A Flexible Hybrid Framework for Modeling Complex Manipulation Tasks

Oliver Kroemer, Jan Peters

***Abstract*—Future service robots will need to perform a wide range of tasks using various objects. In order to perform complex tasks, robots require a suitable internal representation of the task. We propose a hybrid framework for representing manipulation tasks, which combines continuous motion planning and discrete task-level planning. In addition, we use a mid-level planner to optimize individual actions according to the task plan. The proposed framework incorporates biologically-inspired concepts, such as affordances and motor primitives, in order to efficiently plan for manipulation tasks. The final framework is modular, can generalize well to different situations, and is straightforward to expand. Our demonstrations also show how the use of affordances and mid-level planning lead to improved performance.**

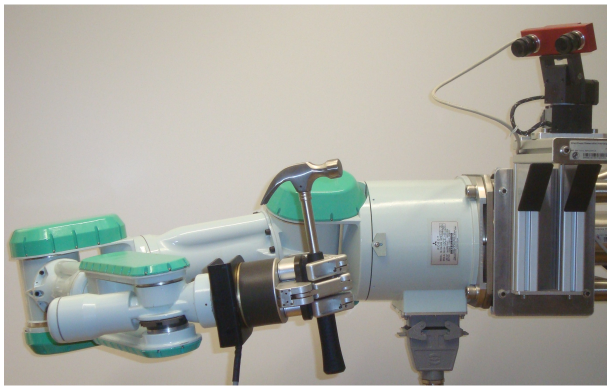
Figure 1. Robot with an optimal grasp of hammer for subsequent pushing and striking actions.

I. Introduction

Autonomous robots will need to deal with a wide range of different tasks and encounter many different objects in unstructured environments. In order to perform such complex tasks, robots must have a representation of the tasks that can be adapted to task variations autonomously.

Given a model of a task, a robot can begin to plan which actions to execute to achieve the specified goals. However, modeling arbitrary manipulation tasks is not trivial. A suitable modeling framework must be sufficiently flexible for representing a wide range of tasks, while simultaneously keeping the problem tractable. To handle new tasks and new objects it needs to be modular and straightforward to extend.

In this paper, we present a first attempt at a modeling framework for representing robot manipulation tasks. The proposed framework combines concepts from hybrid systems [9], [1] with insights into human manipulation and action sequence determination.

Neuroscience experiments indicate that humans appear to use forward models to predict the outcomes of their actions, and plans accordingly [14]. Models of manipulation tasks offer similar benefits for robots. However, rather than considering complex tasks as a single action, people apparently divide these tasks into discrete segments. Such segmenting of actions can be observed in the changing responses to stimulii during a manipulation task [13], [4], as well as on a neurological level[6]. We can model these discrete phases of continuous actions using a hybrid system framework [9], [1].

The concept of motor primitives also considers actions as being formed from a discrete library of elementary behaviors [5], [28], [11]. Such motor primitives are flexible actions that can be easily adapted to new situations. In this manner, only a few motor primitives are needed to perform a wide range of tasks. The proposed framework uses motor primitives to reduce the number of actions that need to be represented. The framework also incorporates a mid-level planner to optimize the motor primitive's hyperparameters according to the task plan.

The theory of affordances is another fundamental aspect of human manipulation skill[7], [23]. A central concept of this theory is that objects are classified by the actions that can be performed with them. For example, a sharp edge affords cutting, and a chair and a table both afford holding up objects. In this manner, humans can directly determine the resources available for performing actions and generalize their manipulation plans between similar objects. In this paper, we adopt an affordance-based representation of objects and their corresponding actions.

II. Modeling of Manipulation Tasks

The proposed framework uses a hierarchical structure to keep the planning problem tractable. In this section, we describe the proposed framework in a bottom-up manner. We begin by representing the objects involved in the task as tokens (see Section II-A). These tokens contain the parameters that define the objects and their state. The objects' actions and interactions are defined as continuous systems controlled by a small set of hyperparameters (see Section II-B). Each action of these actions requires a suitable set of affordance-bearing objects to be executed.

These action representations are incorporated into a planning framework in order to perform more complex tasks (see Section II-C). Given the structured nature of manipulation actions, we can define a network of possible transitions between actions. Once a task plan has been determined, we use a mid-level planner to optimize the actions' hyperparameters

Max Planck Institute, Spemannstraße 38, 72076 Tubingen
{oliverkro, jan.peters}@tuebingen.mpg.de

accordingly (see Section II-D). In Section III, we demonstrate a few key aspects of the framework on two manipulation examples.

