ROS Commander (ROSCo): Behavior Creation for Home Robots

Hai Nguyen[1], Matei Ciocarlie[2], Kaijen Hsiao[2], and Charles C. Kemp[1]

***Abstract*— We introduce ROS Commander (ROSCo), an open source system that enables expert users to construct, share, and deploy robot behaviors for home robots. A user builds a behavior in the form of a Hierarchical Finite State Machine (HFSM) out of generic, parameterized building blocks, with a real robot in the develop and test loop. Once constructed, users save behaviors in an open format for direct use with robots, or for use as parts of new behaviors. When the system is deployed, a user can show the robot where to apply behaviors relative to fiducial markers (AR Tags), which allows the robot to quickly become operational in a new environment. We show evidence that the underlying state machine representation and current building blocks are capable of spanning a variety of desirable behaviors for home robots, such as opening a refrigerator door with two arms, flipping a light switch, unlocking a door, and handing an object to someone. Our experiments show that sensor-driven behaviors constructed with ROSCo can be executed in realistic home environments with success rates between 80% and 100%. We conclude by describing a test in the home of a person with quadriplegia, in which the person was able to automate parts of his home using previously-built behaviors.**

I. Introduction

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 [17]. 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.

[1]H. Nguyen and C. C. Kemp are with the Healthcare Robotics Lab, Robotics and Intelligent Machines Center @ Georgia Tech, haidai at gmail.com, charlie.kemp at bme.gatech.edu

[2]M. Ciocarlie and K. Hsiao are with Willow Garage, matei at willowgarage.com, hsiao at willowgarage.com

Fig. 1. 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.

In this paper, we introduce a system called ROS 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 paper, we use our system to provide evidence for the following claims:

- HFSMs constructed from a relatively compact set of low-level building blocks, and without direct access to code, are 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 2, 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, trajec-

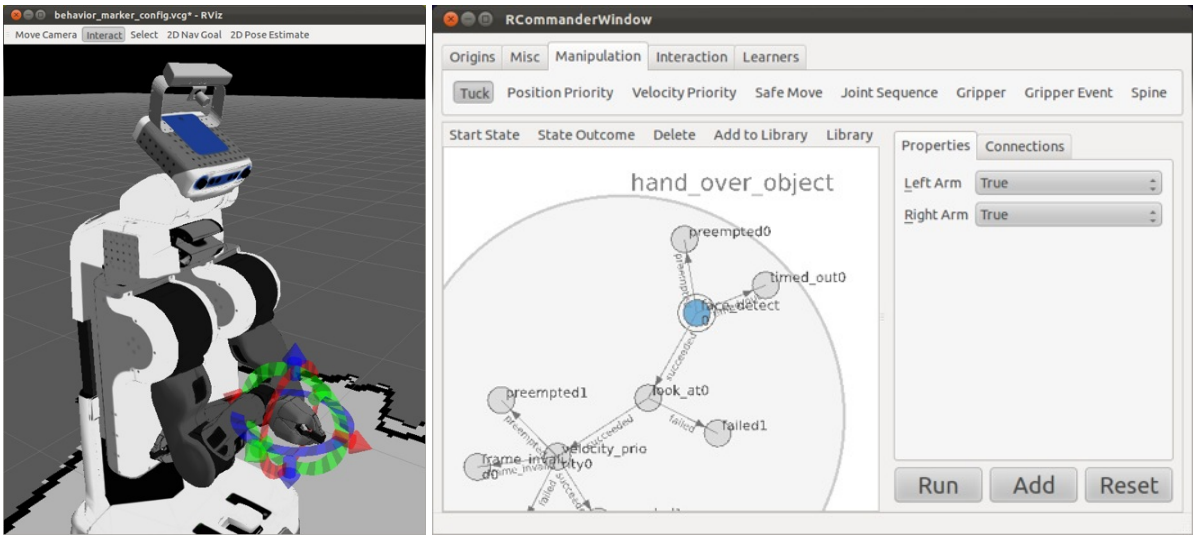

Fig. 2. The HFSM editor that we propose (right side) is paired with ROS's rviz with interactive markers (such as 6-DOF rings-and-arrows controls around grippers) for posing and controlling the PR2 (left side). Combined, we believe that these interfaces will allow quick iterative development of novel parameterized abilities for the PR2.

tory 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 classic AI STRIPS-styled representations, the resulting behaviors of HFSMs can be more predictable from a user's perspective as demonstrated in video game [3], [8] and movie character design–a potentially important property for user interaction. Furthermore, as behaviors generated by users are potentially open-ended, this style of HFSM driven behavior-based robotics does not require users to build explicit models for physical interaction before having robots perform useful actions.

