Decoupling Behavior, Control, and Perception in Affordance-Based Manipulation

Tucker Hermans James M. Rehg Aaron F. Bobick

***Abstract*— A novel mechanism is introduced by which a robot can connect the general notion of an affordance of an object to specific behaviors by which the robot can achieve the desired action. We achieve this by decomposing an affordance-drive behavior into three components. We first define *controllers* that specify how to achieve a desired change in object state through changes in the agent's state. For each controller we develop at least one *behavior primitive* that determines how the controller outputs translate to specific movements of the agent. Additionally we provide at least one *perceptual proxy* that defines the representation of the object that is to be computed as input to the controller during execution. A variety of proxies may be selected for a given controller and a given proxy may provide input more than one controller. Decoupling these components allows the systematic exploration of a variety of strategies when evaluating the affordances of novel objects. We demonstrate the approach using a PR2 robot that executes different combinations of controller, behavior primitive, and proxy to perform a push positioning behavior on a selection of household objects.**

I. Introduction

As the goal of having robots operate in uncontrolled environments becomes more critical to the advancement of robotics, there has been much research on the notion of *affordances* of objects with respect to a robot agent [1]. Within the context of robotics affordances describe the possible actions an agent can take acting upon an object and the resulting outcome [2]. Specific examples might include *graspable* (e.g. [3]) or *pushable* [4] that indicate a particular object can be grasped or pushed, respectively. Because one can cast affordances as state-action pairs that will transform the object state in some way, there has been further work in considering affordance as a basis of planning [5]. If the robot has a goal of clearing the path to an object being fetched, it might first push interfering objects to the side assuming they can be pushed, i.e. have the affordance *pushable*.

However, while a planner may be able to leverage an abstracted description of the affordance as being true or not of an object, or even of having some probability of being true in the case of a probabilistic planner, such a high level description is not sufficient to actually *execute* the action required for the affordance. And, indeed the method of performing the action may vary by object or object state: pushing a round cereal bowl might be quite different than pushing a TV remote control that has rubber buttons that occasionally stick to a table surface.

Tucker Hermans, James M. Rehg, and Aaron F. Bobick are with the Center for Robotics and Intelligent Machines and The School of Interactive Computing, Georgia Institute of Technology, Atlanta, GA. {thermans, rehg, afb}@cc.gatech.edu

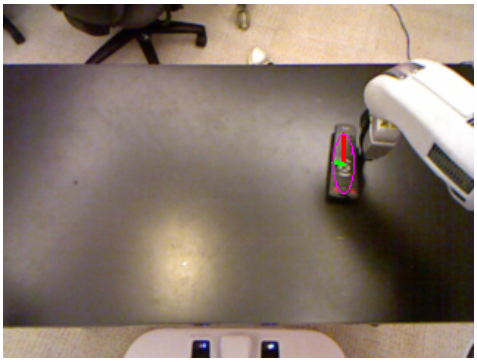

Fig. 1: Example of the robot performing pushing using feedback from visual tracking. The red line represents the dominant orientation of the object computed from the purple ellipse fit to the object's 3D point cloud.

The goal of this paper is to introduce a mechanism by which a robot can attach the general notion of an affordance to a specific method by which the robot can achieve it. We do this by decomposing an affordance-driven behavior into three components. We first define *controllers* that specify how to achieve a desired change in object state through changes in the agent's state. For each controller we develop at least one *behavior primitive* that determines how the controller outputs translate to specific movements of the agent. Additionally we provide at least one *perceptual proxy* that defines the representation of the object that is to be computed as input to the controller during execution. Obviously, the proxy must be sufficiently rich to support estimation of the variables required by the controller. The novelty here is that multiple proxies may support the same controller and a given proxy representation may be selected for use with more than one controller. Additionally, a single behavior primitive may be compatible with multiple controllers. Decoupling these components allows the systematic exploration of a variety of strategies when evaluating the affordances of novel objects.

In this paper we use as an example affordance *push-positionable* where the goal is to move the object to a specified location. We develop two feedback controllers to implement this action using the *overhead push* behavior primitive. Each of these controllers has its own perceptual proxy. These methods require no prior knowledge of the object being pushed and make no estimates of underlying model parameters. We show how a particular controller may succeed or fail on the basis of the proxy computed. Finally, we show how one of these controller-proxy pairs can be utilized by a second behavior primitive, a *sweep push*.

We organize the remainder of our paper as follows. Section II describes relevant past work on the topic of affordance learning and affordance-based planning; we also briefly mention prior methods of controller-based pushing. In Section III we formally define the affordance assertion problem and the push-positioning task. Then in Section IV we present two proposed feedback controllers each based upon a different perceptual proxy and each better suited to different object types. We give details of our implemented proxies in Section V followed by the implemented behavior primitives in Section VI. Section VII presents results of experiments performed on a robot using our proposed system. We conclude with directions for future work in Section VIII.

