Learning Robot In-Hand Manipulation with Tactile Features

Herke van Hoof[1], Tucker Hermans[2], Gerhard Neumann[1] and Jan Peters[1,3]

***Abstract*— Dexterous manipulation enables repositioning of objects and tools within a robot's hand. When applying dexterous manipulation to unknown objects, exact object models are not available. Instead of relying on models, compliance and tactile feedback can be exploited to adapt to unknown objects. However, compliant hands and tactile sensors add complexity and are themselves difficult to model. Hence, we propose acquiring in-hand manipulation skills through reinforcement learning, which does not require analytic dynamics or kinematics models. In this paper, we show that this approach successfully acquires a tactile manipulation skill using a passively compliant hand. Additionally, we show that the learned tactile skill generalizes to novel objects.**

I. Introduction

Object and tool manipulation stands as a fundamental problem in robotics. Often, the manipulated object needs to be held in a specific configuration: cups should be held upright, and screwdrivers must be held firm during use. Objects cannot always be picked up in these configurations; however, in-hand manipulation enables a robot to reconfigure an object [1]. We believe in-hand manipulation to be a vitally important capability for robots to achieve real-world tasks.

Most methods for in-hand manipulation rely on exact models of both the hand and the objects. These models are used by planning, control and optimization methods to manipulate known objects [2–8]. To manipulate unknown objects as well, systems could be designed to reactively adapt to the object rather than requiring an exact model.

Adaptation can be realized passively through compliant hardware. Compliant hands are able to physically adapt to an object's shape. Hence, even with a simple control strategy, many objects can be grasped with such hands [9, 10]. The finger configuration of such hands depends on both the applied controls and the interaction with the environment, which is hard to model.

In addition to passive adaptation, sensory feedback can be exploited to make skills more adaptive. Tactile and haptic feedback are especially useful for in-hand manipulation. Apart from providing information on the object's pose and contact normals [11–17], tactile sensing can provide greater robustness to variations in object properties [18–21], perturbations [22, 23], and sensing errors [10, 18]. Tactile sensing can also aid by detecting grasp instabilities and slippage that can occur while manipulating an object [13, 20, 24, 25].

Fig. 1: Using reinforcement learning and tactile feedback, a rolling primitive can be learned on an under-actuated compliant robot hand.

Tactile and haptic sensors have been shown to improve grasping performance [10, 18–21, 24, 25], and are promising for in-hand manipulation. Strategies for feedback-based in-hand manipulation [13–17] have been applied to unknown objects, but commonly only work for fully actuated hands with known dynamics and kinematics models. Exact models, however, are often not available for compliant robots.

In this paper, we propose to directly learn a control policy for the unknown system. This approach does not rely on knowing the robot's dynamics or kinematics, and can therefore be applied to under-actuated compliant robots. We formalize the in-hand manipulation problem as a Markov Decision Process (MDP). The robot learns a policy for this MDP using non-parametric relative entropy policy search, a reinforcement learning method that combines smooth policy updates with the ability to learn non-linear control policies [26]. We evaluate this method on the task of learning to roll an object between the fingertips of the compliant ReFlex robot hand shown in Fig. 1. To our knowledge, this is probably the first demonstration of reinforcement learning to learn an in-hand manipulation skill.

In the next section, we discuss related work on manipulation. In Sec. III, we explain the platform and the task in more detail. Subsequently, we provide details on the reinforcement learning method. Then, we introduce our experimental set-up and present the results. Finally, we present our conclusions and discuss possible future work.

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007–2013) under grant agreements #610967 (TACMAN) and #270327 (CompLACS).

[1]TU Darmstadt, Computer Science Department.
{hoof, neumann, peters}@ias.tu-darmstadt.de
[2]School of Computing, University of Utah.
thermans@cs.utah.edu
[3]Max Planck Institute for Intelligent Systems.

II. Related work on manipulation and learning

In this section, we discuss related work on in-hand manipulation with and without tactile sensing, as well as reinforcement learning for various kinds of object manipulation.

A. Planning for in-hand manipulation

Many approaches for in-hand manipulation employ a planning perspective. The assumption here is that analytic descriptions of both the hand and the object are known beforehand, so planning approaches and control methods can be used to find open-loop trajectories.

Such approaches have been used successfully for finger gaiting [2], manipulation with a rolling contact [2–5], sliding [5], in-grasp-manipulation [6], and rolling objects over the hand palm [7]. Optimization techniques have also been used for a variety of in-hand motions [8]. We are, however, interested in manipulating objects for which no model is available, using an underactuated hand that is hard to model exactly due to compliance. Therefore, these methods are not applicable.