A. Objects as Object Tokens

Before describing the manipulation of objects and their interaction, we must first define how the objects themselves are represented in our framework. Every object involved in a manipulation task is represented by an *object token*. An object token is an abstract representation of a single object, which defines the properties and state of the object. Using concepts from the theory of affordances, we structure these object tokens' properties according to the actions that the object affords.

The affordances of an object are indicated by the object's token being a member of a corresponding *affordance class*. For example, if an object can be rolled, then the corresponding object token is a member of the ROLLABLE affordance class. This affordance class contains the properties relevant to the affordance; e.g., the ROLLABLE affordance class includes the AXIS OF ROTATION and RADIUS OF ROTATION properties. The properties of an object are thus inherited from its affordance classes. Since most objects afford assorted actions, most tokens will be members of multiple affordance classes. The properties of the object are given by the union of the affordance classes' properties; i.e., if two affordance classes have the same property, then the object only has one instance of the property.

The use of affordance classes does not only link similar objects, but also creates a modular representation of the object's properties. This modularity allows many properties of an object to be defined, but only the few relevant to the current action to be employed. Each affordance class can also be associated with a set of characteristic visual features [22], [8].

The object tokens do not only represent the manipulated objects, but also the parts of the robot used to perform the manipulations. For example, a bimanual robot would have two tokens to represent its two arms and hands. Removing one arm and hand would remove one token.

These tokens represent the capabilities of the robot, and are therefore associated to *capability classes*. Capability classes have all properties of affordance classes but in addition also define *basic actions* such as move the hand up, down, left, and right. The robot has direct access to these basic actions. Examples of capability classes include STATIC ENDEFFECTOR, which can push objects, and 2-VIRTUAL-FINGERED HAND, which can additionally perform basic grasps. Affordance classes and object classes are collectively referred to as *object classes*.

The concept of object tokens was originally inspired by the use of tokens in Petri nets [2], [21]. In Petri nets, tokens do not only allow for an infinite state space, but represent the resources available within the system. Similarly, the proposed object tokens represent the available affordances as resources for performing manipulation tasks.

B. Actions and Interactions as Action Blocks

Having individually modeled objects, the next aim has to be representing higher level behaviors, actions, and the interactions between objects. In the proposed framework, each type of action is represented by an *action block*. Example action blocks include RESTING, PUSHING, ROLLING, and CARRYING. In order to perform an action, the robot must have access to objects that afford the actions. The affordances and capabilities available to the robot are given by the current set of object tokens. The affordances and capabilities required to perform an action are defined by the set of *object slots* of the action block. Each slot is associated with one object class, which defines the type of object needed to perform the action. In order to execute an action, each of its slots must be assigned exactly one object token. The token assigned to a slot must be a member of the object class associated with that slot. In this manner, only the objects that afford a certain action can be assigned to the corresponding action block. Action blocks can have one or multiple slots depending on the requirements of the action. By associating the slots to affordance classes, these symbolic classes become grounded and can even be tested for new objects [18], [3], [8].

Each action block also incorporates a *continuous system*, which defines how the properties of the assigned objects behave. The continuous system is a function mapping the current states and basic actions of the objects to a new set of states. This mapping is applied to the assigned objects at every time step. This mapping function is only defined for the states and objects associated with this action; i.e., we only include the properties and action of the object classes associated to the object's slots. Thus, we use of the object tokens' structure to focus on the relevant properties of the objects. The continuous system allows the state and basic actions of one object to affect the states of other objects. Object tokens must be assigned to all slots of an action block, or else the continuous system is ill-defined.

As the action blocks define the behaviors of the objects, all tokens must be assigned to an action. Objects not being manipulated will often be assigned to the RESTING block. The robot may generate any number of identical copies of an action block. The number of actions that can actually be performed in parallel is limited by the availability of the required object tokens.

If an action block contains one or more slots associated to capabilty classes, then the system incorporates basic actions. In order to define a higher level action, we must define a *policy* for these basic actions. A policy is a function mapping the current state of the continuous system to the execution of basic actions; i.e., they are a form of feedback law. By incorporating these policies into the continuous system, the resulting system becomes more autonomous. Assigning objects to an action block will thus result in the desired action on the objects.