II. Related Work

In early work on affordance prediction described in [6, 7], a humanoid robot learns to segment objects through actions such as poking and prodding. After interaction with a set of objects, the system could learn the rollable affordance for the objects and predict the result of hand-object interactions. The goal was to learn parameters such as initial location of the hand with respect to the orientation of an object that best induce the desired motion. The actions were atomic in the sense that they were applied in their entirety and the results measured. In [8], a classification method is applied to high-level image features to learn the affordance of liftable. Using decision tree classifiers with SIFT and patch features, they demonstrate the ability to learn liftable vs nonliftable objects.

A series of works [9–11] address the task of recognizing the graspable and tappable affordances, based upon experimentation through self-observation of actions. Learning in a Bayesian network is employed to learn cuing rules for actions. The network models the relationship between object appearance and motion, end-effector motion, and action. In [12], a functional approach to affordance learning is developed in which subcategories of the graspable affordance (such as handle-graspable and sidewall-graspable) are learned by observation of human-object interactions. Interaction with specific object parts leads to the development of detectors for specific affordance cues (such as handles). The focus of that work was to learn a mapping from object features to grasp locations without unduly worrying about what method of grasping would work at that location.

Related, Stoytchev [13] describes a method for learning the functionality of a tool through observation of the effects of exploratory behaviors, a process that he termed behavioral babbling. In experiments with a mobile manipulator, the system demonstrated the ability to learn the affordances of a set of tools that could be identified by their color.

With respect to planning, affordance-based modeling of robot-object interaction would allow a planning system to systematically select from a set of actions to achieve desired subgoals. An example of such an approach is given in [5] where the robot arrange plates and bowls on a table. In that work, however, there is an assumption of a priori knowledge as to which behaviors can successfully operate on which objects and what the resulting state of the action will be. The approach presented here would both permit experimental exploration on the part of the robot of the different methods by which an affordance could be realized for a given object and a method for monitoring the effectiveness of the behaviors.

The concept of Instantiated State Transition Fragment (ISTF) is introduced in [14]. It encodes the pairing between an object and an action in the context of the state transition function for a domain-specific planner. The authors describe a process of learning Object Action Complexes (OACs) through generalization over ISTF's. Montesano et. al. [11] present a Bayesian network model that implicitly represents affordances as mappings from action to effect, which are mediated by the visual features of objects. A model for grasping, tapping, and touching actions is learned from both self-observation and imitation of a human teacher. The goal is to leverage such OACs in planning and executing a multi-step task.

Effective pushing behaviors offers a number of benefits in robotics domains which complement standard pick-and-place operations. For example pushing can be used to move objects too large for the robot to grasp, to more quickly move objects to new locations, or to move an object while another object is already grasped. As such there has been considerable work at developing such capability. Early work that analyzed a complete model of the dynamics of pushing was developed by Mason who describes the qualitative rotational changes of sliding rigid objects being pushed by either a single point or single line contact [15]; representative examples of some more recent applications of pushing are available in [4, 16–20].

Notably, Ruiz-Ugalde et al. execute a pushing behavior by determining the static and kinetic friction coefficients for multiple objects with rectangular footprints, both between the robot hand and object and between the object and table [20]. Additionally they present a robust controller using a cart model for the object being pushed. The control takes object velocity as input to control the system to a desired 2D pose, as such the mapping from applied force to velocity is believed known from the estimation and is separate from the control of the object. Their control is the closest approach we have found to the pushing controllers presented in this work. However, their overall approach presumes the ability to predict the resulting action based upon known or learned parameters that characterize the physics of the object.

To address the inherent difficulty in estimating model parameters, there are data-driven methods that use an empirically derived characterization of the outcomes of specific actions applied to the object. For example, Narasimhan uses vision to determine the pose of polygonal objects of known shape in the plane [21]. Three methods were proposed to be able to push objects into the desired location and orientation: a hand coded heuristic that assumes known center of mass (and uniform friction properties), a feedback controller to explicitly rotate and translate an object, and finally a data-driven, learning approach that stores the results of different pushes and uses nearest neighbor to select the action that generates a result closest to the desired outcome. where the

Fig. 2: Initial pose of the food box. The green circle represents the desired position and the green line is the current vector between the object origin and the goal.

states and results of different methods are examined.

Similarly, Salganicoff et al. present a method for learning and controlling the position in image space of a planar object pushed with a single point contact [22]. Slip of the object is avoided by pushing at a notch in the object. Scholz and Stilman learn object specific dynamics models for a set of object through experience [23]. Each object is pushed at a number of predefined points on the perimeter and the robot learns Gaussian models of displacement in (x, y, θ) at each location. These learned models are then used to select the input push location given a desired object pose.

