2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

Online augmentation of learned grasp sequence policies for more adaptable and data-efficient in-hand manipulation

Ethan K. Gordon[1], Rana Soltani Zarrin[2]

***Abstract*— When using a tool, the grasps used for picking it up, reposing, and holding it in a suitable pose for the desired task could be distinct. Therefore, a key challenge for autonomous in-hand tool manipulation is finding a sequence of grasps that facilitates every step of the tool use process while continuously maintaining force closure and stability. Due to the complexity of modeling the contact dynamics, reinforcement learning (RL) techniques can provide a solution in this continuous space subject to highly parameterized physical models. However, these techniques impose a trade-off in adaptability and data efficiency. At test time the tool properties, desired trajectory, and desired application forces could differ substantially from training scenarios. Adapting to this necessitates more data or computationally expensive online policy updates.**

In this work, we apply the principles of discrete dynamic programming (DP) to augment RL performance with domain knowledge. Specifically, we first design a computationally simple approximation of our environment. We then demonstrate in physical simulation that performing tree searches (i.e., *lookaheads*) and policy *rollouts* with this approximation can improve an RL-derived grasp sequence policy with minimal additional online computation. Additionally, we show that pretraining a deep RL network with the DP-derived solution to the discretized problem can speed up policy training.

I. Introduction

Dexterous manipulation focuses on the problem of enabling stable grasping and manipulation of objects using multi-fingered hands [1]. In this broad space, many previous studies address object reposing only [2–4]. In this work, we focus on the specific application of in-hand tool manipulation, which requires the object (i.e., the *tool*) to be held with a grasp suitable for applying an arbitrary wrench (i.e., force and torque) to the external environment.

As shown in Fig. 1, this may require multiple different grasps addressing the sub-problems of acquisition, movement, and use. Therefore, we consider in-hand tool manipulation as a problem of *grasp sequence planning*. Given a desired tool trajectory and application, the robot must determine which grasp to use at each point in time to execute the motion while maintaining stability over the entire episode. This is a difficult problem for traditional physical controllers. The dynamics are complex and highly parameterized. Future grasp performance depends on previous grasps. Prerequisite physical models of the system may be imperfect.

Deep reinforcement learning (DRL) offers an enticing solution by eliminating the need to model the complex physics of the problem [5]. End-to-end systems collect hours of data to directly learn low-level control policies [6]. Hierarchical frameworks, on the other hand, can use RL to learn high-level sub-tasks or primitives and rely on model-based methods for lower level control [7,8]. Without explicit domain knowledge, both of these approaches effectively build physical models from scratch. Therefore, with insufficient data, these systems can struggle to adapt to scenarios that differ significantly from those at training time.

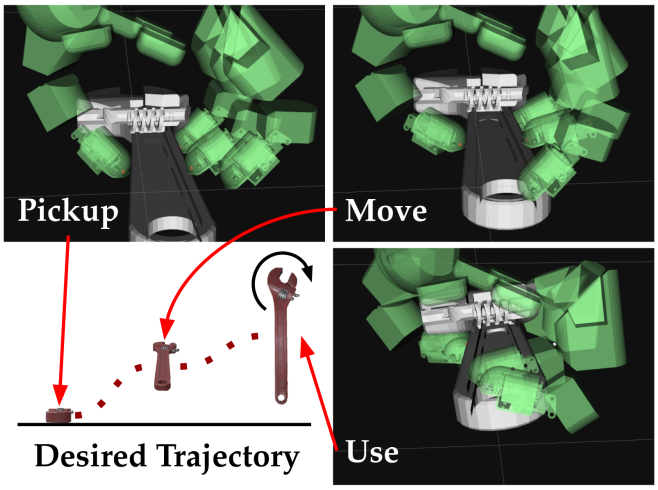


Fig. 1: Given a desired tool trajectory and anticipated external forces, determine a sequence of grasps that continuously maintains force closure and stability. In this example, while moving the wrench, the selected grasp adds the ring finger for stability. To use the wrench for bolt tightening, the fingers gradually transition to a power grasp while the thumb is placed towards the wrench head to apply the desired torque.

Our key insight is that discrete-time DP can be used to introduce domain knowledge to an RL policy at test time via a computationally-inexpensive approximate physical model. This scheme improves RL performance on previously unseen scenarios without additional data collection or computational overhead at test time. In summary, this work presents two primary contributions: (1) a surrogate environment model that can approximate the grasp sequence planning problem, and (2) a scheme that leverages traditional DP to augment test time RL performance using this approximate model. Additionally, we show that DP-derived grasp sequences within the surrogate model can be used as an expert policy to provide a "hot start" to RL training via behavior cloning. Finally, we compare this scheme to RL and DP baselines in physical simulation.

