Learning Generalizable Robot Manipulation Skills with Object Hierarchy

Rafaël Beckers
Supervisors: Zlatan Ajanović, Jens Kober
TU Delft, department of Cognitive Robotics

***Abstract*—Previous work has shown that state abstraction can be an efficient way to plan in robotic environments with continuous actions and long task horizons. Although some of these works learn predicates for state abstractions, they often neglect an important part needed for generalization. Namely objects and their properties, our work shows that object properties can be exploited to increase the generalizability of learned models. We do this by extending a framework based on predicates for state abstraction and introducing *affordances*. Affordances allow us to group objects in a way that also considers possible actions. In order to learn affordance interpretations we make use of querying. First, the agent is tasked with solving a task made to test a specific affordance, then based on its uncertainty about an object's affordance interpretation it may query an expert. For example, when presented with a task where it needs to pick up an object it may query the expert on whether the object is actually grippable. We compare our approach with a baseline without affordances and show that our approach needs fewer operators to plan and that our approach is able to generalize to novel objects.**

I. Introduction

The work described in this paper is a continuation of the work performed in [1], which itself is an extension of the Bilevel planning series of papers [2], [3], [4]. This series of papers focuses on learning all aspects of STRIPS-style symbolic planning. Where it is currently possible to learn entire STRIPS style operators, predicate groundings, as well as samplers used by a controller. The assumption is still made that crude controllers are supplied as well as a deterministic simulator. We however found that object similarities are under-exploited and that a higher degree of generalization can be achieved by exploiting similarities between objects. We propose to extend the approach found in [1] with affordances. In our approach, affordances are used to group objects to be able to plan with these groups instead of object instances. We also combine STRIPS style operators based on affordances and symbolic effects in order to plan more efficiently. We show that our approach indeed needs fewer operators to be able to solve a task and that our approach allows for easy integration with novel objects. In this report, we will first give a brief introduction to Task and motion planning (TAMP) and the concept of affordances. Afterward, we will describe our problem formally in Section II, and the method will be explained in Section III. Experimental details are provided in Section IV, and finally the results are displayed and discussed in Section V

A. Robot Task and Motion Planning Problem

In order for robots to be deployable in unstructured environments such as homes, restaurants, and hospitals they need to know what actions to take in order to achieve their goal. These goals are often highly abstract in nature. For example, tasking a robot to retrieve an item from a drawer. Such tasks are not easy to solve using motion planning alone, as the robot will require different behaviors for moving to the drawer, opening it, and picking the object from the drawer. This problem led to the invention of Task And Motion Planning (TAMP). In TAMP task planning and motion planning are integrated with each other. Task planning is done in a discretized space often through the use of symbols that map to a collection of states. These symbols can then be used in STRIPS-style planning to go from an initial abstract space to an abstract goal state. Such a sequence of abstract states can be thought of as a task plan or plan skeleton, this plan then needs to be refined into low-level continuous states in order to create a full plan. This refinement however is now made much easier as the planner needs to only consider the states within the symbols. In short, the abstract/symbolic plan serves as a guide for the continuous plan. A visual example of such an approach is displayed in Fig. 1.

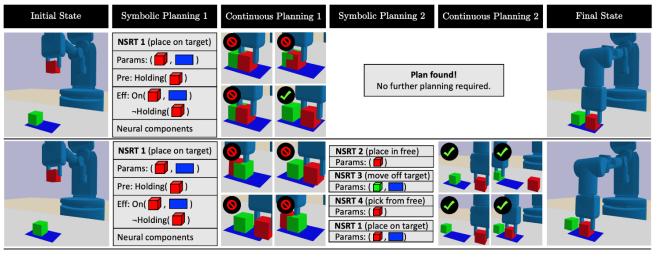

Fig. 1: Overview of the TAMP approach used in [2], note NSRTs are STRIPS operators with neural components such as samplers.

We also make use of TAMP style planning where the abstract space is induced through a set of predicates, previous work has shown that these predicates can be learned [5], however, we keep to a given set of predicates for our experiment. Since state abstractions using predicates are often lossy we decided to introduce the concept of *affordances* to TAMP. In robotics, affordances are defined as a tuple containing: (*object, action, effect*). To give an example say a robot can pick up a cup, and can also pick up a bottle then these two objects are

said to share the pickable affordance. This is because picking up these objects both resulted in them being picked up. By utilizing affordances together with TAMP it becomes easier to plan with object properties in mind, as during planning in the abstract space it is often beneficial to also consider an object's affordances. For example, say we want a robot to clean a table and it has access to two objects a bottle and a dishcloth it needs to figure out which of the two objects to use. Now provide the robot with the object's affordances and it becomes much more clear which object it should use.

II. Problem statement

In this section, we will formalize our problem and give some details about our problem setting in general in Section II-A, and afterward give more details on predicates and affordances in Section II-B.