Rather than specifying fixed policies, the policies can be defined with respect to a small set of hyperparameters. For example, a PUSH block's policy can be fixed to push an object 10cm forward. However, the uses of this action block would be extremely limited. A more efficient approach would be to define DISTANCE and DIRECTION hyperparameters, and adjust the policy accordingly. By using the parameterized policies, the robot can handle the same range of situations using fewer action blocks.

In biology, such flexible policies are known as *motor primitives* [5], [28], and represent building blocks of movement generation in animals. Motor primitives have also been studied for robot applications [12], [27]. These robot motor primitives generalize well to different situations and can be easily acquired from human demonstrations using imitation learning [11]. Individual motor primitives can also be learned and optimized to improve the performance of the action [16].

Once a motor primitive's policy has been defined, the continuous system can autonomously perform the action. The actions are therefore ready to be sequenced using a planner.

C. High-level Task Planning

To perform complex manipulation tasks, the robot must plan a suitable sequence of action blocks. In particular, it must determine how to reassign object tokens from one set of action blocks to another. Each block requires a set of entry and exit conditions in order to use them in a planning framework. The *entry conditions* define the states that objects are allowed to be in when assigned to the action block. Similarly, the *exit conditions* define the states that objects must be in when they are unassigned from the action block. These entry and exit conditions correspond to the pre- and post- conditions of actions used in planning frameworks [20]. A planner can generate plans by chaining together actions with overlapping entry and exit conditions, while generally disregarding the inner workings of the action blocks [17].

When an action begins or ends, it must do so for all objects involved. Therefore, a reassignment can only occur when the entry and exit conditions are fulfilled for all of the action block's slots. Objects must also always be assigned to an action block. Hence, a reassignment of tokens is only possible when both the current exit conditions and the subsequent entry conditions are met. However, these constraints do not require that all of the objects from one action block are reassigned to the same next action block. Therefore, the tokens from multiple blocks can be reassigned to a single block, and tokens from a single block can be reassigned to multiple blocks.

Given the entry and exit conditions, the action blocks could potentially be used for planning. However, manipulation tasks have underlying structures that can be exploited when planning. In particular, actions are often preceded by specific actions, and not all actions can be performed sequentially. For example, an object cannot be held without being grasped first. These underlying structures can be treated as partial plans, and allow for additional levels of abstraction in hierarchical planning [20].

In order to represent the structure of manipulation tasks, we utilize concepts from the hybrid systems literature. Hybrid systems can be modeled as a set of continuous systems connected together as the nodes of a discrete automatum [1]. Similar as inhybrid systems approaches, the object tokens perform discrete reassignments between the continuous systems of the action blocks. Therefore, we define a discrete network of allowed token reassignments between slots.

The *reassignment network* is defined as a set of nodes and edges. The nodes correspond to the slots of the various action blocks. An edge between two nodes indicates that a reassignment between the corresponding slots is valid. An edge may be directed if a reassignment is not reversible. The allowed reassignments of tokens also define the allowed transitions between action blocks. When action blocks are in series, without branching, they represent discrete phases of a larger action [4]. These serial actions do not increase the planning complexity any more than a single action would.

The network of action blocks will grow as the robot learns to perform new action and tasks. The additional complexity of a large network will usually result in slower planning. However, the size of the network can be reduced by only considering the parts of the network associated with the affordances of the objects that need to be manipulated. If the task cannot be accomplished with the reduced network, the network can be expanded by incorporating other available object as tools. In this manner, the robot can effectively generate a network tailored to the current task.

D. Mid-Level Planner

Planning tasks entirely at the level of basic actions is generally infeasible for complex tasks. Similarly, high-level task planning is also inefficient if vast numbers of actions are available. In order to reduce the number of discrete actions, we use motor primitives to create a small set of actions controlled by hyperparameters. Since we are using motorprimitives, we also incorporate a *mid-level planner*. The mid-level planner is responsible for optimizing the hyperparameters used by the action blocks' policies, according to the plan set by the task planner.

Selecting suitable hyperparameters for individual motor primitives has been previously studied [15], [19]. However, humans are known to adapt their actions depending on the following actions [6], [24]. We therefore propose applying similar approaches for optimizing the hyperparameters of entire sequences of motor primitives.