II. Related Work

a) In-Hand Manipulation: While prior strategies for in-hand manipulation commonly use external or contact forces from the environment—such as supporting an object by the hand's palm, using non-prehensile primitives [2,10–

[1] Ethan K. Gordon is with the Department of Computer Science and Engineering, University of Washington, Seattle, WA 98195 ekgordon,@cs.washington.edu. Work done as an intern at the Honda Research Institute.

[2] Rana Soltani Zarrin is with the Honda Research Institute, San Jose, CA 95134 rana_soltanizarrin@honda-ri.com

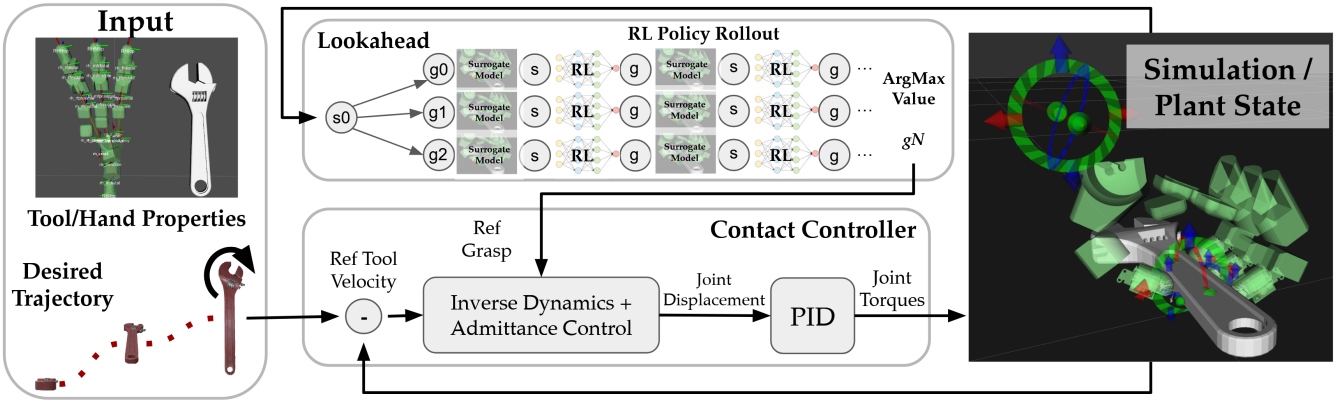


Fig. 2: System diagram. At each time step, we construct a "lookahead" complete search tree of length l for the discrete set of possible actions (i.e., grasps) from the current system state. The outcome of each grasp is approximated by the physical model described in Section V. The terminating cost of each tree leaf is determined by rolling out the RL policy until the end of the episode. The grasp action with the highest value is set as the reference grasp for the Contact Controller from [9], which in turn calculates the desired joint torques.

13], or using underactuated hands to improve stability and robustness [2,14,15]—we perform in-hand manipulation considering internal forces using only fully actuated hands. Model-based approaches have been used to perform in-hand manipulation tasks involving sliding and rolling [16,17] in computer graphics and simple object reorientation applications [13]; however, the difficulty of modeling complex contact interactions and inaccuracies, and the high computation cost of re-solving the problem for variations introduced during runtime, make them unsuitable for real-world robotic applications. As a result, data-driven approaches have been widely explored, albeit mainly for object reorientation. Taking advantage of the extra stability that results from supporting the object using the palm, multiple end-to-end DRL methods have been used for in-hand reorientation of simple objects, such as a cylinder or cube, using vision sensory information [3,6]. However, data dependencies limit the sample efficiency and generalizability of these approaches.

Hierarchical frameworks, i.e., those that learn motion primitives and rely on model-based low-level controllers for primitive execution, can alleviate the issues associated with individual methods. [4] has proposed a hierarchical learning/control scheme, where a higher level learning algorithm plans a sequence of motion primitives (reposing, flipping, sliding) that are executed by a lower level controller. In our recent work [9], we developed a hybrid model- and RL-based grasp sequence planner capable of commanding object reposing and adding/removing/sliding contacts that are executed by a low-level controller; we showed higher robustness to trajectory variations compared to a pre-computed optimal sequence generated by solving a DP. In this paper, by pretraining RL with a DP-generated policy for a nominal environment, we speed up training, and by using an online lookahead and rollout method, we make the planner more robust to trajectory and object variations at runtime.

b) Search-Based Lookahead: The benefit of using search-based methods, a staple of traditional DP, to augment the performance of trained-RL in naturally discrete board game systems (such as the AlphaZero from DeepMind [18] to solve Go) has been demonstrated. These methods were partially formalized in [19], identifying the lookahead and rollout procedure. It contemplates the potential use of the idea in continuous, adaptive control but offers no specific implementation. Our work leverages this idea of augmenting RL with search-based DP: by discretizing the continuous in-hand manipulation problem to enable the use of lookahead, we demonstrate its potential utility for many robotics applications.

c) RL Pretraining: To improve the sample efficiency of RL methods, research has explored augmenting demonstrations [20–22], learning from priors [23], and using model-based RL [24]. However, most of these behavior cloning approaches utilize human-collected trajectories as the expert policy to clone, limiting the amount and variety of data from which to learn. This work instead proposes using the optimal policies generated by DP on a discrete subset of the environment as the expert policy to clone. This obviates the need to commission a human study for data collection.