III. Problem Statement

We define an affordance to exist between a robot and an object, if the robot can select a specific behavior primitive, controller, and perceptual proxy by which it can successfully perform the desired action. We take as an example action that of *push positioning*, where the robot must position an object at an arbitrary location by pushing with its arm. We assume that the object is being pushed over a plane and thus the object state $X = (x, y)$ defines the location of the origin of the object in a 2D space.[1] We denote the goal pose as $X^* = (x^*, y^*)$. This state representation is sufficient at the level of a task level planner, however, a specific controller may require more state variables to be estimated by the relevant perceptual proxy.

The (unknown) dynamics of the pushing system are governed by the nonlinear relation $\dot{X} = h(X, Q, U)$ which defines the interaction dynamics between the object state, the robot configuration Q, and the input to the robot U. Importantly, we make no attempt model h. In developing our visual feedback controllers to achieve the above defined task, we presume we do not have an exact measurement of the object state. Instead we will operate on the estimated state $\hat{X}$ that will be computed at each timestep based upon properties of a perceptual proxy. In this work we control the arm through Cartesian control, both position and velocity, in the robot's task frame. We denote the specific forms of U and X used in our controllers in detail below. Our task thus becomes defining a feedback control law $U = g(\hat{X}, X^*)$ which drives the position error $X_{err} = X^* - \hat{X}$ to zero.

[1]We wish to make clear, that we do not assume objects are flat.

IV. Two Push-Positioning Controllers

In this section we define two visual feedback controller for the robot to push an object to a desired location. Each controller has a necessary set of state variables to be estimated from the perceptual representation that is continuously updated. These representations serve as the proxies for the object with respect to the defined controllers.

A. Spin-Correction Control

Our first method of defining a push-positioning controller relies on the fact that the direction of an object's rotation while being pushed depends on which side of the center of rotation the applied force intersects. This fact is well described by the limit surface formulation [15, 24]. Mason derived the velocity direction of a sliding object as a function of the forces applied by the pushing robot as well as the support locations and mass distribution of the object [15]. These parameters are difficult to know or estimate well for a given object and even when they are known, the exact resulting behavior is often indeterminate [15]. However, we make use of Mason's realization that the resulting rotation of the object abruptly changes direction when the input force passes directly through the center of rotation of the object. As such we can use the direction of the observed rotation of the object to infer which side of the center of rotation the applied forces are currently acting through. We can then correct the direction of our applied forces to compensate for any unwanted rotation of the object.

Since objects tend to rotate less when the input forces as directed near the center of the object our controller attempts to push the object through its center in the direction of the goal position. This gives a simple procedure for determining the initial hand position. We cast a ray from the goal location through the centroid of the object and find its intersection with the far side of the object. This location defines the initial position for the hand. We further orient the hand so that its gripper is facing in the direction of the goal from the initial position. An example image of the initial hand placement can be seen in Figure 2. Once positioned our feedback control process is initiated. The controller is defined in equations 1 and 2 which operates on state $X = (x, y, \theta, \dot{\theta})$ and computes input U of x and y velocity of the end effector in the robot's workspace.

$$u_{\dot{x}} = k_g v_{goal_x} - sin(\phi_g)(v_{rot}) \quad (1)$$

$$u_{\dot{y}} = k_g v_{goal_y} + cos(\phi_g)(v_{rot}) \quad (2)$$

Our control is comprised of two terms. The first pushes through the object driving it to the desired goal, while the second displaces the contact location between the robot and object to compensate for changes in object orientation. The input control defined in equations 3 and 4 commands the robot to push in the direction of the goal. The overall effect of this component is controlled by the positive gain k_g. Since the object lies between the end effector and the goal this

causes the object to translate towards the goal.

$$v_{goal_x} = (x^* - \hat{x}) \tag{3}$$
$$v_{goal_y} = (y^* - \hat{y}) \tag{4}$$

However, since the forces applied by the robot on the object are not pushing directly through the center of rotation, the object will undoubtedly spin. To compensate for this we apply additional input velocities proportional to the observed rotational velocity of the object. We desire not only that the object not rotate, but also that it maintains its initial orientation θ_0. We combine these terms to generate v_{rot}.

$$v_{rot} = k_{sd}\dot{\hat{\theta}} - k_{sp}(\theta_0 - \hat{\theta}). \tag{5}$$

We desire to displace the end effector perpendicular to the current direction of the object's translational motion. Since our estimate of the instantaneous velocity is somewhat noisy, we instead rotate the velocity vector about the angle defined between the center of the object and the goal ϕ_g.

$$\phi_g = \texttt{atan2}(y^* - \hat{y}, x^* - \hat{x}) \tag{6}$$

Our pushing controllers halt once $x_{err} < \epsilon_x$ and $y_{err} < \epsilon_y$. For the purpose of developing this method as well as the controller in Section IV-B, the gains are manually adjusted, but remain fixed for all objects.

