularly sensitive to this parameter however. The training system ran without experimenter intervention except for pausing and resuming when the robot's batteries ran low. In all, we trained 12 classifiers, a result of having 6 devices and a pair of behaviors for each device ($12 = 6 * 2$).

After finishing the training sessions, we evaluated each classifier by running each behavior

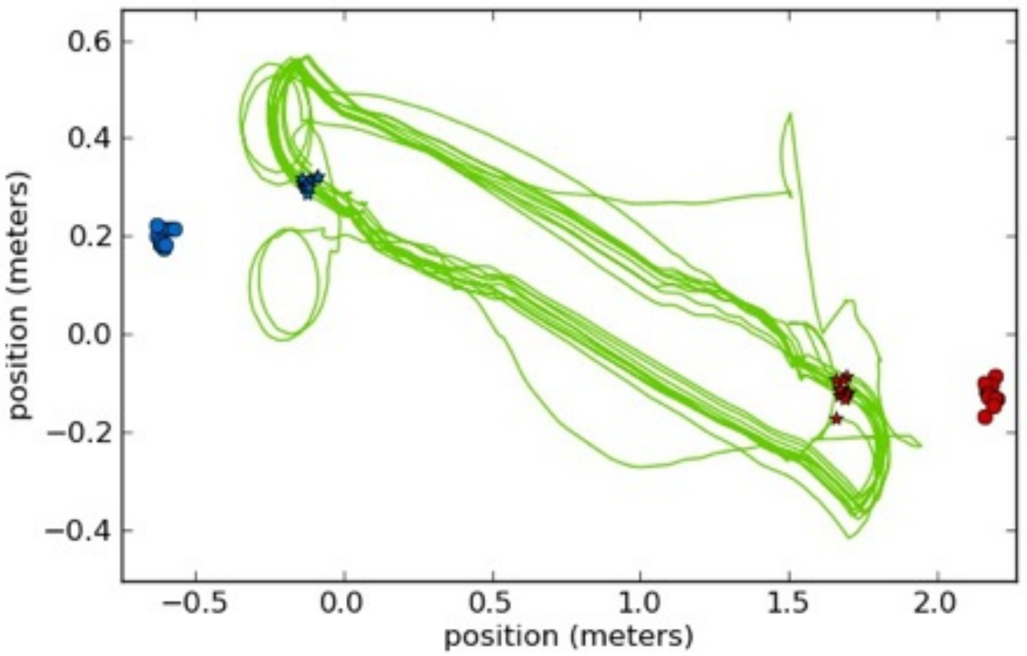


Figure 31: Results of experiments for which we used a motion capture system to track the robot's pose while navigating between two goal poses (blue and red). Green is the path the robot took. Stars indicate the final poses of the robot after it navigated to the goal poses. Circles show a point 50 cm in front of the robot.

multiple times, giving 110 trials in all (110 trials $=$ (5 devices $*$ 2 behaviors $*$ 10 trials) $+$ (1 device $*$ 2 behaviors $*$ 5 trials)). During each trial we allowed the behavior to retry and incorporate information from failures if it did not succeed the first time. However, we discarded any data gathered during the retry procedure by previous trials at the start of each new trial to obtain accurate error statistics for the original classifier.

For the devices we used in our evaluation, the functional components are difficult for the PR2's laser range finder to detect. Light switches only show up as a few protruding 3D points similar to other noisy 3D points produced by the sensor. The rocker switch appears as a flat 2D texture on the 3D point cloud. Drawer handles tend to be metallic and reflective resulting in an absence of 3D points. Using features from RGB images enabled the robot to overcome these challenges.

5.4.0.1 Effects of Navigation Errors

To better understand the variation in the task due to the robot's mobility, we investigated how the pose of the PR2 varies when navigating to a goal pose. Using a room equipped with a NaturalPoint

Table 9: Training examples (abbreviated Ex.) gathered gathered for each action.

Action	Positive Ex.	Negative Ex.	Total
HRL Rocker On	49	96	145
HRL Rocker Off	47	94	141
Aware H. Rocker On	26	47	73
Aware H. Rocker Off	29	52	81
Ikea Drawer Open	23	35	58
Ikea Drawer Close	23	39	62
Brown Drawer Open	21	62	83
Brown Drawer Close	25	46	71
Orange Switch On	17	43	60
Orange Switch Off	20	31	51
Ornate Switch On	38	66	104
Ornate Switch Off	40	76	116

OptiTrak motion capture system, we tracked the pose of the PR2 and commanded the robot to navigate back and forth to two goal poses 10 times each. As the standard deviation of the robot's Cartesian position does not represent angular errors, we calculated errors for a point 50 cm in front of the robot, which is representative of where a device would be located. The standard deviation of the location in front of the robot was 1.85 cm, and 1.79 cm in the x and y directions, respectively. For the second position, the standard deviation was 1.55 cm and 2.38 cm in the x and y directions, respectively. We show the results of this experiment in Figure 31. These errors demonstrate that navigating to a pre-recorded location and moving the arm through a pre-recorded motion would result in large variation that can result in failure. For example, the robot's finger tips are 2.0 cm wide and light switches are only 0.8 cm wide.

5.4.1 Results

Figure 32 shows the locations that the trained SVMs predict will be likely to lead to the success of their associated behaviors. These predictions are solely a function of the visual appearance of each location as represented by its image feature vector. These visualizations of the classifier output demonstrate that the classifiers identify locations relevant to their associated behaviors. For example, the robot autonomously discovers that opening a drawer requires grasping at the location of the drawer handle, while closing a drawer can be performed across the front surface of the drawer. The visualizations also show that different drawer handles can have distinct task-relevant properties.

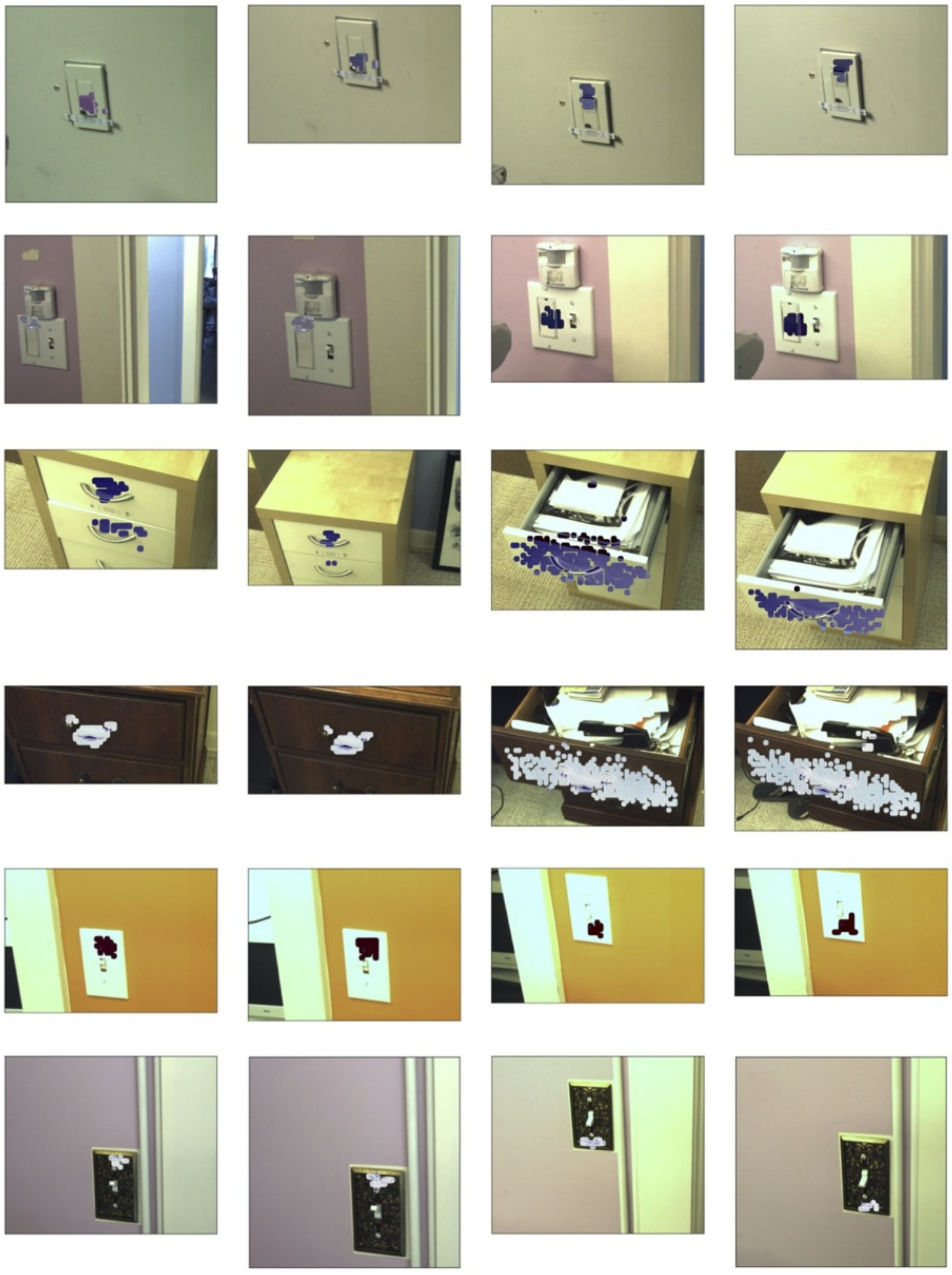


Figure 32: Each pair of images shows classification results of learned detectors just after convergence then on a new test image. Areas with inverted colors mark locations identified as leading to success of associated behaviors. To show these results, we took positively predicted points, placed those points in a grid, then overlaid this grid on the original image. ***Row 1:*** *Detectors for a rocker switch in our lab.* ***Row 2:*** *Detectors for a different rocker switch in the Aware Home.* ***Row 3:*** *Detectors for pushing and pull a wooden drawer.* ***Row 4:*** *Detectors for another dark wooden drawer.* ***Row 5:*** *Detectors for a regular light switch.* ***Row 6:*** *Detectors for an ornate light switch.*

Table 10: For each trained behavior we ran 10 trials. We list the number of tries until success for these trials below.

Action	1^{st} Try	2^{nd} Try
HSI Rocker On	2	3
HSI Rocker Off	4	1
Aware Home Rocker On	10	
Aware Home Rocker Off	9	1
Ikea Drawer Open	10	
Ikea Drawer Close	10	
Brown Drawer Open	10	
Brown Drawer Close	10	
Orange Switch On	8	2
Orange Switch Off	9	1
Ornate Switch On	9	1
Ornate Switch Off	9	1

For example, the opening behavior works best when grasping the middle of the silver handle, but can succeed by grasping the far ends of the brass handle.

Due to the distribution for random sampling including some points on the lower handles for the white drawers, the SVM estimates that success can be achieved by pulling on the top handle or the bottom handle. The illustrates a limitation with our current approach, since the verification function for pulling a drawer open can not tell the difference between the top or the bottom drawer. It also shows the influence of the distribution used to randomly sample 3D locations. At the same time, it suggests that the visual classifiers may have some ability to generalize to distinct objects.

For the light switches, the behaviors slide along the surface of the switch. The robot autonomously discovered that locations that are along the switch plate above and below the switch are likely to lead to success. Additionally, it does not predict success for locations along the wall, which is appropriate since the robot's fingers get caught on the switch plate edge if the robot tries to slide along the wall to the switch.

In Table 9, we show the number of examples collected for each classifier. The median number of examples needed was 77, and the maximum needed was 145 examples. With the rocker switch, where examples are noisy due to the middle of the switch being an unreliable spot to push, the number of examples increased to 145 indicating a sensitivity of our approach to label noise.

Table 10 shows the results of using these trained classifiers after training. Encouragingly, over the 110 trials our behavior execution process attained a 100% success rate after at most two tries. In addition, errors that led to retries usually caused the robot to miss an appropriate location on the device by a small distance.

5.5 Discussion and Conclusions

In general, there are risks for a robot that learns in human environments and an unrestrained learning system can get into situations that are dangerous to itself, to the environment, or to people. We address this issue by limiting the robot to using a few classes of behaviors in parts of the home that users have designated as safe for robot learning. Additionally, the behaviors move the robot's arm compliantly and use haptic sensing to decide when to stop moving. By learning in situ, a robot's data gathering activities do not have to stop after its training phase and can potentially continue for as long as the robot remains in service.

Autonomous learning in human environments is a promising area of research that gives robots methods to cope with devices that they have not encountered before and many forms of real-world variation. We have presented methods that enable a mobile manipulator to autonomously learn to visually predict where manipulation attempts might succeed. As we discussed in the introduction, our work advances autonomous robot learning in three ways. First, our approach uses a robot's mobility as an integral part of autonomous learning, which enables the robot to handle the significant task variation introduced by its mobility. Second, our research demonstrates that by using active learning, a robot can autonomously learn visual classifiers solely from self-generated data in real-world scenarios with a tractable number of examples. Third, our research introduces complementary behaviors to address challenges associated with autonomously learning tasks that change the state of the world.

In this chapter, we have made simplifications where we used behaviors programmed manually as input to our learning system. As described, we expect this learning procedure to function as well with any pair of complementary behaviors that satisfy our assumptions. In the next chapter, we investigate this expectation with behaviors created by subjects in the user study featured in Chapter 3.

CHAPTER VI

INTEGRATING AUTONOMOUS LEARNING INTO ROS COMMANDER

As seen, affixing ARToolKit tags to mechanisms can quickly help users create autonomous behaviors enabling robots to accomplish tasks without human intervention. Being able to localize mechanisms accurately is key to task success when replaying motions as our behaviors rely on the heuristic that motions carried out at the same spatial location with the local environment in a similar state will result in similar outcomes. Crucial for user authoring behaviors, relying on this spatial smoothness of outcomes can allow the robot to succeed despite not modelling why and how user given motions interact with the environment.

In the previous chapter, we explored how knowing an additional piece of information, namely whether the task succeeds after execution, can enable the robot to gather data for identifying visual regions associated with success. Instead of depending on environmental modifications to provide accurate local reference frames, learned classifiers identify not just singular points but entire regions on a mechanism's surface. As shown previously, accurately modeling these regions as opposed to using single point estimates provided by users enables the robot to select alternatives after encountering failure, increasing system robustness.

In addition, while users potentially might not have issues with their living environments being modified to accommodate robotic perception, such aesthetic distractions may have significant impacts on the acceptance of robot systems. From the popularity of home decorating literature and media, it is reasonable to believe that there exist a group of users who would not be accepting of carefully designed spaces being overwhelmed by scattered fiducial patterns.

Taking cues from the previous chapter, a possible solution is to combine autonomous learning and end-user programming. Using robot autonomy enables the system to repeatedly perform actions to capture data intelligently, while open-ended user authoring of behaviors equip the system to perform tasks for which it has not performed before. The resulting behaviors, using self-trained classifiers, can then operate autonomously without need for fiducials such as ARToolKit tags to be

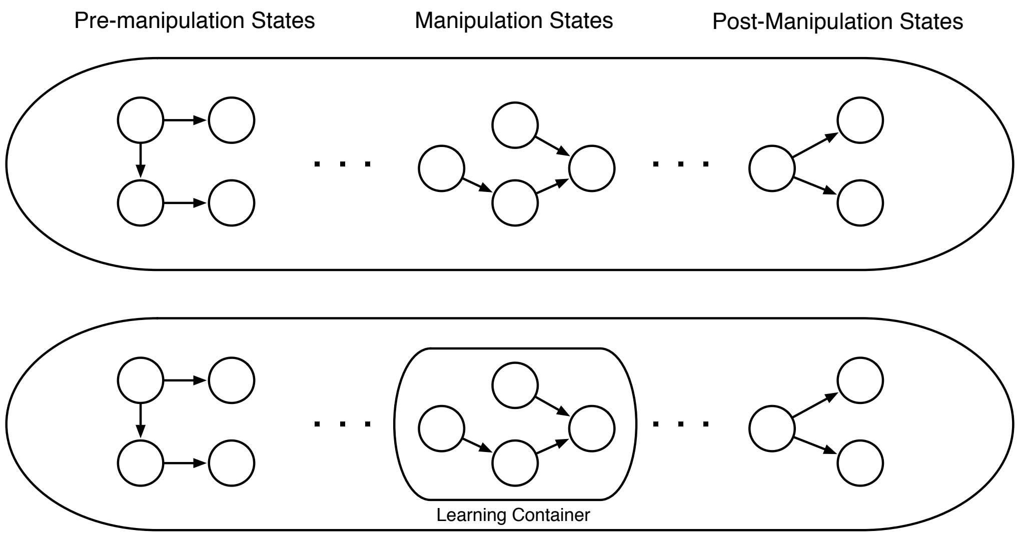


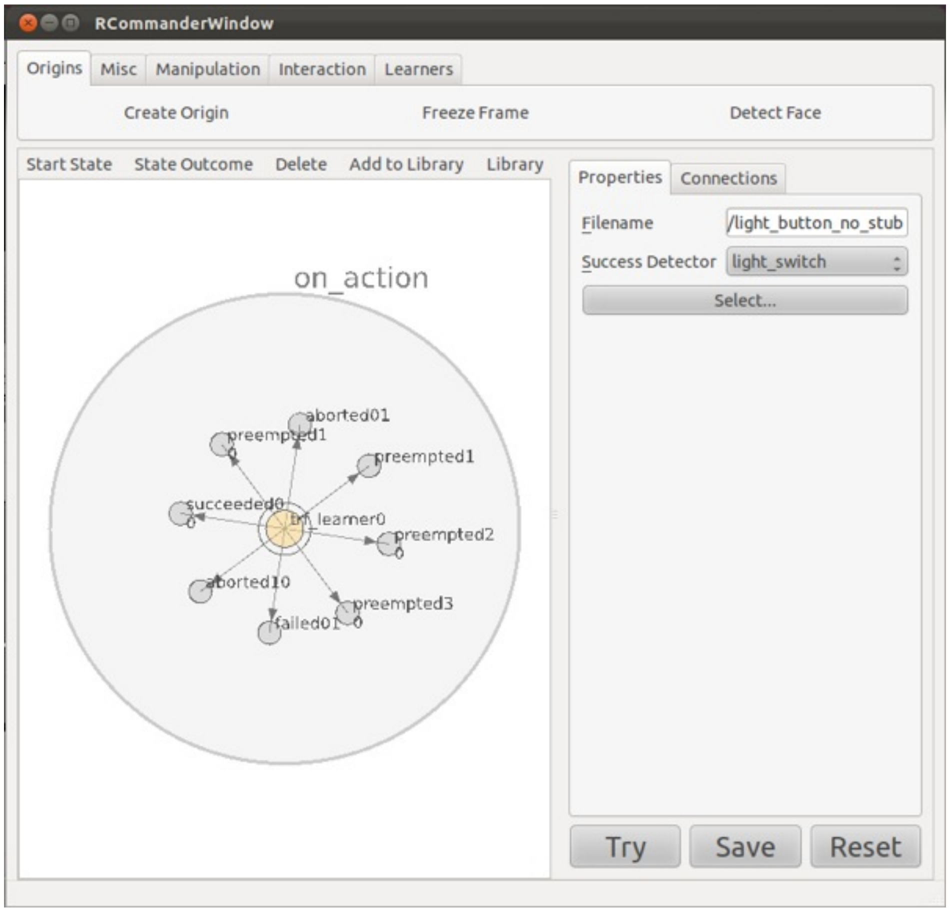
Figure 33: The learning tool that we introduce in this chapter functions as an operator creating a container state that wraps around the cluster of states that perform manipulation in behaviors. This new state uses the set of manipulation states to learn the appearances of 3D points that leads to success in task execution. ***Top:*** *A typical manipulation behavior's HFSM clustered into their relationship to the manipulation performed by the robot.* ***Bottom:*** *The same HFSM after applying the learning operator. When executed, the container state uses its learned classifier to find a 3D point where the manipulation states would succeed.*

affixed to the environment. Then, as the learned mappings identify entire regions associated with success, the robot can explore alternatives should the behavior fail to execute, allowing the overall system to be significantly more robust.

In this chapter, we integrate autonomous learning as a new tool in ROSCo that allows the addition of learning elements into constructed behaviors. We also show experiments demonstrating that, once integrated, the autonomous learning system featured in Chapter 5 can operate not only with behaviors constructed by roboticists but also those constructed by non-roboticists from our study in Chapter 3. Out of the 33 behaviors constructed by our study participants, with each of 11 participants constructing 3 behaviors, we use our system to autonomously train 6 behaviors, 2 from each of 3 categories. We also verify that the mapping created by the resulting classifier allows the robot to succeed with similar success rates to the roboticist-crafted behaviors in Chapter 5.

6.1 Approach Overview

Our method enables users to create ROS Commander behaviors with embedded states that autonomously learn to detect 3D locations where those behaviors would likely succeed, combining



Filename /light_button_no_stub

Success Detector light_switch

Figure 34: ROSCo with the visual appearance learning tool activated on the right. Users specify two parameters, the behavior verification function to use and the name of a saved behavior, to activate it. When activated, our system uses the verification function to determine if the contained behavior succeeds or not.

Chapter 5's learning system and behaviors produced by Chapter 2's expert interface. This allows the learning system to use behaviors constructed by expert users in ROSCo instead of manually coded behaviors (as was used in Chapter 5). By doing so, in contrast to behaviors relying on ARTags in Chapter 2, these new behaviors can function autonomously in unmodified environments.

Our method operates in 3 main phases: construction, initialization, and training. To begin, a user provides a complementary pair of ROSCo behaviors with a special structure that supports autonomous learning. Next with initialization, the procedure requires a user specified area for exploration and examples of locations that lead to success and failure. When the training process starts, the robot autonomously moves to the mechanism repeatedly to gather data until its learner converges.