III. Problem Definition

The problem of interest is in-hand manipulation of a rigid body *tool* through free space such that, once at the goal pose, it can apply a specified wrench on a given quasi-static object in the environment. We are motivated by the following example applications: using a wrench to tighten a nut, using a screwdriver to tighten a bolt, steadying a drill as it screws, or using a crow bar to pry open a box. We simplify this problem slightly and formalize it as:

- **Given:**
  - An environment of static obstacles
  - A rigid tool to manipulate
  - A rigid multi-fingered robotic hand controllable via single DOF joint torques and a base $SE(3)$ free joint
  - A target tool position p_O^d and orientation R_O^d
  - An external wrench w_{ext} (i.e., force and torque) to be applied to the tool at that pose indefinitely
- **Find:** A sequence of joint torques such that
  - The tool reaches its target pose without colliding with the environment or being dropped, and
  - The tool can maintain the target pose indefinitely under the applied w_{ext}.

A. Reduction to Grasp Sequence Planning

The in-hand manipulation problem presented above is very broad. Our work uses a hierarchical planning and control framework [9] to isolate the specific problem of grasp sequence planning. For each episode, a collision-free trajectory is planned for both the tool and the robot hand base through the environment to the target pose. This trajectory and the physical description of the tool (i.e. geometry, mass, moment of inertia, and friction coefficient) constitute the input provided to the grasp sequence planning algorithm at the start of each episode.

Low level continuous joint torque control is provided by the Contact Controller. This is a combination of inverse dynamics calculations, admittance control, and PID control that accepts both a reference tool pose and a reference grasp. It will then output joint torques to realize this pose and grasp given the physical state of the system.

Therefore, the general in-hand tool manipulation problem reduces to grasp sequence planning. At each time step, given the desired tool trajectory and system state, determine the reference grasp that needs to be sent to the Contact Controller.

B. Grasp Sequence MDP Definition

We formulate the Grasp Sequence Planning problem as a finite-horizon Markov decision process (MDP) dependent on the Contact Controller [9].

- **State Space $\mathcal{S}$:**
  - Robot joint positions q
  - Robot base position p_h and orientation R_h
  - Tool pose p_O, R_O, plus linear and angular velocities $\dot{p_O}, \omega_O$
  - Episode Input: desired tool trajectory $\{\widehat{p_O}(t), \widehat{R_O}(t)\}$, stopping time T, tool model, and its estimated inertial properties
- **Action Space $\mathcal{A}$:** Reference Grasp $\hat{G}$ for the Contact Controller
- **Transition Function $\mathcal{S} \times \mathcal{A} \to \mathcal{S}$:** the composition of the Contact Controller and the environment's physical dynamics
- **Reward $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$:** For $t < T$, return $R_{min} << 0$ if the tool or robot collides with the environment, and 0 otherwise. The reward for $t = T$ is described below.

At the end of the reference trajectory, i.e., at time T when the tool should have reached its desired pose $\{\widehat{p_O}(T), \widehat{R_O}(T)\}$, we assume w_{ext} is applied to the tool for an unknown time T_{eval}. Again, under any collision, the episode ends with the minimum reward R_{min}. Otherwise, the following cost is incurred:

$$\int_{t=T}^{T+T_{eval}} \left[\|p_O(t) - \widehat{p_O}(T)\|_2^2 + angle(R_O(t), \widehat{R_O}(T)) \right] dt \tag{1}$$

This mean-squared error calculation captures the ability of the grasp to maintain a constant pose under the desired wrench, which we assert is equivalent to applying that wrench to a static object at the given pose.

Algorithm 1: DP Using Bellman Backups

Input : States $\mathcal{S}$, Actions $\mathcal{A}$, Reward $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, Transition $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$, Horizon T, start state s_0

Output : Policy $\pi : \mathcal{S} \to \mathcal{A}$ and Value $V(s_0)$

$V \leftarrow array(|\mathcal{S}| \times T)$; $\pi \leftarrow array(|\mathcal{S}| \times T)$

$V[:, T-1] \leftarrow$ Terminal Reward for all $s \in \mathcal{S}$

for $t \leftarrow T-2, T-3, ..., 0$ **do**

  for $s \in \mathcal{S}$ **do**

    $V[s,t] \leftarrow \max_a R(s,a) + V[\mathcal{T}(s,a), t+1]$

    $\pi[s,t] \leftarrow \arg\max_a R(s,a) + V[\mathcal{T}(s,a), t+1]$

return $\pi[s_0, 0]$, $V[s_0, 0]$

IV. Preliminaries

A. DP with Bellman Backups

Consider any general MDP defined by a finite state space $\mathcal{S}$ with initial state s_0, a finite action space $\mathcal{A}$, some transition function $a_{t+1} = \mathcal{T}(s_t, a_t, w_t)$ (where $a_i \in \mathcal{A}$, $s_i \in \mathcal{S}$ and w_i is a random variable drawn from a known distribution), some reward function $R(a_t, s_t, w_t)$, and a finite time horizon $t \in [0, T]$. DP [25] can be used to find a policy $a_t = \pi(s_t)$ that maximizes the expected total reward $\sum_{t=0}^{T} \mathbb{E}_w [R(\pi(s_t), s_t, w)]$.