B. Centroid Alignment Control

Our second push-positioning controller replaces the monitoring of object orientation with a strategy based upon the relative locations of the object's centroid, the assumed location of the contact point on the end effector, and the goal position. The simple intuition is that pushing the object can be achieved by positioning the end effector at a location on the object boundary that intersects a line between the goal location and the object centroid.

The robot achieves this behavior by using a control law that includes a velocity term to move toward the goal and one that moves the end effector to the line defined through the goal centroid locations:

$$u_{\dot{x}} = k_{gc}v_{goal_x} + k_c v_{centroid_x} \tag{7}$$
$$u_{\dot{y}} = k_{gc}v_{goal_y} + k_c v_{centroid_y} \tag{8}$$

where v_{goal_x} and v_{goal_y} are as before. The second term provides the additional velocity term toward the goal-centroid line; $v_{centroid_x}$ and $v_{centroid_y}$ are components of perpendicular vector from the presumed end effector contact point to the goal-centroid line. The robot then pushes in the direction of the goal attempting to maintain this collinearity relation. This controller has the state $X = (x, y)$ and computes the same U as in Section IV-A. Additionally, the end effector is initially positioned relative to the object as above.

V. Object Proxies

The two above controllers have modest perceptual requirements. The orientation-velocity controller requires both the location of the object and its orientation whereas the centroid-driven one only requires position as defined by the object centroid. Here we describe the perceptual computations performed and the proxies that satisfy the requirements.

Fig. 3: The first image shows the overhead push behavior primitive pushing the television remote. The second image show the sweep push behavior primitive pushing the dinner bowl. Both objects have the estimated centroid location and ellipse overlayed.

We begin with a simple depth-based segmentation and tracking method that currently assumes only a single object resting on the sliding surface (a table) is in the scene. The input is the RGBD image of a Microsoft Kinect though in this simple implementation only the depth channel is used. We initialize the tracker by moving the robot's arms out of the view of the camera, capture the depth image and then use RANSAC [25] to find the dominant plane in the scene parallel to the ground plane. We then remove all points below the estimated table plane and cluster the remaining points. We filter out clusters with very few points and, because we're assuming only one object is on the table, we accept the cluster with most points as the object.[2] We compute the 3D centroid of the points in the cluster and use the x and y components as the object's location on the table.

Once initialized we track by performing the same procedure with the added step of removing points belonging to the robot from the scene. We project the robot model into the image frame using the forward kinematics of the robot and remove points from the point cloud coincident with the robot arm mask. Because of noise in measurements and other calibration issues points belonging to the robot can sometimes remain. To prevent the tracker from selecting any of these points as the current object we perform nearest neighbor matching between current cluster centroids and the previous object state, selecting the closest as the current object. We then estimate the object velocity using the previous estimate of the object state.

Computing the perceptual proxies needed for each of the controllers is straightforward given the tracker described above. For the centroid based control where the proxy is only the centroid of the object, we can immediately return the x and y values. For the orientation-velocity control we need a proxy that includes an estimate of object orientation, as well as its rotational velocity, with respect to the global robot frame. We fit a 2D ellipse [27] to the x and y values of all points in the object point cloud and use the orientation of the major axis of the ellipse as the objects orientation θ. The change in θ from one frame to the next is the

[2]We note that we [19] and others (e.g. [26]) have previously developed methods for singulating objects form each other by pushing actions.

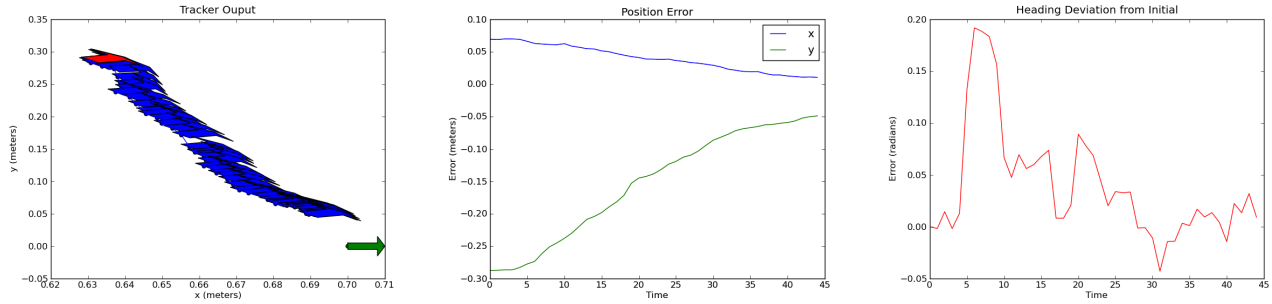

Fig. 4: The first image shows the tracked box pose trajectory. The red error shows the initial pose. The green arrow is the goal pose. The second image is error in the food box position and the third shows change in orientation from the initial orientation.

estimated orientation velocity $\dot{\theta}$. An example of the computed ellipsoidal proxy is shown in Figure 1.