From the end-user's perspective, our learning system appears as a tool in ROS Commander and new controls in RViz that allows the creation of behaviors that support autonomous learning. When given as input the manipulation cluster of states from a behavior, and a verification function–a method for detecting whether the behavior encoded by that HFSM succeeds–our tool creates a new "container" state that modifies the behavior of states within it. We show this modification abstractly in Figure 33. For the purposes of creating behaviors that can be used for learning, we separate states within behaviors into 4 clusters of states depending on whether the state situates the robot, places it in a preparatory manipulation pose, manipulates the target object, or pulls away from the target.

Wrapped around the manipulation states, our learning container alters these states' behavior by shifting the "task_frame" [23, 95], an object centric reference frame specific to each task and used previously in behaviors that relied on ARToolkit Tags. This "task_frame" modification is used both during the subsequent training phase and normal operation. When training, our procedure executes the behavior, repeatedly varying the location and thus "task_frame" given to the behavior, then recording the visual signature around that location and–using the verification function–whether the attempt succeeds. During execution, the container state runs visual classification using recorded data to identify a 3D location and setting a "task_frame" that would most likely succeed according to past successful executions.

After constructing the behavior, to initialize it at a particular location, users select a pair of complementary behaviors then position an inital 6D "task_frame" at which to start exploring, and finally set 4 practice locations for each behavior in the pair. As the classifier only produces a 3D position, our method constructs the final 6D "task_frame" by combining its output 3D point with 3 rotational degrees of freedom from the frame given by users during initialization.

After initialization, in the autonomous training phase, we use the same SVM active learning procedure as described in Chapter 5 and execute the training process until convergence.

When classifying and learning during execution of the container state in each of the above phases, we use the same infrastructure here as was used by the autonomous learning system in Chapter 5. The system takes as input a 2D RGB image and a registered 3D point cloud, then extracts image features using PCA to classify using a Support Vector Machine (SVM). After classification, this pipeline produces suitable 3D locations at which the behavior will likely succeed.

6.1.1 Assumptions

Our method has a number of assumptions derived from its component parts. For autonomous learning, we assume that we have a pair of *complementary* (in the same sense as from Chapter 5) ROS Commander behaviors produced by expert users, and a verification function–a way of reliably detecting whether the behavior executed succeed. Just as in Chapter 5, this function has the form $V(I(b), I(a))$, where $I(x)$ is the array of robot sensor readings when the state of the world is x, with states b and a being states before and after the robot executes its manipulation action.

First, in terms of the behaviors' structure, we assume the supplied behaviors are structured such that our perception learning component wraps around the set of states responsible for the core manipulation action. Second, prior to the perception component executing, we require that the behavior situates the robot such that the target object is detectable with its sensors. Finally, we also assume that the core manipulation states begin and end in states of the world that are discriminable by the particular verification selected $V(I(b), I(a))$.

6.1.2 Integrating Learner into ROSCo

We now discuss modifying behaviors to support autonomous learning. We illustrate in Figure 34 the ROSCo tool that invokes our learning container. It has two input parameters: a file path to the component state machine, and the verification function to use. When activated, it either loads the indicated state machine or creates an empty container state within which other states can be inserted. Just as with other hierarchical structures in ROS Commander, users can examine and edit the child HFSM by double-clicking its container state.

We demonstrate how a user would use this tool with an example using a pair of drawer opening and closing behaviors. In this example, we separate each behavior in the behavior pair as being composed of four functional components: states that first situate the robot (such as tucking the arms, and navigating to the mechanism), states that place the robot in a preparatory manipulation pose (e.g pointing the head, and untucking the arms), states that perform the manipulation (e.g. reaching forward, closing the gripper, and pulling the drawer's handle), and states that place the robot back to a pose where it will be ready to manipulate again (e.g. pulling the gripper away from the handle, and tucking the arms).

With this decomposition in mind, users would then perform the following preprocessing steps:

1. Save states that situate the robot as a standalone behavior.
2. Delete states that situate the robot from the original behavior pair.
3. Invoke the learning tool to add a learning state to the behavior pair, selecting a verification function in the process.
4. Place the states that perform manipulation into the state container created by the learning tool.

To use the above processed behaviors, when learning or running the behavior pair, our program first situates the robot using the standalone behavior (created in Step 1) then executes the rest of the behavior–saved separately–until the controllers activated by the various states finishes. After each execution, the verification function generates the associated labels which are then stored in a database.

In Step 4, we wrapped our learning component only around the set of states that perform manipulation (e.g. those that reaches forward, grasps and pulls the handle) as our learning method and its verification function places some additional restrictions on where they are placed with respect to other states. The first of these restrictions is that the set of states which situates the robot needs to make the mechanism visible to the robot's sensors after executing. Second, while there can be more esoteric requirements placed by a particular verification function, fundamentally, actions of the set of states that manipulate have to be observable using the robot's sensors and in the timespan it takes to execute the behavior's learning container state. For example, the verification function for drawer opening defines success as the robot having a drawer handle in its gripper after pulling, or, programmatically, as having the gripper open more than a threshold distance after pulling. If we were to use this particular verification function, then the manipulation actions inside the learning container would also be required to not release the drawer' handle after pulling.

6.1.3 Frame Mappings

Previously, behaviors constructed in ROSCo use ARToolKit tags to estimate poses of mechanisms that the robot manipulate. In this section, we describe how behaviors created to use these tags are adapted to use reference frames provided by our learned classifiers.

When using ARToolKit tags, our system creates a *perceptual frame* $T_{P_i}^M$ for each tag i detected to map from that tag's frame P_i to *M*, the global map frame. Here, we represent reference frames such as $T_{P_i}^M$ as 4x4 $SE(3)$ homogeneous transformation matrices. During the deployment process, the user selects a behavior j and positions a *task frame* $T_{t_j}^{P_i}$ to relate the N points $p_{t_j}^{1:N}$, representing motion trajectories defined in that behavior's frame t_j, to the perceptual frame P_i. Combined, these frames transform poses $p_{t_j}^{1:N}$ defined with respect to the object and behavior's frame t_j to the map frame *M*:

$$p_M^{1:N} = T_{P_i}^M T_{t_j}^{P_i} p_{t_j}^{1:N} \tag{5}$$

We have a similar situation when using learned perceptual frames:

$$p_M^{1:N} = \tau_{P_j}^M \tau_{t_j}^{P_j} p_{t_j}^{1:N} \tag{6}$$

Here, we use τ to distinguish terms in this system of transforms from the previous case. After classification of the scene, our system selects a perceptual frame $\tau_{P_j}^M$, mapping from the detected frame for a behavior j back to the global map frame *M*. Note that as the classifier detects regions of 2D points, unlike with ARToolKit tags, there is a distribution of possible valid perceptual frames $\tau_{P_j}^M$. Then, using the perceptual frame P_j, we define a frame $\tau_{t_j}^{P_j}$ that maps from an associated task frame t_j back to P_j.

Just as in Section 2.3, we ask users to position an initialization frame–using a 3D rings-and-arrows control–such that its position on the mechanism is at the same approximate location and orientation as when the behavior was created, a piece of information that would be communicated by the person who created the behavior. For example, if the original behavior was defined with its task frame on the drawer's handle, with positive Z pointing out of the plane of the drawer towards the robot, the deployed behavior would also be required to follow the same convention for the initialization frame.

Initialization provides 6 numbers per behavior with 3 positions and 3 orientations that we use to form $\tau_{P_j}^M$, the perceptual frame. During training, our system uses these 3 positions to guide its exploration process, testing only $\tau_{P_j}^M$ candidates in close proximity to the initial location.

As the classification system produces only 3D points, we combine these estimates with the initializer's 3 orientations to create a complete SE(3) perceptual frame $\tau_{P_j}^{M}$ during execution. Additionally, since these produced 3D points are all on surfaces detected by the robot's Kinect sensor, they exist on a 2D subspace reducing the transform $\tau_{P_j}^{M}$ to have only 5 DoF (degrees of freedom). To allow the final product of transforms $\tau_{P_j}^{M}\tau_{t_j}^{P_j}$ to result in a full 6 DoF for SE(3) transformation, we ask users to provide an offset from the 2D surface adding an additional 1 DoF (shown in Figure X). We then set $\tau_{t_j}^{P_j}$ to have the identity transform with the user given offset used as its Z translation. This offset allows usage of behaviors that users defined with the task frame positioned underneath, or above surfaces detected by the robot's 3D sensors.

6.1.4 Autonomous Training

6.1.4.1 Initialization

To begin autonomous training, users would follow a procedure similar to the behavior deployment interface in Chapter 2. Interface-wise, this is similar to ROSCo's deployment process for attaching behaviors to a local reference frame defined by ARTags. However, the behavior's frame in this case is stored in global coordinates defined by the robot's map of the environment instead of coordinates in a reference frame centered at the location of an ARTag. We demonstrate this process in Figure 35.

The procedure starts with users situating the robot close to the mechanism then selecting a behavior pair out of a menu near the robot's head, this creates two controls that represent the two initialization frames for the behavior pair, mentioned in Section 6.1.3. After positioning these frames on the mechanism's surface close to locations where the behaviors would likely to be successful if it were executed, users would next define an offset, used to form $\tau_{t_j}^{P_j}$, into or out of the surface if needed by the behavior.

Finally, the active learning-based training procedure also needs a pair of starting classifiers to begin suggesting points for the robot to execute its complementary pair of behaviors. For each seed classifier, users would repeatedly position a marker to indicate a 3D point for the robot to execute the associated behavior until the robot has at least one positive and one negative example (3rd and 4th panel in Figure 35.

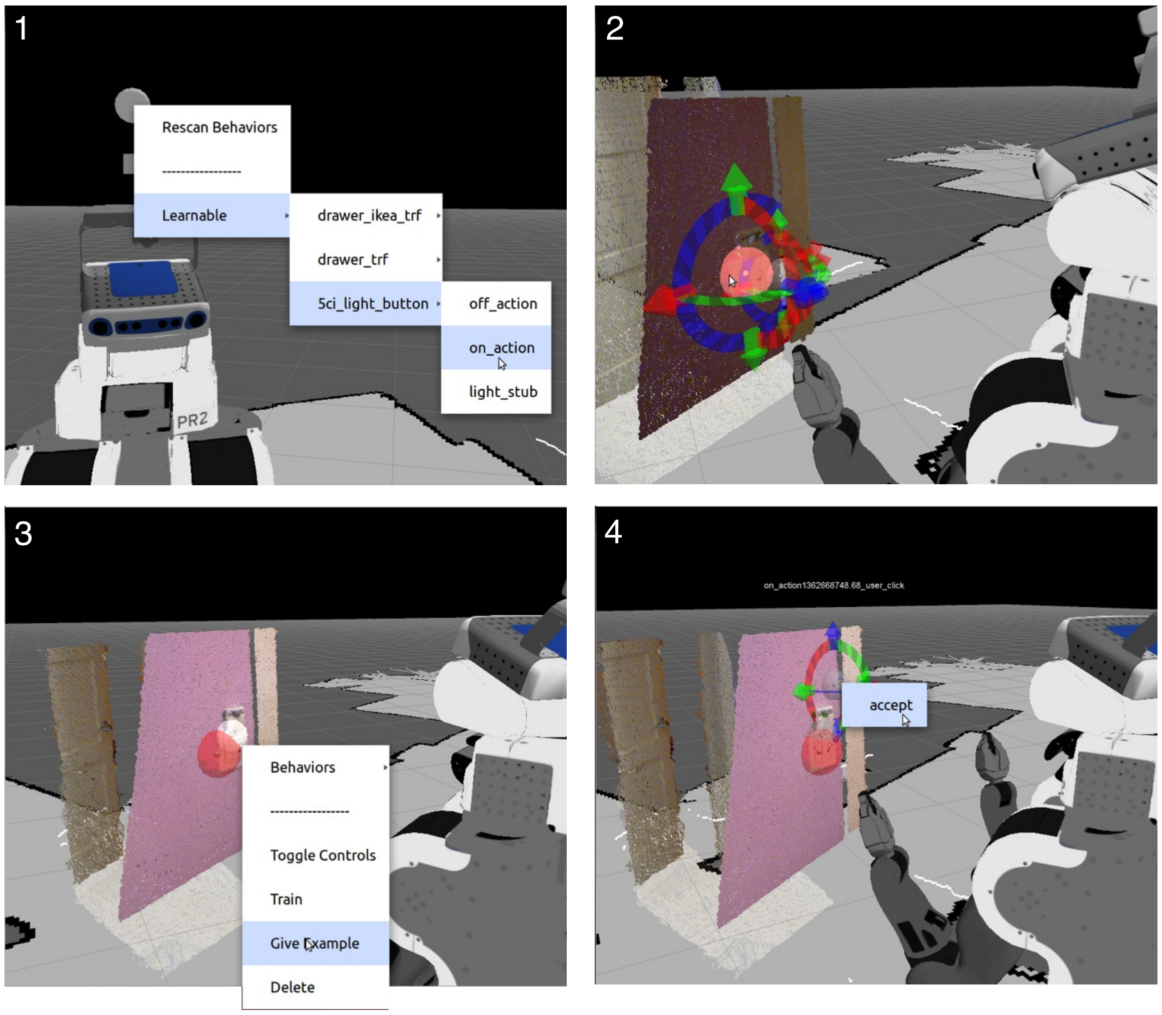


Figure 35: To initialize the training procedure, users (1) select the behavior pair to train, (2) position initial 3D locations for exploration, then (in (3) and (4)) guide the initialization by pointing out prospective points on the surface serve as good positive and negative examples by first selecting "Give Example", position a 3D point, then clicking "Accept".

The steps above for building the seed classifier is different from those presented in Chapter 5 since we have dispensed with the robot attempting to explore blindly and systematically. As without either guidance from past data or the user's help, that process described was likely to require many behavior execution attempts from the robot before it would find a requisite positive example. This is as regions likely to lead to success tend to be spatially compact for some behaviors. Having a person specify where to explore initially helps to speed up initialization considerably.

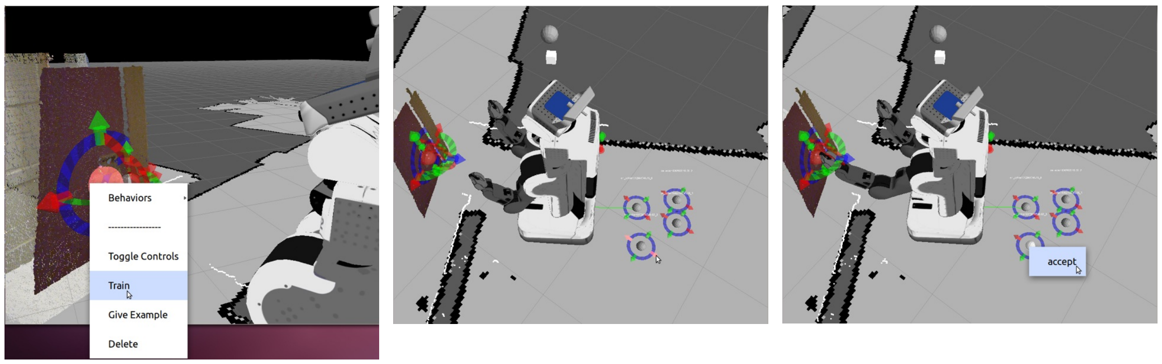


*Figure 36: Interface for starting the training procedure. Users would first (**left**) select the action to train. Then (**middle**) position the practice locations, and (**right**) click "accept" to start training.*

6.1.5 Training Procedure

Our training procedure attempts to replicate the same conditions that the robot would encounter when performing its given task. This is similar to the process discussed in Section 5.2.3.2 of Chapter 5 but with some modifications to accommodate for the execution of ROSCo behaviors. To start training, users would first select the particular action to train. Then position four markers representing practice poses that the robot will use during training, and click "accept." We show this procedure in Figure 36.

To simulate errors in navigation that the system will encounter when executing the behavior normally, our procedure repeats a cycle of navigating to one of the four practice locations then runs the selected ROSCo behavior to collect data. To navigate back to the mechanism, our behavior uses the frames provided during initialization as a guess for the product $\tau_{P_j}^{M}\tau_{t_j}^{P_j}$. Even though any actions carried out, such as autonomous navigation, using this guess would suffer from localization errors, our method succeeds if these actions place the robot close enough to an area where the mechanism can be reached by the end effector and detected using the robot's sensors. When the behavior executes its embedded autonomous learning state, the procedure runs an iteration of active learning by taking an RGBD (RGB plus Depth) image–capturing a 2D image and registered 3D pointcloud–then selecting a 3D point to label according to its learning learning heuristic.

Two outcomes are possible after execution. If the behavior is successful–implying that the state of the world has changed as a result of the robot's actions–we execute that behavior's complement to reset the mechanism's state, else we pick a new 3D point to attempt that behavior again. The procedure repeatedly executes the behavior until it captures a fixed number of examples, after which it navigates to the next practice location.

6.1.6 Execution Procedure

Our execution procedure also proceeds mostly in the same manner as described in Chapter 5 with a few small differences. Here, execution starts with users triggering a behavior through ROSCo's web interface, in the same manner as other behaviors. Similar to the training phase, the behavior executes state by state until it reaches states performing manipulation actions within the learning container. When the learning state executes, it first runs the trained classifier to find positive points around the initial 6D task-frame, calculates the mode of the positive classified points using kernel density estimation, selects a 3D point closest to this mode, then adjusts the perceptual frame $\tau_{P_j}^{M}$ used.

6.2 Robot Experiments

There were 11 users in our study from Chapter 3, with each generating behaviors in 3 categories–light switch flipping, rocker switch pressing, and drawer opening–resulting in 33 behaviors in total. We selected 2 behaviors in each of the 3 categories at random without replacement, resulting in 6 behaviors total, to train. We simulated usage of our tool by users through training on the same mechanisms used in the Georgia Tech Aware Home during our study.

As participants did not have access to our learning tool during the study, we modified the behaviors selected to support autonomous learning using the general procedure described in Section 6.1.2.

We started our preprocessing step by removing the "stub" state from each user created behavior and then saved these states as the standalone behavior used by the training process. Next, we wrapped a learning container state around user created states that define motions with respect to a "task frame". We illustrate the results of this preprocessing on an example behavior in Figure 37.

Additionally, since participants in our study were not asked to construct complements for their behaviors, needed for our training procedure, we supplemented the picked behaviors with complementary behaviors that we constructed using ROSCo. We created a drawer pushing behavior for the two picked drawer pulling behaviors, and a behavior that turns off the lights for the two picked

Table 11: Training examples (abbreviated Ex.) gathered for each action.

Action	Positive Ex.	Negative Ex.	Total
Rocker On (1)	31	59	90
Rocker Off (1)	30	71	101
Rocker On (2)	37	78	115
Rocker Off (2)	36	63	99
Drawer Open (1)	37	50	87
Drawer Close (1)	38	17	55
Drawer Open (2)	33	65	98
Drawer Close (2)	34	18	52
Switch On (1)	35	82	117
Switch Off (1)	32	79	111
Switch On (2)	46	43	89
Switch Off (2)	51	158	209

light switch behaviors. As the constructed rocker switch behaviors press the switch and therefore can function as their own complements, we did not need to construct new behaviors to train them.

With the preprocessed behaviors, we followed the same procedure as was used to evaluate the learning system in the previous chapter. We began our evaluation by running the initialization, then training procedure until convergence for each behavior, stopping and restarting when the robot's battery ran low. In all, we trained 12 classifiers, with 2 classifiers for each of 3 mechanisms.

After training, we evaluated each classifier by running its associated behavior 10 times, 120 trials in all, allowing retries if the behavior does not succeed. Additionally, each trial started with the robot in a random location within the same room as the mechanism that it operated on.

6.2.1 Results

Training was successful for all behaviors except for drawer opening behavior 2 where we intervened 5 times. We intervened during autonomous training since the gripper, as commanded by the behavior, struck the drawer while tucking the robot's arms causing the drawer to close. Without reopening the drawer manually, the training process would have failed completely as it expected the drawer to be open.

Additionally, that particular behavior also had an issue with a gripper command causing it to not execute reliably. With the gripper's controller, if commanded to close for a smaller distance than is physically possible, such as when there is already a drawer handle in between the gripper, the

Table 12: For each trained behavior we ran 10 trials. We list the number of tries until success and failures for these trials below.

Action	1^{st} Try	2^{nd} Try	3^{rd} Try	Failures
Rocker On (1)	8	2		
Rocker Off (1)	10			
Rocker On (2)	9	1		
Rocker Off (2)	10			
Drawer Open (1)	9			1
Drawer Close (1)	10			
Drawer Open (2)	6	2		2
Drawer Close (2)	10			
Switch On (1)	8	1	1	
Switch Off (1)	8	1	1	
Switch On (2)	10			
Switch Off (2)	8	2		

controller would sometimes produce an outcome of "aborted" instead of "succeeded." Many, but not all, users encountered this issue during our study and changed the state transitions to ignore the "aborted" outcome since it did not affect the task's performance. Once we made this change to the behavior, it was able to execute without aborting early, enabling the training process to complete.

We show in Table 11 the number of training examples collected by our training process. Compared to Table 9, the number of examples needed appears higher. Previously, the median number of examples needed was 77 but now it is 98.5. Even so, however, these differences do not seem statistically significant (with $p = .32$) according to a two-tailed t-test that assumes different variances.

In Table 12, we show results of testing each trained classifier 10 times using the mechanisms that they were trained on. We show outputs from the these trained classifiers in Figure 38. There were 3 failures during drawer opening where the robot bumped into the drawer after opening causing it to close preventing retries from occurring. Summing up the second and third retries over all tests, there were 11 retries overall for all the different behaviors trained. Overall, 88.3% (106/120) of the robot's attempts succeeded on the first try. After all retries, our system succeeded in 97.5% (117/120) of its attempts.