The solution relies on the *principle of optimality*. If a hypothetical policy is optimal from some time t until the end, then it contains within it the optimal policy from any intermediate time to the end. This principle allows the original MDP to be broken into simpler sub-problems by starting at the end and optimizing backwards.

One method for doing this is *finite value iteration*, or the *Bellman backup*, outlined in Algorithm 1. Define the *value*, or reward-to-go, of a state at a given time to be the sum of rewards that the optimal policy can obtain by starting at that state. We can write this value as a recursive function dependent on the value of the following states. For simplicity, we consider a deterministic reward and transition function.

$$\begin{aligned} V(s,t) &= \max_{\pi} \sum_{t'=t}^{T} R(\pi(s'_t), s'_t) \\ &= \max_{a} [R(a, s_t) + V(s_{t+1} = \mathcal{T}(a, s_t))] \end{aligned} \tag{2}$$

By beginning at the final time and working backwards until the start state, this algorithm can solve the initial problem in $O(T|\mathcal{S}||\mathcal{A}|)$ time. For very large or continuous state spaces, the set of reachable states can be determined by building a tree with all possible actions. The problem then reduces to an expectimax search over the tree with complexity $O(|\mathcal{A}|^T)$.

B. Online Lookahead and Rollout

DP as described above has several limitations. It requires an accurate model of both the reward and transition functions, discrete state and action spaces, and the computation capacity to evaluate the Bellman backups for all possible states and actions. Additionally, the lack of any function approximation means that small perturbations in the reward model require DP to be completely re-run in order to recover the optimal policy. Therefore, solving the in-hand manipulation problem

Algorithm 2: l-step Lookahead and m-step Rollout

Input : States $\mathcal{S}$, Actions $\mathcal{A}$, Reward $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, Transition $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$, start state s_0

Input : Baseline Policy $\pi : \mathcal{S} \to \mathcal{A}$. *Optional:* Value Estimate $V : \mathcal{S} \to \mathbb{R}$

Input : Lookahead depth $l >= 1$, Rollout depth m

Output : Recommended action $a \in \mathcal{A}$

Function `Rollout` (*State* s) **:**
  $v \leftarrow 0$
  for $t \in [0, m]$ **do**
    $v = v + R(s, \pi(s))$
    $s = \mathcal{T}(s, \pi(s))$
  return $v + V(s)$

Function `Lookahead` (*Current* l', *State* s) **:**
  if $l' = 1$ **then**
    return $\arg\max_a [R(s, a) +$ Rollout$(\mathcal{T}(s, a))]$
  else
    return $\arg\max_a [R(s, a) +$ Lookahead$(l' - 1, \mathcal{T}(s, a))]$

return Lookahead(l, s_0)

`/* Can return argmax or max as needed. */`

using DP is not feasible for real-world applications, where we expect a different context per episode and online variations in the transition and/or reward functions are inevitable.

In general, DRL solutions can overcome all these limitations using the function approximating power of neural networks. However, this approach incurs a trade-off between sample complexity and robustness to distributional shift. RL needs large amounts of valid data to approximate the transition and reward functions, and it can be thwarted by a test-time environment that differs substantially from the training environments. Ideally, we should be able to adapt an RL-trained policy to a new or perturbed environment without incurring significant training costs or resorting to re-calculating the optimal policy from scratch with DP.

To address these limitations, we adopted the online l-step lookahead and m-step rollout algorithm [19] to solve the in-hand manipulation problem in a more data efficient and robust manner. Outlined in Algorithm 2, lookahead can be interpreted as performing DP with a time horizon l (implemented as expectimax search). The terminal reward is determined by rolling out any policy (e.g., an RL-derived policy) for m steps or until the end of the episode. In the former case, a trained value function (common in model-free RL such as PPO [26]) can be used as the terminal reward of the rollout. Importantly, the search tree and rollouts can be executed in a *different* environment from that used to train the RL policy.

Lookahead provides tunable hyperparameters l and m to manage the trade-off between the high computational complexity of DP and any sub-optimality present in the underlying RL model in the novel environment.