A. General problem

We consider a fully observable environment with deterministic transitions in which a robot must solve a manipulation task. A state $x \in \mathcal{X}$ is defined by a set of objects $\mathcal{O}$ and a feature vector for each object. The size of these feature vectors is defined by the object's *type* $\lambda \in \Lambda$. An object's type can be thought of as an object class, a type can additionally contain multiple parent types. Thus a type λ is a tuple containing name, feature vector, and parent types. In order to solve the task the agent must take an action $u \in \mathcal{U}$ which is a controller with discrete and continuous parameters. We also have access to a discrete physics-based simulator $f : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ that predicts the next state given a state and action. The simulator is also available to the agent and can be used for planning.

B. Predicates and affordances

A *predicate* ψ is characterized by a name e.g. `Holding` and a tuple of types. A predicate also contains a classifier $c_\psi : \mathcal{X} \times \mathcal{O} \to \{true, false\}$. Predicates can be lifted or grounded, in the case of a lifted predicate the predicate is defined for typed variables and in the case of a ground atom the predicate is defined for specific objects. For example `Holding(block1)` is a ground atom and `Holding(?block)` is a lifted atom. An *affordance* θ is very similar to a predicate in that they can both be grounded or lifted and they also contain a binary classifier: $c_\theta : \mathcal{X} \times \mathcal{O} \to \{true, false\}$. In addition to this, an affordance also contains a set of $controllers$ which is defined as a low-level controller that can take in continuous parameters. Predicates can be defined for an arbitrary number of objects larger than one, affordances are always defined for one object. The reason for this is that affordances describe what actions can be applied to a specific object.

III. Method

Our method focuses on the reuse of learned concepts such as STRIPS operators. In this section, the method that was used to achieve higher reusability of concepts will be explained. First, the different phases of our approach will be discussed in Section III-A, afterward the bilevel planning/TAMP approach will be discussed in more detail in Section III-B. Lastly, we will discuss how and why we learn affordances, as well as how learn to group objects and STRIPS operators in Section III-C and Section III-D respectively.

A. Approach outline

As can be seen in Fig. 2 the agent is first initialized in an environment where it is provided with demonstrations which consist of a *task* and a *plan*. A task contains an initial state $x_0 \in \mathcal{X}$ and a goal g. The goal is a set of ground atoms that are satisfied in state x if $c^*_\psi(x) = true$ for all ground atoms $\psi \in g$, where c^*_ψ is the oracle's classifier for ψ. The agent also has access to a set of grounded examples for the predicates ψ and affordances θ it needs to learn, where the responses l_ψ and l_θ contain the truth values of the grounded predicates and affordances respectively.

The *Exploration* phase follows the initialization phase in which the agent is initialized in an environment where it can take action. According to an exploration policy π_e the agent will select an action and alter the state. The agent can also query an oracle that has ground truth information of predicate and affordance classifiers for seen and unseen objects. In our experiments, the oracle serves as a stand-in for a human expert. The queries $\mathcal{Q}$ made to the oracle can be in the form of ground atom queries or ground affordance queries. Where the response is a set of the form: (ψ, l_ψ) or (θ, l_θ) in the case of a ground affordance query. The Dataset $\mathcal{D}$ will be updated with each response and attached state, thus $\mathcal{D} \leftarrow \mathcal{D} \cup (x, \psi, l_\psi, \theta, l_\theta)$. The updated dataset will then be used for learning affordance and predicate classifiers, as well as operators, samplers, and object classes. In order to judge the agent's performance it is periodically tested on held-out tasks which include unseen states or entirely novel objects.

B. Learning abstractions for bilevel planning

Bilevel planning is a variation of TAMP where planning is performed in an abstract space and in a continuous state. Where the main idea is to plan in an abstract space first to guide the search in the continuous space. In [1] the authors state two key takeaways these being: predicates induce abstract states, and abstract actions provide transitions between abstract states. Thus we can create an abstract state by applying a set of predicates Ψ to a state x. A formal definition of this is provided in (1) adapted from [1].

$$s(x, \Psi) := \{\psi : c_\psi(x) = true\} \tag{1}$$

The s in (1) is the abstract state and can be described as a set of ground atoms that are either true or false. It should be mentioned that the abstract state tends to lose information, this is due to the lossy nature of the predicates themselves [5]. An abstract action is characterized by an *operator* and a *sampler*. *Operators* contain arguments, preconditions, add effects, delete effects and a controller. Preconditions are sets