From the robot's movements while being directed by its behavior, we expected that both the rocker switch's on and off behaviors would have similarly predicted success regions except for a translational offset. However, the trained classifiers for the on actions identified a strip in the middle that is presumably the light switch with the off actions' classifier finding a small horizontal strip at the bottom of the switch (first two images in Figure 12). The results are similar even training using two different behaviors created by different individuals (first row of Figure 12).

We show results from participants' constructed light switch behaviors in the second row with the first and third image where the light switch is pointed upward. The other two images show learned classifiers for our provided light switch on behavior. As we expected, since these behaviors use an entire side of the gripper to flip the switch, the classifier predicted a wide spatial area compared to the classifiers in Figure 32 from Chapter 5. Since our provided behavior uses more of the front edge of the gripper as opposed to the side edge, the success regions for it are also appropriately shifted to the left. This effect is consistent across the two different learned classifiers in the second and fourth image.

Even so, the classified images show artifacts and errors. The first light switch classifier mislabeled parts of the robot's arm and the adjacent white door frame as being success regions. On the third image, the classifier also misclassified parts of the door frame. Additionally, wide empty regions on the light switch plate in this figure are areas where there are not 3D readings returned by the Kinect sensor, preventing our classifier from running at all.

With drawer opening, results from participants' behaviors are qualitatively similar to those in Chapter 5 where the identified success regions appears just below the drawer handle. For the drawer pushing behavior, we inadvertently oriented the gripper so that it is perpendicular with the floor, not parallel to as was the case in the previous chapter's behavior, leading the process to learn that all 3D points on the drawer would lead to success. Even though 3D points are on top of the set of drawers, using them as the centers of task frames for the behavior still leads to success at pushing. However, since the pushing behavior assumes that the center of the task frame is where the drawer's surface begins, the robot tends to move the gripper with high velocity, slamming into the drawer.

6.3 Conclusion

Enabling end-users to create robot behaviors has the potential to help solve the problem of creating general purpose robots capable of performing many tasks and variations on those tasks. Advancing this goal, we showed the feasibility of using a flexible pipeline for creating task oriented perceptual classifiers from end-user authored behaviors. Being less dependent on explicit environmental modifications, such systems can allow end-users to produce more desirable robot capabilities closer to those produced by expert roboticists.

As far as we know, ours is the first instance of perception learning using behaviors constructed from basic building blocks by non-roboticists. By formatting ROSCo behaviors generated in our study from Chapter 3 into an appropriate structure, we were able to use the perception algorithm shown in Chapter 5 to learn an appropriate image-based classifier predicting regions where those behaviors will succeed. Furthermore, this integration enables our system to operate on the same behaviors as before without relying on fiducials such as ARToolKit Tags as well as retry in case of failures.

In tests, our system learned to predict the success regions for 6 user created behaviors enabling success in 88.3% of cases on the first manipulation attempts. The predicted regions also qualitatively reflect where these behaviors would be expected to succeed. Just as before, the system learned which positions will turn a rocker switch off and which will turn it back on. It adjusted to the new strategy of swiping at the switch plate by labeling points on the switch plate as being success regions. Additionally, having the gripper rotated by 90 $^\circ$ to be perpendicular to the floor caused most points on the drawer to be correctly recognized as leading to success as opposed to just points on the front surface.

There are a few issues with our current approach that we would like to address with future work. Currently, the most prominent issue is our method's dependence on hand-coded verification functions. In our experiments, this factored into failures of one drawer opening behaviors tested where the robot knocked a drawer close causing the system to believe that the drawer has been successfully opened. Using hand-coded verifications limit the applicability of our methods to simple classifiers that work only over low dimensional sensor spaces such as light intensity or the absolute intensity pressure readings. Additionally, employing additional sensor modality would also improve robustness.

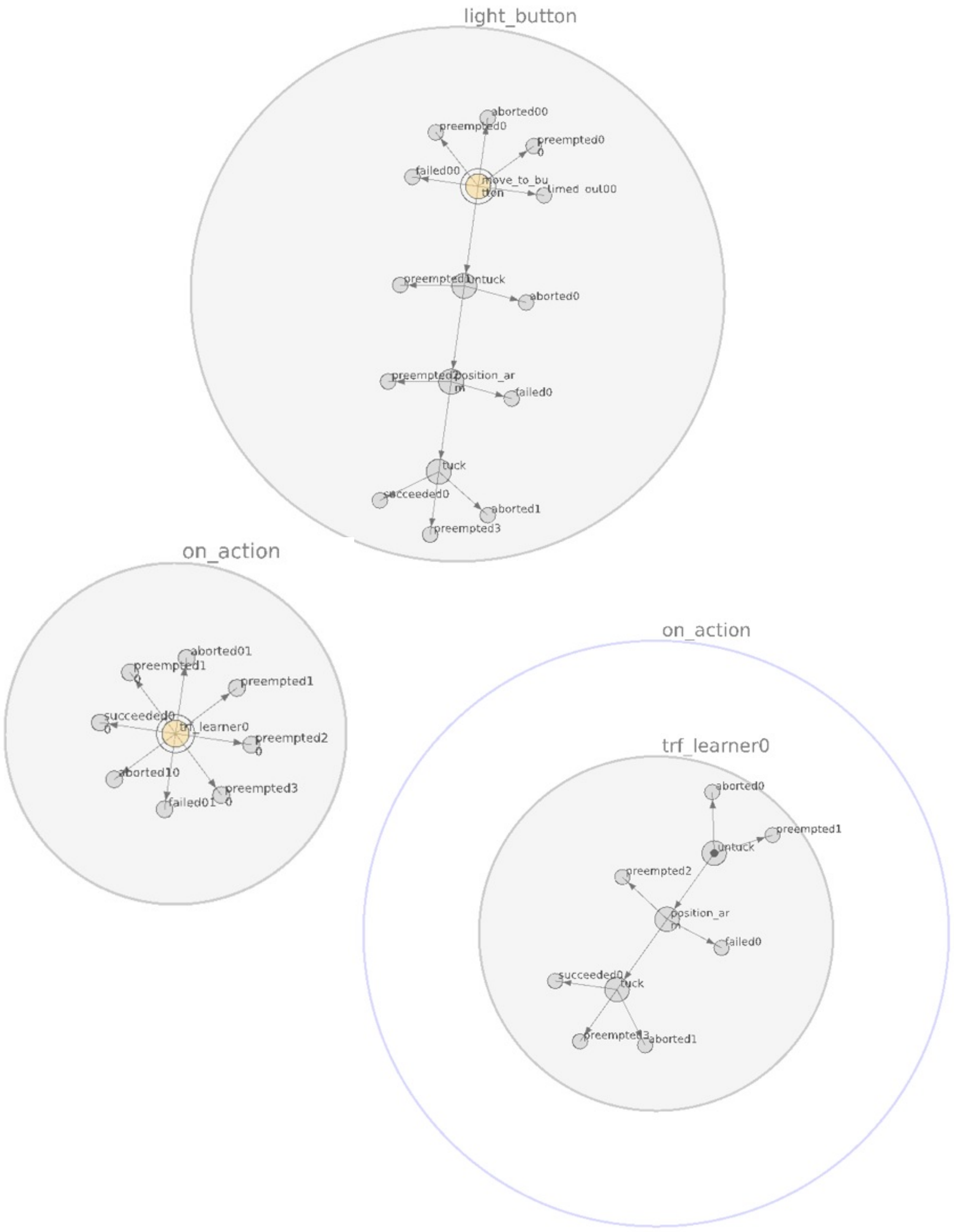
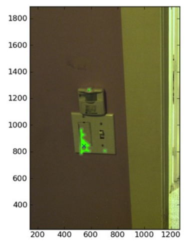
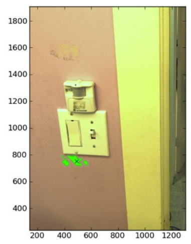
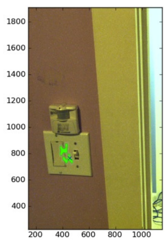
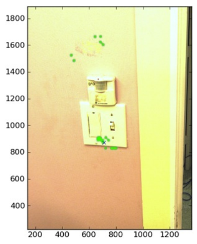


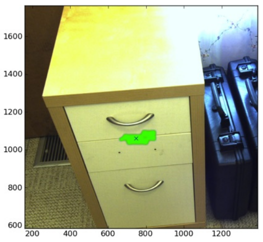
Figure 37: A preprocessed behavior to turn on a rocker switch constructed by a participants in our study from Chapter 3. ***Top:*** *Original behavior before modifications.* ***Middle:*** *The behavior with its stub removed and inserted into a learning container shown with the learning tool's options displayed on the right.* ***Bottom:*** *Double-clicking on the container displays the original behavior inside.*

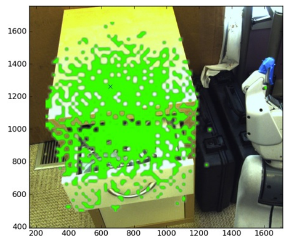

(a) Rocker switch behavior 1 (on)

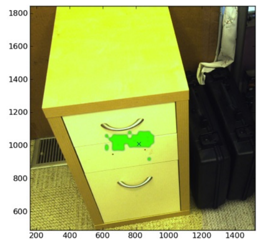

(b) Rocker switch behavior 1 (off)

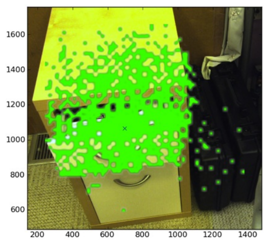

(c) Rocker switch behavior 2 (on)

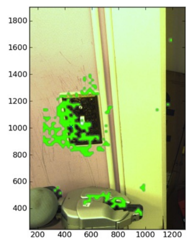

(d) Rocker switch behavior 2 (off)

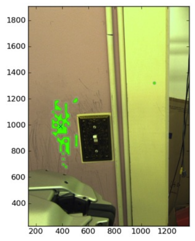

(e) Drawer 1 (pull)

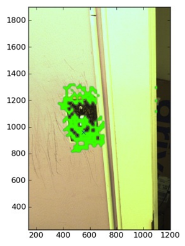

(f) Drawer 1 (push)


(g) Drawer 2 (pull)


(h) Drawer 2 (push)


(i) Light switch 1 (on)


(j) Light switch 1 (off)


(k) Light switch 2 (on)

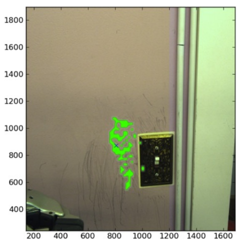

(l) Light switch 2 (off)

Figure 38: We trained two participants' behavior pairs to operate a rocker switch, drawer and light switch. Each pair of images shows classification results from learned detectors for complementary behaviors after training during a test run. Green labeled regions are areas predicted to lead to success, and the blue X's indicate locations that was used as input to the behavior.

CHAPTER VII

CONCLUSION

Creating general purpose robots able to perform autonomously in unconstrained home settings remains a significant challenge. As can be seen with other general-purpose technologies, such as with personal and mobile computing, there are a large variety of tasks that people might want a home robot to perform. Furthermore, with each one of these tasks there are many sources of variations making it difficult to build systems that can operate reliably across a wide range of environments. Even commonplace mechanisms such as, drawer handles and light switches, can vary in style, size, shape, and composition, potentially drastically altering the manipulation and sensing strategy that needs to be used. Additionally, traditional computer vision concerns also apply when operating under lighting conditions that can change hour-to-hour and with occlusions due to clutter. Overtime, as wear and tear sets in, the robot might also need to cope with further changes in the mechanism's mechanical properties and appearances.

In this thesis, we proposed methods which allow end-users to create, modify and adapt behaviors for new tasks, environments, and circumstances. By allowing end-users, and not just roboticists, to author behaviors, we can enable a larger population of users to construct a variety of different behaviors to address the multiplicity of tasks, and the variability within these tasks. As other computer literate users are now enabled control and program robots to perform household tasks, our approach offers a viable path for general purpose robots to inhabit home environments.

Key to its success is that our system relies on human intelligence and domain knowledge to provide for a behavior's overall structure and parameters for its components. However, when components such as perceptual classifiers have larger numbers of parameters that are infeasible to define by hand, we complement human guidance with autonomous learning, enabling the robot to gather data on its own to learn the classifiers instead of requiring constant supervision.

7.1 Robot Behavior Authoring Environments

In ROS Commander, end-users construct behaviors represented as HFSMs (Hierarchical Finite State Machines) by piecing together building blocks encapsulating common algorithms used in mobile manipulation. Not only are the building blocks themselves modular, but HFSMs enable large parts of behaviors to be conserved and reused in new circumstances. Entire behaviors can be used as subtrees of larger, more complex parent behaviors. We used this capability in our user study to provide participants with stubs, or parts of behaviors consisting of sets of states that are commonly used.

Theoretically, while Finite State Machines (FSMs) and HFSMs are more limited than general computing automata such as Turing machines, their implementation in our system and on a robot allow HFSMs to represent concepts that they would not normally be able to for two main reasons. First, HFSMs operating on robots are Turing complete since the robot system is able to use the world as the Turing "tape" to both read from and to store arbitrary state information. Second, even without the world, as the ROS (Robot Operating System) modules with which we construct the HFSMs can store state information in other parts of the system, our particular implementation of HFSMs again become the equivalent of Turing machines.

It is due to the above modularity and ability to perform a variety of different types of computations that we have been able to construct a wide variety of different behaviors in this work. Even though the behaviors that we have shown operate only on static infrastructural elements in homes and with relatively lower level actions such as opening a single refrigerator door or drawer, this is an artifact of limited implementation resources.

The same HFSM architecture, with additional building blocks, can be used to build more complex capabilities. Indeed, using SMACH, the same HFSM library which powers ROSCo, the authors of Meeussen et al. [98] constructed automated recharging behaviors by having the robot plug into an electrical outlet, and Hsiao et al. [63] demonstrated a reactive grasping behavior for household objects.

Most importantly, however, we are able to demonstrate through human subject experiments that HFSMs can be employed by non-roboticists, through our interface, to create robot behaviors that are of similar complexity to behaviors created by experienced roboticists. In subsequent tests with Henry Evans, a man with quadriplegia, the above results with able-bodied participants seem robust as Henry was able to succeed with similar outcomes.

7.2 Augmenting Behaviors with Autonomous Learning

Given a behavior and a mechanism, it is often difficult to know where best to apply that behavior due to complex interactions between the robot's embodiment and that object while executing the behavior. Furthermore it is also difficult for human users to define all the parameters of a visual classifier that would locate such points. In this work, we introduced autonomous learning as a tool in ROSCo to enable the robot to gather data on its own to discover these locations and train a visual classifier that will later identify locations likely to lead to success.

We showed that having access to two complementary behaviors allows a robot to learn autonomously. For example, in order to learn to open a drawer the robot must be able to close the drawer once it has opened that drawer. This is an issue not addressed by other learning work where it is usually assumed that the environment can be modified to suit the robot [115], that the robot can learn in a special playpen [135], that the robot can learn using specific tasks where success does not have lasting effects [136, 79], or that the experimenter can manually intervene [104]. With many tasks and in normal human environments, such assumptions can be untenable.

We showed further that our approach works both with roboticist hand-crafted behaviors and behaviors generated by participants during our user study. Additionally, we demonstrated that the underlying learning mechanic of adding new data samples can be used during operation, allowing the robot to adapt to failed manipulation attempts by retrying. This small step greatly increased our system's success rate from 88.3% to 97.5%.

In the context of end-users constructed behaviors, we believe that robot learning can enable these hand-crafted capabilities to function with more autonomy and robustness while mitigating the need for users to modify their environment. While there are a number of remaining issues to be addressed, we believe this usage of robot autonomy for learning within end-user created behaviors can make a significant impact in enabling robots to function in home environments.

7.3 Contributions

This thesis makes a number of contributions but we will summarize the most salient ones below:

1. We developed a new system for constructing robot behaviors based on Hierarchical Finite State Machines (HFSMs) and introduced general, parameterizable modules that can span a large variety of tasks useful in home environments. We demonstrated that the resulting behaviors are reliable and robust through experiments with various behaviors operating in non-laboratory environments, including: drawer opening, refrigerator opening, turning on the lights with a light switch, unlocking a door using a rocker switch, and handing over an object.

2. Through user studies with able-bodied individuals and a quadriplegic person, Henry Evans, we provided evidence that non-experts are capable of constructing behaviors of comparable complexity and functionality to those created by experts using our behavior interface ROS Commander (ROSCo) or similar systems. We have also shown evidence that even in the restricted settings of our study, with limited time and constrained goals, participants were still able to demonstrate their creativity through producing behaviors that used a variety of different manipulation strategies.

3. We demonstrated that autonomous learning can benefit manually programmed behaviors by using the robot's embodiment to interact with the world. Through these interactions it is then able to discover 3D locations on home mechanisms where those behaviors can attain success. We formalized the concept of complementary behaviors and listed the theoretical conditions under which learning without human involvement can be carried out. We showed empirical results of our system working with behaviors that include drawer opening, light switch flipping, and rocker switch pressing.

4. As far as we know, ours is the first instance of autonomous perception learning using behaviors constructed from basic building blocks by non-roboticists. Past work has only demonstrated learning using roboticist constructed behaviors.

7.4 Future Work

Although a combination of non-experts using ROS Commander and autonomous robot learning can already create many useful household behaviors, a number of extensions may increase the reach and applicability of our system.

7.4.1 Operators in ROS Commander

As ROS Commander uses HFSM as the representation for its behaviors, operators defined on these graphs can be powerful tools for end-users. Adding just a few operators may make the creation of many types of behaviors simpler.

- Loops are a common and powerful control flow structure in many programming languages. Implementing looping in a general manner that enables users to create periodic motions used in applications such as cleaning a surface, carrying multiple objects between two different spaces, rotating a door handle, stirring a pot, or scooping cat litter can dramatically improve the usefulness of ROS Commander.
- Multi-class classification is another primitive that has been found to be useful in many robotic applications. Particularly, being able to detect the current state of an object such as determining visually whether a door is open, or closed enables behaviors to verify that a mechanism is in an appropriate state before operating on it, improving robustness to uncertainty.
- Similarly, determining whether the trace of sensor readings resulting from a behavior's execution is abnormal or not and where in that execution trace it became atypical can function as a generic failure detection mechanism, and thus would help to improve robustness. Human environments are often unpredictable with many failure conditions which are difficult for behavior designers to anticipate, roboticist or not. For example, a door opening procedure can fail due to a myriad of reasons, such as doors being locked, blocked, or broken. However, for the goal of opening doors, it only matters whether the door was opened with all other outcomes being deviations from that result allowing methods based on outlier detection to excel [115].

7.4.2 Online Behavior Warehouse for Common Household Mechanisms

While the various behaviors shown so far in this thesis are useful in their own right and have moderate generalization power–i.e. possibly being able to operate a few novel mechanisms–they are not sufficient to enable general purpose manipulation. The diversity of physical forms and variability of household objects pose a great challenge to the generalization capability of even the most carefully crafted behaviors. Additionally, similar to other applications such as collaborative wiki editing and websites containing user-generated content, we expect that the majority of users will not be behavior creators, but consumers instead.

As such, we believe that having an online repository akin to Google's 3D Warehouse but for behaviors indexed by properties such as the type of mechanism that the behavior can be used with will enable a dramatic increase in the capability of robots to interact with the home. With more robots in use, such a system can begin to gather data during operation, enabling concepts acquired by systems such as the autonomous learner in Chapter 5 to be more general and robust to variations. Similarly, simple success statistics on behaviors constructed, their compositions, and the objects with which they interact gathered across a large range of environments can advance the understanding of household robotics significantly.

7.4.3 Interaction and Visualization Improvements in ROS Commander

As much of the interactions with our system occur while users record, test, and edit motion trajectories, we believe that having more intuitive mechanisms facilitating this process can vastly improve ROS Commander and similar behavior authoring systems. There are other approaches to trajectory modification including using reinforcement learning to reshape [115, 80, 116] movements with task success as a feedback signal and aggregating multiple noisy demonstrations [16] to create a more complete local trajectory model. However, for manipulation in homes where unrestricted autonomous practice can be damaging and a lack of interaction models for everyday objects, an approach with more explicit interactions such as ours can potentially be more powerful and widely applicable.

In future work, we believe exploring different visualizations of the trajectories generated by each module and their properties, such as velocity and forces expected in context of the environment that it interacts with, can provide better means for users to create effective behaviors. Additionally, mechanisms used in traditional media authoring such as splits, fades, joins, and cuts can also potentially be useful in this new medium of robotic manipulation.

7.4.4 Generalization of Autonomous Learning

The system that we have presented so far was intended to demonstrate the potential of fully autonomous learning on a robot, showing that it can lead to greater robustness and capable behaviors. As such, our current method also makes the unnecessary assumption that each new mechanism a given behavior interacts with is completely different from any mechanism that it has interacted with in the past. This need not be the case as the objects that a given behavior interacts with are often related, but are not necessarily identical, in their visual appearance and mechanical properties. For example, although the appearances of light switches can vary drastically between homes, within a single residence their appearances tend to match.

In future work, by exploiting the similarity between these related learning tasks using transfer learning [113] we believe that there is a possibility to further decrease the number of training examples needed for learning each new mechanism. In our setting, when using an over-complete basis feature set we can assume the source and target domain to be identical. This allows the use of many existing inductive transfer learning methods such as those that depends on transferring knowledge of instances [51] or knowledge of SVM weights [58].