Selecting hyperparameters can generally be considered as an optimization or reinforcement learning problem [15], [16]. The main optimization is the minimization of the distance between the final state of the plan and the desired goal state. The by optimizing over hyperparameters, the optimization problem has a smaller dimensionality than when optimizing the entire low level trajectory. Additionally, some hyperparameters will not affect the final goal state and may be selected arbitrarily, thus further reducing the hyperparameter space.

In addition to optimizing the final goal state, individual actions can also be assigned performance measures and, thus, optimized. The performance measure of an action effects not only the optimal hyperparameters of that action, but also the hyper parameters of previous actions. An example of using performance measures for individual actions is given in Section III-B.

E. *Perspectives on Affordances*

In the proposed framework, the affordances of objects are treated as resources for performing actions. In this section, we explain how this approach incorporates the different perspectives of affordances. The three main interpretations of affordances are the environmental perspective, the agent perspective, and the observer perspective [26].

The environmental perspective treats affordances as a set of properties inherent to the objects being manipulated. Thus, a ball affords throwing. This view of affordances is incorporated by the affordance classes of the object tokens. The observer perspective considers pairs of agents and objects as having affordances. Thus, a ball and a robot arm together afford throwing. The slots of the action blocks define the required pairings between objects and agent capabilities. Hence, the proposed framework incorporate the observer's perspective of affordances. The agent perspective is similar to the observer perspective, but only incorporates the affordances specific to that agent. Thus, for a robot arm, a ball affords throwing. These agent-specific affordances are given by the action blocks that can be used with the agent's capability tokens.

The proposed framework also models special cases of these perspectives. One special case of the observer perspective is when the observer has similar capabilities to the observed agent; e.g. a humanoid robot observing a human, or a child observing an adult. Due to their similar capability tokens, it is straightforward to infer that the actions afforded to the acting agent are also afforded to the observer. This special case of the observer perspective is the bases of imitation learning and programming-by-demonstration [11]. In neuroscience, this learning ability is generally attributed to mirror neurons **[25]**.

The agent perspective can also be expanded to incorporate multiple agents. An agent may have insufficient capability tokens to perform certain actions; e.g., lifting a heavy object. However, a group of agents can pool their capability tokens together in order to perform these more demanding actions. Thus, the affordances apply to a group of agents rather than individual agents. This situation is important for scenarios wherein robots must cooperate to efficiently perform a task.

III. Example Manipulation Tasks

In this section, we demonstrate key concepts of the proposed framework through two exemplary case studies. The first example focuses on the use of tokens to represent available affordances. The second one demonstrates the use of the mid-level planner in the context of grasping objects.

A. *Rolling and Tumbling*

In this experiment, we demonstrate how the proposed framework incorporates the concepts of affordances as resources. The robot's task is to move heavy objects in a controlled manner. Some objects, such as cylinders, can be rolled. The rolling action can be performed with a single motor primitive. It is therefore represented in Fig. 2 as a single action block $\mathbf{D}_1$ with one slot for the hand and another for the object being rolled. Other objects, such as rectangular boxes, must be tumbled using two hands. This tumbling action involves

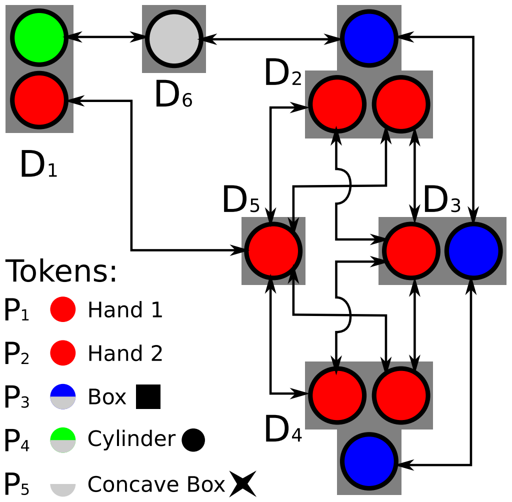

Figure 2. The diagram shows the tumbling and rolling system. Each of the dark gray blocks represents an action block, with the index given next to it. The circles in the blocks represent their slots. The colors of the slots indicate the class they are associated with. The arrows show the allowed reassignments of tokens between slots. In the bottom left is the list of tokens. The colored circles next to the token names mark the classes that the objects are members of. The concave box $\mathbf{P}_5$ does not have a complete circle as its membership to the ROLLING (green) and the TUMBLING (blue) classes is initially unknown. $\mathbf{D}_1$ is ROLLING, $\mathbf{D}_2$ to $\mathbf{D}_4$ are TUMBLING, $\mathbf{D}_5$ REACHING, and $\mathbf{D}_6$ is RESTING.