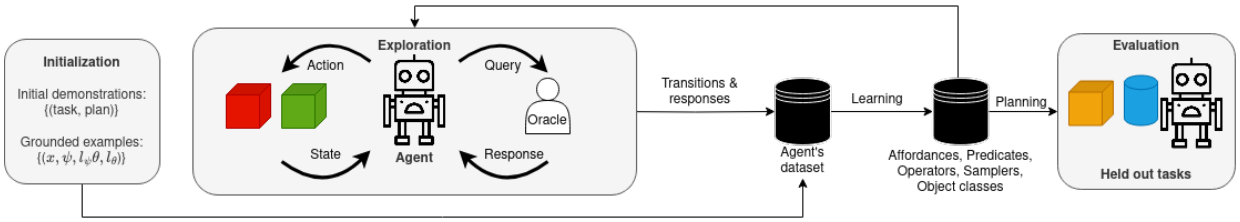

Fig. 2: Approach overview, first the agent is initialized with a small number of expert demonstrations from which it can learn affordances, predicates, operators, samplers, and object classes. It then uses these learned models to explore its environment and query an oracle through which it acquires additional data, improving the models. Expansion on the approach taken in [1]

of lifted atoms over the arguments that must be true for the operator to be applicable. Add effects are lifted atoms that will become true after applying the operator. The inverse is true for the delete effects. The controller serves to connect the abstract action to the action space in the environment, such as joint angles or position, the discrete parameters of the controller are provided by the operator itself. *Samplers* provide a way to fill in the continuous parameters of the controllers such as grasping pose. With these two key concepts an abstract state space s can be created and planned using abstract actions which are created by *grounding* the operators by substituting their arguments for objects of the correct type. The objects are then passed to the sampler as well as the continuous state x. The ground operators can then be used to generate low-level actions through the values returned by the sampler. An abstract plan can thus be created by applying the ground operators to the abstract state: $F(s, a) \rightarrow s'$ yielding a new abstract state s'. This abstract plan, or plan skeleton in the TAMP terminology, can then be refined into continuous actions through the use of the controllers and samplers. As previously mentioned there tends to be a loss of information when going from continuous state x to an abstract state s. This can be alleviated by creating or learning additional predicates, however, predicates should provide information beneficial to solving the task. Otherwise, additional learning needs to take place for little to no benefit. The learning of predicates and what makes good predicates is discussed in more detail in [5]. We instead focus on another area of abstractions, that of object properties, since simple type information is not enough information to create a generalizable plan. For instance, a block the size of 5 cm affords totally different actions than a block of 1 m. In the previous work by Silver et al. these object properties have largely been neglected. Therefore we introduced the concept of *affordances* in order to capture the object properties and be able to plan with object properties in mind. Additionally, we attempt to classify objects based on affordances and data in order to be able to plan with groups of objects in mind instead of object instances.

1) Object and Skill Hierarchy: In order to plan more efficiently use can be made of an object hierarchy or an ontology-like database that includes object relationships. Previous research such as [6] and [7] have shown that using object affordances can help generalization, especially in the case of objects not seen during training. Our approach utilizes affordances to create new object classes based on shared object affordances. As previously mentioned object types can have one or multiple parent types, thus we can create a new type and add to another type's parents. This allows us to plan with the new parent type, additionally, when a novel object shares an affordance with known objects we can add the novel object to the class. An example of this can be found in Fig. 3.

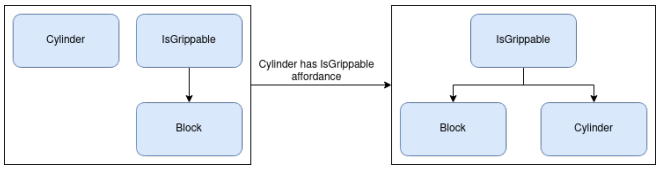

Fig. 3: Figure showing how object classes are detected using affordances.

In the previous papers by Silver et al. STRIPS operators are learned for each object type individually and not for a group of objects at a time. So when presented with a picking demonstration for two similar but different objects, such as a block and a cylinder the agent will learn two separate picking operators. This process is redundant in the case that the operators only differ in object types. In other words, when faced with a set of n objects with m different types, m different operators will be learned. What we propose is to first group similar objects as described in the previous paragraph and to learn an operator for the shared parent class, so instead of learning an operator for the `block` type we learn it for the `IsGrippable` type. An example of these operators can be found in Fig. 4.

Pick operator-block:		**Pick operator-IsGrippable:**	
Arguments:	[?b - block, ?r - robot]	Arguments:	[?g - IsGrippable, ?r - robot]
Preconditions:	{GripperOpen(?r), HandEmpty(?b)}	Preconditions:	{GripperOpen(?r), HandEmpty(?g)}
Add effects:	{Holding(?b)}	Add effects:	{Holding(?g)}
Delete effects:	{GripperOpen(?r), HandEmpty(?r)}	Delete effects:	{GripperOpen(?r), HandEmpty(?r)}
Controller: Pick(?b, ?r, θ)		Controller: Pick(?g, ?r, θ)	

Fig. 4: Examples of simplified STRIPS operators for the block type (left) and for the IsGrippable type (right).

As can be seen in Fig. 4 we have replaced the `block` type with the `IsGrippable` type, which can also be found in Fig. 3. This means that we can now plan with the `IsGrippable` variant of the operator for both the `block` and `cylinder` types. This decreased the number of operators we need to learn for effective planning. Additionally by taking an affordance approach to grouping objects a skill hierarchy is created implicitly. This is due to the nature of affordances as they relate to objects, actions, and effects.

