2015 IEEE International Conference on Robotics and Automation (ICRA)
Washington State Convention Center
Seattle, Washington, May 26-30, 2015

The Affordance Template ROS Package for Robot Task Programming

Stephen Hart[1] and Paul Dinh[2] and Kimberly Hambuchen[3]

Abstract— **This paper introduces the Affordance Template ROS package for quickly programming, adjusting, and executing robot applications in the ROS RViz environment. This package extends the capabilities of RViz interactive markers [1] by allowing an operator to specify multiple end-effector waypoint locations and grasp poses in object-centric coordinate frames and to adjust these waypoints in order to meet the run-time demands of the task (specifically, object scale and location). The Affordance Template package stores task specifications in a robot-agnostic JSON description format such that it is trivial to apply a template to a new robot. As such, the Affordance Template package provides a robot-generic ROS tool appropriate for building semi-autonomous, manipulation-based applications. Affordance Templates were developed by the NASA-JSC DARPA Robotics Challenge (DRC) team and have since successfully been deployed on multiple platforms including the NASA Valkyrie and Robonaut 2 humanoids, the University of Texas Dreamer robot and the Willow Garage PR2. In this paper, the specification and implementation of the affordance template package is introduced and demonstrated through examples for wheel (valve) turning, pick-and-place, and drill grasping, evincing its utility and flexibility for a wide variety of robot applications.**

I. INTRODUCTION

Recent advances in human-robot interface design have led to a number of useful tools for interacting with complex systems, such as humanoid robots. Specifically, the abundance of RGBD and LIDAR devices have resulted in the development of 3D visualization tools that reflect the state of the robot and environment in a virtual space with the aggregate data delivered by these devices. One such tool is RViz, designed for use with Robot Operating System (ROS) enabled systems, which provides extensible interfaces for both visualization and robot control. While RViz delivers a significant amount of off-the-shelf situational awareness to a human supervisor with a point-and-click interface for teleoperation [1], it does not inherently provide a standard tool for directing robots to accomplish more complex tasks that require extended sequences of commands. The Affordance Template framework, briefly introduced in [2], was developed to provide such task-level structure. This paper introduces the corresponding Affordance Template ROS package[1].

This work was supported by the DARPA Robotics Challenge and the NASA Human Robotics Systems project. Special thanks to the UT Austin HCRL lab, Brown University's RLAB, and David Lu!! for their support in this work.

[1]Stephen Hart is with TRACLabs, Inc. Houston, TX 77058, USA swhart@traclabs.com

[2]Paul Dinh is with Oceaneering Space Systems, Houston, TX 77058, USA paul.dinh@nasa.gov

[2]Kimberly Hambuchen is with NASA Johnson Space Center, 2101 NASA Pkwy, Houston, TX 77058, USA kimberly.a.hambuchen@nasa.gov

An *affordance* describes a place or object in the world that affords an action by a particular agent. If an area of the perceived environment affords "sitting," it is not that that area is a chair, but rather that it has a surface upon which that agent could sit, whether it be a chair, a stool, a rock, or the ground. The concept of affordances was initially introduced in the psychological literature by Gibson [3] as a means of describing cognitive structures that assign functional merit to the environment. It is therefore conceptually applicable for programming robots—embodied agents that must perform actions to achieve some discernible objective—where an operator (or the robot itself) can similarly assign such functionality to discernible areas in observed sensor data.

An *affordance template* is a computational construct that exists in a graphical 3D immersive environment to provide human-adjustable robot task goals and parameters in object-centric coordinate frames. If a template is placed in such an environment alongside a robot avatar and its aggregate sensory data, a (human) supervisor can move the template to an appropriate location—that which is hypothesized to afford the task behavior—and adjust the template goals and parameters as needed according to the perceived current run-time context. Template parameters capture the key degrees of freedom of the task in an efficient representation making them useful tools for shared autonomy between an operator and a robot and suitable for control over unreliable networks or in scenarios where degraded communications persist.

The Affordance Template (AT) package is an attempt to standardize application programming for robots with one or more end-effectors. Unlike the *de facto* standard ROS-compatible packages for robot navigation [4], motion planning [5], computer vision (OpenCV), or 3D perception (PCL), affordance templates exist on-top of these tools, integrating such functionality as needed. While the current release incorporates only the MoveIt! package, ongoing work seeks to extend the framework to robot navigation, both wheeled and legged, and to integrate it with autonomous perception to alleviate run-time operator adjustments of task parameters. These topics, and others, will be discussed in more detail in Section VI. In the remainder of the paper, the current release of the AT package is described along with some motivating examples showing its utility in various manipulation demonstrations.

[1]https://github.com/swhart115/affordance_templates. Currently available for both Hydro and Indigo ROS releases.

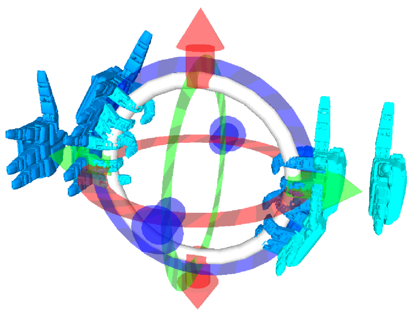
Fig. 1. A wheel-turning AT instantiated for the Valkyrie robot. Each end-effector visualization shows a different waypoint for the robot (pre-grasp, grasp, turn-goal, etc.) and a corresponding grasp pose (open, closed), and in the coordinate frame of the white wheel at the center. These waypoints have an order that allow the operator to step forward or backward along the trajectory, or to "play-through" the full sequence till the end. The entire template can be moved in RViz by the 6-DOF controls shown, or individual waypoints can be adjusted by right-clicking on the hand to expose corresponding controls.