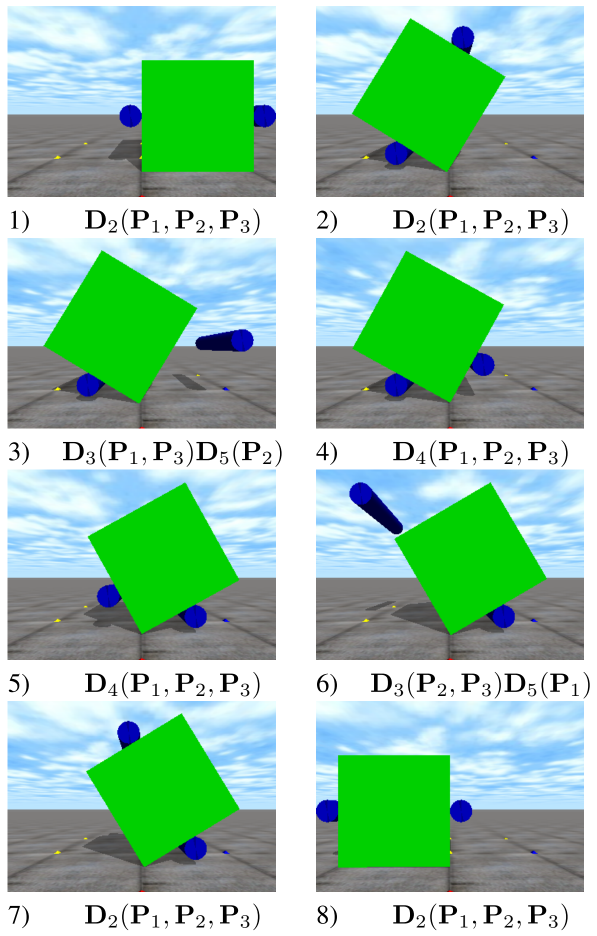
1) $\mathbf{D}_2(\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3)$
2) $\mathbf{D}_2(\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3)$
3) $\mathbf{D}_3(\mathbf{P}_1, \mathbf{P}_3)\mathbf{D}_5(\mathbf{P}_2)$
4) $\mathbf{D}_4(\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3)$
5) $\mathbf{D}_4(\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3)$
6) $\mathbf{D}_3(\mathbf{P}_2, \mathbf{P}_3)\mathbf{D}_5(\mathbf{P}_1)$
7) $\mathbf{D}_2(\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3)$
8) $\mathbf{D}_2(\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3)$

Figure 3. The images show the system in different stages of the two-handed tumbling procedure. The two blue cylinders are the hands $\mathbf{P}_1$ and $\mathbf{P}_2$. The green object is the box $\mathbf{P}_3$. Under each image is the step number, and the allocation of tokens to the continuous systems.

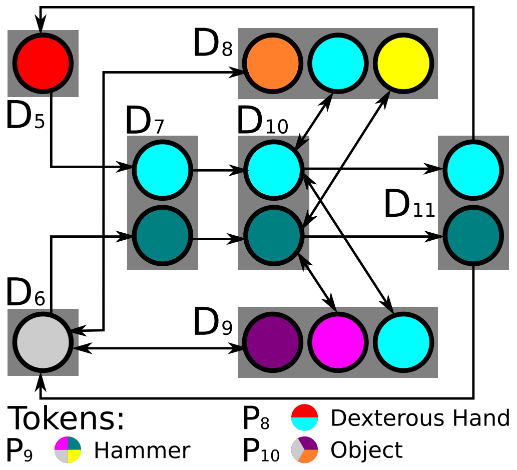

Figure 4. The diagram shows the pushing and striking system. Each of the dark gray blocks represents a continuous system, with the index given next to it. The circles in the blocks represent their slots. The colors of the slots indicate the class they are associated with. The arrows show the allowed reassignments of tokens between slots. At the bottom is the list of tokens. The colored circles next to the token names mark the classes that the objects are members of. $\mathbf{D}_5$ and $\mathbf{D}_6$ are identical to those in Fig. 2. $\mathbf{D}_7$ is GRASPING. $\mathbf{D}_8$ is STRIKING. $\mathbf{D}_9$ is PUSHING. $\mathbf{D}_{10}$ is CARRYING. $\mathbf{D}_{11}$ is RELEASING.

tipping the box onto an edge, repositioning the hands, and then tipping the box over completely. An example of this procedure is shown in Fig. 3.