C. Learning affordances

As previously mentioned affordances are very similar to predicates, therefore learning affordances can also be done in largely the same manner. When referring to affordance learning we mean the affordance classifier c_θ, we assume for now that affordance symbols and a ground truth classifier are given for now. While this is a limiting assumption previous work by [5] has shown that predicates can be learned from the ground up, meaning a similar approach can likely be taken for affordances in the future. Affordances can be viewed as extensions of an object's properties indicating whether it affords a certain action, in this way affordances act in a similar way as the precondition predicates in an operator. These extra preconditions are required since not every object in the object type necessarily has the same affordances. For example, a mug with a diameter of 6 cm in diameter would be graspable by a robot with a gripper 8 cm in width, but a mug with a diameter of 12 cm would not be graspable. Therefore despite the two objects being of a similar type this being a mug, they do not have the same affordances. In order to plan effectively in a varied environment these intricacies need to be taken into account as well. An important part of anything learning-related is data collection, as the data in large part shapes how learning can take place. Recall that we intend to learn the affordance's classifier c_θ, for this, we will need grounded examples of an affordance and its truth value, together with the state that produced it. This is the same way predicates are learned in previous works [1]. In the initialization, the agent only gets to see a limited set of objects for which it will get the affordances and the truth values for those affordances. The exploration phase in our implementation is different from that found in [1]. Instead of simply running the current models on training tasks as is done in [1] we opt to generate new tasks that are specifically constructed such that an affordance can be tested. For example when we are trying to test an affordance related to grasping we generate a grasping task. This difference is because affordances are closely related to actions, so in order to discover an object's affordances it will need to attempt it first. This type of exploration is typically referred to as play and actually mirrors the way babies and children learn. In our experiments, we emulate play by generating a task made to test a specific affordance. In order to collect additional information about object affordances queries are made to the oracle on whether a certain affordance holds. The agent will first query until it has both positive and negative examples for its affordances. Afterward, it will query based on entropy. This is done so the agent will only query about objects it is uncertain about.

D. Grouping objects and operators

In our experiments a group of operators is first extracted that can explain the underlying abstract state transitions, this process is described in detail in [3]; note in this paper operators are referred to as NSRTs. We have extended this method by also making it include affordances. The set of operators that is extracted using the aforementioned method contains duplicate operators for each object type seen in the data. In our new algorithm we go through the set of operators and when we detect they have different object types but the same effects, preconditions, and affordances we create a new object type including the object types from the reference operators, and then create a new operator with the same preconditions, effects, and affordances as the references but with the new object type. An overview of this algorithm is given in Algorithm 1. For clarity, a visual example is also provided in Fig. 5.

Algorithm 1: `Combine`: algorithm for combining operators and generating a new set of operators for new shared super types based on shared affordances, takes a set of operators Op and the current types $\mathcal{T}$ as input.

input : $Op, \mathcal{T}$
begin
  $Op^{'} \leftarrow \varnothing$ `// initialization`
  $\mathcal{T}^{'} \leftarrow \varnothing$
  foreach $o_1 \in Op, o_2 \in Op$ **do**
    `// Check similarity between operators`
    if `IsSimilar`(o_1, o_2) **and** $o_1 \neq o_2$ **then**
      $\mathcal{T}^{'} \leftarrow$ `CreateType`(o_1, o_2)
      $Op^{'} \leftarrow Op^{'} \cup$ `CreateOp`$(o_1, o_2, \mathcal{T}^{'})$
      $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{T}^{'}$
  return $Op', \mathcal{T}$

As can be seen in Fig. 5 a new type `IsGrippable` is created based on the shared affordance. This new object type can also be planned with and additional object types can be added when the object is found to have the same affordance. This has the potential to speed up learning as we need only find out an object's affordance in order to be able what learned operators are applicable to it.

IV. Experiments and Implementation

In this section, we will discuss our experiments and give more details on our implementation of the approach discussed in the method. In Section IV-A we will first go over the simulator used, next we will discuss the used affordances and how they affect operators in Section IV-B. After this, we will discuss the objects used during the experiments in Section IV-C. Finally, we discuss the experimental details for the exploration phase and the experiment as a whole in Section IV-D and Section IV-E respectively.