Our experiments show that by constructing HFSMs using states created by generic ROSCo modules, our system can generate a variety of robust autonomous behaviors. We present trials where we execute each of 6 behaviors 10 times, with 80% to 100% success rates. These behaviors include opening a refrigerator with two arms, flipping a light switch, unlocking a door with a push switch, opening a drawer, and handing objects over to a person.

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" [29] 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.

II. 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 [23], [30], [31], [16], [20], [12]. Planning approaches for manipulation go as far back as the Handey system [30] for flexible pick-and-place tasks. More recently, the CRAM toolbox [15] 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 [32], [34], [28], [35], [19], [37]. A number of approaches [34], [28], [35] use a combination of reinforcement learning and Dynamic Movement Primitives [25] to learn demonstrated dynamic motions such as playing pool [34], playing with a ball-in-cup toy [28], and hitting a ball [35]. Cakmak and Thomaz [19] investigate guidelines for designing asking behaviors for improving robot skill learning from human kinesthetic demonstrations. Finally, Tenorth et al. [37] 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 [9], [18]). 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 [36] (used as a basis for the popular Lego Mindstorms kit) and Microsoft Robotics Studio's Visual Programing Language [10], 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 [13] was designed for creating new behaviors for reactive mobile robot navigation. The Robonaut 2 Command and Control Interface [24] 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) [22] 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 [11], Havok [3], and Unity3D [8] 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.

III. 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 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.

A. Robot Behavior Creation: Basic Concepts

Our user interface for authoring behaviors is shown in Figure 2, 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 [21].

Visually, the ROSCo interface for editing state machines is divided into three primary areas (right side of Figure 2). 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 [17]. ROS Commander state machines are compiled into Python SMACH state machines, which allows direct execution on the robot.

B. 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.

C. 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 [27] 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 [14]). 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 4). 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 [26]).

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 [33]. 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

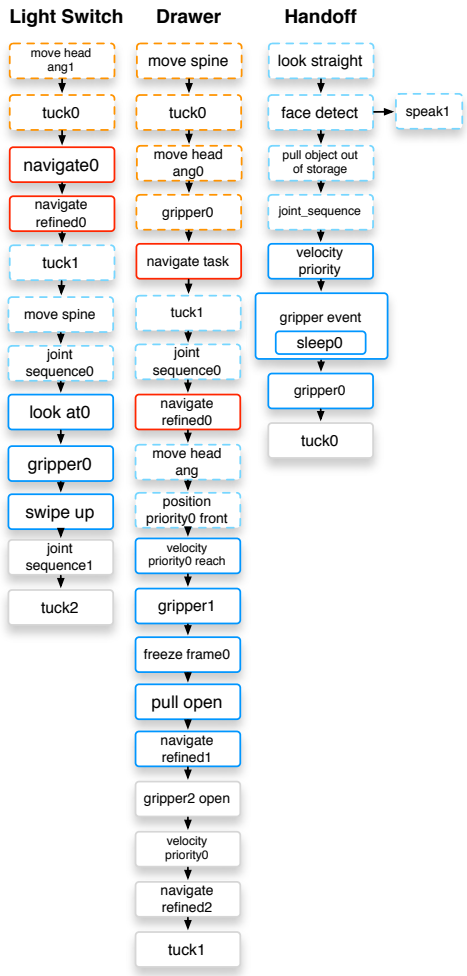

Fig. 3. HFSMs for three prototypical behaviors with states colored based on which phase of the task they represent. Dotted orange boxes are states that put the robot into a pre-navigation pose. Red states are navigation states. Dotted blue states are pre-manipulation states. Blue states perform the manipulation action for that behavior. Finally, grey states place the robot back in its starting pose.

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.

D. Example ROS Commander Behaviors in Detail

We now illustrate the anatomy of three typical ROSCo behaviors (Figure 3) 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).

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 3 also has this canonical structure.

With the object hand-off behavior in Figure 3, 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.

