Learning to Design and Use Tools for Robotic Manipulation

Anonymous Author(s)
Affiliation
Address
email

Abstract: When limited by their own morphologies, humans and some species of animals have the remarkable ability to use objects from the environment toward accomplishing otherwise impossible tasks. Robots might similarly unlock a range of additional capabilities through tool use. Recent techniques for jointly optimizing morphology and control via deep learning are effective at designing locomotion agents. But while outputting a single morphology makes sense for locomotion, manipulation involves a variety of strategies depending on the task goals at hand. A manipulation agent must be capable of *rapidly prototyping* specialized tools for different goals. Therefore, we propose learning a *designer policy*, rather than a single design. A designer policy is conditioned on task information and outputs a tool design that helps solve the task. A design-conditioned controller policy can then perform manipulation using these tools. In this work, we take a step towards this goal by introducing a reinforcement learning framework for jointly learning these policies. Through simulated manipulation tasks, we show that this framework is more sample efficient than prior methods in multi-goal or multi-variant settings, can perform zero-shot interpolation or fine-tuning to tackle previously unseen goals, and allows tradeoffs between the complexity of design and control policies under practical constraints. Finally, we deploy our learned policies onto a real robot. See our supplementary video and website at https://robotic-tool-design.github.io/ for visualizations.

Keywords: tool use, manipulation, design

1 Introduction

Humans and some species of animals are able to make use of tools to solve manipulation tasks when they are constrained by their own morphologies. Chimpanzees have been observed using tools to access food and hold water [1], and cockatoos are able to create stick-like tools by cutting shapes from wood [2] with their beaks. To flexibly and resourcefully accomplish a range of tasks comparable to humans, embodied agents should also be able to leverage tools.

But critically, while any object in a human or robot's environment is a potential tool, not every object is directly a useful aid for the task goal at hand. How can a robotic system also acquire the extraordinary ability of animals to create an appropriate tool to help solve a task after reasoning about a scene's physics and its own goals? In this work, we investigate not only how agents can perform control using tools, but also how they can learn to *design* appropriate tools when presented with a particular task goal, such as a target position or object location.

Prior works have studied joint learning of agent morphologies and control policies for locomotion tasks [3, 4, 5, 6, 7]. However, these approaches optimize designs for a single, predefined *generic* goal, such as maintaining balance or forward speed. In this work, we take a step towards agents that can learn to rapidly prototype *specialized* tools for different goals or initial configurations.

Submitted to the 7th Conference on Robot Learning (CoRL 2023). Do not distribute.

We propose tackling this challenge by learning *designer* and *controller* policies, that are conditioned on task information, solely from task rewards via reinforcement learning (RL). We find that when trained with a multi-stage Markov decision process (MDP) formulation, these policies can be efficiently learned together in a high-dimensional combined space. We train agents on multiple instantiations of each task so that they can learn to produce and manipulate designs best suited to each situation.

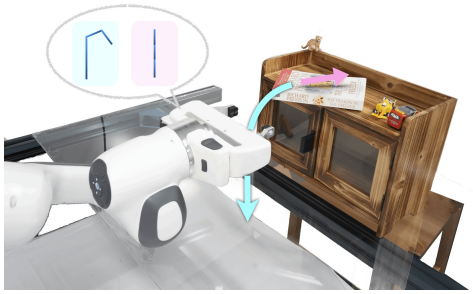


Figure 1: A robot may need to use different tools to fetch an out-of-reach book (blue) or push it into the bookshelf (pink). It should *rapidly prototype* the tool it needs.

We argue that there are three important properties of an embodied agent that designs and uses tools in realistic settings. Firstly, it must be able to learn design and control policies without explicit guidance, using signals specified only on task progress. Furthermore, it should form specialized tools based on the task goal at hand, as motivated by Figure 1. Finally, it should adjust to real-world constraints, rather than creating infeasible designs.

Our main contribution is a learning framework for embodied agents to design and use tools *based on the task at hand*. We demonstrate that our approach can jointly learn these policies in a sample-efficient manner from only downstream task rewards for a variety of simulated manipulation tasks, outperforming existing stochastic optimization approaches. We empirically analyze the generalization and few-shot finetuning capabilities of the learned policies. By introducing a tradeoff parameter between the complexity of design and control components, our approach can adapt to fit constraints such as material availability or energy costs. Finally, we demonstrate the real-world effectiveness of the learned policies by deploying them on a real Franka Panda robot.

2 Related Work

Computational approaches to agent and tool design. Many works have studied the problem of optimizing the design of robotic agents, end-effectors, and tools via model-based optimization [8, 9], generative modeling [10, 11], evolutionary strategies [3, 5], stochastic optimization [12], or reinforcement learning [13]. Li et al. [14] use differentiable simulation to find parameters of a tool that are robust to task variations. These methods provide feedback to the design procedure by executing pre-defined trajectories or policies, or performing motion planning. In contrast, we aim to *jointly* learn control policies along with designing tool structures.