By exploiting the visual similarities between mechanisms that a behavior operates, there is a possibility that we can enable robots to operate on novel devices with very few learning attempts, and reduces the amount of set up time needed to enable a robot to operate in novel home environments. More generally, such capabilities enables progress towards a robot capable of lifelong learning, a topic of recent interest in the robotics community.

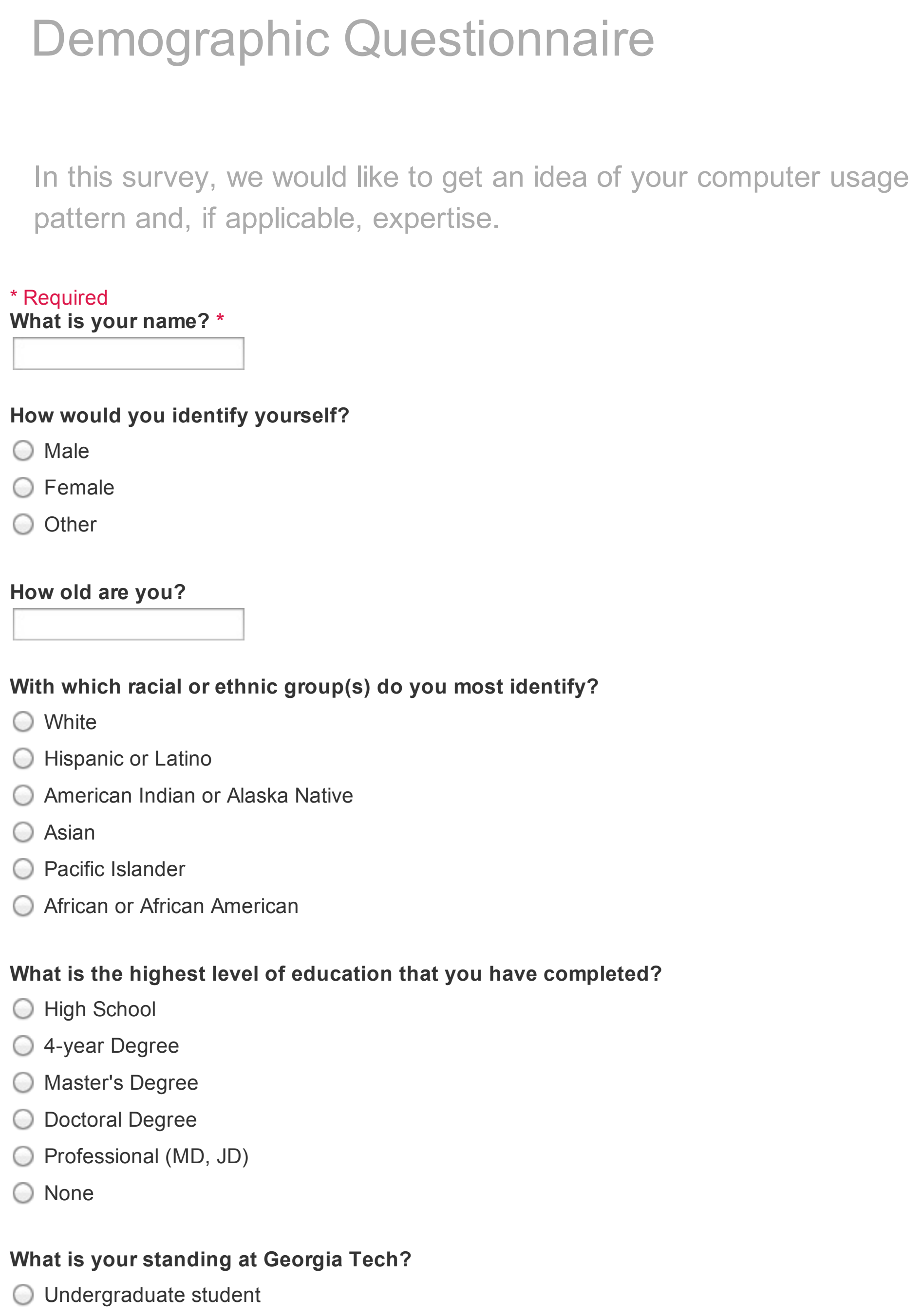
7.5 Conclusion

There is a great deal of work remaining to create a practical home robot, ranging from creating smaller, more robust mobile manipulators to better error detection and perception algorithms. However, we believe that a combination of software authoring environments along with autonomous learning and data collection can result in capabilities diverse, robust and reliable enough to create a robot that can exist in an everyday home environment.

APPENDIX A

ROSCO USER STUDY SURVEYS

A.1 Demographic Questionnaire



Demographic Questionnaire

In this survey, we would like to get an idea of your computer usage pattern and, if applicable, expertise.

* Required

What is your name? *

How would you identify yourself?

- ○ Male
- ○ Female
- ○ Other

How old are you?

With which racial or ethnic group(s) do you most identify?

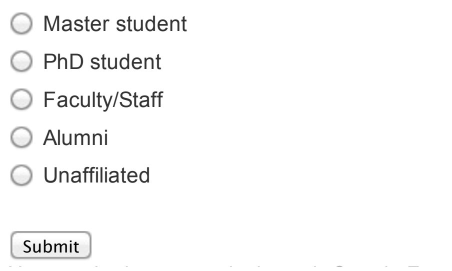
- ○ White
- ○ Hispanic or Latino
- ○ American Indian or Alaska Native
- ○ Asian
- ○ Pacific Islander
- ○ African or African American

What is the highest level of education that you have completed?

- ○ High School
- ○ 4-year Degree
- ○ Master's Degree
- ○ Doctoral Degree
- ○ Professional (MD, JD)
- ○ None

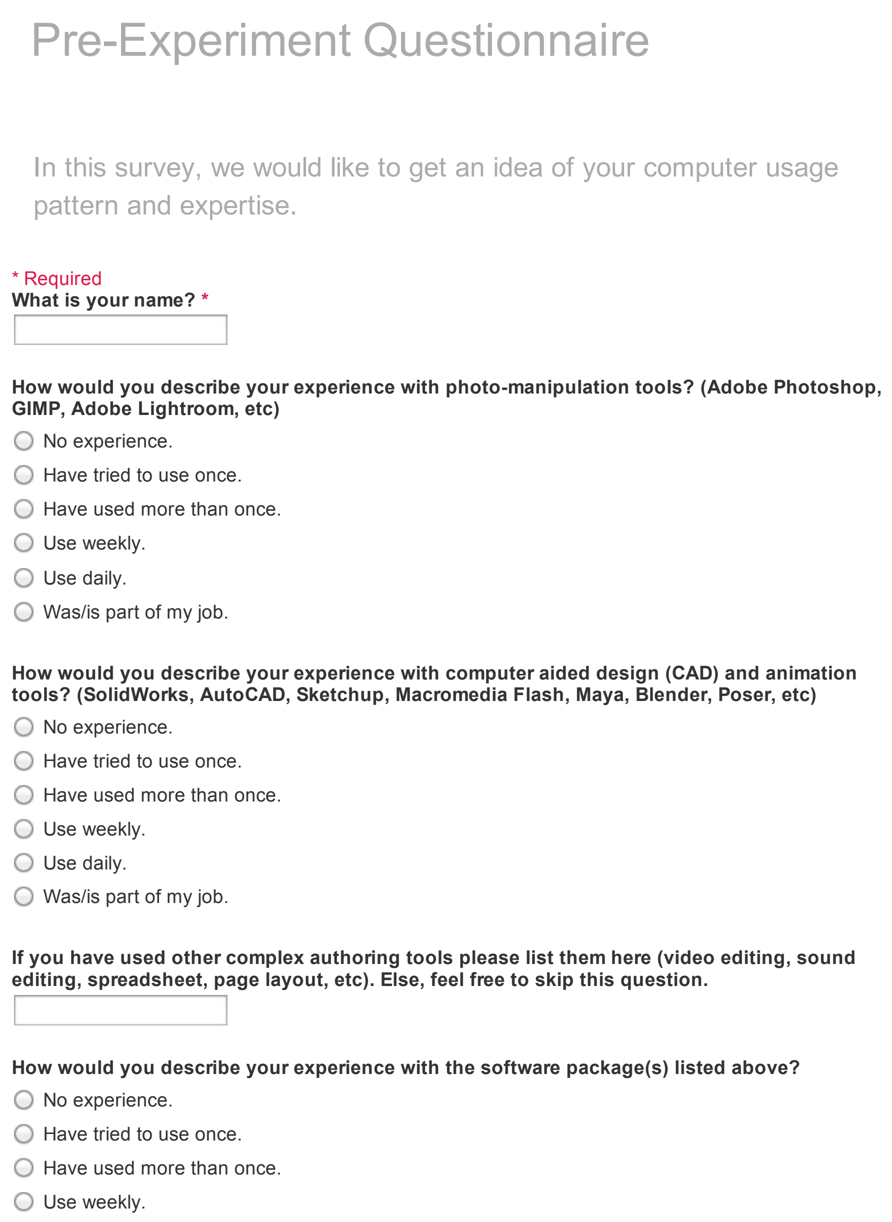
What is your standing at Georgia Tech?

- ○ Undergraduate student



- Master student
- PhD student
- Faculty/Staff
- Alumni
- Unaffiliated

A.2 *Pre-experiment Questionnaire*



Pre-Experiment Questionnaire

In this survey, we would like to get an idea of your computer usage pattern and expertise.

* Required

What is your name? *

How would you describe your experience with photo-manipulation tools? (Adobe Photoshop, GIMP, Adobe Lightroom, etc)

- No experience.
- Have tried to use once.
- Have used more than once.
- Use weekly.
- Use daily.
- Was/is part of my job.

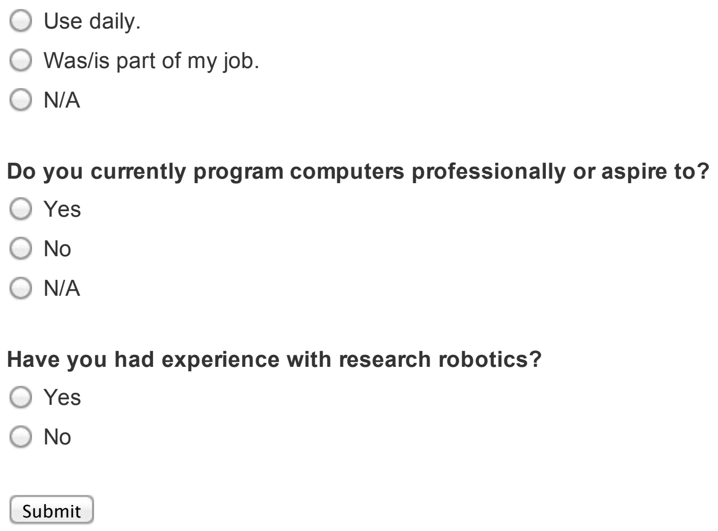
How would you describe your experience with computer aided design (CAD) and animation tools? (SolidWorks, AutoCAD, Sketchup, Macromedia Flash, Maya, Blender, Poser, etc)

- No experience.
- Have tried to use once.
- Have used more than once.
- Use weekly.
- Use daily.
- Was/is part of my job.

If you have used other complex authoring tools please list them here (video editing, sound editing, spreadsheet, page layout, etc). Else, feel free to skip this question.

How would you describe your experience with the software package(s) listed above?

- No experience.
- Have tried to use once.
- Have used more than once.
- Use weekly.



○ Use daily.
○ Was/is part of my job.
○ N/A

Do you currently program computers professionally or aspire to?

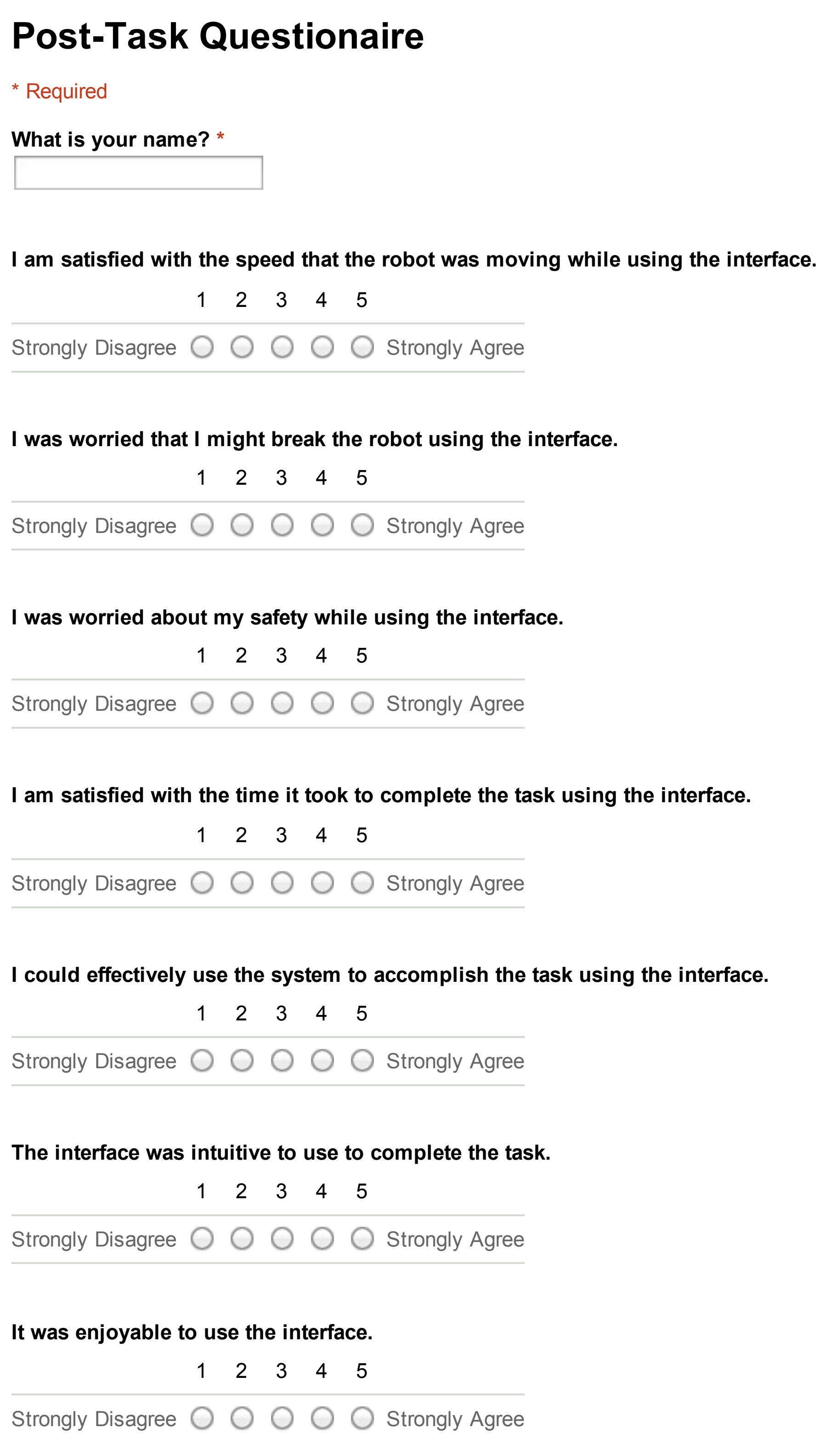
○ Yes
○ No
○ N/A

Have you had experience with research robotics?

○ Yes
○ No

Submit

A.3 Post-experiment Questionnaire



Post-Task Questionaire

* Required

What is your name? *

I am satisfied with the speed that the robot was moving while using the interface.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

I was worried that I might break the robot using the interface.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

I was worried about my safety while using the interface.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

I am satisfied with the time it took to complete the task using the interface.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

I could effectively use the system to accomplish the task using the interface.

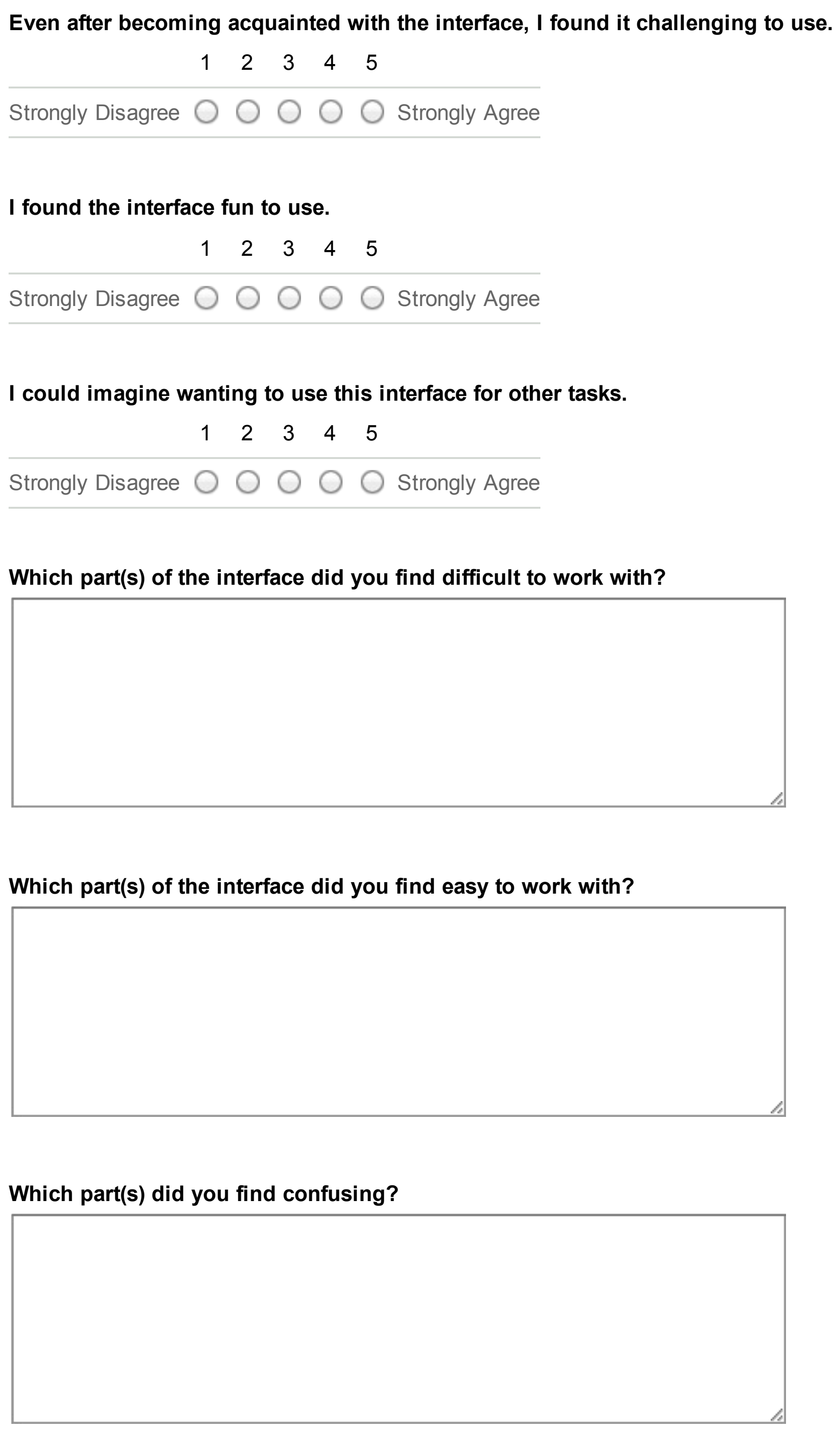
	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

The interface was intuitive to use to complete the task.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

It was enjoyable to use the interface.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree



Even after becoming acquainted with the interface, I found it challenging to use.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

I found the interface fun to use.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

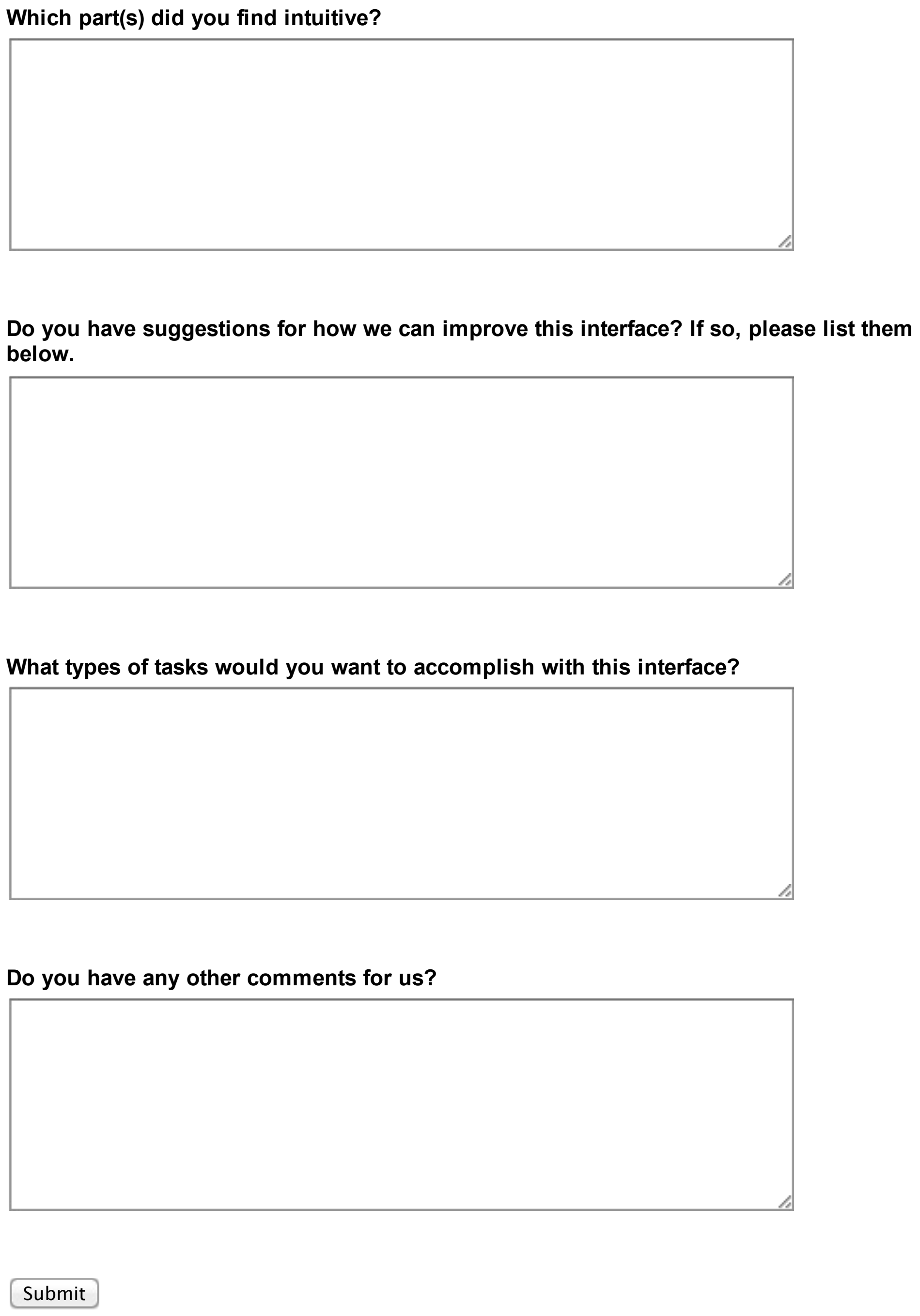
I could imagine wanting to use this interface for other tasks.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

Which part(s) of the interface did you find difficult to work with?