B. In-hand manipulation using sensor feedback

Another perspective is to use sensory feedback, especially tactile sensing, to adapt to unknown object properties. The sensed contact locations can be used to define a 'virtual object frame' (the centroid of contacts), to act as a proxy for the object's location, helping in executing in-grasp manipulations [13–16]. A similar method is used in [17] for local control, augmented by a finger gaiting strategy for larger movements. Instead of tactile sensing, in-grasp control of an object can be realized using only internal joint angle and angular velocity sensors [27]. These methods, however, still assume knowing the hand dynamics and kinematics, whereas for our under-actuated and compliant hand exact models are not available.

If no accurate model of the robot is available, iterative learning control can be used to learn feedback controllers for manipulation [28]. However, this requires knowing the trajectory of the contact points in advance.

Sensory feedback has also been used in another way. When manipulating a held object, through planning methods or otherwise, grasp instabilities or slippage can occur. Multiple researchers, e.g. [13, 20, 24, 25], focused on detecting slippage and instabilities. Furthermore, [13] subsequently used tactile sensing to find stable grasps similar to the current grasp, and adapt the current grasp. These methods can be used to augment other strategies.

C. Reinforcement learning for manipulation

In previous work, reinforcement learning (RL) has been used to learn how to manipulate objects on a symbolic level [29]. For low-level motor control, however, a sub-symbolic level is more suitable. Continuous RL has been used for reaching and grasping objects [30–32], as well as for the transportation of grasped objects [12, 32–35]. These methods are driven by feedback from tactile sensing [12, 35], Cartesian- and joint-space coordinates [33, 34], or both [32]. However, these approaches have not been used to learn *in-hand manipulation*, i.e., changing the object's pose with respect to the hand.

The problems of grasping and door opening have been addressed in [36]. Like in-hand manipulation, these tasks require stability and force control. For that reason, [36] uses RL to optimize both a Cartesian trajectory as well as desired forces along this trajectory. However, the disadvantages of optimizing a trajectory is that it assumes a fixed starting location. In this paper, we will instead focus on learning time-independent policies.

III. Learning policies with tactile features

In this paper, we aim at demonstrating the possibility of learning feedback controllers for in-hand manipulation using reinforcement learning on an underactuated, compliant platform. First, we will describe the platform that we use and the task we want to perform. Then, we will address how we represent the task as a Markov Decision Process (MDP). In Sec. IV, we will explain how non-parametric policy search can be used to solve this MDP.

A. Compliant robotic platform with tactile sensing

For the experiment we use the three fingered, underactuated ReFlex robot hand. A single tendon drives each two-link finger, which include position sensors on the tendon spool and proximal joint. The hand has nine MEMS barometers embedded in each finger to generate tactile feedback signals. Two of the fingers can additionally be controlled by a single, coupled pre-shape joint.

Difficulty arises in applying more conventional methods to this hand, as the state of the compliant connection between the proximal and distal finger segments depends on the history of the executed commands and there is no joint sensor measuring the angle between these two links. Underactuation also poses a problem for any method that relies on workspace control. Moreover, most of the sensors are quite noisy.

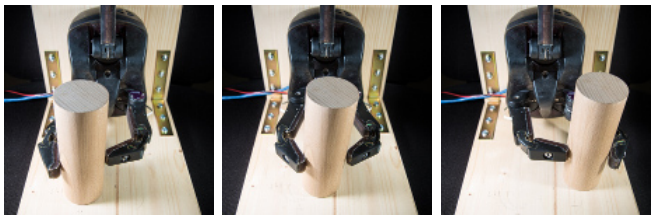
The hand is mounted with the fingers pre-shaped so that two fingers oppose each other. The third finger is unused in our experiments. The set-up is shown is Fig. 1.

B. In-hand manipulation task

We desire our robot to perform a rolling task on an object grasped between its two fingers. Since the robot can only control the individual fingers along one axis, the object is rolled horizontally between the robot's fingers, while sliding along the ground plane, as shown in Fig. 2. To keep control of the object, the robot should maintain contact with both of its fingers, while moving them from one side of its workspace to the other in a coordinated manner.

C. MDP formalization

In an MDP, an agent in state $\mathbf{s}$ selects an action $\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})$ according to a (possibly stochastic) policy π and receives a reward $\mathcal{R}_{\mathbf{s}}^{\mathbf{a}} \in \mathbb{R}$. We will assume continuous state-action spaces: $\mathbf{s} \in \mathcal{S} = \mathbb{R}^{D_\mathrm{s}}$, $\mathbf{a} \in \mathcal{A} = \mathbb{R}^{D_\mathrm{a}}$. If the Markov decision



(a) (b) (c)

Fig. 2: Illustration of the rolling task: from an initial state (a), the robot has to coordinate finger movement (b) to move the object to the goal location (c), while maintaining pressure. Compliance and under-actuation make such motions difficult to plan.