In settings where the desired design is known but must be assembled from subcomponents, geometry [15] and reinforcement learning [16] have been used to compose objects into tools. In this work, we address the fabrication stage of the pipeline using rapid prototyping tools (e.g. 3D printing).

Learning robotic tool use. Several approaches have been proposed for empowering robots to learn to use tools. Affordance learning is one common paradigm [17, 18, 19, 20]. Noguchi et al. [21] integrate tool and gripper action spaces in a Transporter-style framework. Learned or simulated dynamics models [22, 23, 24, 25] have also been used for model-based optimization of tool-aided control. These methods assume that a helpful tool is already present in the scene, whereas we focus on optimizing tool design in conjunction with learning manipulation, which is a more likely scenario for a generalist robot operating for example in a household.

Joint optimization of morphology and control. One approach for jointly solving tool design and manipulation problems is formulating and solving nonlinear programs, which have been shown to be especially effective at longer horizon sequential manipulation tasks [26, 27]. In this work, we aim to apply our framework to arbitrary environments, and so we select a purely learning-based approach at the cost of increasing the complexity of the search space.

Reinforcement learning-based approaches have also been applied to jointly learn morphology and control. These include policy gradient methods with either separate [28] or weight-sharing [29] design and control policies or methods that consider the agent design as a computational graph [30].

Evolutionary or Bayesian optimization algorithms have also been combined with control policies learned through RL to solve a variety of locomotion and manipulation tasks [31]. Luck et al. [4] use an actor-critic RL formulation with a graph neural network (GNN) value function for improved sample efficiency. Pathak et al. [32] learn to design modular agents by adding morphology-modifying actions to an MDP. Yuan et al. [7] provide a generalized formulation with a multi-stage MDP and GNN policy and value networks. These methods have demonstrated promising performance on locomotion tasks. In this work, we focus on developing tools for manipulation, where the challenge is learning designer and controller policies that can create and operate different tools *depending on the task variation at hand*. Furthermore, it is less feasible to add actuated joints in these settings.

3 Problem Setting

Our framework tackles learning tool design and use for agents to solve manipulation problems, without any supervision except for task progress. We represent the agent's environment as a two-phase MDP consisting of the "design phase" and "control phase". We use environment interactions from both phases to jointly train a designer policy and controller policy.

At the start of each episode, the environment begins in the **design phase**, visualized at the top of Figure 2. During the design phase, the action $a_d \in \mathcal{A}_D$ specifies the parameters of the tool that will be used for the rest of the episode. The environment state $s_0 \in \mathcal{S}_{task}$ consists of a vector of task observations: the positions and velocities of objects in the scene and the Cartesian end-effector position of the robot if present.

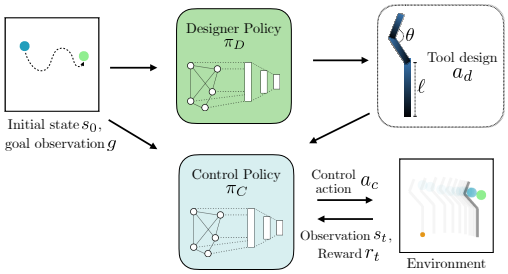


Figure 2: Solving a task using learned designer and controller policies. During the design phase, the designer policy observes the task at hand and outputs the parameters for a tool. In the control phase, the controller policy outputs motor commands given the tool structure, task specification, and observation.

After a single transition, the MDP switches to the **control phase**, illustrated in the lower half of Figure 2. During the control phase, the actions $a_c \in \mathcal{A}_C$ represent inputs to a position or velocity controller actuating the base of the previously designed tool. The control phase state is the concatenation of the task observation $s_t \in \mathcal{S}_{task}$ and the previously taken design action a_d. That is, the control state space $\mathcal{S}_C = \mathcal{S}_{task} \times \mathcal{A}_D$. The agent receives rewards r_t at each timestep t based on task progress (e.g., the distance of an object being manipulated to the target position). The control phase continues until the task is solved or a time limit is reached, and the episode then ends.

To learn design and control policies for multiple goals, we condition our policies on a supplied parameter g from a goal or task variation space $\mathcal{G}$. In this paper, we choose goals that represent the final position of an object to be manipulated or the number of objects that must be transported. The objective is then to find the optimal *goal-conditioned* designer and controller policies $\pi_D^\star(a|s,g)$, $\pi_C^\star(a|s,g)$ that maximize the expected discounted return of a goal-dependent reward function $R(s,a,g)$.

4 Instantiating Our Framework

Next, we concretely implement our framework toward solving a series of manipulation tasks. Specifically, we select a **tool design space**, **policy learning procedure**, and **auxiliary reward function**.