V. Scheme and Surrogate Model

We propose to solve the MDP outlined in Section III-B with a trained RL policy augmented at test time (i.e., online) with the DP-based procedures described above. The actual lookahead and rollout occurs by alternating between the RL policy (given a state, output a grasp) and a surrogate environment model (given a grasp, predict the next state) implemented in OpenAI's Gym [27]. This model is populated with the episode input (i.e., desired trajectory and description of the robot and tool) at test time and utilizes discretization and a shaped dense reward function that are relatively quick to compute. Even given these approximations, we hypothesize that the result is an augmented RL policy that can adapt to previously unseen episodes without additional training.

A. Discrete Action Space

We rely on the assumption that each link between the robot and the contactable portion of the tool is convex so we can guarantee that each link makes contact with the tool at only one point. From there, the complete set of possible reference tool grasps G is discretized by defining a finite countable set of contacts (consisting of one point on the robot and one on the tool) for each robot link, including a "null" contact that is the removal of the given link from the tool.

Due to the current structure of the Contact Controller, this assumption is subject to the restriction that only one robot link can be modified (either added, removed, or slid) at a time. Therefore, in addition to the null action that maintains the current reference grasp, an action consists of selecting which link to modify and its new reference contact point.

B. Shaped Reward Function

The reward function is constructed to be correlated with the odds of grasp failure (i.e., the tool falling) without full physical simulation by splitting kinematics and dynamics calculations into *IK Error* and *Wrench Error* respectively.

a) IK Error: IK is performed iteratively by collision-avoiding gradient descent from some fixed starting joint configuration, targeting tool contact points p_O. The result is a new joint configuration and a set of effective contact points p_R on the link in the world reference frame. In this context, IK Error is defined as $\max_{all\ links} \|p_R - p_O\|$, i.e., the worst error between the target contact point on the link and the target contact point on the tool.

b) Wrench Error: This is the solution to the following constrained optimization problem: find the set of contact forces within the friction cone (or on the edge in case of sliding contact) that minimizes the difference between the net wrench on the tool and the desired net wrench. An additional term in this metric penalizes the sum of the square norm of each contact force, which in turn penalizes grasps with few links and high contact forces.

Thus, the final reward function is constructed as follows:

- R_{min} if either IK fails with collision (including self-collision) or any *Wrench Error* exceeds a falling threshold; otherwise:
- *Wrench Error* against gravity and desired tool motion after realizing the new grasp at the current tool position with IK +

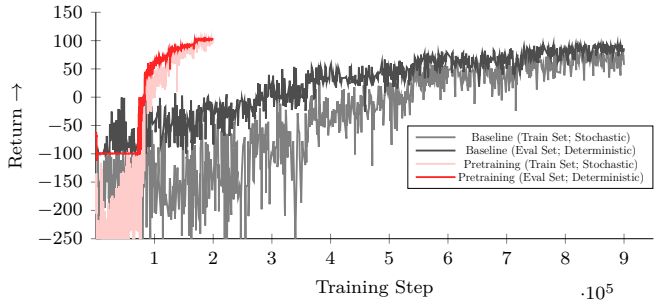


Fig. 3: Training curves for the RL model with and without DP pretraining. Lighter lines are the performance of the stochastic policy on the training set. Darker lines are the performance of the deterministic policy on the validation set, used to determine the training stop time. With DP pretraining, the RL policy can achieve the same return in four times fewer training iterations.

- *IK Error* realizing the new grasp at the desired tool position +
- *Wrench Error* against just gravity after this second IK calculation +
- *Wrench Error* against w_{ext} at end time T +
- A small penalty for redundant actions (i.e., commanding a non-null action that does not result in a changed grasp)

State transition is performed simultaneously with reward calculation. The output of the IK procedures sets the joint positions of the robot hand that are used to determine the approximate system state without full physical simulation.

C. Discrete State Space

The discrete grasp command and the trajectory waypoints themselves do not make the state space discrete. No restrictions are imposed on the observed tool position or joint states, which remain continuous at test time. In fact, a discrete state space is not necessary for lookahead, rollout, or RL training (which consists of rollouts to collect data for policy iteration).

However, discretization is necessary to tractably use Bellman backups to completely "solve" an environment, and we use it for DP pretraining and some of our baseline policies. We do this by making the following assumptions:

- The tool perfectly adheres to the reference trajectory. Therefore, all possible current tool positions come from a discrete set of waypoints.
- A given grasp is always realized as well as possible, as defined by running the IK procedure in Section V-B with a fixed initial joint configuration. This defines a unique joint configuration for a given discrete tool position and reference grasp.

From these assumptions as well as a given reference trajectory, grasp, and discrete time step, we can determine a unique *discrete state*. For Bellman-backup-based DP, we can operate entirely on these discrete states.