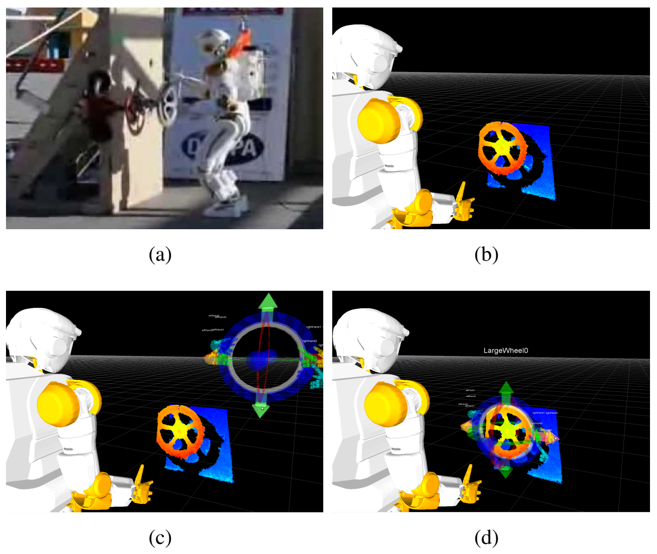

Fig. 2. Placing a wheel-turning AT in RViz for use with the Valkyrie robot.

II. RELATED WORK

J.J. Gibson initially proposed the "theory of affordances," suggesting that organisms perceive their environment in terms of their ability to interact with it [3]. It is thus a natural area of study for embodied learning agents. Applying this theory to robotics, researchers have examined how a robot can learn affordances for pushing and grasping objects [6], tool use [7], and navigation [8]. More generally, various researchers have modeled affordances probabilistically in terms of likely effect of robot behavior [9], as planning operators that can be used in extended sequences of manipulation tasks [10], or in collections that can be acquired through intrinsically motivated reinforcement learning [11], [12]. This work focuses on computationally modeling affordances and is interesting from a theoretical perspective, but automatic recognition of a robot's affordances remains an area of ongoing research. The affordance templates introduced in this paper follow a more pragmatic approach that takes inspiration from the theory, but provides a human-in-the-loop solution: allow an operator to assess a robot's environment through its sensor data and assign affordances manually to bootstrap application and task programming.

A number of large-scale projects have focused on building rich and general application tools and libraries so that developers do not always have to start from scratch to integrate functionality developed by the community. In particular, MATLAB Robotics Toolkit [13], Microsoft Robotics Developer's Studio [14], BRICS [15], ROS [16], YARP [17], and OPRoS [18] have supported a variety of cross-language and cross-platform tools that promote building libraries of functionality and the best practices of component-based software design. These projects often include GUI tools to manage and configure deployed components. ROSCo [19], in conjunction with SMACH [20] provide ROS-based utilities for developing and deploying ROS-based hierarchical applications, but are limited in their ability to modify task structure at run-time. Robot Task Commander (RTC), provides a similar IDE for application development for use with multiple middlewares, but development still requires a certain amount of expert or domain knowledge [21]. These utilities largely remain engineering tools, more useful for *a priori* application development, not flexible tools that allow non-experts to program or adjust applications on-line. Such an approach suggests a more *shared* level of programming where the robot can provide a level of expertise (either pre-programmed or acquired through experience) for decision-making or to handle more of the fine-details of a task, alleviating the load on the operator.

Shared autonomy approaches to robotic operation have become favored methods owing to the desired semi-autonomous nature of many systems. Pitzer *et al.* describe a shared autonomy system in which a human operator assists a robot with manipulation tasks by perceiving and detecting objects in the robot's environment [22]. Witzig *et al.* demonstrate a shared autonomy approach to grasp planning in which the human operator provides contextual information that the robot cannot perceive [23]. Shared autonomy grasping has been demonstrated with RViz interactive markers by Gossow *et al.* [1]; however, the most interaction the human has with the system is either in adjusting single-grasp locations or in confirming that a location is acceptable. Both the MIT and IHMC DRC teams used similar methods to the AT framework. MIT developed the *Object Template Description Format* to direct their user interface [24]. This format provides a representation suitable for use by a human in the shared control of the ATLAS robot. The human operator uses this format to perform "affordance fitting" on their interface, which can also be done autonomously by the robot using its perceptual data. Interactivity is flexible for the operator, providing different levels of shared control. IHMC used a "coactive design" approach that integrates with their visualization tool [25]. This approach provides a similar user interface to ATs to allow for shared autonomy by combining

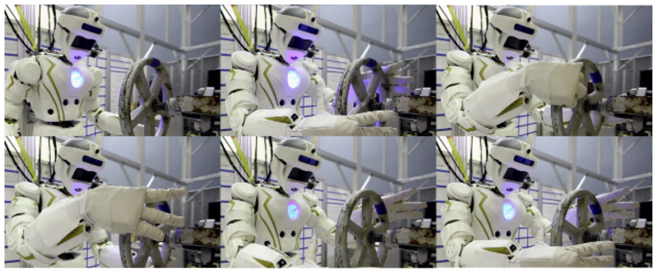

Fig. 3. Valkyrie turning a valve using a wheel-turning template.

human perception of visualized data and robot planning of mobility and grasping. However, neither of these methods provide a level of adjustment and interactivity that the AT framework can provide, which increases the flexibility of both human operation and robot task activity, nor are they provided as general purposes tool that are readily available for use with other robots (at this point). However, the MIT and IHMC approaches demonstrate a convergence of user interface methods for shared autonomy between human and robots that make the contribution of an open-source tool such as the Affordance Template ROS package timely.