Tool design space. The design space significantly impacts the difficulty of the joint design and control optimization problem. When the set of possible designs is large but many of them are unhelpful for *any* task, the reward signal for optimization is sparse. Thus, we would like to select a design parameterization that is low-dimensional but can also enable many manipulation tasks. Furthermore, we prefer designs that are easy to deploy in the real world.

Toward these goals, we consider tools composed of rigid links. While simple, we find that this parameter space includes tools that are sufficient to help solve a variety of manipulation tasks. They

can also be easily deployed in the real world on soft robots [12] or through rapid fabrication techniques like 3D printing. However, we note that our framework is not limited to this design space.

For example, the parameterization used in our 2D environments consists of three links attached end-to-end as shown in Figure 2, where a tool is represented by a vector $[l_1, l_2, l_3, \theta_1, \theta_2] \in \mathbb{R}^5 = \mathcal{A}_D$, where l represents each link length and θ represents the relative angle between the links.

Policy learning. Similarly to Yuan et al. [7], we interactively collect experience in the environment using the design and control policies, where each trajectory spans the design and control phase. We then train the policies jointly using proximal policy optimization (PPO) [33], a popular policy gradient method. We adopt the graph neural network (GNN) policy and value function architecture from Yuan et al. [7]. When training in goal-conditioned environments, we supply the policies with randomized goals sampled from the environment for each interaction episode.

Auxiliary reward. An embodied agent that creates and uses tools in the real world must also consider resource constraints. Many prior co-optimization procedures assume that actuated joints or body links can be arbitrarily added to the agent morphology. However, as an example, when an agent solves manipulation tasks in a household environment, it may not have access to additional motors or building materials. On the other hand, when possible, constructing a larger tool may reduce the amount of power expended for motor control, especially if a task must be completed many times.

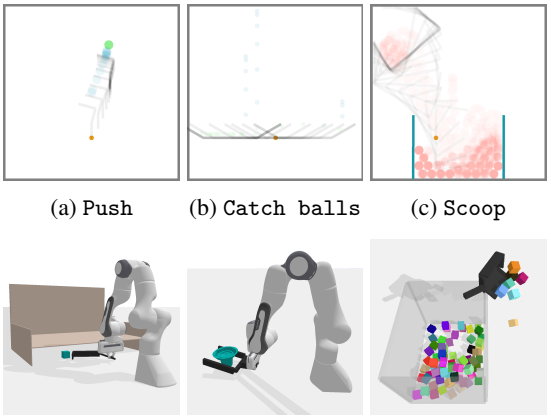
We enable our framework to accommodate preferences in this trade-off between design material cost and control energy consumption, proxied by end-effector velocity, using a parameter α that adjusts an auxiliary reward that is added to the task reward at each environment step:

$$r_{\text{tradeoff}} = K\left[1 - \left(\frac{\alpha \cdot d_{\text{used}}}{d_{\text{max}}} + \frac{(1-\alpha) \cdot c_{\text{used}}}{c_{\text{max}}}\right)\right], \tag{1}$$

where K is a scaling hyperparameter, $\alpha \in [0, 1]$ controls the emphasis on either the control or design component, d_{used} and d_{max} represent the utilized and maximum possible combined length of the tool components in the design respectively, and c_{used} and c_{max} represent the control velocity at the current step and the maximum single-step control velocity allowed by the environment. Intuitively the agent favors using less material for tool construction when α is large, and less energy for the control policy when α is small. Except in Section 5.3, we use $K = 0$ to isolate this reward's effects.

5 Experiments

For our experiments, we introduce three 2D manipulation environments in the Box2D simulator [34] and three 3D environments in PyBullet [35]. These tasks showcase the advantages of using different tools when there does not exist a single tool that can solve all instances. The six tasks are shown in Figure 3. For each task, we initialize the designed tool by matching a fixed point on the tool to a fixed starting position regardless of the goal. During the control phase, we simulate a scenario in which a robot has grasped the tool and manipulates it, via end-effector velocity control in 2D or position control in 3D. All 2D tasks use the 3-link chain parameterization described in Section 4. A short description of each task is as follows, with additional details in the Appendix:



(a) `Push` (b) `Catch balls` (c) `Scoop`

(d) `Fetch cube` (e) `Lift cup` (f) `Scoop (3D)`

Figure 3: Simulated manipulation environments.

- `Push (2D)`: Push a round puck using the tool such that it stops at the specified 2D goal location.
- `Catch balls (2D)`: Use the tool to catch three balls that fall from the sky. The agent's goal is to catch all three balls, which start from random locations on the x-y plane.
- `Scoop (2D)`: Use the tool to scoop balls out of a reservoir containing 40 total balls. Here we specify goals of scooping $n \in \{1, 2, ..., 7\}$ balls.