IV. Deploying Robots in New Homes: Associating Behaviors with Locations

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

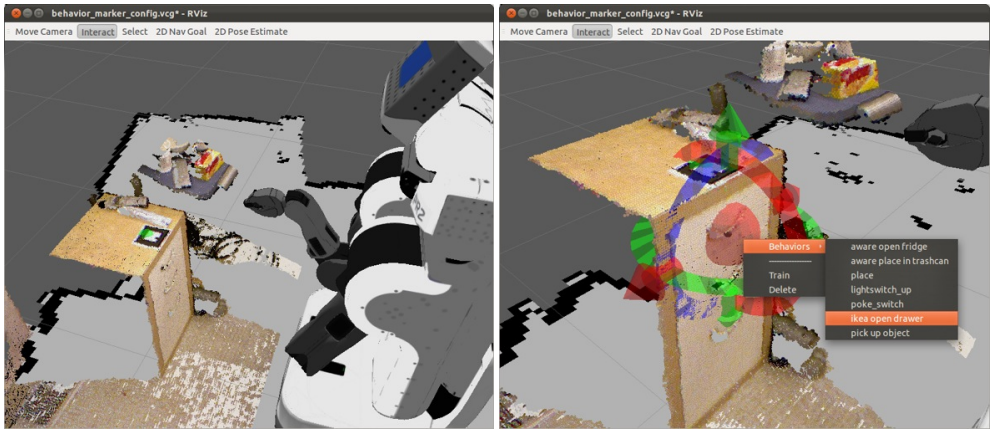

Fig. 4. Associating behaviors with new locations. Using as a reference point an AR ToolKit tag placed near the drawer (**Left**), the user can define a task frame centered on the handle (**Right**), and then start the appropriate behavior.

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 4.

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.

V. 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 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).

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 I

SUCCESS RATES FROM ALL TRIALS.

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

TABLE II

STATE TYPES USED BY EACH BEHAVIOR.

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 1). 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 III-D: 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 II (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 I.

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.

Fig. 5. Behaviors executed by Henry Evans (leftmost image), a person with quadriplegia, at this home: opening a refrigerator, opening a drawer, turning on the lights, and placing an object.

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.

We have also performed preliminary tests with these behaviors in the home of Henry Evans, a person with quadriplegia who can use computers by moving a mouse cursor with a head tracker, and also by clicking the left mouse button through limited use of his hand. With guidance, Henry was able to associate pre-constructed behaviors with mechanisms in his house, using the interface for associating behaviors with locations (discussed in Section IV).

Henry's primary challenge was in positioning the task frame for each behavior in the 3D interface. This is because placing the task frame on an obvious spot, such as the center of the drawer handle as it appears in the Kinect's point cloud, might not work due to issues such as poor robot calibration or limitations of the sensor. However, adjusting the saved location of the task frame slightly until the behavior worked was typically enough to overcome such issues. We enabled Henry to autonomously succeed in opening a refrigerator then grasp a bottle using teleoperation. He also placed that bottle on a table in his home, opened a drawer, and turned on a light switch (Figure 5).

We used nearly identical behaviors at Henry's home as in the Aware Home, which shows that our behaviors as created can generalize across fairly different environments. There were some minor differences: the behaviors in the Aware Home use a more general head-pointing behavior that can point the Kinect at the last known location of the AR tag prior to manipulation. At Henry's, this was stored as a static set of pan-tilt angles, only because the more general behavior had not been created yet. While, based on this data, we are not yet in a position to make a strong claim for the behaviors' ability to generalize, the results suggest that behaviors constructed using ROSCo can be ported with small adjustments to different homes.

VI. Conclusion & Future Work

In this paper we have 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, and handing an object off to a person. 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).

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 future work, we would like to investigate how non-roboticists interact with our system, and which changes, if any, are needed to enable the creation of useful robot capabilities by end-users. 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.

VII. Acknowledgements

We gratefully acknowledge support from Willow Garage and National Science Foundation (NSF) grants CNS-0958545 and IIS-1150157.