III. AFFORDANCE TEMPLATES

This section describes the Affordance Template framework and details the components within. Figures 1—3 provide a motivating example for the framework in which a wheel-turning template (Figure 1) is registered to the NASA-JSC Valkyrie robot's sensory data in order to have it turn a valve. In Figure 2(a), the NASA-JSC Valkyrie robot is shown standing in front of a valve. In (b), the RViz display window shows the robot avatar and the point cloud data received from Valkyrie's head-mounted Ensenso sensor. From this view, an operator can clearly identify the valve in the robot's workspace, and can manually register the wheel template to the location of the valve, as seen in (c), adjusting its size as necessary via a slider in a custom RViz panel (not shown). Additionally, the wheel template allows the operator to use RViz interactive markers to adjust the hand pre-grasp, grasp, and turn-goal waypoint locations (shown as different colored end-effector overlays, and defined in the coordinate frame of the wheel). When the goal locations are set, as in (d), the corresponding robot arm trajectories can be viewed in RViz to provide user feedback on what the robot's movement will look like, if a path is found. These trajectories are computed using MoveIt! When the operator is satisfied with the displayed trajectories the RViz panel can be used to execute these trajectories—one or multiple waypoints at a time, forwards or backwards. Figure 3 shows Valkyrie using the template to turn a valve.

A. Structure

The structure of an affordance template is shown in Figure 4(a). Each affordance template, $a \in A$, is a directed acyclic graph of *display objects*, O_{obj}, and ordered sequences of *end-effector waypoints*, W_{ee}, expressed in coordinate frames defined by the display objects. Each sequence of waypoints represents a set of end-effector configurations that, when followed, achieve the AT's intended task. For each template there must be one object $o_{root} \in O_{obj}$, designated the "root" object of the template, with a coordinate frame ${}^{robot}F_{root}$ expressed in the robot frame, and some set (possibly empty) of child objects arranged in a tree structure descending from that root object. As the root object is defined such that it has no incoming edges, each a forms a rooted tree. Every $wp \in W_{ee}$ also has a single parent object in O_{obj} and corresponding Cartesian pose $({}^{obj}p_{wp}, {}^{obj}R_{wp})$ expressed in the coordinate frame of this parent object. Each display object may also have a shape associated with it. In practice, this shape can be a primitive shape such as a box or cylinder, or a mesh shape in the form of an STL or Collada model.

The relationship between display objects and waypoints in a thus described represents the core structure of the AT. The details about object scales, template location, and how the end-effector waypoints map to specific robot control groups must be determined upon instantiation of each template. Specifically, each end-effector waypoint, wp, has an abstract ID associated with it (for convenience chosen to be in the set of natural numbers $\mathbb{N}$), that represents its corresponding robot end-effector, a sequence ID, and a grasp pose. For example, if a particular template is to be applied to a bi-manual system, such as Valkyrie, the waypoints in that template are assigned an end-effector ID of "0" or "1", the former mapping to the robot's left hand, the latter to its right. The grasp poses will similarly map to common configurations such as "hand closed" or "hand open"[2]. In Figure 4(a), each waypoint is labeled as $wp_{<ee\_id>:<seq\_id>}$, where $<ee\_id>$ is the end-effector ID, and $<seq\_id>$ is the ID of that waypoint in the ordered sequence for that end-effector. When enabling a robot to be used in the AT framework, a robot configuration file (as described below) needs to be constructed that defines the mappings between abstract IDs and the end-effector groups and grasp poses.

Finally, a key advantage of the affordance template methodology is that parameters can be scaled at run time to allow the more intuitive ability of overlaying objects onto (possibly variable) shapes in the robot's environment. Scaled parameters include the sizes of display objects and the distance between these objects and their children. A template, such as the wheel template introduced in Figure 1, can be scaled and positioned by the operator to allow it to be used for a large class of turning tasks with minimal effort. The strategy for scaling is shown in Figure 4(b). As the operator adjusts the overall scale s_{obj} of an object, the magnitude of the vector pointing to the coordinate frame of any waypoint expressed in the frame of that object, ${}^{obj}p_{wp}$, is scaled accordingly. This strategy is used to adjust the location of child display objects, as well as waypoint goals, when the parent object is scaled.

[2]Ultimately, an "automatic grasp" option seems desirable in order to let the robot determine suitable configrations on its own, but this is beyond the scope of the initial AT presentation.