process is ergodic, for each policy π, there exists a stationary distribution $\mu_\pi(\mathbf{s})$ such that $\int_\mathcal{S}\int_\mathcal{A}\mathcal{P}^\mathbf{a}_{\mathbf{ss}'}\pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s})\mathbf{dads} = \mu_\pi(\mathbf{s}')$, where $\mathcal{P}^\mathbf{a}_{\mathbf{ss}'} = Pr(\mathbf{s}'|\mathbf{a},\mathbf{s})$. The goal of a reinforcement learning agent is to choose a policy such that the joint state-action distribution $p(\mathbf{s},\mathbf{a}) = \mu_\pi(\mathbf{s})\pi(\mathbf{a}|\mathbf{s})$ maximizes the average reward $J(\pi) = \int_\mathcal{S}\int_\mathcal{A}\ \pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s})\mathcal{R}^\mathbf{a}_\mathbf{s}\mathbf{dads}$.

To apply an RL method to solve this task, we have to formalize the task as an MDP by defining a state and action representation and a reward function. The transition probability is implicitly defined by the physical system, and not explicitly known.

D. State and action representation

The state of the system comprises the robot state as well as the relation between the robot and the object, as observed through the tactile sensors. Therefore, we define a six-dimensional state representation $\mathbf{s}$, where s_1, s_2 represent the tactile sensors, s_3, s_4 represent the two proximal link joint angles and s_5, s_6 represent the distal link angles.

The proximal link angles are given encoders q_l and q_r of the left and right finger, so $s_3 = q_\mathrm{l}, s_4 = q_\mathrm{r}$. The ReFlex hand has no sensor for the distal link angles, but the difference between the joint encoders and the motor encoders $m_\mathrm{l}, m_\mathrm{r}$ can be used as proxy for the distal segment's position, so $s_5 = m_\mathrm{l} - q_\mathrm{l}$, $s_6 = m_\mathrm{r} - q_\mathrm{r}$.

The tactile sensing is hardest to encode meaningfully, as for higher forces the sensor response signal can be highly non-linear. We obtained the best results by applying the following non-linear functions to sensor vectors $\mathbf{x}_\mathrm{l}$ and $\mathbf{x}_\mathrm{r}$

$$s_1 = \frac{2}{\pi}\arctan\left(\frac{\pi}{80}|\mathbf{x}_\mathrm{l}|_1\right), \quad s_2 = \frac{2}{\pi}\arctan\left(\frac{\pi}{80}|\mathbf{x}_\mathrm{r}|_1\right),$$

which scale the vector-valued sensors to single scalar values for the left and right distal sensors. Taking the l1-norm sums the absolute sensor value of all tactile sensors. This is important, as both positive or negative change in sensor pressure can occur depending on where contact is made. The factor of $\pi/80$ makes sure the $\arctan$ function is close to linear near 0, where differences are meaningful, and flattens out for large pressures where more of the variance is due to noise. Finally, the result is scaled between 0 and 1.

We define actions $\mathbf{a} = [a_\mathrm{l}, a_\mathrm{r}]^T$ as the velocity applied by the robot for two time steps (0.04 s), with limits of $\pm 2.5\mathrm{s}^{-1}$ for each finger. After each action, the system pauses for 0.06 seconds such that despite any observation delays, the correct resulting state is recorded.

E. Reward function

We want the robot to roll the object between its fingers to a goal location while maintaining pressure. To avoid unnecessarily large actions, we also punish the squared magnitude of the actions. Combined, this gives the reward function

$$r(\mathbf{s},\mathbf{a}) = w_1|\mathbf{a}|_2^2 + w_2 r_\text{press}(\mathbf{s}) + w_3 r_\text{goal}(\mathbf{s}),$$
$$r_\text{press} = |s_1 - s_\mathrm{d}| + |s_2 - s_\mathrm{d}|,\ r_\text{goal} = |s_3 - q_\mathrm{l,d}| + |s_4 - q_\mathrm{r,d}|,$$

where $s_\mathrm{d} = 0.7$ is the desired pressure value, $q_\mathrm{l,d} = 1.8$ and $q_\mathrm{r,d} = 1.0$ are the goal locations for the fingers, and w_1, w_2, and w_3 are trade-off factors. We found setting $w_1 = 0.8, w_2 = 80$, and $w_3 = 30$ achieved a good balance between the different goals and yielded good learning progress in practice.

IV. Learning optimal control policies

There are many approaches to solving MDPs. In this paper, we will focus on one of these, Relative Entropy Policy Search [37], as it provides an information-theoretic bound on the size of the policy update that provides smooth and safe updates in robotic experiments. Recently, we proposed a non-parametric version, NPREPS [26], that has the advantages of not requiring a parametric form of the value function or policy to be provided, and of being robust to noisy state transitions. As a consequence, the user does not need to define non-linear features for the sensors by hand, reducing design effort. Note that many RL algorithms require a discretization of the state space. Even though we only use two fingers, the state space is six-dimensional which makes discretization of the state-space impracticable: just four bins in each dimensions would already yield 4096 discrete states; but would still be too coarse for fine coordination.