The reassignment network of this model is visualized in Fig. 2, including the allowed reassignments of tokens. The tumbling action is divided into three action blocks. Action block $\mathbf{D}_2$ represents tipping the box on its edge using two hands on opposite sides of the box. Action block $\mathbf{D}_4$ represents a similar tipping movement, but with the hands on adjacent sides of the box. Finally, action block $\mathbf{D}_3$ represents tilting the box with one hand on a side adjacent to the pivotal edge. Action block $\mathbf{D}_3$ has a slot for the hand, and another for the tumbling object. Blocks $\mathbf{D}_2$ and $\mathbf{D}_4$ have additional slots for the second hand. The RESTING action is given by action block $\mathbf{D}_6$, and a curved reaching movement is given by action block $\mathbf{D}_5$. This figure also shows the tokens used in the initial setup, including a box $\mathbf{P}_3$ and a cylindrical object $\mathbf{P}_4$. The box with concave sides $\mathbf{P}_5$ is novel, and its membership to the ROLLING and TUMBLING affordance classes is unknown.

A key contribution of this paper is the concept of robot capabilities as a complement to object affordances. We will therefore first demonstrate how the affordances of objects change depending on the object capabilities. Given only one hand token, the robot cannot access the tumbling actions even if a suitable box is available. Thus, the boxes do not afford any interaction for the one-handed robot. However, if we supply the robot with a second hand-arm system, then the number of hand tokens is increased to two. From the perspective of this bimanual robot, the box affords being tumbled, as described by Fig. 3. If we further increase the available number of hands to four, we can tumble two boxes simultaneously. Alternatively, the robot can use the four hands to tumble the box faster. Rather than repositioning the hands on the box with action blocks $\mathbf{D}_3$ and $\mathbf{D}_4$, the robot can tilt the box using $\mathbf{D}_2$ and switch between the pair of hands holding the box. This

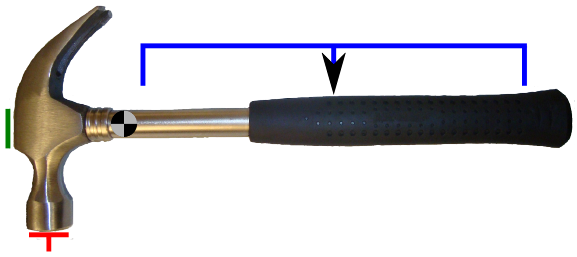
Figure 5. The hammer $\mathbf{P}_9$ used in the experiment. The **blue** marking defines the part of the handle that affords grasping. The **red** marking defines the impact point used for striking objects. The **green** marking is the region of the hammer's head used for pushing objects. The **black and gray** circle indicates the center of mass.

exchanging of hands is similar to jumping from stage 2 to stage 7 in Fig.3.

The robot can also use its knowledge of classes to label new objects, such as the box with convex sides $\mathbf{P}_5$. By morphing the geometry of the labeled objects $\mathbf{P}_3$ and $\mathbf{P}_4$ to the new object $\mathbf{P}_5$ [10], we can predict suitable hand placements for the new object. The affordances can then be tested by applying the corresponding actions. When tested, $\mathbf{P}_5$ will not roll, but can be tumbled. Thus, the robot can determine the affordances of new objects.

B. Task-specific Grasping

In this demonstration, we focus on using the mid-level planner to determine the optimal grasps of a hammer, given different tasks. While most objects afford a range of grasps, only a few may be suitable for the subsequent actions. The system of action blocks is shown in Fig. 4, wherein $\mathbf{D}_5$ and $\mathbf{D}_6$ are the same as in Fig. 2. The relevant actions for this experiment are grasping $\mathbf{D}_7$, striking $\mathbf{D}_8$, and pushing $\mathbf{D}_9$. The dexterous hand is a member of two classes, since it has additional grasping capabilities.