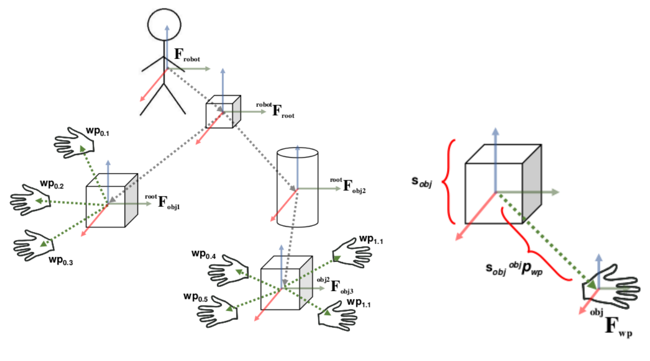

Fig. 4. Diagram (a) shows the generic structure of ATs. Diagram (b) shows the template scaling methodology.

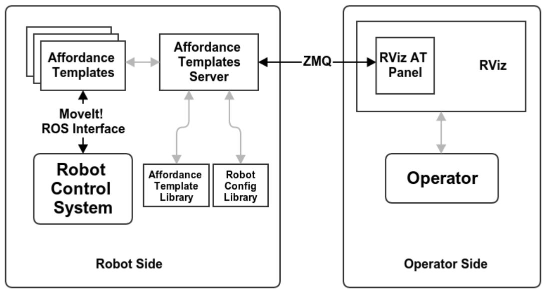

Fig. 5. The AT implementation architecture. Dark, labeled arrows indicate network communication (ROS or ZMQ), light arrows indicate code or operator interfaces.

B. Implementation Architecture

The implementation of the AT package is structured as seen in Figure 5. An affordance template *server* ROS node loads the available templates and robot configurations stored in libraries on disk, and sends this information to a custom RViz panel over ZeroMQ (ZMQ) [26] using JSON data structures[3]. From the RViz panel, an operator can choose which template they want to use and for which robot. This information is sent back to the server accordingly, which then instantiates a new template as a Python class, running as a separate process within the ROS node. This instantiated class uses the mappings defined in the robot configuration file to display RViz interactive markers for the display objects and robot end-effectors at the locations and in the configurations defined by the template. This process is what allows the instantiation of the wheel-turning AT shown in Figure 1 to display the white wheel at the center and the displays of Valkyrie's hands in the open and closed configurations at the specified waypoint locations (for grasping and turning the wheel).

Once the template is instantiated in RViz, the operator can then interact with it, moving it to the desired location, scaling any display objects as necessary, and adjusting the locations of the end-effector waypoints. When configured satisfactorily, MoveIt! generated `JointTrajectory` messages are computed and sent to the robot in order to follow the waypoint trajectories at the operator's discretion (from the panel to the server to the template). If the operator wishes to modify template or robot configuration parameters, updated information can be sent back to the server to be stored in the libraries for future use. The next sections discuss the structure of the robot configuration and AT files that are stored in the libraries.

C. Affordance Template Description Format

Affordance templates are stored in the AT library using an *Affordance Template Description Format* (ATDF) syntax implemented in JSON. Every ATDF file starts with the `name` of the template and an `image` for display in the RViz panel. Next follows the description of display objects and end-effector waypoints. Every object must have a unique `name`, `shape`, `origin`, and `controls` element. If no `parent` element is provided (referring to the name of a different display object), the origin is taken to be with respect to the root AT coordinate frame as specified in the robot configuration file (defined below). If a parent is given, the origin will be in the coordinate frame of that object. The geometry element allows the developer to chose primitive shapes such as cubes, spheres, and cylinders, or meshes. The controls element is a 6-DOF binary `mask` defining which dimensions the operator can adjust the object along in the RViz window (ordered as [x y z roll pitch yaw]). For example, if only translation controls are desired, the RPY values would be set to "0 0 0" and orientation controls would not be provided to the operator for the corresponding object in RViz. The following example shows the display object for the wheel template. It uses a Collada-format mesh (`torus.dae`) that is pitched 90 degrees, and allows full 6-DOF position/orientation controls. The scale factor is useful to appropriately scale the size of the IM controls, specifically for clutter reduction in the RViz window.

```
"name": "Wheel",
"image": "wheel.png",
"display_objects": [
  {
    "name": "wheel",
    "shape": {
      "type": "mesh",
      "data": "torus.dae",
      "size": [1,1,1]
    },
    "origin" : {
      "xyz": [0,0,0], "rpy": [0,1.57,0]
    },
    "controls": {
      "mask": [1,1,1,1,1,1], "scale": 0.3
    }
  }
```

Any number of display objects are acceptable for each template, in whatever tree-based relationship specified. There must be one object that has no parent (the root object), and thus is defined in the root frame of the template.

[3]The decision to use ZeroMQ as the transport layer between "robot-side" and "operator-side" was made to provide a simple, but robust protocol, more flexible than ROS service calls, appropriate for use with degraded or unreliable networks or with multi-master ROS environments.

End-Effector waypoints are defined similarly. Multiple different waypoint trajectories can be stored in the ATDF for each template, distinguished by the `name` tag, which the operator can switch between at run-time using a right-click context menu. Different trajectory options could allow a template to store different ways of accomplishing at task suitable for different types of (or particular) robots. For example, the wheel-turning template could store one-handed trajectories for uni-manual robots in addition to the two handed strategy shown in Figure 1.

```
"end_effector_trajectory": [
{
  "name": "Two-Hand Counter-Clockwise Turn",
  "end_effector_group" : [
   {
       {
         "ee_pose": 0,
         "display_object": "wheel",
         "origin": {
           "xyz": [0,0.2,-0.1], "rpy": [0,-1.57,0]
         },
         "controls": {
           "mask": [1,1,1,0,1,1], "scale": 0.25
         }
       },
       {
         "ee_pose": 2,
         "display_object": "wheel",
         "origin": {
           "xyz": [0,0.2,-0.1], "rpy": [0,-1.57,0]
         },
         "controls": {
           "mask": [1,1,1,0,1,1], "scale": 0.25
         }
       },
.
.
```

The end-effector ID that will be mapped to the specific robot group is defined by the order of the `end_effector_group` fields. Similarly, the waypoint sequence ID for each waypoint is defined by the order each is provided in this file. The configuration (i.e., hand closed, hand open) is defined by the `ee_pose` tag. The display object that the waypoint's location is defined with respect to is defined as `display_object`. The `origin` and `controls` tags capture similar information as for the display objects.