References

[1] Ecto. http://plasmodic.org/.
[2] Gostai Studio. http://www.gostai.com/products/jazz/gostai_studio/.
[3] Havok behavior. http://www.havok.com/products/behavior.
[4] Kismet. http://www.unrealengine.com/features/kismet/.
[5] Massive. http://massivesoftware.com.
[6] Robotflow. http://robotflow.sourceforge.net/.
[7] Ros commander for the pr2. http://ros.org/wiki/rcommander_pr2.
[8] Unity character animation. http://video.unity3d.com/video/4655480/unity-character-animation-gdc.
[9] Visual language research bibliography. http://web.engr.oregonstate.edu/~burnett/vpl.html.
[10] Vpl introduction. http://msdn.microsoft.com/en-us/library/bb483088.aspx.
[11] xaitcontrol. http://www.xaitment.com/english/products/xaitcontrol-for-unity.html.
[12] R. Alami, A. lie Clodic, V. Montreuil, E. A. Sisbot, and R. Chatila. Toward human-aware robot task planning. In *AAAI*, 2006.
[13] R. Arkin. Missionlab v7.0. http://www.cc.gatech.edu/ai/robot-lab/research/MissionLab/, July 2006.
[14] D. Ballard. Task frames in robot manipulation. In *AAAI*, 1984.
[15] M. Beetz, L. Mosenlechner, and M. Tenorth. Cram a cognitive robot abstract machine for everyday manipulation in human environments. In *IROS*, 2010.
[16] D. Benjamin, D. Lyons, and D. Lonsdale. Adapt: A cognitive architecture for robots. In *Conference on Cognitive Modelling*, 2004.
[17] J. Bohren and R. Rusu. Towards autonomous robotic butlers: Lessons learned with the pr2. In *ICRA*, 2011.
[18] M. Boshernitsan and M. Downes. Visual programming languages: A survey. Technical report, University of California Berkeley, 2004.
[19] M. Cakmak and A. Thomaz. Social machine learning , "designing robot learners that ask good questions. In *HRI*, 2012.
[20] S. Cambon, F. Gravot, and R. Alami. A robot task planner that merges symbolic and geometric reasoning. In *European Conference on Artificial Intelligence*, 2004.
[21] M. Ciocarlie, K. Hsiao, A. Leeper, and D. Gossow. Mobile manipulation through an assistive home robot. In *IROS*, In Press.
[22] R. G. Dromey, J. He, and Z. Liu. Formalizing the Transition from Requirements to Design. In *Mathematical Frameworks for Component Software Models for Analysis and Synthesis*, pages 156–187. 2006.
[23] R. Fikes and N. N.J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. In *Technical Note 43r. AI Center, SRI*, 1971.
[24] S. Hart, J. Yamokoski, and M. Diftler. Robonaut 2: A new platform for human-centered robot learning. In *RSS Workshop on Mobile Manipulation*, 2011.
[25] A. J. Ijspeert, J. Nakanishi, and S. Schaal. Learning attractor landscapes for learning motor primitives. In *NIPS*, 2003.
[26] A. Jain and C. C. Kemp. El-e: An assistive mobile manipulator that autonomously fetches objects from flat surfaces. In *Autonomous Robots*, 2010.
[27] H. Kato and M. Billinghurst. http://www.hitl.washington.edu/artoolkit.
[28] J. Kober and J. Peters. Imitation and reinforcement learning - practical algorithms for motor primitive learning. In *IEEE RAM*, 2010.
[29] K. Liu, D. Sakamoto, M. Inami, and T. Igarashi. Roboshop: multi-layered sketching interface for robot housework assignment and management. In *CHI*. 2011.
[30] T. Lozano-Perez, J. Jones, E. Mazer, and P. O'Donnell. *HANDEY: A Robot Task Planner*. M.I.T. Press, Cambridge, MA, 1992.
[31] C. McGann, F. Py, K. Rajan, H. Thomas, R. Henthorn, and R. McEwen. T-rex: A model-based architecture for auv control. In *ICAPS*, 2007.
[32] G. Metta, G. Sandini, D. Vernon, L. Natale, and F. Nori. The icub humanoid robot: an open platform for research in embodied cognition. In *PerMIS: Performance Metrics for Intelligent Systems Workshop*, 2008.
[33] H. Nguyen, T. Deyle, M. Reynolds, and C. C. Kemp. Pps-tags: Physical perceptual and semantic tags for autonomous mobile manipulation. In *IROS workshop*, 2009.
[34] P. Pastor, M. Kalakrishnan, S. Chitta, E. Theodorou, and S. Schaal. Skill learning and task outcome prediction for manipulation. In *ICRA*, 2011.
[35] J. Peters and S. Schaal. Reinforcement learning of motor skills with policy gradients. In *Neural Networks*, 2008.
[36] M. Portsmore. Robolab: Intuitive robotic programming software to support life long learning. In *APPLE Learning Technology*, 1999.
[37] M. Tenorth, U. Klank, D. Pangercic, and M. Beetz. Web-enabled robots – robots that use the web as an information resource. In *IEEE RAM*, 2011.