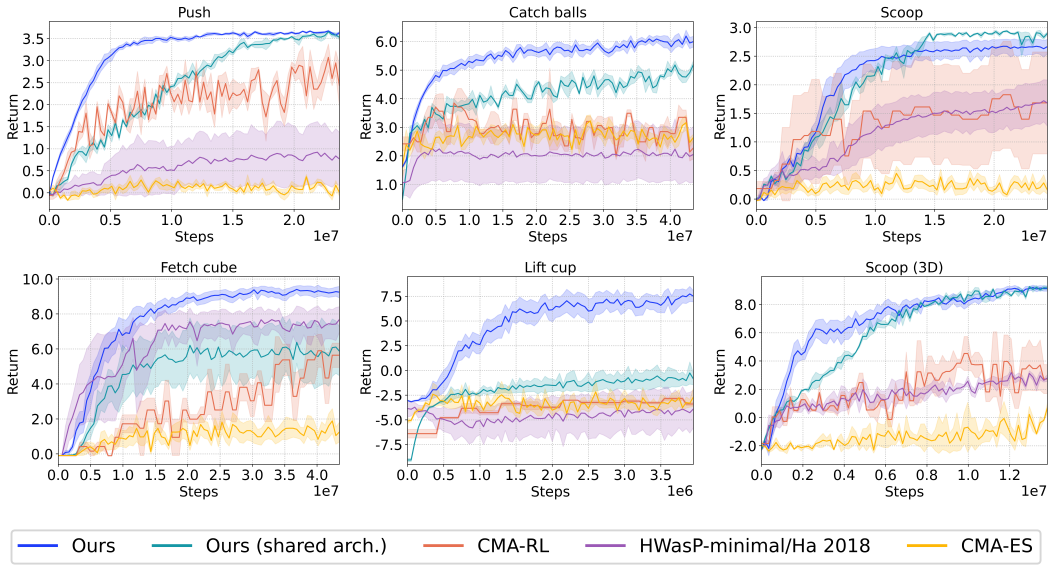


Figure 4: Learning curves for our framework, prior methods, and baselines. Across all tasks, our framework achieves improved performance and sample efficiency. Shaded areas indicate standard error across 6 random seeds on all methods, except the `scoop (3D)` task where we use 3 seeds due to computational constraints.

- `Fetch cube (3D)`: Use the tool to retrieve an object randomly positioned beneath an overhang. This task is challenging because the end effector is restricted to a $0.8\text{m} \times 0.2\text{m}$ region of the x-y plane to avoid collision with the overhang. The tool is a three-link chain where each link is a box parameterized by its width, length, height, and relative angle to its parent link.
- `Lift cup (3D)`: Use the tool to lift a cup of randomized geometry from rest to a certain height. This task requires careful tool design to match cup geometry. The tool is a four-link fork with two prongs symmetrically parameterized by their separation, tilt angle, width, length, and height.
- `Scoop (3D)`: An analog of the 2D `scoop` task (with the same goal space), but the tool in 3D is a six-link scoop composed of a rectangular base plate parameterized by its length and width, and four rectangular side plates attached to each side of the base plate, parameterized by their height and relative angle to the bottom plate. A fixed-dimension handle is attached to one side plate.

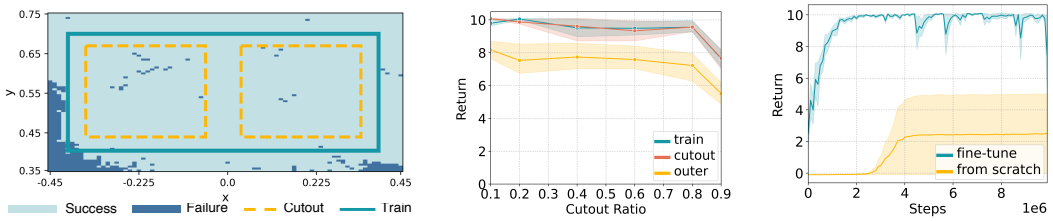
In our experiments, we analyze whether the instantiation of our framework on these manipulation tasks has the following four desirable properties:

- Can our framework jointly learn designer and controller policies in a sample-efficient manner, using just rewards based on task progress?
- Do learned designer and controller policies generalize or enable fine-tuning for unseen goals?
- Can the adjustable parameter α enable agents to trade off design and control complexity?
- Can designer and controller policies learned by our framework be deployed on a real robot?

5.1 Evaluating sample efficiency

Sample efficiency is critical for a performant joint tool design and control learning pipeline, as many sampled designs will be unhelpful for *any* goal or initialization. As prior joint optimization works do not handle situations where diverse designs may be produced depending on the particular task variation, we compare to the following prior methods and baselines (details in Appendix B.2):

- **Bi-level optimization (CMA-RL):** This follows the common bi-level paradigm for joint optimization [4]. We use CMA-ES [36] to perform outer-loop stochastic design optimization and learn design-conditioned policies with PPO in the inner loop, as the reward signal for design.
- **Hardware-as-Policy Minimal [30]/Ha [29]:** The variant of HWasP without differentiable physics assumptions (HWasP-minimal) and the method from Ha [29] both perform policy-gradient optimization of a single set of design parameters together with the control policy jointly using RL.