D. Robot Configuration File

To instantiate an AT for a particular robot, a simple YAML configuration file is required to map the abstract declarations of the robot-generic template defined in the ATDF to the specific exigencies of each robot. This ensures that specific end-effectors (*e.g.* left or right hands) will be mapped to the appropriate template waypoints and that the template will exist in the appropriate robot base coordinate system. This configuration file provides the name of the robot, the name of the robot's MoveIt! configuration package (generated through the MoveIt! "setup wizard"), and some basic information about the robot that needs to be determined for use with the AT package. This information includes the root offset from the robot's base frame to where any AT is first placed, transformation coordinates to align the robot's end-effectors with the abstract AT end-effector frames (such that hands are oriented appropriately), and mappings between the abstract template end-effector labels and the corresponding end-effector group names. For example, the YAML config file for Robonaut 2 is as follows:

```
robot_name: r2
moveit_config_package: r2_moveit_config
frame_id: r2/robot_base
root_offset: [0.4,0,-0.2,3.14,0,0]
end_effector_map:
  - name: left_hand
    id: 0
    pose_offset: [0,0,0,1.57,0,0]
  - name: right_hand
    id: 1
    pose_offset: [0,0,0,-1.57,0,0]
end_effector_pose_map:
  - name: Right Hand Open
    group: right_hand
    id: 0
  - name: Right Hand Close
    group: right_hand
    id: 1
  - name: Right Hand Point
    group: right_hand
    id: 2
    .
    .
```

In this example, `end_effector_map` is used to map the abstract end-effector labeled "0" in the ATDF will to the R2 MoveIt! group `left_hand`. Correspondingly, end-effector "1" will be mapped to `right_hand`. These group names must be defined in the generated SRDF file created by the MoveIt! setup process. Each group also has a `pose_offset` field necessary to re-orient the command and display of the end-effectors into the robot-agnostic AT frame convention. Similarly, `end_effector_pose_map` is used to map the abstract AT end-effector pose identifiers of the ATDF to poses stored in the SRDF. For example, the stored pose `Right Hand Close` for end-effector group `right_hand` is here mapped to the AT ID "1". Additional pose mappings are stored in this file as well, but omitted for reasons of space. When configuring a new robot for use with the AT package, the developer must determine the appropriate settings for these values. An automatic "wizard" tool could be useful to facilitate this process, but such a tool not been developed at this point.

IV. RVIZ INTERFACE

Two complimentary methods are provided to create, modify, or interact with ATs in order to execute task behavior on a robot: a custom RViz panel and context-menus available for each template after it has been instantiated.

A. Affordance Template Panel

Figure 6 shows the RViz Affordance Template panel. In the main RViz window, the UT Dreamer robot is shown with a pick-and-place AT. Basic information about AT server connectivity is shown in the right-side panel, along with which templates have been initiated. Multiple instances of any template can be instantiated at any given time. In this panel, the operator can also delete existing templates. A second hidden tab in the top section allows the operator to assign robots to the ATs, as well as to change and save parameters of that robot's configuration YAML (Section III-D). The lower half of this panel shows the AT library. To instantiate a new template, the operator need only select an icon from this list and the corresponding template appears in the RViz window for the appropriate robot. Shown are templates for a door handle, the R2 ISS Taskboard (discussed in more detail in Section V-D) and pick-and-place.

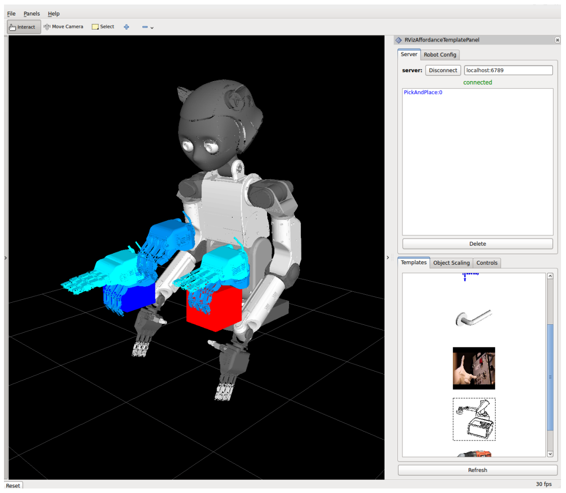

Fig. 6. Dreamer in RViz with a pick-and-place template, and the AT panel on the right.

Other tabs in the bottom half of the panel are seen for scaling objects (as described in Section III-A) or managing the end-effector waypoint trajectory of the template. The operator has the option of commanding the robot to move through these trajectories any number of steps at a time in any direction, or for de-coupling the commands to the end-effectors (so that they can be run in conjunction or in isolation), if desired. For considerations of space, explicit images of these panels are omitted.