Note that these two simple proxies — the first merely a centroid, the other the 2D ellipse — are intended to be available for *any controller* for which these inputs are sufficient. And, indeed a robot may have a variety of proxies that can yield a set of controller input variables. Later, when we discuss future work of learning the affordances of objects, we will return to this point.

VI. Pushing Behavior Primitives

We performed pushing with two behavior primitives: an overhead push and a sweep push. The overhead push has the robot place its hand such that the fingertips are in contact with the table with the wrist directly above. The sweep push places the length of the hand on the table with the flat of the hand facing the object. As our controllers operate only within the 2D pose of the hand (x, y, θ), the configuration of the end effector with respect to the arm and object remain fixed during operation. Specifically that means that the wrist remains above the hand throughout pushing for the overhead push. Likewise the sweep push keeps the long side of the robot hand along the table with the broad side of the hand perpendicular to the surface during pushing. Images of the robot operating with these behavior primitives can be seen in Figure 3.

For both primitives the arm is moved to the initial pushing pose using Cartesian position control. The arm is first moved to a position directly above the table at the desired pushing pose and desired orientation. The hand is then lowered in a straight line to the initial pushing pose. We use a Jacobian inverse controller to control the Cartesian velocity of the end effector during feedback control. We push objects with the robots right arm when the angle towards the goal pose move left in the workspace and use the left arm in the opposite case.

VII. Experimental Validation

We implemented our system on a Willow Garage PR2 robot augmented with a Microsoft Kinect for visual input. We experiment with different combinations of proxies, control laws, and behavior primitives in pushing a television remote, a food box, and a dinner bowl. In all experiments $\epsilon_x = \epsilon_y = 0.05$ meters.

(a) (b)

Fig. 5: The top and bottom of the television remote. The support distribution of the remote is much more complex than a simple polygon. Additionally the narrow end is significantly more massive than the wider end owing to the batteries inside.

A. Goal Position Controller Evaluation

We first show an example of pushing a television remote using the overhead push controlled by the spin correction controller. The perceptual proxy used is the ellipse model. The TV remote has a rather complicated set of support points and far from uniform mass or friction distributions. We show an up close picture of the remote in Figure 5. The tracked trajectory of the TV remote as well as the pose errors are shown in Figure 6. Midway through the pushing trajectory the remote becomes partially occluded by the robot arm which causes a jump in the estimated position. Figure 7 shows the controller compensating for this change which induces a larger velocity in the object, including its rotation. We show the velocities for the remote in Figure 8. Note that after the increased rotational velocity the controller most apply larger input velocities to maintain the initial orientation and continue pushing towards the goal. Regardless, the remote control converges within the desired bounds of the goal pose and the execution is successful.

We now show that this same affordance instantiation of overhead push, ellipse proxy, and spin correction controller

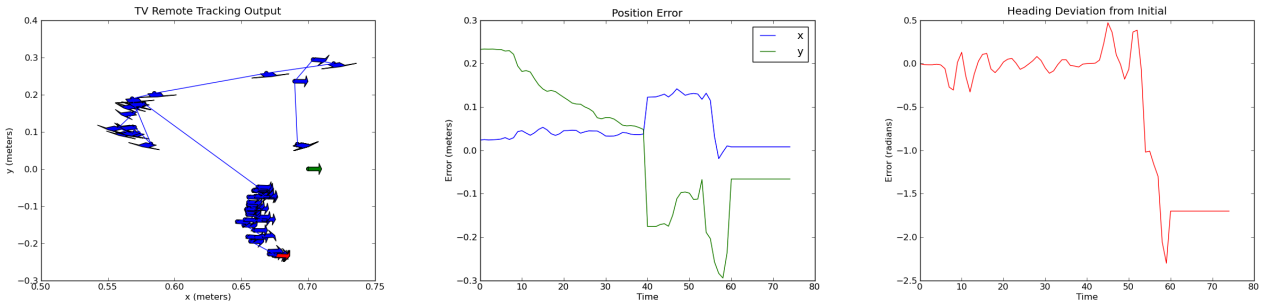

Fig. 6: The first image shows the tracked TV remote pose trajectory. The red error shows the initial pose. The green arrow is the goal pose. The large jump in error near time 40 is a result of the TV remote becoming partially occluded by the robot arm, which results in poor visual tracking performance.

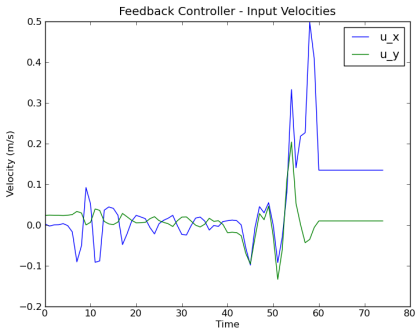

Fig. 7: Plot of the input velocities to the arm controller commanded by our feedback controller during pushing of the TV remote.