(a) Initialization ranges and zero-shot performance when cutting out 60% of the area of the entire possible training region.

(b) Returns for policies trained with varying relative cutout region area.

(c) Fine-tuning performance compared to learning from scratch across 4 target goals.

Figure 6: We find that our policies can solve instances of the `fetch cube` task unseen at training time either zero-shot or by rapid finetuning. In (a), we plot the success and failure goal regions. Areas within the dotted yellow borders and outside the teal region are unseen during training. The region within the teal border (but outside cutout regions) is the training region. Training curves are averaged over 3 runs; shaded regions denote standard error.

- **Single-trajectory CMA-ES baseline:** We optimize a set of design and control actions for an episode independent of the goal or starting task state, demonstrating the required policy reactivity.
- **Ours (shared arch.):** An ablation of our framework that uses a single policy network with design and control heads, demonstrating the importance of separate designer and controller architectures.

In Figure 4, we compare the learning curves of our method and demonstrate results for all six tasks. Our method strongly outperforms the prior methods and baselines, achieving superior final performance in fewer samples. It does so by producing specialized tools to solve each task instance, while the other methods optimize for a single design across all task variations. The ablation shows the importance of learning separate designer and controller policies.

We present qualitative examples of tool designs outputted for different goals on the `push` task in Figure 5. The designer policy outputs a range of tools depending on the goal location.

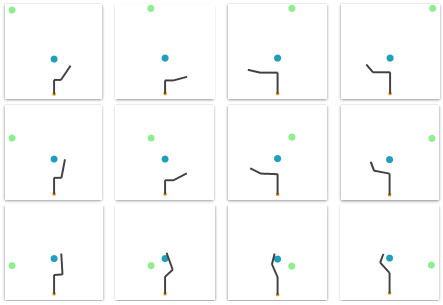


Figure 5: Tool designs outputted by a single learned designer policy for the `push` task as the goal position, in green, varies. The range of outputted designs enable the agent to push the ball in the desired direction.

5.2 Generalization to unseen task variations

In simulation, agents can experience millions of trials for a range of task variations. However, when deployed to the real world, designer and controller policies cannot be pre-trained on all possible future manipulation scenarios. In this section, we test the ability of our policies to generalize to task variations unseen during training. For these experiments, we focus on the `fetch cube` task because it has an initial pose space that can be manipulated in a semantically meaningful way. We train policies using our framework on a subset of initial poses from the entire initial pose space by removing a region of the space, which we call the "cutout" region. (see Figure 6a). Then, we evaluate the generalization performance of learned policies on the initial poses from the cutout region and outside the training region in two scenarios.

Zero-shot performance. In the first scenario, we test the ability of the designer and controller policy to tackle a previously unseen initial pose directly. Using a policy trained on the initial pose space with a cutout region removed, we evaluate the zero-shot performance on unseen initial poses.

In Figure 6a, we visualize the zero-shot performance of our design and control policies on initial poses across the entire environment plane, finding that our policies are able to solve even task variations outside the training region boundaries. We also analyze how decreasing the number of possible training poses affects generalization performance. In Figure 6b, we show the returns over the training region, cutout region, and the regions outside of the training region, as the size of the cutout region for training poses varies.

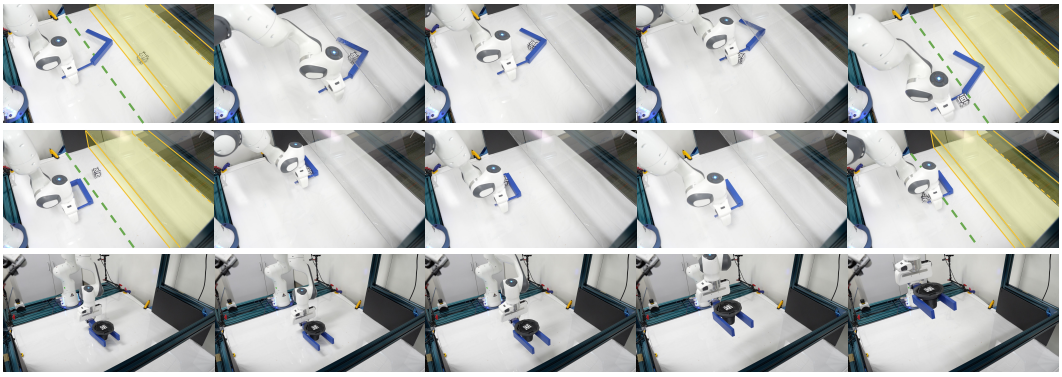
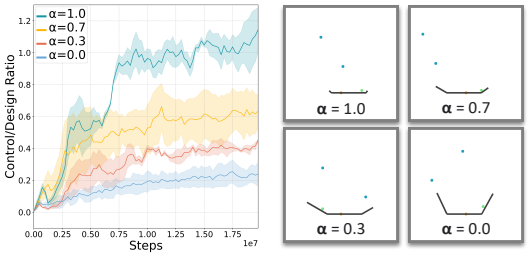


Figure 8: Real world rollouts for the `fetch cube` (top) and `lift cup` (bottom) tasks. For `fetch cube`, we show the task success threshold in green and the volume covered by the overhang in yellow. The design policy generates specialized tools, and the control policy adjusts its strategy to use the tool to complete each task.