A. Simulation environment

For our experiments, we used a 7 DOF panda robotic arm simulated in pybullet[8], a physics-based simulator accessible

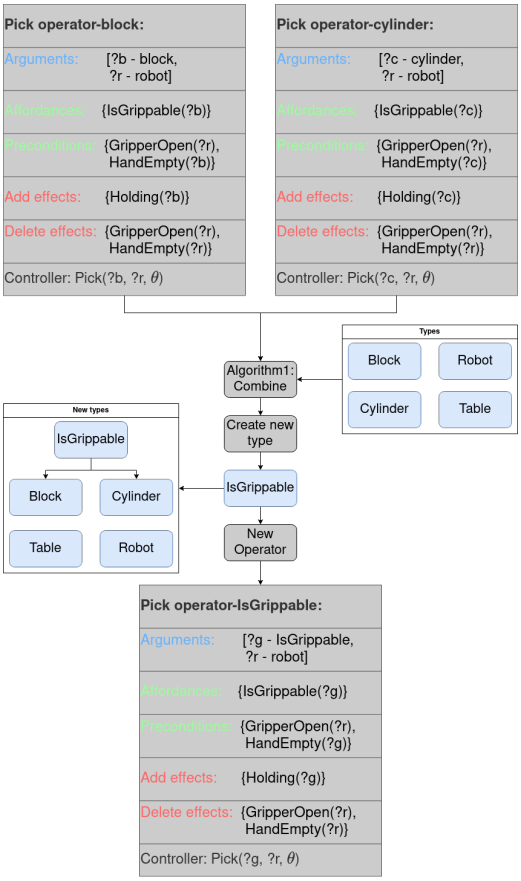

Fig. 5: Visual example of Algorithm 1.

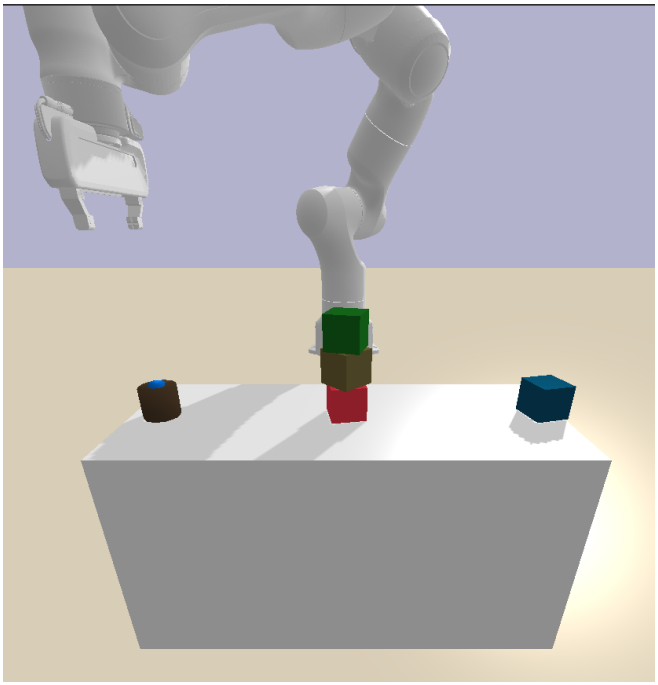

Fig. 6: Example scene from the test tasks.

Pick operator-block:	
Arguments:	[?b - block, ?r - robot]
Affordances:	{IsGrippable(?b)}
Preconditions:	{GripperOpen(?r), HandEmpty(?b)}
Add effects:	{Holding(?b)}
Delete effects:	{GripperOpen(?r), HandEmpty(?r)}
Controller: Pick(?b, ?r, θ)	

Fig. 7: Example of our extended operators including affordances.

from Python. This approach is different than the previous papers by Silver et al. where the authors used a fully deterministic hand-crafted simulator for each domain. Removing the need for such a simulator makes our approach easier to integrate with new domains, however, we also found that the simulator can be somewhat unreliable from time to time. An example scene from one of the test tasks is displayed in Fig. 6

B. Extended operators and affordances

In order to be able to use affordances from a planning standpoint they need to be included in the operators. This was done by treating them as extra preconditions on the manipulated objects. An example of the extended operators used can be seen in Fig. 7. In our experiments, we used three affordances: `IsGrippable`, `IsStackable`, and `IsPlacealbe` representing whether an object is grippable, can be stacked, or can be placed respectively. Each of the affordances also contains a ground truth classifier given to the oracle, in order for the agent to be able to get accurate queries on the actual affordance. An object is said to be grippable or placeable when it is under a certain size, and an object is said to be stackable when the top of the object is flat.

C. Object Hierarchy

As mentioned in the method we will focus more on object properties and grouping objects to be able to generate more general plans. In order to do this we work with three different manipulable object types, these being: `block`, `cylinder`, and `bottle`.

As seen in Fig. 8 `block` type objects have a square cross-section, `cylinder` type objects have a circular cross-section. The `bottle` type objects are similar to the `cylinder` type but also have a smaller cylinder on top to represent the bottle cap. All the objects have the same dimension feature vector, the features of which are displayed in Table I.

As can be seen in Table I the objects have features important for manipulation, such as position, orientation, and size. But they also contain more abstract features such as Shape and concavity. These features are there to help the samplers and affordance classifiers to differentiate between the different object types, as these do not get access to the type information directly.