Which part(s) of the interface did you find easy to work with?

Which part(s) did you find confusing?



Which part(s) did you find intuitive?

Do you have suggestions for how we can improve this interface? If so, please list them below.

What types of tasks would you want to accomplish with this interface?

Do you have any other comments for us?

Submit

APPENDIX B

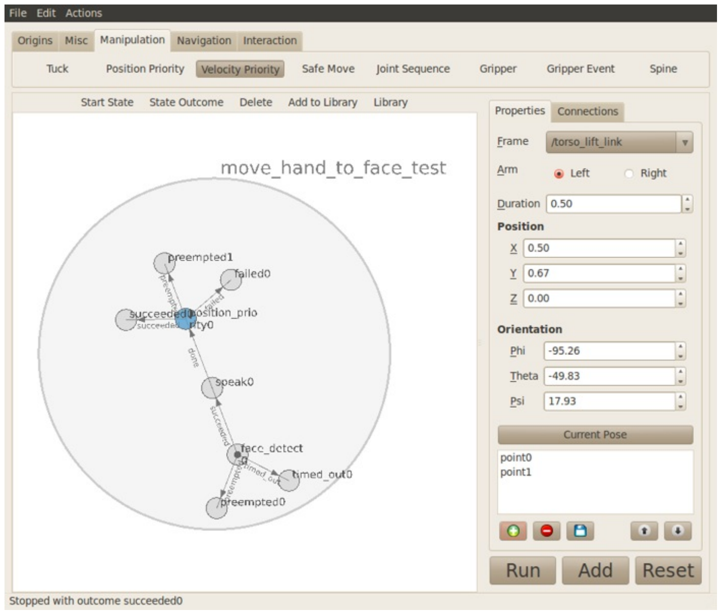
ROSCO MANUAL

The online version of this manual is located at `http://wiki.ros.org/rcommander_core` and `http://wiki.ros.org/rcommander_pr2`.

ROSCo is a stack for graphical editing of SMACH state machines. The hope is that ROSCo-style expert interfaces will allow end-users (non-programmers) to comfortably program new robot behaviors. ROSCos infrastructure allow construction of graphical user interfaces that allow editing of parameters in SMACH states, sequence and compose those states as well as save and load them from disk. The state machines constructed are compatible with all code that takes SMACH state machines as input.

Currently the main focus is on the PR2 robot but there is emerging support for the Turtlebot platform, and hopefully there will be others. The tutorials and documentation here will help you to adapt ROSCo to new platforms.

B.1 Users Guide



Frame /torso_lift_link
Arm [✓ Left] Right
Duration 0.50
Position
X 0.50
Y 0.67
Z 0.00
Orientation
Phi -95.26
Theta -49.83
Psi 17.93
point0
point1

Stopped with outcome succeeded0

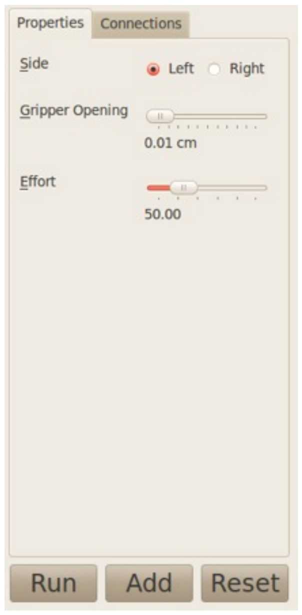
Figure 39: Main ROSCo interface.

Above is the basic ROSCo interface. ROSCo is a tool for creating robot behaviors. Each robot behavior is represented as a network of linked actions where each action (or node) is a circle and arrows indicate the ordering in which those actions occur. The main job of the interface is to allow you to specify parameters that determine what each action will do and in which order.

There are three main regions to the interface itself:

- **Tools Panel:** Top tabbed area with each tab containing a grid of buttons where each button activates its corresponding tool.
- **Properties and Connections:** This area is shown on the right hand side. When either a tool button or action is selected this area fills up with the current parameters in effect. The properties tab contain parameters of the currently selected object. The connections tab contain information about how the current node is connected to other nodes.
- **Current Behavior:** This is the white area containing little circles where each circle represents a node created through one of the tools in the Tools Panel.

B.1.1 Controlling the Robot



Side: [✓ Left] Right
Gripper Opening: 0.01 cm
Effort: 50.00

Figure 40: Gripper block.

Each button in the tool panel activates a robot action. Try this out by clicking on the Gripper tool (in the Manipulation tab). After activating the Gripper tool (Figure 40), the properties panel populate with controls for opening and closing the gripper. For instance, the properties panel show options for selecting either the left or right gripper under Side, how wide to open/close the gripper under Gripper Opening, and how hard the gripper should close under effort (this number does not have physical meaning, the default 50 should be fine for gripping most things).

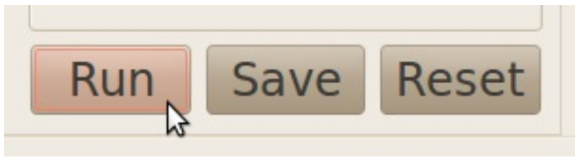


Figure 41: State run button.

Now click Run (Figure 41). The robot should either open or close its gripper in response. The status bar at the bottom of ROSCo should read ”succeeded0”.

B.1.2 Creating Your First Node

Controlling the gripper was great but let's combine it with other actions. Activate the Gripper tool again. Now, Select the settings desired using its controls. Next, click add. Clicking add creates a new cluster of nodes with the gripper action at the center highlighted in blue. Feel free to click on the other nodes. Notice that clicking on a node causes it to be highlighted in blue. Another change that happens when clicking is that the properties and connections tab fill up with the appropriate information for that particular node. Unlike gripper0, the other three nodes on screen preempted0, succeeded0, and aborted0 represents outcomes that can occur after running gripper0 and that, accordingly, there are no properties to edit nor connections to make with them.

B.1.3 Connecting a 2nd Action

With the first action added, now click on Tuck to activate the tuck tool and change the dialog box options to False for both left and right arm. Now click add. This creates another cluster of nodes on screen separate from the first cluster. To connect the first cluster, select gripper0, Connections, then under succeeded select tuck0. The graph will now show that after executing gripper0 successfully, the robot will now execute tuck0 as indicated by the arrows. Your robot behavior should now look like Figure 42.

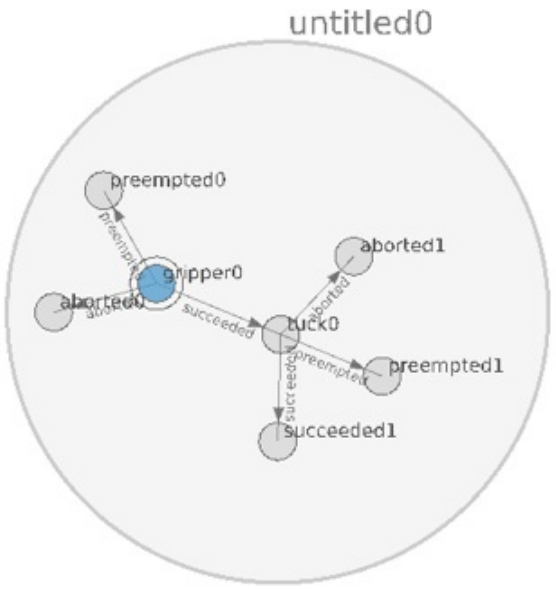


Figure 42: The state machine that you should see.

If you are operating in a simulator, also connect aborted0 to tuck0 to avoid any troubles (this is due to a small bug with the gripper action in simulation). Now run the behavior by clicking on gripper0 and selecting Start State. This sets gripper0 as the starting point of our behavior (the display reflects this by adding an extra circle around gripper0), meaning that when we run this network gripper0 will be the first to run. Next, click on Actions then Run. The robot will now execute all the actions starting with gripper0 and other actions according to the direction of the arrows. The display will indicate which action is executing by highlighting the currently active action in green.

B.1.4 Running a Node

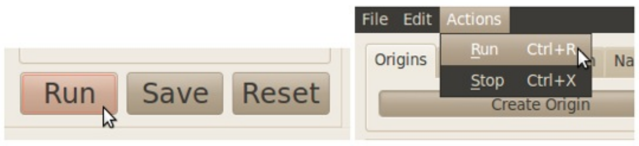


Figure 43: Running a node.

In normal circumstances, you might not want to execute both actions, or the entire behavior as shown but just an individual action. To do so, click on the action to run and click on the large Run button under the properties and connections panel (left in Figure 43). Notice that this Run has a very different functionality from the Run under the Actions menu (right in Figure 43). The large run button only runs the node selected. The run menu option, on the other hand, runs the entire sequence of nodes starting with nodes that has a double circle.

B.1.5 Stopping Execution

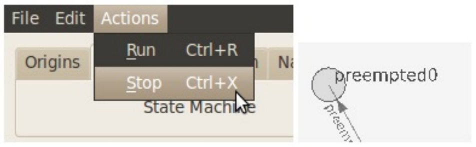


Figure 44: Stopping the execution of a node.

While running with either of the methods above, you might want to cancel execution. To do so, select Actions then Stop. This should cause whatever is running to transition to the preempted output (right in Figure 44).

B.1.6 Modifying the parameters of a Node

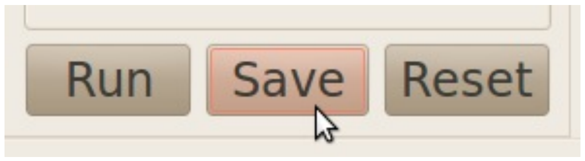


Figure 45: Saving a node.

If you now want to modify one of the properties of either tuck0 or gripper0, select it, make your changes in the properties or connections box then click Save at the bottom of the screen (Figure 45). To preview your changes prior to saving, you can also click Run.

B.1.7 Reset

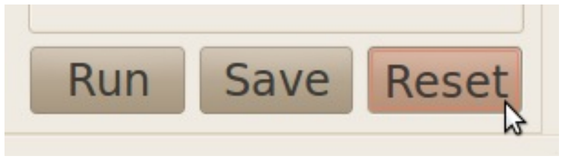


Figure 46: Saving a node.

The big Reset button (Figure 46) at the bottom when clicked will reset the properties panel back to its default setting. However, reset does not save those changes. To make reset's changes permanent, click Save.

B.1.8 Deleting Unwanted Nodes

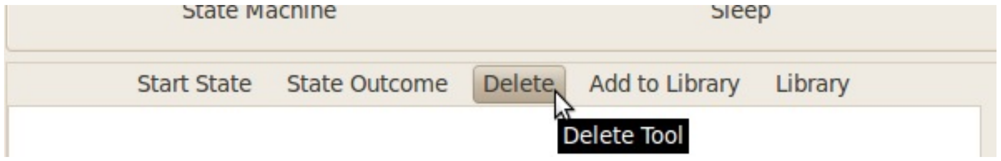


Figure 47: Saving a node.

To delete nodes that are no longer wanted, select that node, then click Delete underneath the tool bar (Figure 47). Deleting an action node such as tuck0 or gripper0 will also delete all its outcomes. However, deleting an outcome will merely cause another outcome node to take its place as outcome nodes are place holders.

B.1.9 Saving and Loading Your Work

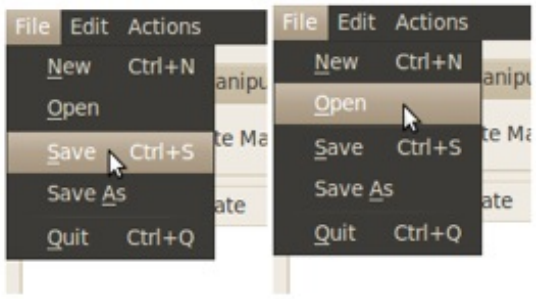


Figure 48: Saving a node.

ROSCo works like most normal applications. To save or load your work, click on File then Save or Open as needed (Figure 48).

B.1.10 States Library

Creating gestures such as in the last tutorial is a labor intensive process. The library allow storing individual actions inside it so that this effort can be reused in the future.

To store an action in the library, select that action then click on Add to Library underneath the toolbar (Figure 49).

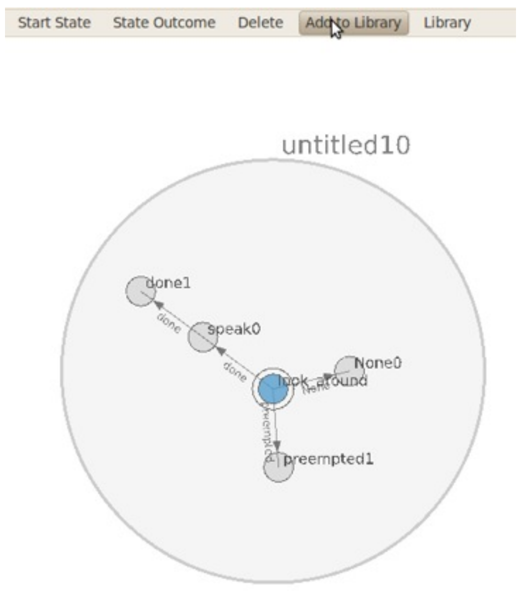


Figure 49: Adding a node to the states library.

Later, clicking on Library will show a list of all actions that have been saved to it (Figure 50). Selecting a action and clicking on Add will add a copy of it to your current behavior. Also, clicking on Run will run the currently selected action in the library.

B.2 Building Blocks

In the following sections, we will discuss each building block in ROS Commander in more details.

B.2.1 State Machine

The state machine tool (Figure 51) allows embedding another behavior inside the one that is currently active. State machine in this context just refers to the formal classification of behaviors created in ROSCo.

As an example of how to use this, let's say you have constructed a greeting behavior that says hello and waves after it detects a person and now you would like to reuse that entire behavior in a series of new behaviors where each starts with a robot greeting. With this tool you can embed the greeting behavior into your new behaviors.

To begin, first click the Select button and navigate to a location on disk where the other behavior resides. Next, the file path to that behavior should appear in the Filename field. Now click the add button. A new yellow node should appear like the above. All of the possible outcomes of your

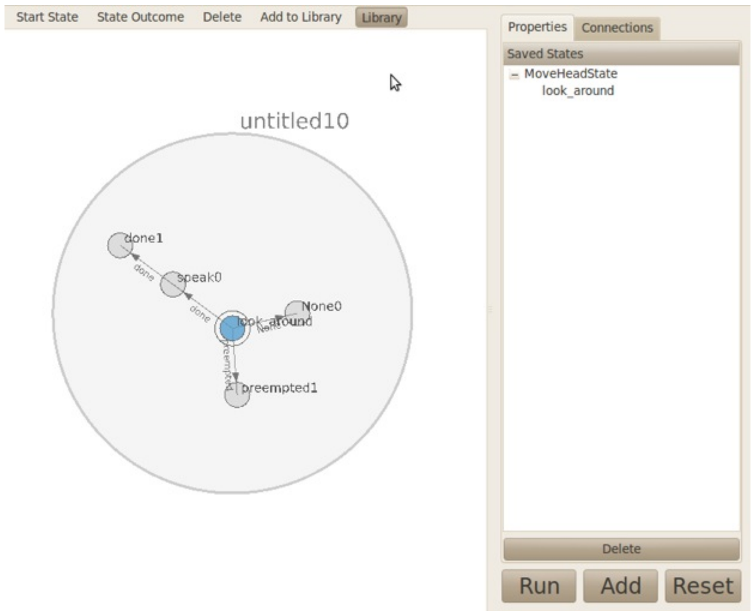


Figure 50: State added to states library.

behavior should appear as outcomes of this new node.

The inserted behavior is still available in our current behavior. To open it up, double click on the yellow node (Figure 52). You will now see the contents of the inserted state machine. Additionally, there will be an extra circle around it to indicate that this is an expanded view. To go back up to where we were, double click anywhere between the outer and inner circle.

B.2.2 Freeze Frame

This tool creates a frame out of any other moving frame (shown in Figure 53). These frames are meant to be used in situations where you want to specify a motion relative to a the position of a robot's body part at a time in the past.

For example, you might want to record a generic pulling behavior that pulls for 15 cm starting at the left gripper's current position. In this case, you would create a Freeze Frame action for l_gripper_tool_frame then create either a Velocity Priority or Position Priority node that pulls for 15 cm using the frozen frame as a reference point. If you attempted this using the l_gripper_tool_frame as a reference point it would not work as the l_gripper_tool_frame itself would be continuously moving!

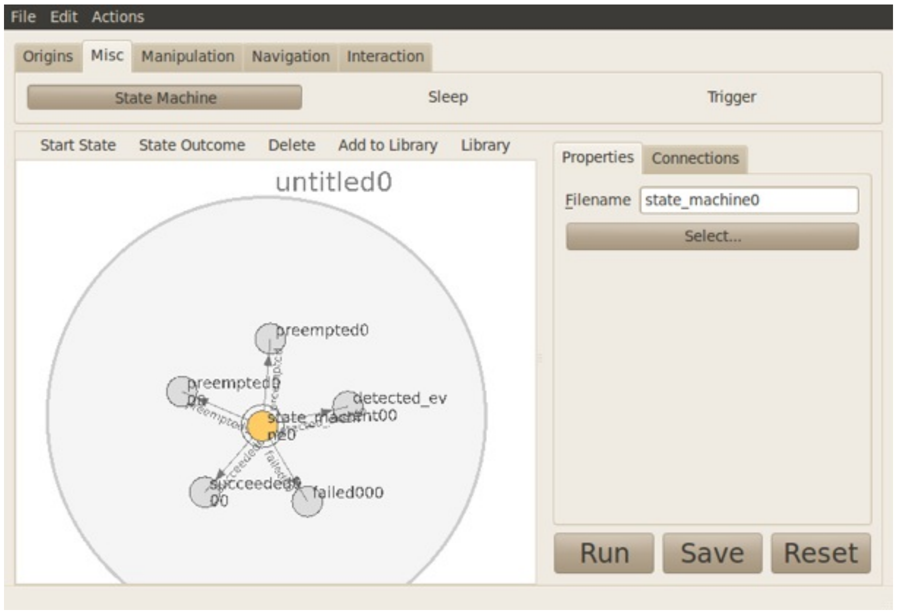


Figure 51: A hierarchical state.

B.2.3 Create Origin

The create origin tool allows you to create a custom 3D frames of the same name as the node's name (shown in Figure 54). If you know the precise position of the frame enter it into given fields. The orientation of that frame will be inherited from the frame specified under theFrame field.

However, it can be difficult to provide a 3D position so to get these filled out automatically you can use the RViz broadcast position option. Do this by clicking the Get Point button.

Next, broadcast points from RViz by right clicking the PR2's head, selecting Take Snapshot, then right clicking on a point in the snapshot that you want to broadcast and select Broadcast Click Position (Figure 55). You might have to do this more than once. The Create Origin tool will now be filled with your point. If this node then executes it will create a frame located at the given position with the inherited orientation.

Another use for this tool is to provide frames at run time for a behavior allowing entire motions to be described relative to a Broadcast Click Position command. For example, you can create a door opening or drawer opening behavior by creating a frame placed at the door or drawer's handle then creating a motion described in this new frame with the Velocity Priority tool.