When the cutout region is very small, the performance of our learned policies on seen and unseen poses is similar. As the area of the cutout region increases, the performance on unseen goals degrades gracefully and can still solve a significant portion of unseen tasks.

Fine-tuning performance. Sometimes, new task variations cannot be achieved zero-shot by designer and controller policies. In this section, we test whether our design policies and controller policies can still serve as good *instantiations* for achieving these variations. We test this by pre-training policies with our framework on the entire training region and fine-tuning them to solve initial poses outside that region.

In Figure 6c, we show the results of the fine-tuning experiment. We find that even for poses that are far away from the initial training region, our policies are able to learn to solve the task within a handful of gradient steps, and is much more effective than learning from scratch.



(a) Control/Design ratio with different α.

(b) Tools produced at different α values.

Figure 7: Examples of tools generated by varying the tradeoff parameter α. As α increases, the created tools have shorter links at the sides to decrease material usage. With lower α, large tools reduce the control policy's required movement.

5.3 Trading off design and control complexity

In this section, we aim to determine whether our introduced tradeoff parameter α (defined in Equation 1) can actualize preferences in the tradeoff between design material cost and control energy consumption. For this experiment, we focus on the `catch balls` task, because the tradeoff has an intuitive interpretation in this setting: a larger tool can allow the agent to catch objects with minimal movement, while a smaller tool can reduce material cost but requires a longer trajectory with additional energy costs. We train four agents on `catch balls` with different values of α.

In Figure 7a, we plot the ratio $\frac{d_{\text{used}}/d_{\text{max}}}{c_{\text{used}}/c_{\text{max}}}$, where d represents the combined length of all tool links and c is the per-step control velocity. d_{max} and c_{max} indicate the maximum tool size and control velocity allowed by the environment. We find that this ratio indeed correlates with α, which indicates that agents that are directed to prefer saving either material or energy are doing so, at the cost of the other. We also visualize the outputted tool designs in Figure 7b.

5.4 Evaluation on a real robot

Next, we provide a demonstration of our pipeline on a real robot by transferring learned policies in two of the 3D environments, `fetch cube` and `lift cup`, directly to the real world. We use a Franka Panda 7-DoF robot arm equipped with its standard parallel jaw gripper (Figure 11). Five RealSense D435 RGBD cameras perform object tracking using AprilTags [37].

Fabricating tools. In order to evaluate our policies in the real world, we fabricate the tools that are outputted by the design policy via 3D printing. Specifically, based on an initial state observation and/or goal, we convert tool parameters from design policy outputs into meshes and construct them from PLA using consumer Ender 3 and Ender 5 fused deposition modeling (FDM) 3D printers. See Figure 9 for examples of printed tools generated by our policy.

Real robot evaluation. For the `fetch cube` and `lift cup` tasks, we evaluate policies on four initial cube positions and four cup geometries respectively. For each task instance, we fabricate the designed tool and evaluate control over five trials. We report success rates for each tool in Table 1, finding that our policies successfully transfer their performance to the real world.

Figure 8 shows qualitative examples of real rollouts. In the top row, the robot designs an 'L'-shaped hook. Because the end-effector has a constrained workspace due to the risk of collision with the overhang, the robot employs a two-stage strategy – first hooking the block from under the overhang, and then using the backside of the tool to drag it closer to itself. In the second row, the robot uses a 'U'-shaped tool to quickly retrieve the nearby block and finally pushes it to the goal with the gripper fingers when it is within reach. This exemplifies that our framework flexibly allows an agent to use its original morphology in combination with designed tools when needed. Finally, in the `lift cup` task, the design policy selects appropriate distances between the tool arms and an angle of approach to hold the cup securely when elevated.

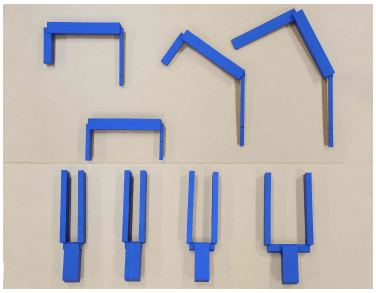


Figure 9: 3D-printed tools for `fetch cube` (top) and `lift cup` (bottom).

For `fetch cube`, we further analyze the performance of the control policy when using tools for initial positions that they were **not directly designed for**. Specifically, we take the four generated tools and evaluate how well the control policy can use them to solve 12 tasks on a 12cm $\times$ 85.6cm grid of initial cube positions.