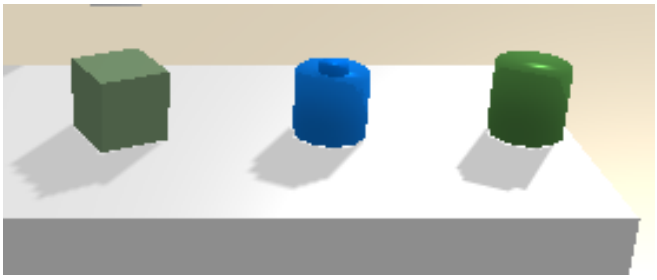

Fig. 8: Image showing the panda robot arm and the three object types block(left), bottle(middle), and cylinder(right) on a table.

Feature name	Description
Position	x,y, and z position in Cartesian coordinates.
Orientation	Quaternion
Size	Bounding box size in the three principal directions.
Color	RGB color data.
Shape	Information about the number of sides on an object, 4 for a block, -1 for cylindrical shaped objects.
Bottle-like	Binary flag indicating whether an object is bottle-like.
Constant	Binary flag indicating if the object's cross-section is constant.
Concavity	Feature indicating the shape of the top of the object, -1 for convex, 0 for flat, and 1 for concave.

TABLE I: Breakdown of manipulable object features and their meanings.

D. Exploration

The exploration phase of learning is used to learn the affordance classifiers $c_\theta(x)$. The classifiers are learned as an ensemble of binary classifiers each with different initial weights in order to be able to use entropy as a metric for when to query. For our experiment, we used 10 members for the ensemble and 50000 training iterations for each member. To learn an object's affordance a specific task related to that affordance is generated, for example, to test the `IsGrippable` affordance a picking task is generated where the robot must attempt to pick up the object. These tasks are short horizon involving only the bare minimum of objects. This approach is taken as to not provide full-length demonstrations in order to show that generalizability can be improved by only considering part of a task and refining it. The task is then solved by a bilevel planner, which is the same algorithm used to solve full tasks. This approach is different than the one found in [1], where the authors instead try to find states the agent is uncertain about. Since we are only concerned about the affordance of the object this is not important to us. Currently, the task generation is hard-coded, however, this could be done automatically in the future by using the add and delete effects of operators the affordance appears in and constructing an initial and goal state based on these predicates.

E. Experimental details

Our Experimental domain consists of an object rearranging task in a restricted 3D environment, the robot can only place the object in a line on the table but it is free to move in the entire space. The objects in the scene consist of the robot itself, a table, and the three objects described previously. Where the actions present in the demonstrations are: picking an object from the table, placing an object on the table, unstacking an object, and stacking an object. An initial set of 50 demonstrations is given from which the agent can learn an initial set of operators and affordance classifiers. The set of objects in the demonstrations is limited to the `block` and `cylinder` types of limited dimensions such that they can always fit in the robot's gripper. During the exploration phase, the full object set is used without the restriction on object size. For each of the three different affordances, we generate ten exploration tasks designed to test that specific affordance. After the exploration phase, the agent is tested on 30 held-out tasks and the cycle of exploration and evaluation repeats, we limit the number of cycles to 10. The test tasks are similar to the test tasks, but they can also involve objects with different dimensions than those seen in the initial demonstrations, additionally, there is a 50 percent chance of also including `bottle` type objects. The agent has access to the following set of predicates and their respective ground truth classifiers: {`Clear(x1)`, `Holding(x1)`, `On(x1,x2)`, `OnStatic(x1)`, `GripperOpen(x1)`}. The hyperparameters for the learned models are unchanged from those found in [1], except for the number of maximum iterations for the affordance classifiers which is limited to 50000.

V. Results & Discussion

In this section, we will highlight the results of the experiments we performed and discuss them. The results will be compared to a baseline. First, the baseline approach will be highlighted in Section V-A, afterward the difference in the learned operators will be displayed and discussed in Section V-B, and finally the generalizability of our approach will be highlighted in Section V-C.

A. Baseline comparison

In order to show the benefits of our approach we compare it with a baseline. The baseline chosen was an approach that does not use affordances, and can therefore not make use of our operator combining algorithm (1). Because the approach does not use affordances there is also no point in having an exploration phase. Thus the baseline approach is an approach without affordances, and without an exploration phase. All other aspects of the approach remain the same, such as the predicates, the objects, and the training and test tasks. A quick overview of the two approaches and their features can be found below:

- **Main** Our full approach with affordances, extended operators, dynamic object types, and the learning loop displayed in Fig. 2.

- **Baseline** A stripped-down approach without affordances, dynamic object types, and also without an exploration phase.

B. Learned operators

In order to highlight the difference between our approach and the baseline we will show the number of learned operators required for solving the training tasks. The number of operators required to solve the tasks can be found in Table II.

Main	Baseline
4	12

TABLE II: Number of operators required for solving the training tasks per approach.