B. Context Menus

Right-clicking on display objects or waypoints in the main RViz window provides the operator a number of useful features, including the following.

- **Hide/Show Controls:** To reduce clutter in the RViz environment, the operator has the option of not displaying the controls at all times. Exposure of the controls is used occasionally for placing the template in the appropriate location or for fine-tuning end-effector positions.
- **Add/Delete/Move Waypoints:** The operator has options to add waypoints to the existing end-effector trajectory (either before or after existing goals), delete waypoints, or swap the sequence order of any pair of waypoints.
- **Change Grasp Pose:** For waypoints, the operator has the option to alter which stored grasp pose is commanded at each waypoint. All poses available in the robot's MoveIt!-generated SRDF file are provided as options.
- **Save Modifications:** The operator can save any changes to the template back to the AT library for future use.
- **Change Trajectory:** This options lets the operator choose which of the multiple trajectories associated with a template is displayed at any given time.

V. Examples

In this section, we briefly explore a number of motivating examples demonstrating some key features of the AT package and how they apply to different robots and contexts.

A. *MoveIt! Integration*

Manual placement of an AT in a location does not explicitly determine if the corresponding waypoint trajectory is within the workspace of the robot, nor does it generate or show the path the robot would follow to achieve that trajectory. To these ends, the MoveIt! motion planning library was integrated into the AT package to provide both of these capabilities inherently. In general, any motion planning library could be used for these purposes, but MoveIt! provides an off-the-shelf solution that is widely used by the ROS community. Figure 7 shows an RViz screen capture of the wheel-turning template in action instantiated for the R2 robot. The MoveIt!-generated path from the robot's current location through the first two waypoints (for each arm's sequence) is overlaid in purple. From the control panel, the operator can choose how many waypoint steps "ahead" in each sequence are planned, displayed, and executed at a time. Integrating MoveIt! quickly allows an operator to gain introspection on the achievability of the task at hand.

B. *Robot Generalization*

Because each template stores only abstract information about a task, a robot configuration YAML description (Section III-D) is required to instantiate the template for a given context. Figure 8 shows a drill AT instantiated for the R2, Valkyrie, and Dreamer systems. If the trajectory stored in the

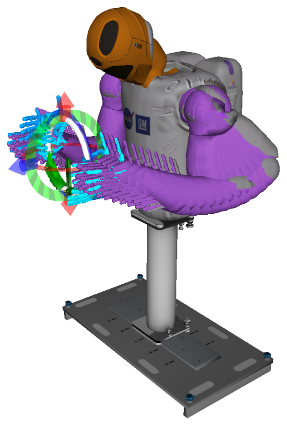

Fig. 7. Visualization of incremental path demonstrated when using the wheel turning AT with the R2 system. The MoveIt! generated path is display between the robot's current position and the first end-effector waypoint for each arm. The wheel template has been overlaid on top of the RGBD data observed by the robot showing a green valve in its workspace.

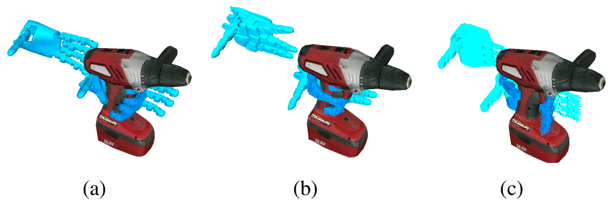

Fig. 8. A drill grasping AT instantiated for R2, Valkyrie, and Dreamer.

ATDF is suitable for a given robot "off-the-shelf," no further programming is necessary, only than manual registration of the template onto the robot's 3D data in RViz. If a robot-specific trajectory is needed, the operator is free to expose the individual waypoint controls, modify a stored trajectory as needed, and save the modified template for future use.

C. Object Scaling

With the scaling procedure discussed in Section III-A, the wheel template can be used in different run-time contexts. Figure 9 shows this template (instantiated for R2) in an array of sizes. The size of the template was adjusted using a slider in the "Object Scaling" tab in the bottom half of the RViz AT panel (Figure 6), which was generated automatically from the ATDF definition.

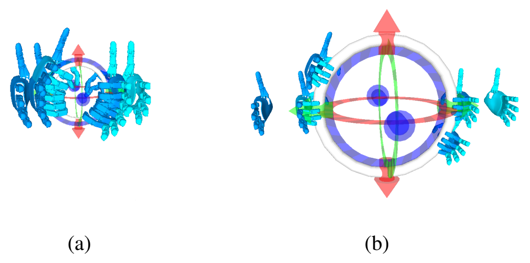

Fig. 9. The wheel-turning template scaled to smaller and larger sizes.

D. Multi-Trajectory Storage

Figure 10 shows three waypoint trajectories stored in the ISS Taskboard AT. The taskboard represents a set of canonical manipulation tasks (switches, buttons, handles, etc.) present on the ISS that is used for R2 development and practice as the robot's operators learn how to control a humanoid robot in space. Each of the three images shown in the figure displays a different waypoints trajectory, each one for pushing a different taskboard button. As there are dozens of different tasks on the taskboard, it is natural to allow the single taskboard AT to store strategies for achieving these tasks, as opposed to storing many separate templates in the library. In this manner, an operator can choose from the AT context menu which task to perform at any given time.