In this experiment, we use the hammer $\mathbf{P}_9$ shown in Fig. 5. The striking action is a standard hammering movement. The corresponding hyperparameters include the target point to strike. The pushing action uses the head of the hammer to push the object in a straight line, with variable initial and final states. The grasping policy consists of a reaching motion and a closing of the hand. The graspable points of an object define the exit conditions of the grasping action. The grasping action's hyperparameters define the grasp location. The hammer affords being grasped along the length of its handle, as shown by Fig. 5 in blue.

The performance measure of the grasping action is linear in the amount of contact between the hand and the object that is compliant. The striking action has a performance that is quadratic in the final velocity of the tool's impact point (see Fig. 5), minus a linear term for the final velocity of the hand. The performance measure of the pushing reward function is quadratic in the torque applied to the hand, multiplied by the distance pushed.

The system has been tried out with three different plans following grasping: pushing only, striking only, and pushing then striking. The goal state of the object has been varied

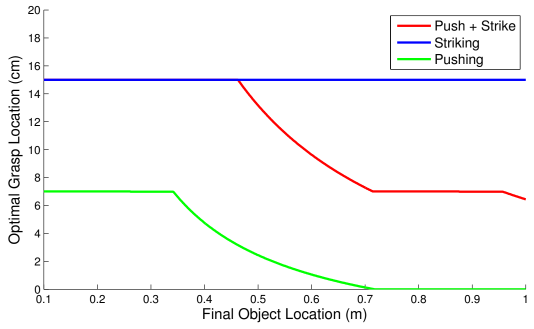

Figure 6. The plot shows how the optimal grasp of the hammer varies on the following actions. As the final object location increases, the object had to be pushed further from its initial 0.1m position. The length of the handle limits the grasp location to 15cm.

between 0.1m and 1m from the origin. When the plan involved pushing, the object started at a distance of 0.1m from the origin and had to be pushed to the final location. When only striking, the object started at the final location. Due to the low dimensionality of the hyperparameter space, we could perform the optimization by directly sampling from the space. More complex systems may require more advanced methods.

The results are shown in Fig. 6. The results show that the hammer is held closer to its center of gravity when used for pushing, and closer to the end of the handle for striking. The optimal grip depends on how far the object will be pushed. When multiple actions are performed, the grasp is automatically selected to suit the entire set of actions. Thus, the system adapts its grasps depending on the subsequent actions. The rewards can be easily transferred to other objects with the same class memberships. This ability to adapt grasps is only possible because of the flexibility of the motor primitive policies and the mid-level planner.

IV. Conclusion

The framework proposed in this paper represents manipulation tasks in a modular manner. Objects are represented as interchangeable tokens, which allows actions to be compactly represented as continuous systems. These actions form the foundation of an hierarchical, and hybrid, representation of tasks. By representing objects as tokens, the resources available for manipulating are straightforward to define. The experiments indicate that the framework may be applicable to assorted manipulation tasks and adaptive to variations in the task.

References

[1] Michael S. Branicky. Introduction to hybrid systems. In Dimitrios Hristu-Varsakelis and William S. Levine, editors, *Handbook of Networked and Embedded Control Systems*, pages 91–116. Birkhäuser, 2005.

[2] Christos G. Cassandras and Stephane Lafortune. *Introduction to Discrete Event Systems*. Springer-Verlag New York, Inc., NJ, USA, 2006.

[3] Paul Fitzpatrick, Giogio Metta, Lorenzo Natale, Sajit Rao, Giulio Sandini, and Giulio S. Learning about objects through action - initial steps towards artificial cognition. In *In Proceedings of the 2003 IEEE International Conference on Robotics and Automation*, pages 3140–3145, 2003.

[4] J. R. Flanagan, M. C. Bowman, and R. S. Johansson. Control strategies in object manipulation tasks. *Curr Opin Neurobiol*, 16(6):650–659, December 2006.

[5] Tamar Flash and Binyamin Hochner. Motor primitives in vertebrates and invertebrates. *Current Opinion in Neurobiology*, 15(6):660 – 666, 2005. Motor sytems / Neurobiology of behaviour.