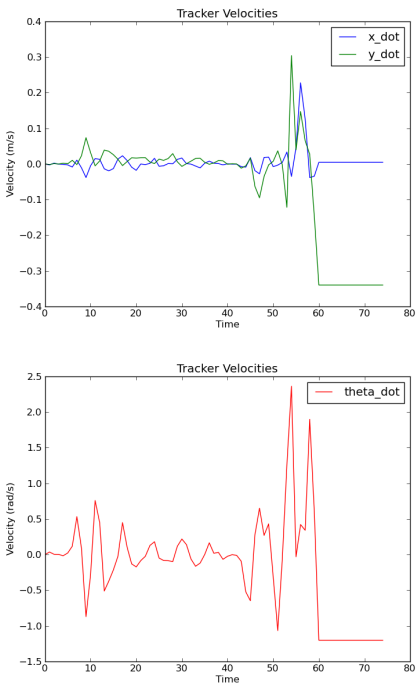

Fig. 8: Plot of the tracked object velocities of the TV remote.

can be used for a different object of a simple food box. The robot successfully pushed this box using the same components as with the television remote. The results for a trial with this setting are shown in Figure 4. The box can be seen in Figure 2.

We investigate the same manipulation settings of overhead push, ellipse proxy, and spin correction controller to a simple, white dinner bowl. We show the robot pushing this bowl in Figure 3. The position error for the bowl and the input velocities during control are shown in Figure 9. We show the tracker output and rotational velocity estimates of the bowl in Figure 10. Applying this method to the bowl fails to push the bowl to the desired location. This failure can be attributed to the symmetric appearance of the bowl, which causes instability in estimating the object's orientation by the ellipse perceptual proxy. However, by pushing the bowl with the overhead push controlled by the centroid controller the robot can correctly position the object. We show error results and input velocities for these settings in Figure 11.

Following the success of the centroid controller in pushing the bowl, we investigate its use with the overhead pushing behavior primitive to push the television remote. Unsurprisingly, the centroid controller quickly loses contact with the remote since the visually estimated centroid is not the center of rotation and trying to push in line with it fails to compensate for the object's rotation. Figure 12 shows position errors and input velocities from the experiments.

B. Behavior Primitive Evaluation

We now examine using the sweep push behavior primitive with the controller proxy pairs. Following the success of positioning the TV remote with the spin correction controller and overhead push, we tried the same setup with the sweep push behavior primitive. This performed quite poorly. Partially at fault was the occlusion of the remote by the arm causing unstable state estimates. Additionally the spin compensating control input, v_{rot} caused somewhat volatile control of the sweeping end effector, that was much smoother with the overhead push. This could have perhaps been fixed by changing controller gains, however, we did not investigate this.

We also examined pushing the bowl using the centroid controller and the sweep push. This method successfully positioned the bowl. We show the position error and input velocities in Figure 13. The error results and control velocities were quite similar to those seen in pushing with the

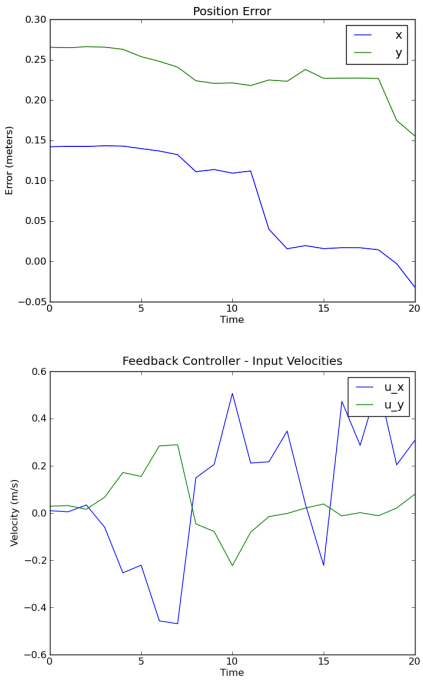

Fig. 9: Position error and input velocities for pushing the bowl using the spin correction controller with the overhead push.

overhead push.

VIII. Conclusions and Future Work

We have presented a novel mechanism by which a robot can connect the general notion of the affordances of an object to specific methods by which the robot can perform the necessary actions. We decompose affordance actions into behavior primitives, controllers, and perceptual proxies. This not only simplifies developing these capabilities but also allows a robot to systematically explore the affordances of objects. In the near future we will incorporate this approach into a learning paradigm where a robot not only learns the affordance of novel objects but also attempts to learn perceptual markers that will permit transfer of affordance knowledge between objects.