Tool number	1	2	3	4
`Fetch cube` (single init.)	5/5	5/5	5/5	5/5
`Fetch cube` (grid inits.)	10/12	4/12	6/12	5/12
`Lift cup`	5/5	5/5	5/5	5/5

Table 1: **Real world evaluation performance.** "Single init" denotes the evaluation of a tool on the initial environment state it was designed for. "Grid of inits" evaluates the same tool on a range of initializations.

The results are presented in Table 1. While the control policy can reuse tools to solve new tasks, no tool can solve all the tasks. Tool 1 solves the most tasks, but tool 4 solves both tasks that tool 1 cannot. We also find that the policy takes a greater number of timesteps to solve each task with tool 1 compared to tools specialized for those initializations. These experiments demonstrate that a diverse set of tools is indeed important for different variations of this task.

6 Conclusion

We have introduced a framework for agents to jointly learn design and control policies, as a step towards generalist embodied manipulation agents that are unconstrained by their own morphologies. Because the best type of tool and control strategy can vary depending on the goal, we propose to learn designer and controller policies to generate useful tools based on the task at hand and then perform manipulation with them. Our work is a step towards building embodied agents that can reason about novel tasks and settings and then equip themselves with the required tools to solve them, without any human supervision. Such systems may lead the way towards autonomous robots that can perform continuous learning and exploration in real-world settings.

Limitations. In this work, we focus on rigid, non-articulated tools composed of linked primitive shapes. A promising direction for future work is to explore other parameterizations. In addition, we address the problem of learning design and control rather than fabrication: we use 3D-printing for prototyping and do not consider the problem of constructing tools from a set of available objects. An eventual joint framework for design, construction, and control could unify these directions.

References

[1] J. Goodall. Tool-using and aimed throwing in a community of free-living chimpanzees. *Nature*, 201, 1964.

[2] A. M. Auersperg, S. Borasinski, I. Laumer, and A. Kacelnik. Goffin's cockatoos make the same tool type from different materials. *Biology Letters*, 12, 2016.

[3] K. Sims. Evolving virtual creatures. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '94, page 15–22, New York, NY, USA, 1994. Association for Computing Machinery. ISBN 0897916670.

[4] K. S. Luck, H. B. Amor, and R. Calandra. Data-efficient co-adaptation of morphology and behaviour with deep reinforcement learning. In *3rd Annual Conference on Robot Learning, CoRL 2019, Osaka, Japan, October 30 - November 1, 2019, Proceedings*, volume 100 of *Proceedings of Machine Learning Research*, pages 854–869, 2019.

[5] D. Hejna, III, P. Abbeel, and L. Pinto. Task-agnostic morphology evolution. In *International Conference on Learning Representations*, 2021.

[6] A. Gupta, S. Savarese, S. Ganguli, and L. Fei-Fei. Embodied intelligence via learning and evolution. *Nature Communications*, 2021.

[7] Y. Yuan, Y. Song, Z. Luo, W. Sun, and K. M. Kitani. Transform2act: Learning a transform-and-control policy for efficient agent design. In *International Conference on Learning Representations*, 2022.

[8] K. Kawaharazuka, T. Ogawa, and C. Nabeshima. Tool shape optimization through backpropagation of neural network. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8387–8393, 2020.

[9] K. R. Allen, T. Lopez-Guevara, K. Stachenfeld, A. Sanchez-Gonzalez, P. Battaglia, J. Hamrick, and T. Pfaff. Physical design using differentiable learned simulators. *arXiv preprint arXiv: Arxiv-2202.00728*, 2022.

[10] Y. Wu, S. Kasewa, O. Groth, S. Salter, L. Sun, O. P. Jones, and I. Posner. Imagine that! leveraging emergent affordances for 3d tool synthesis. *arXiv preprint arXiv: Arxiv-1909.13561*, 2019.

[11] H. Ha, S. Agrawal, and S. Song. Fit2form: 3d generative model for robot gripper form design. In *4th Conference on Robot Learning, CoRL 2020, 16-18 November 2020, Virtual Event / Cambridge, MA, USA*, volume 155 of *Proceedings of Machine Learning Research*, pages 176–187. PMLR, 2020.

[12] I. Exarchos, K. Wang, B. H. Do, F. Stroppa, M. M. Coad, A. M. Okamura, and C. K. Liu. Task-specific design optimization and fabrication for inflated-beam soft robots with growable discrete joints. In *2022 International Conference on Robotics and Automation, ICRA 2022, Philadelphia, PA, USA, May 23-27, 2022*, pages 7145–7151. IEEE, 2022.

[13] Y. Li, T. Kong, L. Li, Y. Li, and Y. Wu. Learning to design and construct bridge without blueprint. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2021, Prague, Czech Republic, September 27 - Oct. 1, 2021*, pages 2398–2405. IEEE, 2021.