D. DP Pretraining Procedure

Deep RL algorithms that follow the "actor-critic" model involve estimating two functions: given any state, the *policy function* recommends an action, and the *value function* estimates the value (i.e. sum of future rewards). By performing DP to completely solve the discretized environment and recording intermediate results, we can determine the optimal action and the theoretical value of each state. In effect, we generate a behavior cloning dataset that can be used to directly train these networks with any supervised learning algorithm.

DP (base)	RL	$l = 1$	$l = 2$	DP (opt)
1.34s	1.23s	2.68s	4.30s	151.79s

TABLE I: Online Wall-clock Time (s) Per Unseen Trajectory

This procedure yields two key benefits. First, while the DP solution is not optimal for the non-discrete setting, the resulting policy is likely closer to optimal than just using random initial network weights. This is supported empirically by the good performance of the DP solution within the non-discrete simulation (see Fig. 4b). Second, running cross-validation on the DP-generated dataset can allow for easy hyperparameter tuning. For example, in our case, we used DP pretraining to tune our actor and critic network sizes and select ReLu over Tanh as the activation function. As a result, we could train our RL models to the same validation-set performance with 4x fewer training steps, as shown in Fig. 3.

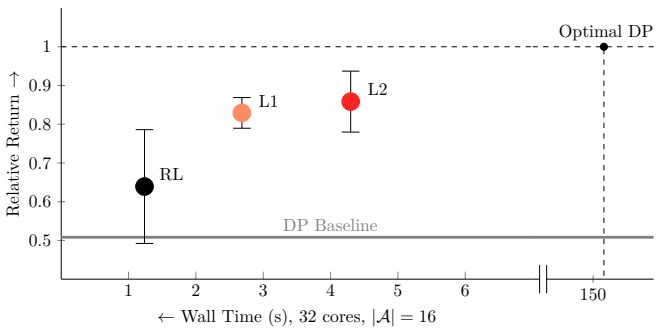
VI. Evaluation

We evaluated our surrogate model and online augmentation scheme in simulation with Honda's 4-fingered robotic hand [28] and a rigid 0.35kg (m_0) adjustable wrench, as shown on the left side of Fig. 2. The number of possible contact points per link were defined as follows: thumb distal (3 points), index distal (3 points), index medial (2 points), middle distal (4 points), middle medial (2 points), ring distal (2 points).

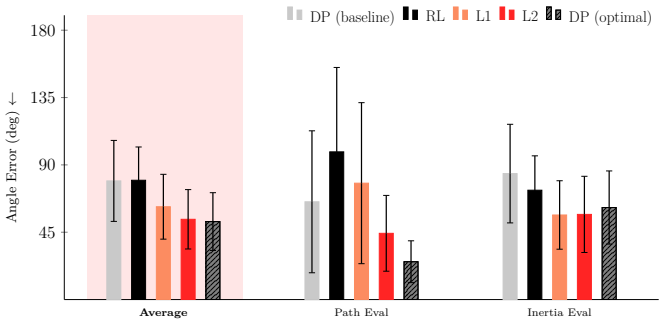
The product of these contact points yields 288 discrete states and reference grasps, and the sum yields 16 possible actions (i.e., moving only one finger at a time). All wall-clock timing calculations were done on a 32-core Intel(R) Xeon(R) Silver 4216 clocked at 2.10GHz.

A. Experiment Design

We evaluated this approach on a series of tool trajectories. The *nominal reference trajectory* started the wrench on a planar surface at the origin, i.e., 15cm below the palm of the robot. The first waypoint was at $(0, 0, 0.014)$, and 15 more waypoints moved the wrench to $(0, 0, 0.06)$ according to the reference velocity and acceleration before applying a positive 1Nm torque about the Z-axis relative to the center of mass. From the nominal trajectory, we varied the height of both the initial and final waypoints, torque direction, tool mass, and center-of-mass y-offset to create a training set and two evaluation sets. The training set included 9 reference trajectories: the nominal trajectory, the nominal trajectory with a negative final torque, and height variations from -6mm to +8mm in increments of 2mm, where negative height variants (-6, -4, and -2mm) experienced a negative final torque, and vice versa. The first evaluation set measured robustness to path deviation and included 7 trajectories: height variation from -6mm to +8mm in increments of 2mm (excluding the nominal 0mm), where the *positive* height variants (+2, +4, +6, and +8mm) experienced a negative final torque, and vice versa. The second evaluation set measured robustness to mass and center-of-mass deviation and included 20 trajectories: masses varied from $0.5m_0$ to $2m_0$ in increments of $0.5m_0$, and the center-of-mass y-offset varied from $-2cm$ to $2cm$ in increments of $1cm$.



(a) Average return (relative to the optimal policy) of RL, 1-step Lookahead (L1), and 2-step Lookahead (L2), as evaluated within the surrogate environment model. DP Baseline is the optimal policy across previously seen trajectories. Error bars represent 95% confidence. Lookahead significantly outperformed RL ($P < 0.05$) with only a few seconds more computation time on a 32-core machine.