References

[1] J. J. Gibson, "The theory of affordances," in *Perceiving, Acting, and Knowing: Toward an Ecological Psychology*, R. Shaw and J. Bransford, Eds. Hillsdale, NJ: Lawrence Erlbaum, 1977, pp. 67–82.

[2] F. Iida, R. Pfeifer, and L. Steels, *Embodied Artificial Intelligence International Seminar, Dagstuhl Castle, Germany, July 7-11, 2003, Revised Papers*, ser. Lecture Notes in Computer Science. Springer, 2004, vol. 3139.

[3] A. N. Erkan, O. Kroemer, R. Detry, Y. Altun, J. Piater, and J. Peters, "Learning probabilistic discriminative models of grasp affordances under limited supervision," *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1586–1591, Oct. 2010.

[4] D. Katz, Y. Pyuro, and O. Brock, "Learning to Manipulate Articulated Objects in Unstructured Environments Using a Grounded Relational Representation," in *Proceedings of Robotics: Science and Systems IV*, Zurich, Switzerland, June 2008, pp. 254–261.

[5] J. Barry, K. Hsiao, L. P. Kaelbling, and T. Lozano-Perez, "Manipulation with multiple action types," in *International Symposium on Experimental Robotics*, no. 1122374, 2012, pp. 1–15.

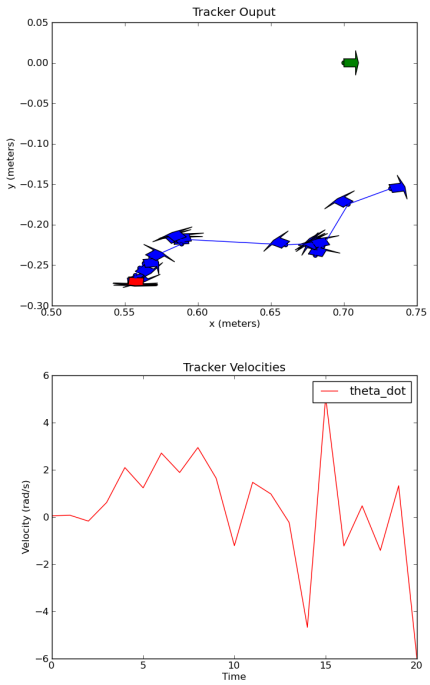

Fig. 10: Position and orientation estimates as well as rotational velocities estimates for pushing the bowl using the spin correction controller with the overhead push.

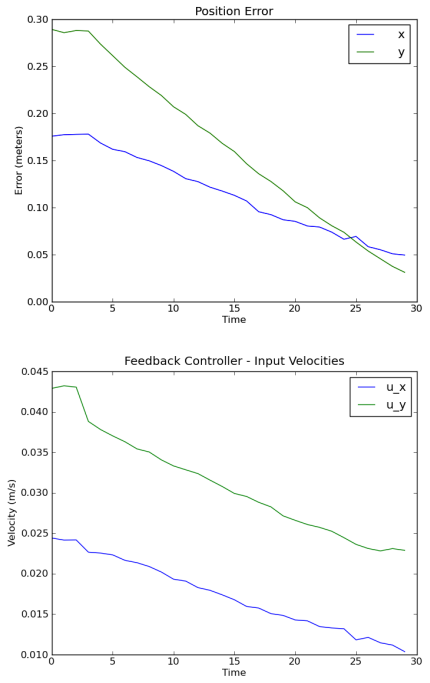

Fig. 11: Position error and input velocities for pushing the bowl using the centroid controller with the overhead push.

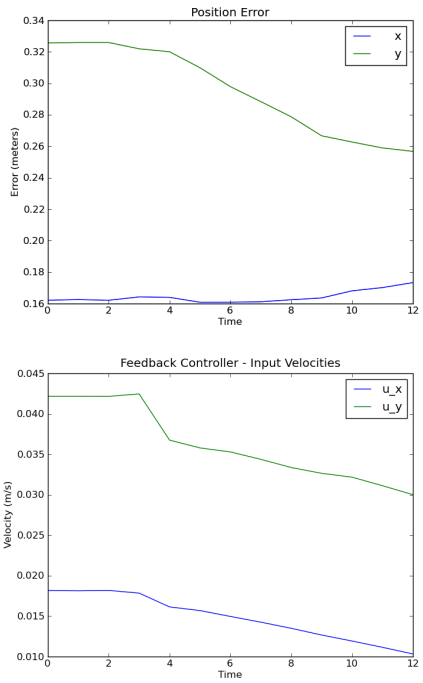

Fig. 12: Position error and input velocities for pushing the television remote using the centroid controller with the overhead push.