To do this, instead of clicking Get Point, check the Wait for Point box. The Time Out field now

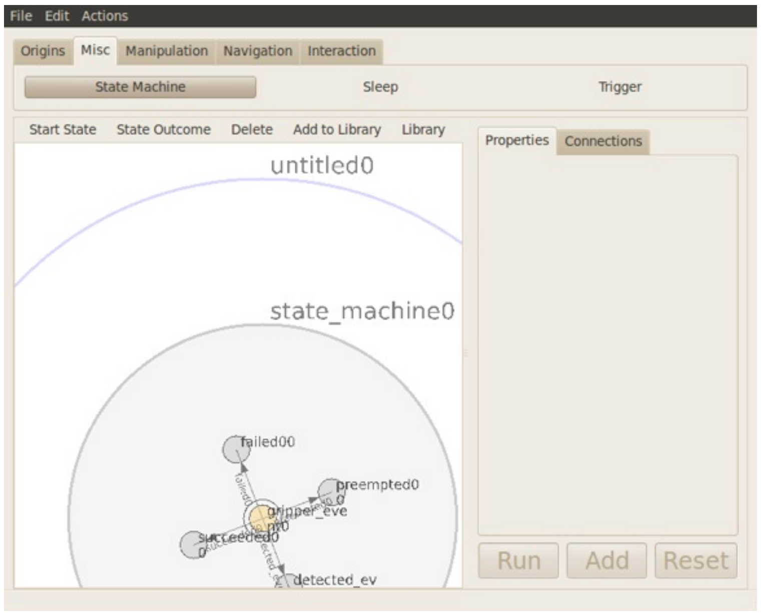


Figure 52: An expanded hierarchical state

specifies how long this node should wait for a Broadcast Click Position event to occur.

B.2.4 Trigger

The trigger tool creates a node that pauses the robot's behavior until the behavior receives a trigger signal (Figure 56. It gives up waiting for the trigger signal after the amount of time indicated in the Timeout field.

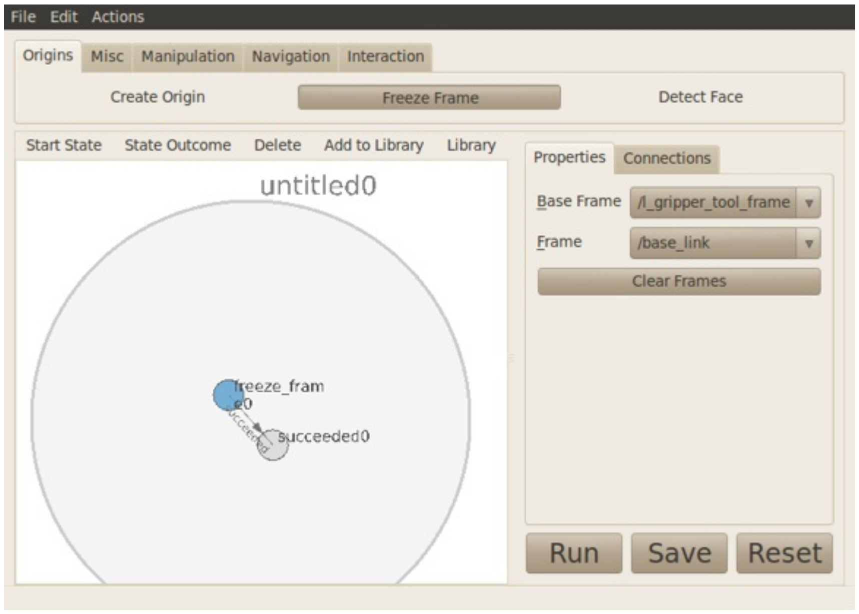
To send the trigger signal while this action is active, use the Send Trigger button.

B.2.5 Sleep

The sleep tool (Figure 57) creates a node that pauses the robot behavior for the given amount of time.

B.2.6 Joint Sequence

The joint sequence tool (shown in Figure 58) allow users to move the PR2's arm to given joint configuration. A joint configuration is a collection of angles that describe the robot's joints with respect to each other. In the dialog shown above, each field of number is displayed next to the name of its joint. For example, shoulder_pan_joint shows the current angle of the shoulder pan joint on the PR2. Each angle here is displayed in degrees (not radians). Although it is possible to type in



Base Frame: /l_gripper_tool_frame
Frame: /base_link
Clear Frames

Figure 53: Freeze Frame block

each joint configuration, the simplest way to assign a motion is to turn on the Live Update mode and move the arms of the robot using RViz (or physically) to desired configurations, clicking the green add button at each configuration. When executed this action will try to move the arm smoothly through the given joint configurations.

Unlike either the Position Priority tool or the Velocity Priority tool, the smooth motion produced only smooths each joint individually, without guaranteeing anything about the gripper's motion. If you need to move the gripper itself smoothly in a line, use either of the tool mentioned.

B.2.7 Move Head

The move head tool (Figure 59) functions the same way as the joint sequence tool except that it allows movement of the PR2's head.

B.2.8 Gripper

The gripper tool (Figure 60) controls the size of the gripper's opening and how hard the gripper should open and close. Here Side selects which gripper to control, Gripper Opening controls how wide the gripper opens or closes (in centimeters), and Effort controls how hard the gripper should close or open if something is in the way.

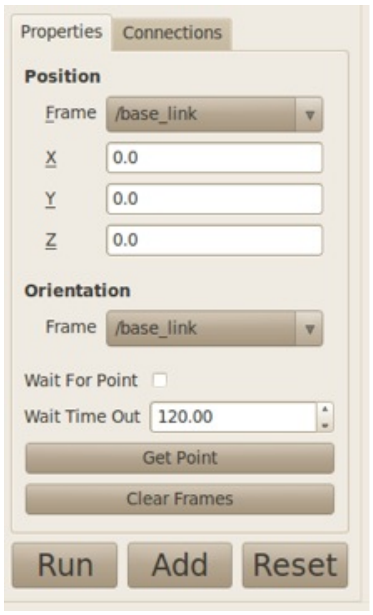

Properties
Connections
Position
Frame: /base_link
X: 0.0
Y: 0.0
Z: 0.0
Orientation
Frame: /base_link
Wait For Point: ☐
Wait Time Out: 120.00

Figure 54: Create Origin block

B.2.9 Gripper Event

The gripper event tool (Figure 61) allows monitoring the sensors in the gripper for bumps to the gripper. The action produced can detect bumps to the gripper that activates the accelerometer inside the gripper or the pressure sensors at the tip of the robot's fingers depending on the settings given.

The Event drop-down allows selection of the type of event that this action will detect. The Acceleration sliders allow selection of the accelerometer threshold to use if detecting accleration at the gripper. This value sets the magnitude of the bump that will be considered as an event. The higher the value the more insensitive the action will be to bumps. However, if this value is set too low, minor vibrations by the robot while moving will also register as a bump to the gripper.

Similarly, the Slip slider sets the sensitivity to slip events. Slip events occur when the robot's gripper have closed on an object and that object is slipping out of the gripper. These events allow the robot to detect slips and, hopefully, respond appropriately. The slip slider sets a value where the higher the value the more insensitive to slippage between the fingers.

Gripper events are designed to be used in conjunction with other actions in ROSCo by adding another outcome to other actions. In the above screenshot we have a gripper action with the three usual outcomes: aborted, preempted, and succeeded. We then add event detector by selecting the

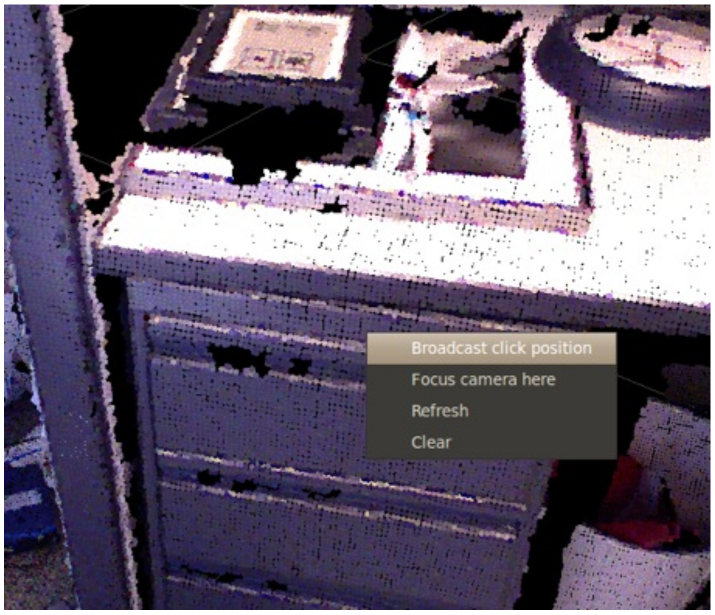


Figure 55: Broadcasting click position.

gripper action (highlighted in blue), clicking Gripper Event, then clicking Add.

After adding, the original gripper action turns yellow with an added outcome that says detected event. Double-clicking on this yellow node will show you the original gripper action allowing you to make changes (if needed) similar to the nodes produced by the state machine tool (Figure 62). Connect another node to the detected event outcome to respond to detected events.

The gripper event action is not restricted to just encapsulating the gripper action. In fact, it will work on virtually any other action in ROSCo including things such as sleep actions.

B.2.10 Safe Move

Sends the arm to a set of static joint angles, will try to keep the arm from colliding with other objects. However, the motion will fail if the arm is initially in contact with something (such as holding an object) or the destination set of joint angles causes the arm to be in contact. It might also fail if the robot hallucinates obstacles.

B.2.11 Spine

The spine tool allows movement of the PR2's spine, either raising or lowering the robot's upper body. It has one simple slider to set the height of the PR2's spine when it executes (shown in Figure 63).

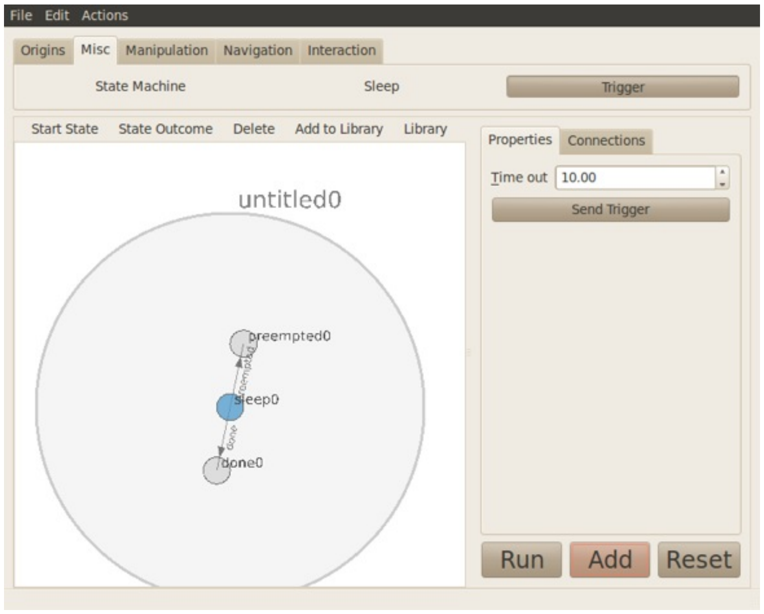


Figure 56: Trigger block.

B.2.12 Tuck

This tool produces an action that allows the robot to tuck and untuck its arms (shown in Figure 64). As the robot is fairly large, tucking allows the robot to shrink down making it easier to move in small spaces. Tuck has two options, Left Arm and Right Arm with the True option specifying that the associated arm should be tucked and vice versa for False.

The tuck tool will not work if the arms are currently under control by RViz. To see if this is the case look for either tri-colored circles around any of the grippers or red circles around the shoulders of the robot in RViz. If tuck nodes seem to hang, click on the appropriate arm part to turn off all the circles in RViz.

The tucking action will also abort if the arms hit and gets stuck on other objects in the environments.

B.2.13 Velocity Priority

The velocity priority tool creates an action that moves the gripper to a series of points (Figure 65). Unique to this tool is the ability to send commands to both arms at once. To use this tool, select the desired frame then iteratively pose either of the robot's arms, click Current Pose, then click the red add button until there are enough points in the trajectory.

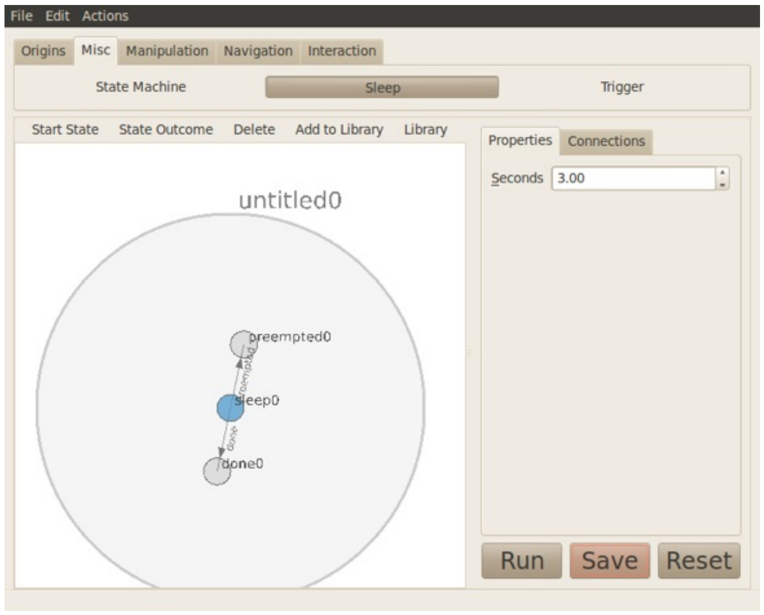


Figure 57: Sleep block.

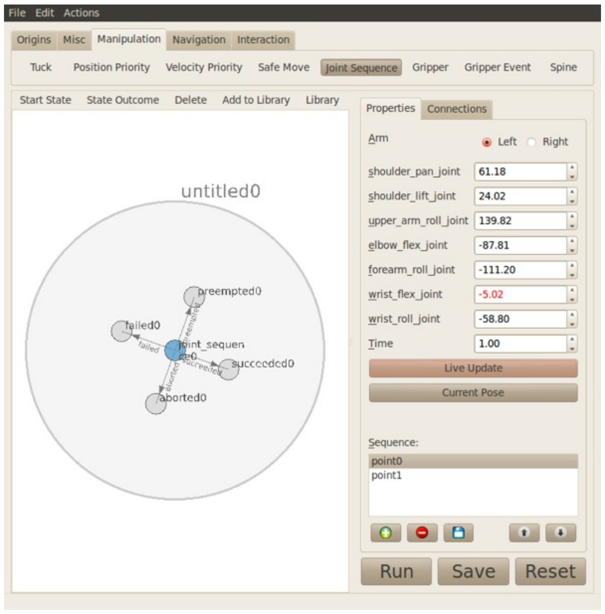
Unlike the Joint Sequence tool, this motion only records the position of the gripper but the recorded motion can be more flexible. For example, if the recorded motion's frame is a frame defined by a person's face, the motions produced by this tool can be retargeted to the face every time it moves while the joint sequence tool's motions will be fixed to its original recorded trajectory.

Also, unlike the Position Priority tool, this tool will allow movements in sequence to an arbitrary number of points but will send points according to a stringent time schedule abandoning points that it cannot reach due to obstructions. This property makes the tool much more suited to usage for tasks that involve manipulation while being in contact such as opening cabinets or drawers.

B.2.14 Position Priority

The position priority node (Figure 66) produces an action that moves the robot's gripper to a new location in a straight line. Unlike the Velocity Priority tool's action, this action will only move to one new point and if the gripper's path is obstructed it will try to push through and will take more time to execute if needed.

This tool provides a number of options to customize the motion it produces. Frame here is the standard frame option, enabling the motion to be described in the frame of reference of choice. Unlike the velocity priority tool, it is safe to to describe motions with respect to the either of the


File Edit Actions
Origins Misc Manipulation Navigation Interaction
Tuck Position Priority Velocity Priority Safe Move Joint Sequence Gripper Gripper Event Spine
Start State State Outcome Delete Add to Library Library
Properties Connections
untitled0
preempted0
failed0
joint_sequence0
succeeded0
aborted0
Live Update
Current Pose
Run Save Reset

Arm: [✓ Left] Right
shoulder_pan_joint 61.18
shoulder_lift_joint 24.02
upper_arm_roll_joint 139.82
elbow_flex_joint -87.81
forearm_roll_joint -111.20
wrist_flex_joint -5.02
wrist_roll_joint -58.80
Time 1.00

Sequence:
point0
point1

Figure 58: Joint sequence block.

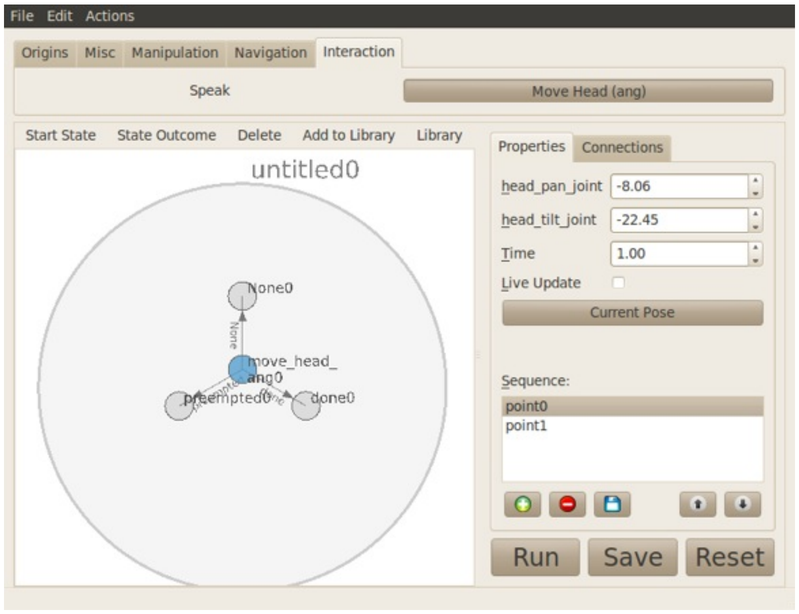
gripper's frame (l_gripper_tool_frame or r_gripper_tool_frame) of references.

The Motion section allows users to vary the rotational and translational speed of the motion. The position and orientation section allows users to manually specify the destination point to move to. However, unless the motion is simple (i.e. pull back X meters, push by Y meters) it is easier to pose the robot in RViz and use the Current Pose button to copy in the information.

B.2.14.1 Mini-Tutorial

In this tutorial we will make a motion that moves the gripper forward by 15 cm in a straight line. First, position the gripper that you want to move in RViz such that it is in front of the robot's chest, as close as possible, with the tip pointed forward. Make sure that the gripper has room to move forward by at least 15 cm. Now select torso_lift_link from the Frame field, then the arm that the gripper is on in the Arm field, and finally click on Current Pose. This will populate the Position and Orientation fields with the gripper's current pose described in the torso_lift_link frame (located in between the two arms of the robot). Next, add .15 to the X field (as all distance displays are in meters) then add this action to the current behavior with the Add button. Then, clicking Run should move the robot's gripper by 15 cm to that new position.

To execute this action again, manually pull the gripper back using RViz then execute the action.



head_pan_joint -8.06
head_tilt_joint -22.45
Time 1.00
Live Update
Sequence:
point0
point1

Figure 59: Move head block.

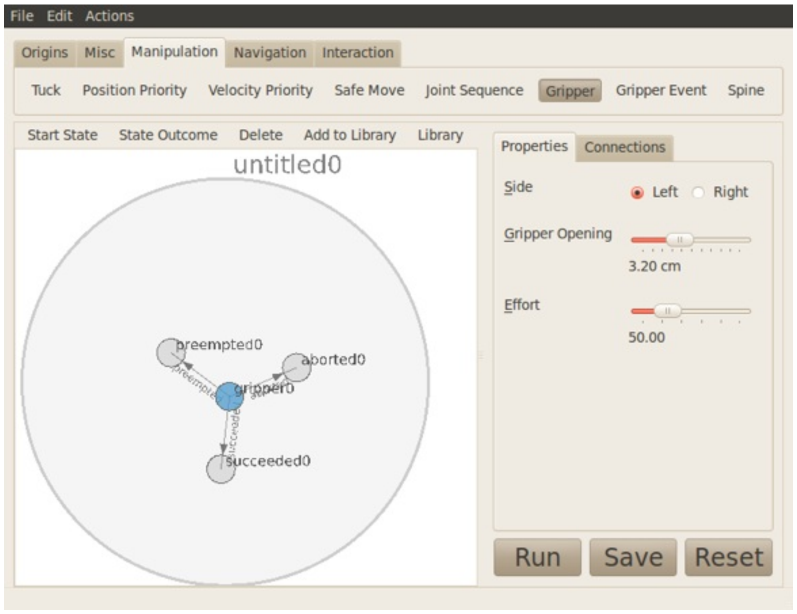
An alternative way to accomplish this is by following the above instructions but instead use either the l_gripper_tool_frame or r_gripper_tool_frame. When you click Current Position all the fields will read 0 (this is normal as the position of the gripper is always at the center of its own frame). Then enter .15 into the X field. Clicking run on this action will now move the gripper forward by 15 cm. However, if you click run a second time the gripper will move forward again by 15 cm from the last position (effectively 30cm!). This is as the command, when expressed in the gripper's reference frame, is saying "wherever the gripper is, move it with respect to this current position in the positive X direction by 15 cm" and in the first case we are saying "wherever the gripper is, move it to this position from the point of view of the torso".

B.2.15 Speak

The speak tool (Figure 67) allows the robot to say out loud whatever is in the text box Say using the sound font specified in Voice.

B.2.16 Detect Face

This tool can be used to detect the location and presence of faces in ROSCo (Figure 68). After detecting a face, it will create a frame with the same name as the node's using the orientation given in Orientation Frame. Here, the timeout field sets how long the action will wait for a face. If this


File Edit Actions
Origins Misc Manipulation Navigation Interaction
Tuck Position Priority Velocity Priority Safe Move Joint Sequence Gripper Gripper Event Spine
Start State State Outcome Delete Add to Library Library
untitled0
preempted0
aborted0
gripper0
succeeded0
Properties Connections
Run Save Reset

Side: [✓ Left] Right
Gripper Opening: 3.20 cm
Effort: 50.00

Figure 60: Gripper block.