(b) Average angle error of the tool while applying desired final torque. DP (optimal) was solved on each evaluation trajectory. DP (baseline) was solved only on the training trajectories. Error bars represent 95% confidence. These results suggest that our surrogate model is correlated with the simulation environment and that lookahead (L1, L2) can approach DP optimal performance. Note that DP "optimal" is no longer guaranteed to be optimal in the full simulation, as it is solved on the surrogate model.

Fig. 4: Results on the 27 evaluation episodes for (*left*) surrogate environment model and (*right*) physical simulation.

B. Algorithms

Table I shows the online computation time (i.e., excluding training time) for each algorithm. For reference, each trajectory took approximately 2 minutes to run in simulation at 1x (i.e. real-world) speed.

a) DP Baselines: For each trajectory, the reward associated with each discrete state-action-time triplet was recorded, for a total of approximately 73k (288 states $\times$ 16 actions $\times$ 16 time steps) executions of `env.step()`. Unreachable states (e.g., requiring moving more than one finger at a time) were assigned a reward of $-\infty$. We then ran DP on this data to determine the optimal grasp sequence for that trajectory under discretization. We refer to the DP solution on the nominal trajectory as the "DP Baseline" and on each evaluation trajectory as "Optimal DP." In Table I and Fig. 4a, DP solution computation time counts against "Optimal DP," which must be solved during test time after the evaluation trajectory is revealed, but not against "DP Baseline," which is solved at training time. DP benefits from a 32x speedup from parallelization on our system.

b) RL Baseline: We used the PPO Clip Agent from TFAgents [29] 3-layer feed-forward policy and value networks and an Adam optimizer with a learning rate decreasing piecewise-linearly. We generated a gym training environment that cycled through all training trajectories, and RL states were not limited to the discrete states of DP.

c) Lookahead Algorithms: We investigated the 1- and 2-step lookahead algorithms ("L1" and "L2," respectively). Since the episode lengths were short (at 16 time steps maximum), we ran our RL-baseline policy all the way to the end of the episode for the rollout portion of the algorithm (i.e., $m = T - t - l \quad \forall t$). Each rollout can run in parallel, yielding a 16x ($|\mathcal{A}|$) speedup for L1 and a full 32x speedup for L2 (number of cores).

C. Results

1) Surrogate Model: We evaluated all trained algorithms on the evaluation trajectories in our surrogate environment model. Fig. 4a presents return results, with significance determined by running a 1-way ANOVA with standard weights and samples correlated (by trajectory). All algorithms significantly outperformed the DP Baseline, and lookahead significantly outperformed RL ($P < 0.05$). While we only tested up to L2, the results are consistent with the idea that lookahead experiences diminishing returns with larger trees [19]. Therefore, most of the improvement comes from first step (L1), requiring a relatively small amount of additional computation.

2) Simulation: We evaluated the applicability of the surrogate model to real manipulation in a full physics simulation using a rigid-body physics engine [9]. The original MDP reward (Equation 1) is implemented by manually applying the external wrench w_{ext} at the end of the trajectory and measuring position and angle error. Since w_{ext} consisted of only a torque term, there was little difference in the position term. Therefore, Fig. 4b shows only the angle error term. If the tool collided with the floor, the maximum error of 180 degrees was assigned.

The fact that "Optimal DP" performed best across the evaluation set suggests that the shaped reward of our surrogate model is correlated with the desired MDP reward function. Additionally, these results are consistent with the previous ones in that the lookahead algorithm generally outperformed baseline DP and RL, approaching the optimal DP performance.

VII. Conclusion

We proposed a data-efficient and robust approach for solving the in-hand tool manipulation problem and evaluated our results in simulation using a dexterous hand that performs tool-use under modeling uncertainties and task variations. Results show that, given even a discrete approximate environment model, DP can be used to augment both the training and execution of RL policies with minimal online computational overhead. We believe that these insights can be applicable to any problem where such a model can be constructed.

Future work opportunities include action space expansion to enable performing additional manipulation primitives. Though a larger or continuous action space will make the lookahead tree more complex to evaluate, we could remedy this using a heuristic-based search technique. Implementing the proposed algorithms on the hardware is underway, and we will then evaluate results on the real system.

REFERENCES

[1] A. Okamura, N. Smaby, and M. Cutkosky, "An overview of dexterous manipulation," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 1, Apr. 2000, pp. 255–262 vol.1, iSSN: 1050-4729.

[2] H. Van Hoof, T. Hermans, G. Neumann, and J. Peters, "Learning robot in-hand manipulation with tactile features," in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*. IEEE, 2015, pp. 121–127.

[3] V. Kumar, E. Todorov, and S. Levine, "Optimal control with learned local models: Application to dexterous manipulation," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 378–383.