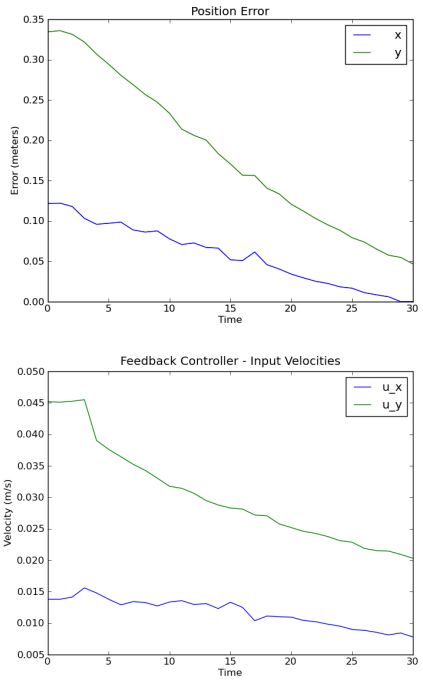

Fig. 13: Position error and input velocities for pushing the bowl using the centroid controller with the sweep push.

[6] P. Fitzpatrick, G. Metta, L. Natale, S. Rao, and G. Sandini, "Learning about objects through action - initial steps towards artificial cognition," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, vol. 3, Sept 2003, pp. 3140–3145.

[7] G. Metta and P. Fitzpatrick, "Early integration of vision and manipulation," *Adaptive Behavior*, vol. 11, no. 2, pp. 109–128, 2003, special Issue on Epigenetic Robotics.

[8] G. Fritz, L. Paletta, R. Breithaupt, E. Rome, and G. Dorffner, "Learning predictive features in affordance based robotic perception systems," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Beijing, China, Oct 2006, pp. 3642–3647.

[9] M. Lopes, F. S. Melo, and L. Montesano, "Affordance-based imitation learning in robots," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, San Diego, CA, 2007, pp. 1015–1021.

[10] L. Montesano, M. Lopes, A. Bernardino, and J. Santos-Victor, "Modeling object affordances using bayesian networks," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2007.

[11] ——, "Learning object affordances: From sensory-motor coordination to imitation," *IEEE Trans. on Robotics*, vol. 24, no. 1, pp. 15–26, Feb 2008.

[12] M. Stark, P. Lies, M. Zillich, J. Wyatt, and B. Schiele, "Functional object class detection based on learned affordance cues," in *Sixth Intl. Conf. on Computer Vision Systems (ICVS 08)*, Santorini, Greece, May 2008, pp. 435–444.

[13] A. Stoytchev, "Behavior-grounded representation of tool affordances," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2005.

[14] C. Geib, K. Mourao, R. Petrick, N. Pugeault, M. Steedman, N. Krueger, and F. W org otter, "Object Action Complexes as an Interface for Planning and Robot Control," in *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, Genova, Italy, Dec 4-6 2006.

[15] M. T. Mason, "Mechanics and Planning of Manipulator Pushing Operations," *The International Journal of Robotics Research (IJRR)*, vol. 5, pp. 53–71, September 1986.

[16] D. Omrčen, C. Böge, T. Asfour, A. Ude, and R. Dillmann, "Autonomous Acquisition of Pushing Actions to Support Object Grasping with a Humanoid Robot," in *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, Paris, France, 2009.

[17] M. Dogar and S. Srinivasa, "Push-Grasping with Dexterous Hands: Mechanics and a Method," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robotics and Systems (IROS)*, 2010.

[18] A. Cosgun, T. Hermans, V. Emeli, and M. Stilman, "Push Planning for Object Placement on Cluttered Table Surfaces," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robotics and Systems (IROS)*, 2011.

[19] T. Hermans, J. M. Rehg, and A. Bobick, "Guided Pushing for Object Singulation," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robotics and Systems (IROS)*, 2012.

[20] F. Ruiz-Ugalde, G. Cheng, and M. Beetz, "Fast Adaptation for Effect-aware Pushing," in *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, 2011.

[21] S. Narasimhan, "Task Level Strategies for Robots," Ph.D. dissertation, Massachusetts Institute of Technology, 1994.

[22] M. Salganicoff, G. Metta, A. Oddera, and G. Sandini, "A vision-based learning method for pushing manipulation," in *AAAI Fall Symposium on Machine Learning in Computer Vision*, 1993.

[23] J. Scholz and M. Stilman, "Combining Motion Planning and Optimization for Flexible Robot Manipulation," in *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, 2010.

[24] S. Goyal, A. Ruina, and J. Papadopoulos, "Limit Surface and Moment Function Descriptions of Planar Sliding," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 1989.

[25] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, pp. 381–395, June 1981.

[26] L. Y. Chang, J. R. Smith, and D. Fox, "Interactive Singulation of Objects from a Pile," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

[27] A. W. Fitzgibbon and R.B.Fisher, "A Buyers Guide to Conic Fitting," in *British Machine Vision Conference*, 1995, pp. 513–522.