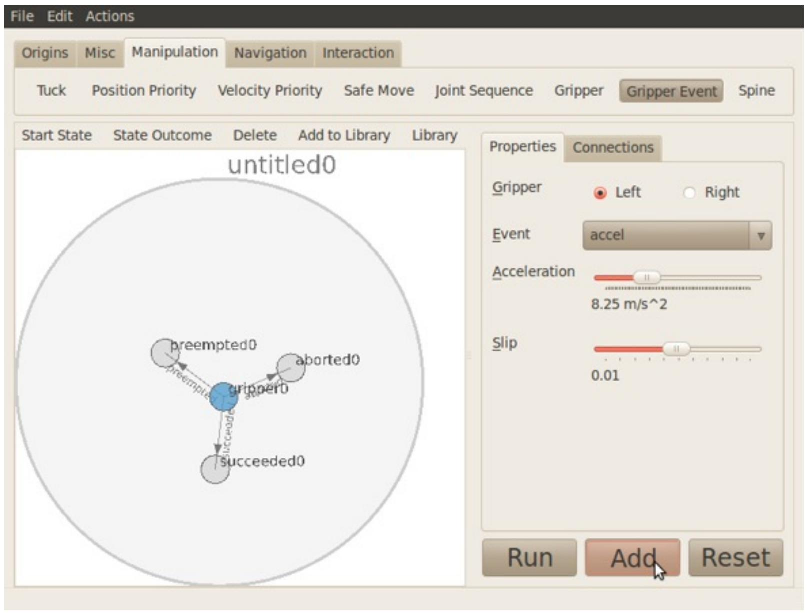
action does not detect a face in the duration given, it will transition to the outcome timed_out.

B.2.17 Base Movement

Base movement can be accomplished through either the navigate or precise navigate blocks.

Navigate (planner): Coarsely moves the robot using its base to another location while avoiding obstacles. To use this command, the arms must first be in a tucked state. Most common failures occur when the tilting laser range finder (on the robot's chest) hallucinate obstacles. To fix this, move the robot around a little bit with the base navigation arrows. The command can be canceled by clicking on the big red sphere that appears around the robot.

Navigate (precise): Moves the robot to another location but does not avoid obstacles. However, it is much more accurate at positioning the robot. As this tool completely ignores all obstacles, it's a good idea to only move the robot a few centimeters at first. NOTE: THIS TOOL CAN BE EXTREMELY DANGEROUS.


File Edit Actions
Origins Misc Manipulation Navigation Interaction
Tuck Position Priority Velocity Priority Safe Move Joint Sequence Gripper Gripper Event Spine
Start State State Outcome Delete Add to Library Library
untitled0
preempted0
aborted0
gripper0
succeeded0
Properties Connections
Run Add Reset

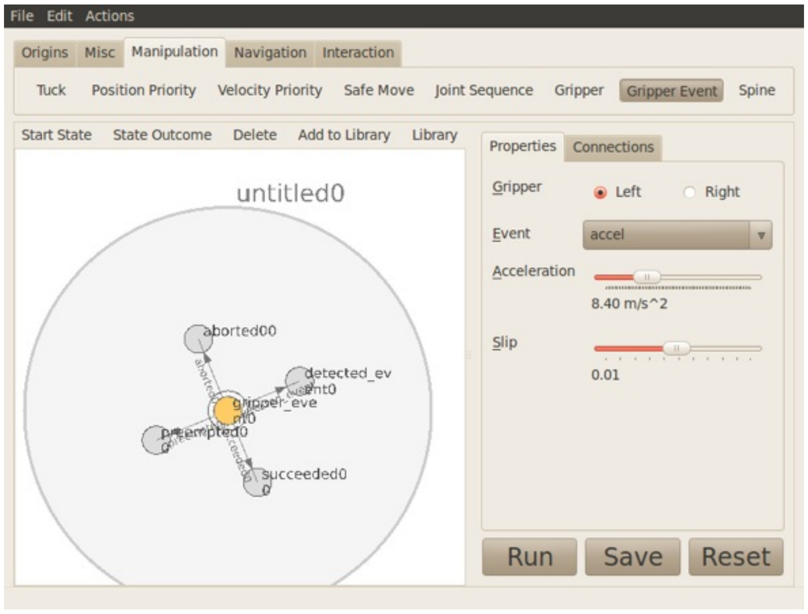
Gripper [✓ Left] Right

Event accel

Acceleration 8.25 m/s^2

Slip 0.01

Figure 61: Gripper event block with a state selected for wrapping.


File Edit Actions
Origins Misc Manipulation Navigation Interaction
Tuck Position Priority Velocity Priority Safe Move Joint Sequence Gripper Gripper Event Spine
Start State State Outcome Delete Add to Library Library
untitled0
aborted00
detected_event0
gripper_event0
preempted00
succeeded00
Properties Connections
Run Save Reset

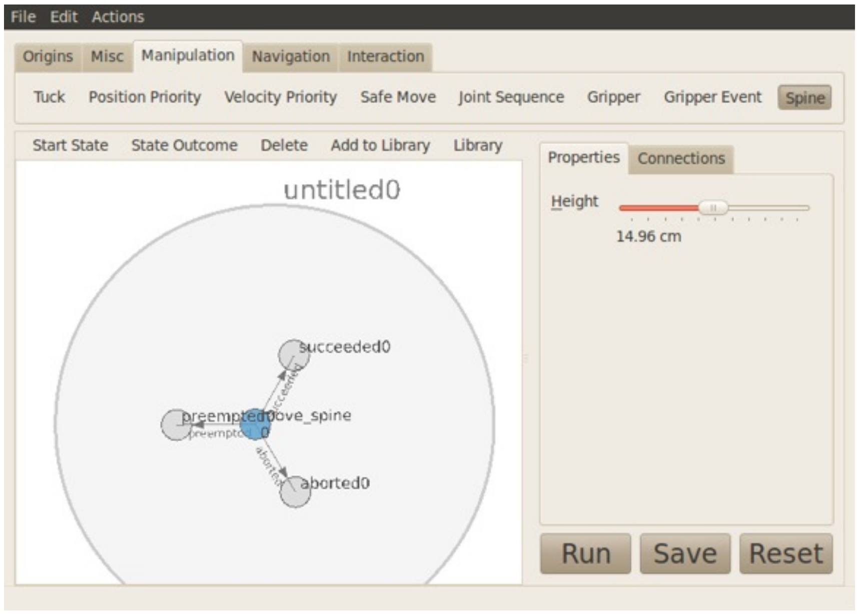
Gripper [✓ Left] Right

Event accel

Acceleration 8.40 m/s^2

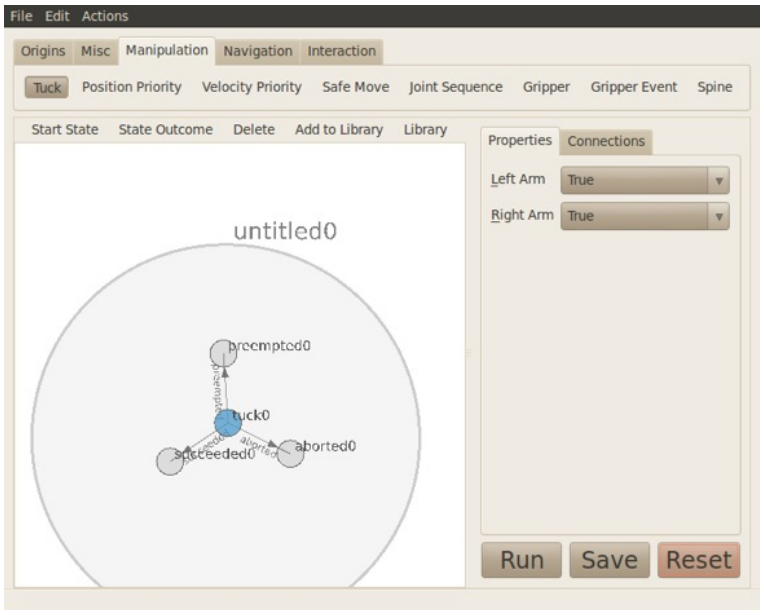
Slip 0.01

Figure 62: Gripper event block with selected state wrapped.


File Edit Actions
Origins Misc Manipulation Navigation Interaction
Tuck Position Priority Velocity Priority Safe Move Joint Sequence Gripper Gripper Event Spine
Start State State Outcome Delete Add to Library Library
untitled0
succeeded0
preempted0
move_spine0
aborted0
Properties Connections
Run Save Reset

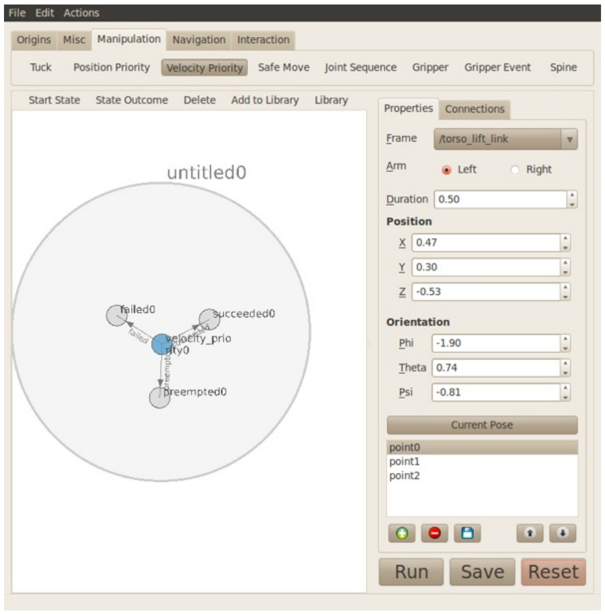
Height 14.96 cm

Figure 63: Spine block.


File Edit Actions
Origins Misc Manipulation Navigation Interaction
Tuck Position Priority Velocity Priority Safe Move Joint Sequence Gripper Gripper Event Spine
Start State State Outcome Delete Add to Library Library
untitled0
preempted0
tuck0
succeeded0
aborted0
Properties Connections
Run Save Reset

Left Arm True
Right Arm True

Figure 64: Tuck block.


File Edit Actions
Origins Misc Manipulation Navigation Interaction
Tuck Position Priority Velocity Priority Safe Move Joint Sequence Gripper Gripper Event Spine
Start State State Outcome Delete Add to Library Library
Properties Connections
untitled0
failed0
succeeded0
velocity_priority0
preempted0
Current Pose
Run Save Reset

Frame: /torso_lift_link
Arm: [✓ Left] Right
Duration: 0.50

Position

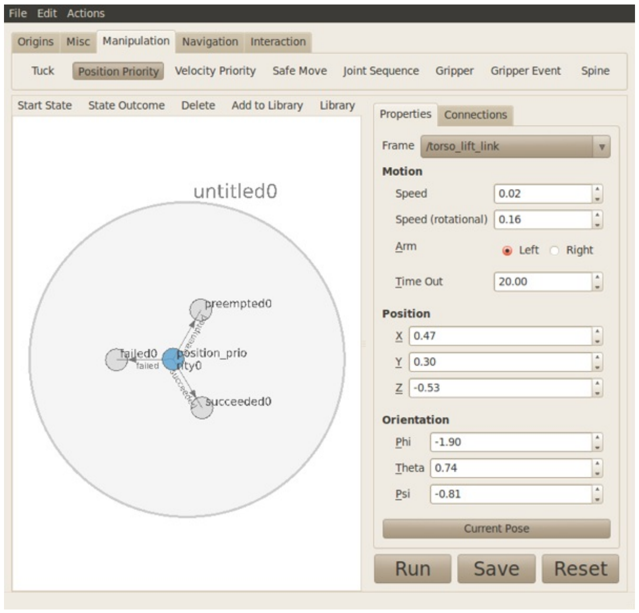
X: 0.47
Y: 0.30
Z: -0.53

Orientation

Phi: -1.90
Theta: 0.74
Psi: -0.81

point0
point1
point2

Figure 65: Velocity priority block.


File Edit Actions
Origins Misc Manipulation Navigation Interaction
Tuck Position Priority Velocity Priority Safe Move Joint Sequence Gripper Gripper Event Spine
Start State State Outcome Delete Add to Library Library
Properties Connections
untitled0
preempted0
failed0
position_priority0
succeeded0
Current Pose
Run Save Reset

Frame: /torso_lift_link

Motion

Speed: 0.02
Speed (rotational): 0.16
Arm: [✓ Left] Right
Time Out: 20.00

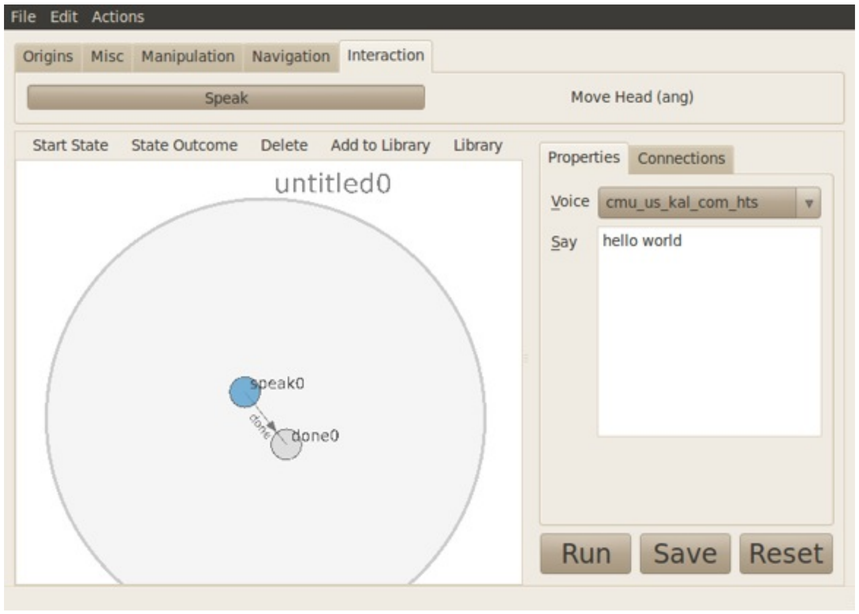
Position

X: 0.47
Y: 0.30
Z: -0.53

Orientation

Phi: -1.90
Theta: 0.74
Psi: -0.81

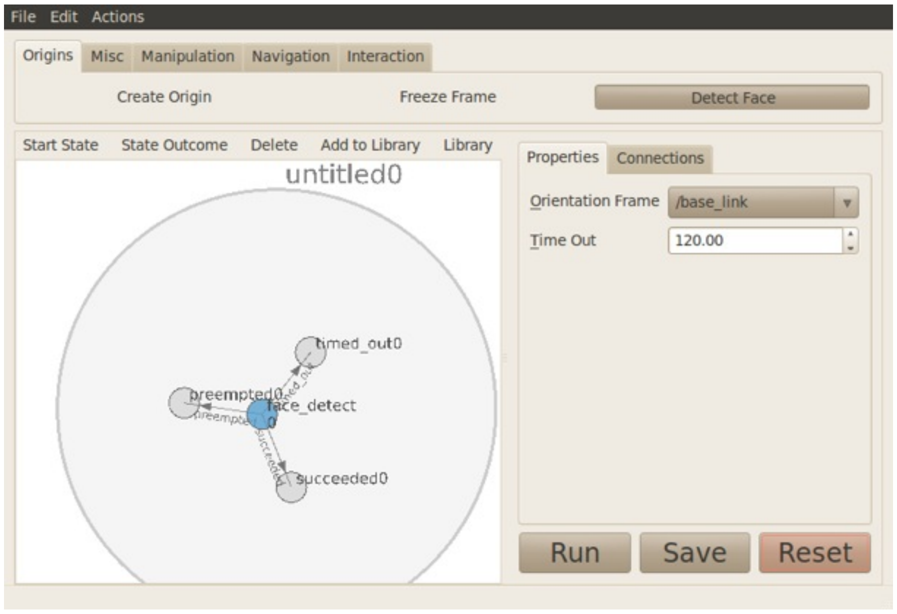
Figure 66: Position Priority block.


File Edit Actions
Origins Misc Manipulation Navigation Interaction
Speak
Move Head (ang)
Start State State Outcome Delete Add to Library Library
untitled0
speak0
done
done0
Properties Connections
Run Save Reset

Voice cmu_us_kal_com_hts

Say hello world

Figure 67: Speak block.



Orientation Frame /base_link

Time Out 120.00

Figure 68: Detect Face block.

REFERENCES

[1] "Ecto." http://plasmodic.org/. 15

[2] "Gostai Studio." http://www.gostai.com/products/jazz/gostai_studio/. 6, 15

[3] "Havok behavior." http://www.havok.com/products/behavior. 6, 13, 15

[4] "Kismet." http://www.unrealengine.com/features/kismet/. 15

[5] "Massive." http://massivesoftware.com. 15

[6] "Robotflow." http://robotflow.sourceforge.net/. 15

[7] "Ros commander for the pr2." http://ros.org/wiki/rcommander_pr2. 11, 26

[8] "Scipy gaussian kde function." http://www.scipy.org/doc/api_docs/SciPy.stats.kde.gaussian_kde.html. 93

[9] "Unity character animation." http://video.unity3d.com/video/4655480/unity-character-animation-gdc. 13, 15

[10] "Visual language research bibliography." http://web.engr.oregonstate.edu/~burnett/vpl.html. 14

[11] "Vpl introduction." http://msdn.microsoft.com/en-us/library/bb483088.aspx. 14

[12] "Willow Garage." http://www.willowgarage.com. 81, 96

[13] "xaitcontrol." http://www.xaitment.com/english/products/xaitcontrol-for-unity.html. 6, 15

[14] "Semantic robot vision challenge." http://www.semantic-robot-vision-challenge.org/, 2007. 83

[15] "Global health and ageing," in *WHO; US National Institute of Aging*, 2011. 3

[16] ABBEEL, P., COATES, A., QUIGLEY, M., and NG, A. Y., "An application of reinforcement learning to aerobatic helicopter flight," in *NIPS*, 2006. 56, 128

[17] ABBEEL, P., QUIGLEY, M., and NG, A. Y., "Using inaccurate models in reinforcement learning," in *International Conference on Machine Learning*, 2006. 83

[18] AKGUN, B., CAKMAK, M., JIANG, K., and THOMAZ, A. L., "Keyframe-based learning from demonstration," in *International Journal of Social Robotics*, 2012. 56

[19] ALAMI, R., LIE CLODIC, A., MONTREUIL, V., SISBOT, E. A., and CHATILA, R., "Toward human-aware robot task planning," in *AAAI*, 2006. 14

[20] ARGALL, B. D., CHERNOVA, S., VELOSO, M., and BROWNING, B., “A survey of robot learning from demonstration,” in *Robotics and Autonomous Systems*, 2008. 82

[21] ARGALL, B. D., CHERNOVA, S., VELOSO, M., and BROWNING, B., “A survey of robot learning from demonstration,” in *Robotics and Autonomous Systems*, 2009. 5, 56

[22] ARKIN, R., “Missionlab v7.0.” `http://www.cc.gatech.edu/ai/robot-lab/research/MissionLab/`, July 2006. 6, 15

[23] BALLARD, D., “Task frames in robot manipulation,” in *AAAI*, 1984. 19, 109

[24] BARRIUSO, A. and TORRALBA, A., “Notes on image annotation,” in *MIT Tech Note*, 2012. 83

[25] BECKER, J., BERSCH, C., PANGERCIC, D., PITZER, B., RUHR, T., SANKARAN, B., J. STURM, C. S., BEETZ, M., and BURGARD, W., “Mobile manipulation of kitchen containers,” in *IROS The PR2 Workshop*, 2011. 4

[26] BEETZ, M., KLANK, U., KRESSE, I., MALDONADO, A., MÖSENLECHNER, L., PANGERCIC, D., RÜHR, T., and TENORTH, M., “Robotic roommates making pancakes,” in *International Conference on Humanoid Robots*, 2011. 4, 5

[27] BEETZ, M., MOSENLECHNER, L., and TENORTH, M., “Cram a cognitive robot abstract machine for everyday manipulation in human environments,” in *IROS*, 2010. 5, 14

[28] BENJAMIN, D., LYONS, D., and LONSDALE, D., “Adapt: A cognitive architecture for robots,” in *Conference on Cognitive Modelling*, 2004. 14

[29] BERGER, A., PIETRA, V. D., , and PIETRA, S. D., “A maximum entropy approach to natural language processing,” in *Computational Linguistics*, 1996. 85

[30] BERTHOUZE, L., BAKKER, P., and KUNIYOSHI, Y., “Learning of oculo-motor control: a prelude to robotic imitation,” in *International Conference on Robotics and Intelligent Systems*, 1997. 82

[31] BERTHOUZE, L. and KUNIYOSHI, Y., “Emergence and categorization of coordinated visual behavior through embodied interaction,” in *Machine Learning*, 1998. 82

[32] BOHREN, J. and RUSU, R., “Towards autonomous robotic butlers: Lessons learned with the pr2,” in *ICRA*, 2011. 5, 11, 17, 55, 56

[33] BOLLINI, M., BARRY, J., and RUS, D., “Bakebot: Baking cookies with the pr2,” in *IROS The PR2 Workshop*, 2011. 4

[34] BOSHERNITSAN, M. and DOWNES, M., “Visual programming languages: A survey,” tech. rep., University of California Berkeley, 2004. 14

[35] BUTKO, N. J. and MOVELLAN, J. R., “Developmental robotics architecture for active vision and reaching,” in *International Conference on Development and Learning*, 2010. 82

[36] BUTKO, N. J. and MOVELLAN, J. R., “Learning to look,” in *International Conference on Development and Learning*, 2010. 82

[37] CAKMAK, M. and THOMAZ, A., "Social machine learning , "designing robot learners that ask good questions.," in *HRI*, 2012. 14