As can be seen in Table II the number of operators required for our approach is half that of the baseline. This is due to the fact that our approach combines operators that have the same symbolic effect. It should be mentioned that in our approach operators are only combined when a shared affordance is also part of the operator. This is not strictly necessary but this approach made creating new types easier. Not only does our approach use fewer operators but the operators are also more general, using high-level classes that contain multiple object types instead of object types. Our approach also does not contain duplicate operators that achieve the same symbolic result but for different object combinations. An example of this can be found in Fig. 9

As can be seen in Fig. 9 the baseline approach requires an operator for stacking a block unto a cylinder and another operator for stacking a cylinder unto a block, whereas our approach only needs one for any combination of stackable objects. The reduction in the number of operators has two key benefits, namely less sampler learning needed, and more general applicability of existing operators.

C. Generalization to novel objects

Since our operators are learned for groups of objects we can transfer learned behaviors to novel objects. Recall that the `bottle` type was not present in the training data, and therefore the agent has not learned an operator for it. However, the agent has encountered the `bottle` type during the exploration phase, when it does the agent is likely to query as it has not seen this object before leading to a high uncertainty. After the agent is done with the exploration phase it will review its query responses and check the object types present for each affordance. If it detects a previously unseen object type in the affordance's related object type it will add the missing type to that object type. For example, when the agent finds that an object of the `bottle` type has the affordance `IsGrippable` it will add the `bottle` type to the group of grippable types. This means that all operators that can be applied to objects of the `IsGrippable` type now also apply to the previously unseen `bottle` type. The results of our experiments are displayed in Fig. 10.

As can be seen in Fig. 10 the baseline is able to solve 14 out of the 30 tasks. This is due to the fact that the baseline

Baseline:

Stack operator-block-cylinder:	Stack operator-cylinder-block:
Arguments: [?b - block, ?c - cylinder, ?r - robot]	Arguments: [?c - cylinder, ?b - block, ?r - robot]
Affordances: {}	Affordances: {}
Preconditions: {Holding(?b), Clear(?c)}	Preconditions: {Holding(?c), Clear(?b)}
Add effects: {On(?b, ?c), Clear(?b), GripperOpen(?r)}	Add effects: {On(?c, ?b), Clear(?c), GripperOpen(?r)}
Delete effects: {Holding(?b), Clear(?c)}	Delete effects: {Holding(?c), Clear(?b)}
Controller: Stack(?b, ?c, ?r, θ)	Controller: Stack(?c, ?b, ?r, θ)

Ours:

Stack operator-stackable:
Arguments: [?s1 - stackable, ?s2 - stackable, ?r - robot]
Affordances: {IsStackable(?s1), IsStackable(?s1)}
Preconditions: {Holding(?s1), Clear(?s2)}
Add effects: {On(?s1, ?s2), Clear(?s2), GripperOpen(?r)}
Delete effects: {Holding(?s1), Clear(?s2)}
Controller: Stack(?s1, ?s2, ?r, θ)

Fig. 9: Image showing an example of the duplicate operators found in the baseline approach versus our approach.

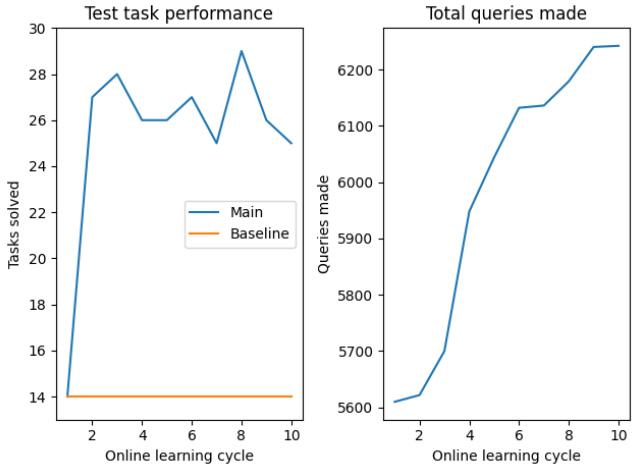

Fig. 10: Results of our experiments, showing our approach can solve tasks including unseen objects during training (left), and number of queries made over time (right).

approach can not solve tasks with novel objects, meaning it is only able to solve about half of the test tasks. Our approach initially fails to solve tasks including bottles as it is still gathering affordance data on the objects. At online learning cycle 2 however our approach has learned the bottle's affordances sufficiently and is able to plan with them as well. The deviation in performance seen in Fig. 10 can be attributed

to either faults in the simulator or bad samplers. The controller for the robot can also sometimes get stuck and this can cause issues when refining the plan. These factors lead to some variability in the number of tasks that can be refined into a full plan and successfully executed. Our approach however is able to find an abstract plan for every task from online learning cycle going forward. Meaning with a more reliable simulator and controller it should be able to solve every test task. In Fig. 10 we can also see that initially the agent makes many queries, this is because it is only initialized with one positive example for each affordance and is trying to get negative examples as well. When finally presented with negative examples the agent starts querying based on entropy and we see that the number of queries made decreases by a factor 10. The number of queries also tends to decrease as the number of cycles continuous, indicating that the agent's uncertainty about the affordances is decreasing as it is exposed to a wider set of object variations. It also becomes clear from Fig. 11 that the agent starts querying objects with features it hasn't seen before more often.