VI. ONGOING WORK

The AT framework follows a supervisory control paradigm of *Plan, Teach, Monitor, Intervene, and Learn* (PTMIL) [27]. Currently, the framework supports only the TMI steps of this paradigm as it only allows the supervisor to place a template in an appropriate location, adjust certain parameters, and execute the resulting plan on the robot, intervening if necessary to make spatial adjustments. Currently, the framework is being extended along multiple dimensions.

- **Planning:** Integration of motion planning tools such as MoveIt! allow an extra level of verification, intelligence, and visualization for the supervisor concerning the robot's expected motion for accomplishing the goal(s) of the AT. Additional uses of robot planning could also be used to eliminate the necessity for the supervisor to define the waypoint trajectories, to set *all* task parameter (as is currently necessary), or to chain ATs together to accomplish multiple extended behaviors (*e.g.* walk up to an object, pick it up, use it for some purpose).
- **Learning:** ATs could monitor the user choices of object placement and waypoint adjustment over the course of multiple executions to provide statistical feedback on how best to "set-up" a template such that it is likely to lead to task success, correlating template parameters with information like the robot's perceived sensory signals. The metric of success in such learning could be supervised (*i.e.* let the supervisor indicate task outcome after execution) or unsupervised depending on the task's requirements.
- **Perceptual Registration:** Rather than relying on the supervisor to register templates with sensor data (as in Figure 2), autonomous point cloud registration techniques such as ICP could be used to accomplish this task autonomously. Additionally, the robot could monitor its environment and provide guesses about likely affordances, initiating corresponding ATs with pre-adjusted parameter settings in RViz accordingly.
- **Navigation & Mobility Tasks:** The AT package as described supports only manipulation-based tasks. Extending the AT package to allow locomotion-based templates (*e.g.* walking up stairs, driving through a cluttered environment) would be desirable to unify mobility and manipulation in the same application framework. Incorporation of the ROS Navigation Stack [4] is currently being investigated as it provides advanced mapping and mobility tools. Applying such templates to multi-legged locomotion is also desirable, though potentially more challenging.
- **Force-Based Tasks:** Currently, existing ATs only allow the supervisor to set spatial parameters. However, force- or contact-based goals (*e.g.* apply a force along an axis of a certain magnitude, turn the wheel with a desired torque) could also be set in the framework. Such specifications will ultimately be necessary for guiding a robot to accomplish sophisticated manipulation tasks in real-world contexts. The visualization of these parameters in a 3D spatial environment is an interesting problem on its own terms, and currently various approaches are under investigation.

Although some of these extensions individually may require significant research contributions, the AT framework provides an efficient representation for aggregating this functionality together in a unified structure. By keeping the supervisor "in the loop" and phasing in more autonomy as appropriate—whether in terms of planning, perception, or learning—the framework supports multiple levels of intervention that can ensure task success in multiple contexts.

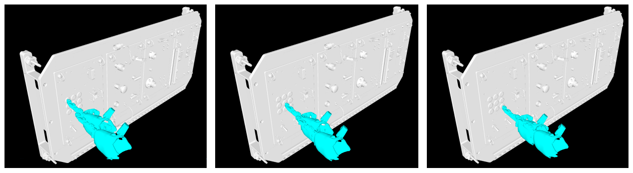
(a) (b) (c)

Fig. 10. AT trajectories for pushing different buttons on the ISS Taskboard.

VII. CONCLUSION

This paper introduces the Affordance Template ROS package for creating, executing, and supervising task-level applications on manipulation-based robotic systems. The package provides a robot-generic framework suitable for use in position-based tasks, though flexible enough to be modified easily to meet the demands of specific contexts. Although the package has clear limitations, it provides a core library in which further extensions can be made as described in Section VI. While the current implementation provides a set of tools that various members of the community are converging on, such as 3D tools for placing and adjusting robot end-effector goals amidst overlaid RGBD sensor data and a robot avatar, visualization of robot trajectories in anticipation of commanded movement, object-centric coordinate frame goal definitions (cf. [25], [24]), the AT package provides the first open-source ROS package that is, by construction, applicable to multiple robots, easily extensible to new tasks, and built using common tools such as RViz interactive markers and MoveIt! A common toolkit for robot application development will greatly progress robot task development in the community, while mitigating redundant development by various groups in the field. By choosing a ROS grounding for the package, it is also the hope that the package will be familiar and easy to use to prevent a steep learning curve for new users.