[38] CAMBON, S., GRAVOT, F., and ALAMI, R., "A robot task planner that merges symbolic and geometric reasoning," in *European Conference on Artificial Intelligence*, 2004. 14

[39] CHANG, C.-C. and LIN, C.-J., "Libsvm : a library for support vector machines." http://www.semantic-robot-vision-challenge.org/, 2001. 94, 97

[40] CHATZILARI, E., NIKOLOPOULOS, S., PAPADOPOULOS, S., and KOMPATSIARIS, C. Z. Y., "Semi-supervised object recognition using flickr images," in *International Workshop on Content-Based Multimedia Indexing*, 2011. 83

[41] CHEN, T. L., CIOCARLIE, M., COUSINS, S., GRICE, P., HAWKINS, K., HSIAO, K., KEMP, C. C., KING, C.-H., LAZEWATSKY, D. A., LEEPER, A., NGUYEN, H., PAEPCKE, A., PANTOFARU, C., SMART, W. D., and TAKAYAMA, L., "Robots for humanity: Using assistive robotics to empower people with disabilities," in *IEEE Robotics and Automation Magazine*, 2013. 59, 72

[42] CHITTA, S., JONES, E. G., CIOCARLIE, M., and HSIAO, K., "Perception, planning, and execution for mobile manipulation in unstructured environments," in *IEEE Robotics and Automation Magazine*, 2011. 5

[43] CHRISTIAN SCHEIER, D. L., "Categorization in a real-world agent using haptic exploration and active perception," in *International Conference on Simulation of Adaptive Behavior*, 1996. 83

[44] CIOCARLIE, M., HSIAO, K., LEEPER, A., and GOSSOW, D., "Mobile manipulation through an assistive home robot," in *IROS*, In Press. 13, 17, 62, 66, 67

[45] COELHO, J., PIATER, J., and GRUPEN, R., "Developing haptic and visual perceptual categories for reaching and grasping with a humanoid robot," in *Robotics and Autonomous Systems*, 2001. 83

[46] COHEN, B., BENAMY, D., COWLEY, A., MCMAHAN, W., and ROMANO, J., "Poop scoop: Perception of offensive products and sensorized control of object pickup," in *IROS The PR2 Workshop*, 2011. 4

[47] COHN, D., ATLAS, L., and LADNER, R., "Improving generalization with active learning," in *Machine Learning*, 1994. 85

[48] CONWAY, L., VOLZ, R. A., and WALKER, M. W., "Teleautonomous systems: Projecting and coordinating intelligent action at a distance," in *IEEE Transactions on Robotics and Autonomation*, 1990. 65

[49] CULOTTA, A. and MCCALLUM, A., "Reducing labeling effort for stuctured prediction tasks," in *Conference on Artificial Intelligence (AAAI)*, 2005. 85

[50] CUSUMANO-TOWNER, M., SINGH, A., MILLER, S., O'BRIEN, J., and ABBEEL, P., "Bringing clothing into desired configurations with limited perception bringing clothing into desired configurations with limited perception bringing clothing into desired configurations with limited perception," in *ICRA*, 2011. 4, 32

[51] DAI, W., YANG, Q., XUE, G., and YU, Y., "Boosting for transfer learning," in *Proc. 24th Intl Conf. Machine Learning*, 2007. 129

[52] DI MARCO, D., TENORTH, M., HÄUSSERMANN, K., ZWEIGLE, O., and LEVI, P., "Roboearth action recipe execution," in *Frontiers of Intelligent Autonomous Systems*, pp. 117–126, Springer, 2013. 5

[53] DOGAR, M., KOVAL, M., TALLAVAJHULA, A., and SRINIVASA, S., "Object search by manipulation," in *IEEE International Conference on Robotics and Automation*, May 2013. 56

[54] DOGAR, M. and SRINIVASA, S., "A framework for push-grasping in clutter," in *Robotics: Science and Systems VII* (HUGH DURRANT-WHYTE, N. R. and ABBEEL, P., eds.), MIT Press, July 2011. 13

[55] DROMEY, R. G., HE, J., and LIU, Z., "Formalizing the Transition from Requirements to Design," in *Mathematical Frameworks for Component Software Models for Analysis and Synthesis*, pp. 156–187, 2006. 15

[56] DUNN, G. and SEGEN, J., "Automatic discovery of robotic grasp configuration," in *International Conference on Robotics and Automation*, 1988. 82

[57] ERKAN, A., KROEMER, O., DETRY, R., ALTUN, Y., PIATER, J., and PETERS, J., "Learning probabilistic discriminative models of grasp affordances under limited supervision," in *IROS*, pp. 1586–1591, IEEE, 2010. 83

[58] EVGENIOU, T. and PONTIL, M., "Regularized multi-task learning," in *ACM SIGKDD Intl Conf. Knowledge Discovery and Data Mining*, 2004. 129

[59] FIKES, R. and N.J NILSSON, N., "Strips: A new approach to the application of theorem proving to problem solving," in *Technical Note 43r. AI Center, SRI*, 1971. 14

[60] GUPTA, M. and SUKHATME, G., "Duplo bricks sorting by pr2," in *IROS The PR2 Workshop*, 2011. 4

[61] HART, S. and STAVELAND, L., "Development of nasa-tlx (task load index): Results of empirical and theoretical research," in *Human mental workload*, 1988. 38, 43

[62] HART, S., YAMOKOSKI, J., and DIFTLER, M., "Robonaut 2: A new platform for human-centered robot learning," in *RSS Workshop on Mobile Manipulation*, 2011. 6, 15

[63] HSIAO, K., CHITTA, S., CIOCARLIE, M., and JONES, G., "Contact-reactive grasping of objects with partial shape information," in *IROS*, 2010. 99, 124

[64] HSIAO, K., CIOCARLIE, M., and BROOK, P., "Bayesian grasp planning," in *ICRA Workshop on Mobile Manipulation*, 2011. 4

[65] IJSPEERT, A. J., NAKANISHI, J., and SCHAAL, S., "Learning attractor landscapes for learning motor primitives," in *Advances in Neural Information Processing Systems (NIPS)*, 2003. 14, 82

[66] JAIN, A. and KEMP, C. C., "Behavior-Based Door Opening with Equilibrium Point Control," in *RSS Workshop: Mobile Manipulation in Human Environments*, 2009. 66

[67] JAIN, A. and KEMP, C. C., "El-e: An assistive mobile manipulator that autonomously fetches objects from flat surfaces," in *Autonomous Robots*, 2010. 20, 84

[68] JAIN, A. and KEMP, C. C., "Pulling open doors and drawers: Coordinating an omni-directional base and a compliant arm with equilibrium point control," in *ICRA*, 2010. 4

[69] JAIN, A., KILLPACK, M. D., EDSINGER, A., , and KEMP, C. C., "Reaching in clutter with whole-arm tactile sensing," in *The International Journal of Robotics Research*, 2013. 56

[70] JAIN, A., NGUYEN, H., RATH, M., OKERMAN, J., and KEMP, C. C., "The complex structure of simple devices: A survey of trajectories and forces that open doors and drawers," in *IEEE RAS/EMBS International Conference on Biomedical Robotics and Biomechatronics (BIOROB)*, 2010. 70

[71] JAIN, P., VIJAYANARASIMHAN, S., and GRAUMAN, K., "Hashing hyperplane queries to near points with applications to large-scale active learning," in *Advances in Neural Information Processing Systems (NIPS)*, 2010. 85

[72] JIANG, Y., ZHENG, C., LIM, M., and SAXENA, A., "Learning to place new objects," in *RSS Workshop on Mobile Manipulation*, 2011. 4

[73] KAELBLING, L. P. and LOZANO-PEREZ, T., "Integrated task and motion planning in belief space," in *International Journal of Robotics Research*, 2013. 5

[74] KATO, H. and BILLINGHURST, M., "http://www.hitl.washington.edu/artoolkit." 19

[75] KATZ, D. and BROCK, O., "Manipulating Articulated Objects With Interactive Perception," in *ICRA*, 2008. 84

[76] KATZ, D., KENNEY, J., and BROCK, O., "How Can Robots Succeed in Unstructured Environments?," in *RSS, Robot Manipulation Workshop*, 2008. 84

[77] KEMP, C. C. and EDSINGER, A., "Robot Manipulation of Human Tools : Autonomous Detection and Control of Task Relevant Features," in *ICDL*, 2006. 84

[78] KLINGBEIL, E., SAXENA, A., and NG, A. Y., "Learning to Open New Doors," in *RSS Workshop on Robot Manipulation*, 2008. 4, 83, 84

[79] KOBER, J. and PETERS, J., "Imitation and reinforcement learning - practical algorithms for motor primitive learning," in *IEEE RAM*, 2010. 8, 14, 125

[80] KOBER, J., OZTOP, E., and PETERS, J., "Reinforcement Learning to adjust Robot Movements to New Situations," in *RSS*, 2010. 5, 82, 83, 128

[81] KORNER, C. and WROBEL., S., "Multi-class ensemble-based active learning," in *European Conference on Machine Learning (ECML)*, 2006. 85

[82] KRAFT, D., DETRY, R., PUGEAULT, N., BAESKI, E., GUERIN, F., PIATER, J., and KRGER, N., "Development of object and grasping knowledge by robot exploration," in *IEEE Transactions on Autonomous Mental Development*, 2010. 83

[83] KRICHMAR, J. L. and EDELMAN, G. M., "Machine psychology: autonomous behavior, perceptual categorization and conditioning in a brain-based device," in *Cerebral Cortex*, 2002. 83

[84] KUNZE, L., HAIDU, A., and BEETZ, M., "Making virtual pancakes – acquiring and analyzing data of everyday manipulation tasks through interactive physics-based simulation," in *German Conference on Artificial Intelligence*, 2012. 13

[85] LAFFERTY, J., MCCALLUM, A., , and PEREIRA, F., "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *International Conference on Machine Learning (ICML)*, 2001. 85

[86] LEEPER, A., CHAN, S., HSIAO, K., CIOCARLIE, M., and SALISBURY, K., "Constraint-based haptic rendering of point data for teleoperated robotic grasping," in *Haptics Symposium*, 2012. 60

[87] LEWIS, D. and CATLETT, J., "Heterogeneous uncertainty sampling for supervised learning," in *International Conference on Machine Learning (ICML)*, 1994. 85

[88] LIU, K., SAKAMOTO, D., INAMI, M., and IGARASHI, T., "Roboshop: multi-layered sketching interface for robot housework assignment and management," in *CHI*, 2011. 13

[89] LOZANO-PEREZ, T., JONES, J., MAZER, E., and O'DONNELL, P., *HANDEY: A Robot Task Planner*. Cambridge, MA: M.I.T. Press, 1992. 14

[90] LUNGARELLA, M. and METTA, G., "Beyond gazing, pointing, and reaching: A survey of developmental robotics," in *EPIROB*, 2003. 82

[91] LUNGARELLA, M., METTAY, G., PFEIFERZ, R., and SANDINI, G., "Developmental robotics: a survey," in *Connection Science*, 2003. 82

[92] MAITIN-SHEPARD, J., CUSUMANO-TOWNER, M., LEI, J., and ABBEEL, P., "Cloth Grasp Point Detection based on Multiple-View Geometric Cues with Application to Robotic Towel Folding," in *ICRA*, 2010. 7, 13, 84

[93] MARDER-EPPSTEIN, E., BERGER, E., FOOTE, T., GERKEY, B. P., and KONOLIGE, K., "The office marathon: Robust navigation in an indoor office environment," in *International Conference on Robotics and Automation*, 2010. 19, 40, 67, 101

[94] MARJANOVIC, M., SCASSELLATI, B., and WILLIAMSON, M., "Self-taught visually-guided pointing for a humanoid robot," in *International Conference on Simulation of Adaptive Behavior*, 1996. 82

[95] MASON, M., "Compliance and force control for computer controlled manipulators," in *IEEE Transactions on Systems, Man, and Cybernetics*, 1981. 109

[96] MCCALLUM, A. and NIGAM, K., "Employing em in pool-based active learning for text classification," in *International Conference on Machine Learning (ICML)*, 1998. 85

[97] MCGANN, C., PY, F., RAJAN, K., THOMAS, H., HENTHORN, R., and MCEWEN, R., "T-rex: A model-based architecture for auv control," in *ICAPS*, 2007. 14

[98] MEEUSSEN, W., WISE, M., GLASER, S., CHITTA, S., MCGANN, C., MIHELICH, P., MARDER-EPPSTEIN, E., MUJA, M., ERUHIMOV, V., FOOTE, T., HSU, J., RUSU, R. B., MARTHI, B., BRADSKI, G., KONOLIGE, K., GERKEY, B. P., and BERGER, E., "Autonomous door opening and plugging in with a personal robot," in *ICRA*, 2010. 4, 5, 7, 13, 124

[99] METTA, G. and FITZPATRICK, P., "Early integration of vision and manipulation," in *Adaptive Behavior*, 2003. 82

[100] METTA, G., SANDINI, G., and KONCZAK, J., "A developmental approach to visually-guided reaching in articial systems," in *Neural Networks*, 1999. 82

[101] METTA, G., SANDINI, G., VERNON, D., NATALE, L., and NORI, F., "The icub humanoid robot: an open platform for research in embodied cognition," in *PerMIS: Performance Metrics for Intelligent Systems Workshop*, 2008. 14

[102] MICHAEL A. GOODRICH, DAN R. OLSEN JR., J. W. C. and PALMER, T. J., "Experiments in adjustable autonomy," in *Conference on Artificial Intelligence (AAAI)*, 2001. 65

[103] MONTESANO, L. and LOPES, M., "Learning grasping affordances from local visual descriptors," *ICDL*, pp. 1–6, 2009. 82

[104] MUELLING, K., KOBER, J., KROEMER, O., and PETERS, J., "Learning to select and generalize striking movements in robot table tennis," *International Journal of Robotics Research*, no. 3, pp. 263–279, 2013. 8, 125

[105] MUSLEA, I., MINTON, S., and KNOBLOCK, C., "Active + semi-supervised learning = robust multi-view learning," in *International Conference on Machine Learning (ICML)*, 2002. 85

[106] N, M., JENKINS, O., KALLMANN, M., and M.J. MATARIC, M., "Motion capture from inertial sensing for untethered humanoid teleoperation," in *IEEE International Conference on Humanoid Robots*, 2004. 60

[107] NAKANISHI, J., MISTRY, M., and SCHAAL, S., "Comparative experiments on task space control with redundancy resolution," in *IEEE International Conference on Intelligent Robots*, 2005. 19, 40, 53, 99

[108] NGUYEN, H., DEYLE, T., REYNOLDS, M., and KEMP, C. C., "Pps-tags: Physical perceptual and semantic tags for autonomous mobile manipulation," in *IROS 2009 workshop: Semantic Perception for Mobile Manipulation*, 2009. 20

[109] NGUYEN, H., JAIN, A., ANDERSON, C., and KEMP, C. C., "A clickable world: Behavior selection through pointing and context for mobile manipulation," in *IROS*, 2008. 87, 90

[110] NGUYEN, H. and KEMP, C. C., "Autonomous active learning of task-relevant features for mobile manipulation," in *RSS 2011 Workshop on Mobile Manipulation: Learning to Manipulate*, 2011. 82

[111] NILSSON, N. J., "Shakey the robot," tech. rep., SRI International, April 1984. 5

[112] OKADA, K., KOJIMA, M., SAGAWA, Y., ICHINO, T., SATO, K., and INABA, M., "Vision based behavior verication system of humanoid robot for daily environment tasks," in *Humanoid Robots*, 2006. 84

[113] PAN, S. J. and YANG, Q., "A survey on transfer learning," in *IEEE Transactions on Knowledge and Data Engineering*, 2010. 129

[114] PAOLINI, R., RODRIGUEZ, A., SRINIVASA, S. S., and MASON, M. T., "A data-driven statistical framework for post-grasp manipulation," in *International Symposium on Experimental Robotics*, 2012. 84

[115] PASTOR, P., KALAKRISHNAN, M., CHITTA, S., THEODOROU, E., and SCHAAL, S., "Skill learning and task outcome prediction for manipulation," in *International Conference on Robotics and Automation*, 2011. 5, 8, 14, 60, 82, 83, 125, 127, 128

[116] PETERS, J., KOBER, J., MUELLING, K., NGUYEN-TUONG, D., and KROEMER, O., "Towards motor skill learning for robotics," in *International Symposium on Robotics Research (ISRR)*, 2009. 128

[117] PETERS, J. and SCHAAL, S., "Reinforcement learning of motor skills with policy gradients," in *Neural Networks*, 2008. 14

[118] PFEIFER, R. and SCHEIER, C., "Sensory-motor coordination: the metaphor and beyond," in *Robotics and Autonomous Systems*, 1997. 82

[119] PONCE, J., BERG, T., EVERINGHAM, M., FORSYTH, D., HEBERT, M., LAZEBNIK, S., MARSZALEK, M., SCHMID, C., RUSSELL, B., TORRALBA, A., WILLIAMS, C., ZHANG, J., and ZISSERMAN, A., "Dataset issues in object recognition," in *Toward Category-level Object Recognition*, 2006. 83

[120] PORTSMORE, M., "Robolab: Intuitive robotic programming software to support life long learning," in *APPLE Learning Technology*, 1999. 14

[121] PRATKANIS, A., LEEPER, A. E., , and SALISBURY, K., "Replacing the ofce intern: An autonomous coffee run with a mobile manipulator," in *International Conference on Robotics and Automation*, 2013. 55

[122] PRATS, M., *Robot Physical Interaction through the combination of Vision, Tactile and Force Feedback.* PhD thesis, Jaume-I University, 2009. 4

[123] PULLIN, G. and NEWELL, A., "Focussing on extra-ordinary users," in *International Conference on Universal Access in Human-Computer Interaction*, pp. 253–262, 2007. 60

[124] ROY, N. and MCCALLUM, A., "Toward optimal active learning through sampling estimation of error reduction," in *International Conference on Machine Learning (ICML)*, 2001. 85

[125] SALGANICOFF, M., UNGAR, L. H., and BAJCSY, R., "Active learning for vision-based robot grasping," *Machine Learning*, 1996. 83

[126] SANKARAN, B., PITZER, B., and OSENTOSKI, S., "Failure recovery with shared autonomy," in *IEEE International Conference on Intelligent Robots and Systems*, 2012. 78

[127] SAXENA, A., DRIEMEYER, J., and NG, A. Y., "Robotic Grasping of Novel Objects using Vision," *IJRR*, 2008. 82, 83

[128] SCHEIN, A. and UNGAR, L., "Active learning for logistic regression: An evaluation," in *Machine Learning*, 2007. 85

[129] SCHOHN, G. and COHN, D., "Less is More : Active Learning with Support Vector Machines," in *ICML*, 2000. 85, 86, 91, 95

[130] SETTLES, B. and CRAVEN, M., "An analysis of active learning strategies for sequence labeling tasks," in *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2008. 85

[131] SETTLES, B., *Active Learning*. Morgan & Claypool Publishers, 2012. 84

[132] STEIN, M. R., *Behavior-Based Control for Time-Delayed Teleoperation*. PhD thesis, University of Pennsylvania, 1994. 65, 66

[133] STOBER, J. and KUIPERS, R. M. B., "Learning geometry from sensorimotor experience," in *International Conference on Development and Learning*, 2011. 83

[134] STORY, M. F., "Maximizing usability: The principles of universal design," vol. 10, 1998. 60

[135] STOYTCHEV, A., "Baby gym for robots: A new platform for testing developmental learning algorithms," in *National Conference on Artificial Intelligence (AAAI)*, 2011. 8, 125

[136] SUKHOY, V. and STOYTCHEV, A., "Learning to Detect the Functional Components of Doorbell Buttons Using Active Exploration and Multimodal Correlation," in *IEEE International Conference on Humanoid Robots*, 2010. 8, 84, 125

[137] TENORTH, M., KLANK, U., PANGERCIC, D., , and BEETZ, M., "Web-enabled robots robots that use the web as an information resource," in *IEEE Robotics & Automation Magazine*, 2011. 14

[138] THRUN, S., MONTEMERLO, M., DAHLKAMP, H., STAVENS, D., ARON, A., DIEBEL, J., FONG, P., GALE, J., HALPENNY, M., HOFFMANN, G., LAU, K., OAKLEY, C., PALATUCCI, M., PRATT, V., , STROHBAND, P. S. S., DUPONT, C., JENDROSSEK, L.-E., KOELEN, C., MARKEY, C., RUMMEL, C., VAN NIEKERK, J., JENSEN, E., , BRADSKI, P. A. G., DAVIES, B., ETTINGER, S., KAEHLER, A., , and MAHONEY, A. N. P., "Stanley: The robot that won the darpa grand challenge," in *Journal of Field Robotics*, 2006. 5

[139] TONG, S. and KOLLER, D., "Support vector machine active learning with applications to text classification," in *Conference on Machine Learning (ICML)*, 2000. 85

[140] TORRALBA, A. and EFROS, A., "Unbiased look at dataset bias," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011. 83

[141] VAN HOOF, H., KROEMER, O., AMOR, H. B., and PETERS, J., "Maximally informative interaction learning for scene exploration," in *IROS*, 2012. 84

[142] ZHANG, J. and RSSLER, B., "Self-valuing learning and generalization with application in visually guided grasping of complex objects," *Robotics and Autonomous systems*, vol. 47, 2004. 82

[143] ZHU, X., LAFFERTY, J., and GHAHRAMANI, Z., "Combining active learning and semi-supervised learning using gaussian fields and harmonic functions," in *ICML Workshop on the Continuum from Labeled to Unlabeled Data*, 2003. 85