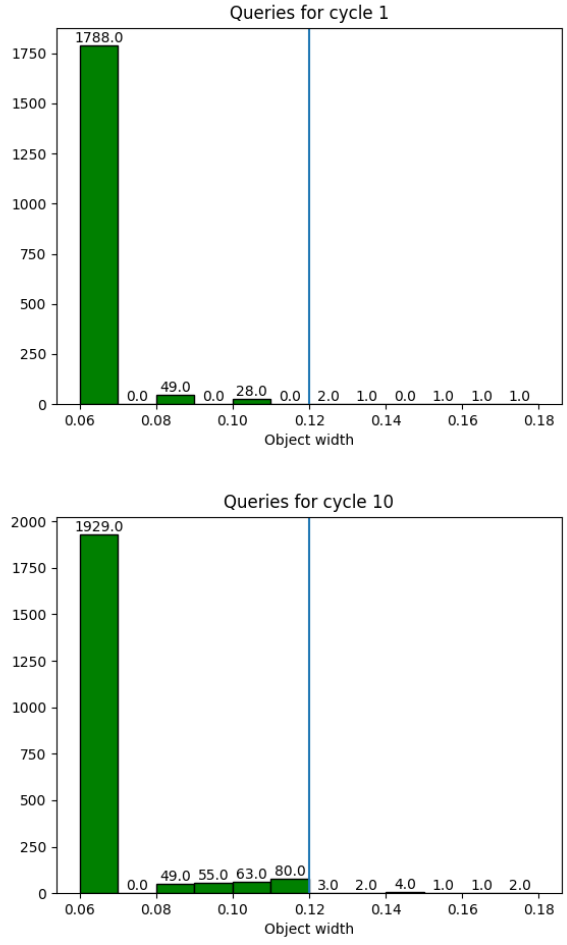

Fig. 11: Graph displaying the number of `IsGrippable` affordance queries made for objects of a certain width. Where the blue line indicates past which object width `IsGrippable` no longer holds. Green indicates positive responses and red indicates negative responses.

We can see in Fig. 11 that as the agent acquires experience it starts to query objects with larger widths more often. This indicates that it is more uncertain about these objects and is trying to refine its classifier. It should be mentioned that the large number of queries for the first bin is due to the fact that many queries are made until the agent has at least one negative example of each affordance.

VI. Conclusion and Future Work

In this paper we showed that our approach can handle novel objects, without having to relearn its operators. This was made possible by leveraging an affordance class-based approach to the objects used for experimentation. With this approach, it is possible to plan with affordance classes instead of object instances. Additionally, we can detect which objects are suitable for a given action without the need for additional predicates. This means that with our approach an agent can reuse its learned operators/skills more easily and does not require much new data for transferring its learned skills.

A. Future Work

In our experiments we assumed a set of affordances is given, however, this means that an expert still needs to reason about an object's affordance from the agent's point of view. While not as hard as designing good predicates it still requires care and may not always be intuitive. In the future similarities between objects and possible affordances should be discovered from data. This could be done by evaluating the similarity of learned operators for each object type, similar to our Algorithm 1. Additionally, during exploration, the agent is simply provided with an object's affordance labels if it queries the oracle. In the future, the agent should not have to query but should be able to discover affordances based on changes in the symbolic states.

VII. Acknowledgments

I would first of all like to thank my supervisor Zlatan Ajanović and Professor Jens Kober for their patience and invaluable feedback. Writing this thesis would not have been possible without your guidance. I would also like to thank my family and friends for believing in me throughout this journey.

References

[1] A. Li and T. Silver, "Embodied active learning of relational state abstractions for bilevel planning," *arXiv preprint arXiv:2303.04912*, 2023.

[2] T. Silver, R. Chitnis, J. Tenenbaum, L. P. Kaelbling, and T. Lozano-Pérez, "Learning symbolic operators for task and motion planning," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 3182–3189.

[3] R. Chitnis, T. Silver, J. B. Tenenbaum, T. Lozano-Perez, and L. P. Kaelbling, "Learning neuro-symbolic relational transition models for bilevel planning," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 4166–4173.

[4] T. Silver, A. Athalye, J. B. Tenenbaum, T. Lozano-Pérez, and L. P. Kaelbling, "Learning neuro-symbolic skills for bilevel planning," *arXiv preprint arXiv:2206.10680*, 2022.

[5] T. Silver, R. Chitnis, N. Kumar, W. McClinton, T. Lozano-Pérez, L. Kaelbling, and J. B. Tenenbaum, "Predicate invention for bilevel planning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 10, 2023, pp. 12 120–12 129.

[6] J. Borja-Diaz, O. Mees, G. Kalweit, L. Hermann, J. Boedecker, and W. Burgard, "Affordance learning from play for sample-efficient policy learning," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 6372–6378.

[7] P. Mandikal and K. Grauman, "Learning dexterous grasping with object-centric visual affordances," in *2021 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2021, pp. 6169–6176.

[8] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2023.