[4] T. Li, K. Srinivasan, M. Q.-H. Meng, W. Yuan, and J. Bohg, "Learning Hierarchical Control for Robust In-Hand Manipulation," *arXiv:1910.10985 [cs]*, Oct. 2019, arXiv: 1910.10985. [Online]. Available: http://arxiv.org/abs/1910.10985

[5] C. Yu and P. Wang, "Dexterous Manipulation for Multi-Fingered Robotic Hands With Reinforcement Learning: A Review," *Frontiers in Neurorobotics*, vol. 16, 2022. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnbot.2022.861825

[6] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray *et al.*, "Learning dexterous in-hand manipulation," *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, 2020.

[7] S. Nasiriany, H. Liu, and Y. Zhu, "Augmenting reinforcement learning with behavior primitives for diverse manipulation tasks," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 7477–7484.

[8] O. Kroemer, S. Niekum, and G. Konidaris, "A review of robot learning for manipulation: Challenges, representations, and algorithms," *The Journal of Machine Learning Research*, vol. 22, no. 1, pp. 1395–1476, 2021.

[9] R. S. Zarrin, K. Yamane, and R. Jitosho, "Hybrid learning- and model-based planning and control of in-hand manipulation," 2022. [Online]. Available: https://arxiv.org/abs/2209.10040

[10] N. C. Dafle, A. Rodriguez, R. Paolini, B. Tang, S. S. Srinivasa, M. Erdmann, M. T. Mason, I. Lundberg, H. Staab, and T. Fuhlbrigge, "Extrinsic dexterity: In-hand manipulation with external forces," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 1578–1585.

[11] N. Chavan-Dafle and A. Rodriguez, "Prehensile pushing: In-hand manipulation with push-primitives," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 6215–6222.

[12] Y. Karayiannidis, K. Pauwels, C. Smith, D. Kragic *et al.*, "In-hand manipulation using gravity and controlled slip," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 5636–5641.

[13] Y. Bai and C. K. Liu, "Dexterous manipulation using both palm and fingers," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 1560–1565.

[14] M. Liarokapis and A. M. Dollar, "Deriving dexterous, in-hand manipulation primitives for adaptive robot hands," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 1951–1958.

[15] S. Abondance, C. B. Teeple, and R. J. Wood, "A dexterous soft robotic hand for delicate in-hand manipulation," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5502–5509, 2020.

[16] C. K. Liu, "Dextrous manipulation from a grasping pose," in *ACM SIGGRAPH 2009 papers*, 2009, pp. 1–6.

[17] I. Mordatch, Z. Popović, and E. Todorov, "Contact-invariant optimization for hand manipulation," in *Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation*, 2012, pp. 137–144.

[18] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm," Dec. 2017. [Online]. Available: https://arxiv.org/abs/1712.01815v1

[19] D. Bertsekas, "Lessons from AlphaZero for Optimal, Model Predictive, and Adaptive Control," *arXiv:2108.10315 [cs, math]*, Aug. 2021, arXiv: 2108.10315. [Online]. Available: http://arxiv.org/abs/2108.10315

[21] R. Jena, C. Liu, and K. Sycara, "Augmenting GAIL with BC for sample efficient imitation learning," Jan. 2020. [Online]. Available: https://arxiv.org/abs/2001.07798v4

[22] I. Radosavovic, X. Wang, L. Pinto, and J. Malik, "State-only imitation learning for dexterous manipulation," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 7865–7871.

[23] A. Singh, H. Liu, G. Zhou, A. Yu, N. Rhinehart, and S. Levine, "Parrot: Data-driven behavioral priors for reinforcement learning," *arXiv preprint arXiv:2011.10024*, 2020.

[24] A. Nagabandi, K. Konolige, S. Levine, and V. Kumar, "Deep dynamics models for learning dexterous manipulation," in *Conference on Robot Learning*. PMLR, 2020, pp. 1101–1112.

[25] D. Bertsekas, *Dynamic Programming and Optimal Control: Volume I*. Athena Scientific, 2012, google-Books-ID: qVBEEAAAQBAJ.

[26] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017. [Online]. Available: https://arxiv.org/abs/1707.06347

[27] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[28] T. Hasegawa, H. Waita, T. Kawakami, Y. Takemura, T. Ishikawa, Y. Kimura, C. Tanaka, K. Sugiyama, and T. Yoshiike, "Powerful and dexterous multi-finger hand using dynamical pulley mechanism," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 707–713.

[29] S. Guadarrama, A. Korattikara, O. Ramirez, P. Castro, E. Holly, S. Fishman, K. Wang, E. Gonina, N. Wu, E. Kokiopoulou, L. Sbaiz, J. Smith, G. Bartók, J. Berent, C. Harris, V. Vanhoucke, and E. Brevdo, "TF-Agents: A library for reinforcement learning in tensorflow," https://github.com/tensorflow/agents, 2018, [Online; accessed 25-June-2019]. [Online]. Available: https://github.com/tensorflow/agents