[6] Leonardo Fogassi, Pier Francesco Ferrari, Benno Gesierich, Stefano Rozzi, Fabian Chersi, and Giacomo Rizzolatti. Parietal Lobe: From Action Organization to Intention Understanding. *Science*, 308(5722):662–667, 2005.

[7] James J. Gibson. *The Ecological Approach To Visual Perception*. Lawrence Erlbaum Associates, new edition edition, September 1986.

[8] Shane Griffith, Jivko Sinapov, Matthew Miller, and Alexander Stoytchev. Toward interactive learning of object categories by a robot: A case study with container and non-container objects. In *Proc. IEEE 8th International Conference on Development and Learning*, pages 1–6, Washington, DC, USA, 2009. IEEE Computer Society.

[9] T. A. Henzinger. The theory of hybrid automata. In *LICS '96: Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science*, page 278, Washington, DC, USA, 1996. IEEE Computer Society.

[10] Ulrich Hillenbrand. Non-parametric 3d shape warping. In *Proc. International Conference on Pattern Recognition*, 2010.

[11] A. Ijspeert, J. Nakanishi, and S. Schaal. learning attractor landscapes for learning motor primitives. In *advances in neural information processing systems 15*, pages 1547–1554. cambridge, ma: mit press, 2003.

[12] A. Ijspeert, J. Nakanishi, and S. Schaal. learning attractor landscapes for learning motor primitives. In *advances in neural information processing systems 15*, pages 1547–1554. cambridge, ma: mit press, 2003.

[13] Roland S. Johansson and J. Randall Flanagan. Coding and use of tactile signals from the fingertips in object manipulation tasks. *Nat Rev Neurosci*, 10(5):345–359, 2009.

[14] Mitsuo Kawato. Internal models for motor control and trajectory planning. *Current Opinion in Neurobiology*, 9(6):718–727, 1999.

[15] J. Kober, E. Oztop, and J. Peters. reinforcement learning to adjust robot movements to new situations. In *proceedings of robotics: science and systems (r:ss)*, 2010.

[16] J. Kober and J. Peters. practical algorithms for motor primitive learning in robotics. (2):55–62, 2010.

[17] George Konidaris and Andrew Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 1015–1023. 2009.

[18] Dirk Kraft, Nicolas Pugeault, Emre Baseski, Mila Popovic, Danica Kragic, Sinan Kalkan, Florentin Wörgötter, and Norbert Krüger. Birth of the object: Detection of objectness and extraction of object shape through object-action complexes. *I. J. Humanoid Robotics*, 5(2):247–265, 2008.

[19] O. Kroemer, R. Detry, J. Piater, and J. Peters. combining active learning and reactive control for robot grasping. (9):1105–1116, 2010.

[20] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[21] Drighiciu Mircea, Manolea Gheorghe, Petrisor Anca, and Popescu Marius. On hybrid systems modeling with petri nets. In *Proc. international conference on System science and simulation in engineering*, pages 73–78, Stevens Point, Wisconsin, USA, 2008. World Scientific and Engineering Academy and Society.

[22] Luis Montesano and Manuel Lopes. Learning grasping affordances from local visual descriptors. In *Proc. IEEE 8th International Conference on Development and Learning*, pages 1–6, Washington, DC, USA, 2009. IEEE Computer Society.

[23] Erhan Oztop, Hiroshi Imamizu, Gordon Cheng, and Mitsuo Kawato. A computational model of anterior intraparietal (aip) neurons. *Neurocomputing*, 69(10-12):1354–1361, 2006.

[24] Jennifer Randerath, Yong Li, Georg Goldenberg, and Joachim Hermsdörfer. Grasping tools: Effects of task and apraxia. *Neuropsychologia*, October 2008.

[25] Giacomo Rizzolatti and Laila Craighero. The mirror-neuron system. *Annual Review of Neuroscience*, 27:169–192, 2004.

[26] Erol Sahin, Maya Cakmak, Mehmet R. Dogar, Emre Ugur, and Gokturk Ucoluk. To Afford or Not to Afford: A New Formalization of Affordances Toward Affordance-Based Robot Control. *Adaptive Behavior*, 15(4):447–472, December 2007.

[27] S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert. learning movement primitives. In *Proc. of International Symposium on Robotics Research*. Springer, 2004.

[28] K. A. Thoroughman and R. Shadmehr. Learning of action through adaptive combination of motor primitives. *Nature*, 407(6805):742–7, |2000|.