[14] M. Li, R. Antonova, D. Sadigh, and J. Bohg. Learning tool morphology for contact-rich manipulation tasks with differentiable simulation. *arXiv preprint arXiv:2211.02201*, 2022.

[15] L. Nair, N. Shrivatsav, and S. Chernova. Tool macgyvering: A novel framework for combining tool substitution and construction. *arXiv preprint arXiv: Arxiv-2008.10638*, 2020.

[16] S. K. S. Ghasemipour, S. Kataoka, B. David, D. Freeman, S. S. Gu, and I. Mordatch. Blocks assemble! learning to assemble with large-scale structured reinforcement learning. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 7435–7469. PMLR, 2022.

[17] K. Fang, Y. Zhu, A. Garg, A. Kurenkov, V. Mehta, L. Fei-Fei, and S. Savarese. Learning task-oriented grasping for tool manipulation from simulated self-supervision. In *Robotics: Science and Systems XIV, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, June 26-30, 2018*, 2018.

[18] Z. Qin, K. Fang, Y. Zhu, L. Fei-Fei, and S. Savarese. Keto: Learning keypoint representations for tool manipulation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7278–7285. IEEE, 2020.

[19] J. Brawer, M. Qin, and B. Scassellati. A causal approach to tool affordance learning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8394–8399, 2020.

[20] D. Xu, A. Mandlekar, R. Martín-Martín, Y. Zhu, S. Savarese, and L. Fei-Fei. Deep affordance foresight: Planning through what can be done in the future. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 6206–6213. IEEE, 2021.

[21] Y. Noguchi, T. Matsushima, Y. Matsuo, and S. S. Gu. Tool as embodiment for recursive manipulation. *arXiv preprint arXiv: Arxiv-2112.00359*, 2021.

[22] K. R. Allen, K. A. Smith, and J. Tenenbaum. Rapid trial-and-error learning with simulation supports flexible tool use and physical reasoning. *Proceedings of the National Academy of Sciences*, 2019.

[23] A. Xie, F. Ebert, S. Levine, and C. Finn. Improvisation through physical understanding: Using novel objects as tools with visual foresight. In *Robotics: Science and Systems XV, University of Freiburg, Freiburg im Breisgau, Germany, June 22-26, 2019*, 2019.

[24] R. Girdhar, L. Gustafson, A. Adcock, and L. van der Maaten. Forward prediction for physical reasoning. *arXiv preprint arXiv: Arxiv-2006.10734*, 2020.

[25] X. Lin, Z. Huang, Y. Li, J. B. Tenenbaum, D. Held, and C. Gan. Diffskill: Skill abstraction from differentiable physics for deformable object manipulations with tools. In *International Conference on Learning Representation (ICLR)*, 2022.

[26] M. Toussaint, K. R. Allen, K. A. Smith, and J. B. Tenenbaum. Differentiable physics and stable modes for tool-use and manipulation planning. In *Proc. of Robotics: Science and Systems (R:SS)*, 2018.

[27] M. Toussaint, J.-S. Ha, and O. S. Oguz. Co-optimizing robot, environment, and tool design via joint manipulation planning. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2021.

[28] C. B. Schaff, D. Yunis, A. Chakrabarti, and M. R. Walter. Jointly learning to construct and control agents using deep reinforcement learning. In *International Conference on Robotics and Automation, ICRA 2019, Montreal, QC, Canada, May 20-24, 2019*, pages 9798–9805. IEEE, 2019.

[29] D. Ha. Reinforcement learning for improving agent design. *arXiv:1810.03779*, 2018.

[30] T. Chen, Z. He, and M. Ciocarlie. Hardware as policy: Mechanical and computational co-optimization using deep reinforcement learning. *Corl*, 2020.

[31] J. Bhatia, H. Jackson, Y. Tian, J. Xu, and W. Matusik. Evolution gym: A large-scale benchmark for evolving soft robots. *Advances in Neural Information Processing Systems*, 34, 2021.

[32] D. Pathak, C. Lu, T. Darrell, P. Isola, and A. A. Efros. Learning to control self-assembling morphologies: A study of generalization via modularity. *arXiv preprint arXiv: Arxiv-1902.05546*, 2019.

[33] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv: Arxiv-1707.06347*, 2017.

[34] E. Catto. Box2d. URL https://box2d.org/.

[35] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2019.

[36] N. Hansen and A. Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 312–317, 1996. doi:10.1109/ICEC.1996.542381.

[37] E. Olson. Apriltag: A robust and flexible visual fiducial system. In *2011 IEEE International Conference on Robotics and Automation*, pages 3400–3407, 2011.

[38] Y. Lin, A. S. Wang, G. Sutanto, A. Rai, and F. Meier. Polymetis. https://facebookresearch.github.io/fairo/polymetis/, 2021.