REFERENCES

[1] D. Gossow, A. Leeper, D. Hershberger, and M. T. Ciocarlie, "Interactive markers: 3-D user interfaces for ros applications [ros topics]," *IEEE Robotics & Automation Magazine*, vol. 18, no. 4, pp. 14–15, 2011.
[2] S. Hart, P. Dinh, and K. Hambuchen, "Affordance templates for shared robot control," in *Artificial Intelligence and Human-Robot Interaction, AAAI Fall Symposium Series*, Arlington, VA. USA, November 2014.
[3] J. J. Gibson, "The theory of affordances," in *Perceiving, acting and knowing: toward an ecological psychology*. Hillsdale, NJ: Lawrence Erlbaum Associates Publishers, 1977, pp. 67–82.
[4] E. Marder-Eppstein, E. Berger, T. Foote, B. Gerkey, and K. Konolige, "The office marathon: Robust navigation in an indoor office environment," in *International Conference on Robotics and Automation*, 2010.
[5] I. A. Sucan and S. Chitta. MoveIt! [Online]. Available: http://moveit.ros.org
[6] P. Fitzpatrick, G. Metta, L. Natale, S. Rao, and G. Sandini, "Learning about objects through action: Initial steps towards artificial cognition," in *IEEE International Conference on Robotics and Automation*, Taipei, May 2003.
[7] A. Stoytchev, "Toward learning the binding affordances of objects: A behavior-grounded approach," in *Proceedings of the AAAI Spring Symposium on Developmental Robotics*, Stanford University, 2005.
[8] J. Modayil and B. Kupiers, "Autonomous development of a grounded object ontology by a learning robot," in *Proceedings of the Twenty-Second Conference on Artificial Intelligence (AAAI-07)*, 2007.
[9] E. Şahin, M. Çakmak, M. Doğar, E. Uğur, and G. Üçoluk, "To afford of not to afford: A formalization of affordances toward affordance-based robot control," *Adaptive Behavior*, vol. 4, no. 15, pp. 447–472, 2007.
[10] N. Krüger, J. Piater, F. Wörgötter, C. Geib, R. Petrick, M. Steedman, A. Ude, T. Asfour, D. Kraft, D. Omrcen, B. Hommel, A. Agostino, D. Kragic, J. Eklundh, V. Kruger, and R. Dillmann, "A formal definition of object action complexes and examples at different levels of the process hierarchy," http://www.paco-plus.org, 2009.
[11] S. Hart and R. Grupen, "Intrinsically motivated affordance learning," in *2009 Workshop on Approaches to Sensorimotor Learning on Humanoids at the IEEE Conference on Robots and Automation (ICRA)*, Kobe, Japan, 2009.
[12] ——, "Intrinsically motivated affordance discovery and modeling," in *Intrinsically Motivated Learning in Natural and Artificial Systems*, G. Baldassarre and M. Mirolli, Eds. Springer Berlin Heidelberg, 2013, pp. 279–300.
[13] P. I. Corke, *Robotics, Vision & Control: Fundamental Algorithms in Matlab*. Springer, 2011.
[14] Microsoft Co., "Microsoft robotics developers studio," http://msdn.microsoft.com/en-us/robotics/, 2008.
[15] H. Bruyninckx, M. Klotzbücher, N. Hochgeschwender, G. Kraetzschmar, L. Gherardi, and D. Brugali, "The BRICS component model: A model-based development paradigm for complex robotics software systems," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, ser. SAC '13. New York, NY, USA: ACM, 2013, pp. 1758–1764. [Online]. Available: http://doi.acm.org/10.1145/2480362.2480693
[16] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.
[17] G. Metta, P. Fitzpatrick, and L. Natale, "YARP: yet another robot platform," *Int. Journal on Advanced Robotics Systems, Special Issue on Software Development and Integration in Robotics*, March 2006.
[18] C. Jang, S.-I. Lee, S.-W. Jung, B. Song, R. Kim, S. Kim, and C.-H. Lee, "OPRoS: A new component-based robot software platform," *ETRI journal*, vol. 32, no. 5, pp. 646–656, 2010.
[19] H. Nguyen, M. Ciocarlie, J. Hsiao, and C. C. Kemp, "ROS Commander (ROSCo): Behavior creation for home robots," in *Proceedings of the IEEE Conference on Robotics and Automation (ICRA)*. ICRA, 2013.
[20] J. Bohren and S. Cousins, "The smach high-level executive," *Robotics & Automation Magazine, IEEE*, vol. 17, no. 4, pp. 18–20, 2010.
[21] S. Hart, P. Dinh, J. Yamokoski, B. Wightman, and N. Radford, "Robot Task Commander: A framework and IDE for robot application development," in *International Conference on Intelligent Robots and Systems (IROS)*. Chicago, IL. USA: IEEE/RSJ, September 2014.
[22] B. Pitzer, M. Styer, C. Bersch, C. DuHadway, and J. Becker, "Towards perceptual shared autonomy for robotic mobile manipulation," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, May 2011, pp. 6245–6251.
[23] T. Witzig, J. Zollner, D. Pangercic, S. Osentoski, R. Jakel, and R. Dillmann, "Context aware shared autonomy for robotic manipulation tasks," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, Nov 2013, pp. 5686–5693.
[24] M. Fallon, S. Kuindersma, S. Karumanchi, M. Antone, T. Schneider, H. Dai, C. Perez D'Arpino, R. Deits, M. DiCicco, D. Fourie, T. Koolen, P. Marion, M. Posa, A. Valenzuela, K.-T. Yu, J. Shah, K. Iagnemma, R. Tedrake, and S. Teller, "Affordance-based Perception and Whole-body Planning in the DARPA Robotics Challenge," MIT, Cambridge, MA, Tech. Rep. MIT-CSAIL-TR-2014-003, 2014.
[25] T. Koolen and J. Smith, "Summary of Team IHMC's Virtual Robotics Challenge Entry," in *IEEE-RAS International Conference on Humanoid Robots*. Atlanta, Georgia: IEEE-RAS, 2013.
[26] iMatix Corporation. (2007) ZeroMQ: Distributed Computing Made Simple. [Online]. Available: http://zeromq.org
[27] T. B. Sheridan, *Telerobotics, Automation, and Human Supervisory Control*. The MIT Press, 1992.