A. Non-parametric REPS

REPS is formulated as the optimization problem

$$\max_{\pi,\mu_\pi} J(\pi) = \max_{\pi,\mu_\pi}\iint_{\mathcal{S}\times\mathcal{A}}\pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s})\mathcal{R}^\mathbf{a}_\mathbf{s}\mathbf{dads}, \tag{1}$$

$$s.t. \quad \iint_{\mathcal{S}\times\mathcal{A}}\pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s})\mathbf{dsda} = 1, \tag{2}$$

$$\forall s' \quad \iint_{\mathcal{S}\times\mathcal{A}}\mathcal{P}^\mathbf{a}_{\mathbf{ss}'}\pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s})\mathbf{dads} = \mu_\pi(\mathbf{s}'), \tag{3}$$

$$\mathrm{KL}(\pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s})||q(\mathbf{s},\mathbf{a})) \le \epsilon, \tag{4}$$

where Eqs. (1-3) specify the general reinforcement learning objective in a slightly unusual form, which will be convenient in deriving our algorithm. Equation (1) states that the joint state-action distribution should maximize the expected average reward, and Equation (2) constraints $\pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s})$

to be a probability distribution. With just these constraints, an optimizer could choose any state distribution, whereas in reality the state distribution is specified by the policy and the real-world transition dynamics $\mathcal{P}^{\mathbf{a}}_{\mathbf{ss}'}$ of fingers and object. Equation (3) enforces $\mu_\pi(\mathbf{s})$ to be the stationary distribution under $\pi(\mathbf{a}|\mathbf{s})$ under the actual system dynamics.

Equation (4) specifies an additional bound on the KL divergence between the proposed state-action distribution and sampling distribution q, that ensures smooth policy updates. In this equation,

$$\mathrm{KL}(p||q) = \int p(x) \log(p(x)/q(x)) \mathrm{d}x.$$

Reference distribution q is usually set to the state-action distribution induced by previous policies. Learning starts with samples from an initial explorative policy $\tilde{\pi}_0$, usually chosen to be a wide, uninformed distribution. The variance typically shrinks after every iterations, such that the policy converges to a (locally) optimal deterministic policy.

The solution to the optimization problem obtained through Lagrangian optimization is given by

$$\pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s}) \propto q(\mathbf{s},\mathbf{a}) \exp\left(\frac{\delta(\mathbf{s},\mathbf{a},V)}{\eta}\right), \text{with}$$
$$\delta(\mathbf{s},\mathbf{a},V) = \mathcal{R}^{\mathbf{a}}_{\mathbf{s}} + \mathbb{E}_{\mathbf{s}'}[V(\mathbf{s}')|\mathbf{s},\mathbf{a}] - V(\mathbf{s}) \quad (5)$$

where $V(\mathbf{s})$ and η denote Lagrangian multipliers [37]. The Lagrangian multiplier $V(\mathbf{s})$ is a function of $\mathbf{s}$ and resembles a value function, so that δ can be interpreted as the Bellman error. Therefore, the policy can be seen as a re-weighted version of the old distribution, with weights given by a soft-max of the advantage function: in other words, for higher expected future rewards, the action will be taken with a higher probability.

The Lagrangian multipliers are obtained through minimization of the dual function

$$g(\eta, V) = \eta\epsilon + \eta \log\left(\sum_{i=1}^{n} \frac{1}{n} \exp\left(\frac{\delta(\mathbf{s}_i,\mathbf{a}_i,V)}{\eta}\right)\right), \quad (6)$$

where the samples $(\mathbf{s}_i, \mathbf{a}_i)$ are drawn from $q(\mathbf{s},\mathbf{a})$. To calculate the Bellman error δ, the transition distribution is required. As this distribution is generally not known, δ needs to be approximated. In earlier work [26], we showed that this approximation can be done efficiently using kernel methods

$$\delta(\mathbf{s},\mathbf{a},\boldsymbol{\alpha}) = \mathcal{R}^{\mathbf{a}}_{\mathbf{s}} + \boldsymbol{\alpha}^T\left(\tilde{\mathbf{K}}_{\mathrm{s}}\boldsymbol{\beta}(\mathbf{s},\mathbf{a}) - \mathbf{k}_{\mathrm{s}}(\mathbf{s})\right), \text{with}$$
$$\boldsymbol{\beta}(\mathbf{s},\mathbf{a}) = (\mathbf{K}_{\mathrm{sa}} + \lambda I)^{-1}\mathbf{k}_{\mathrm{sa}}(\mathbf{s},\mathbf{a}). \quad (7)$$

In this equation, $\mathbf{K}_{\mathrm{sa}}$ and $\tilde{\mathbf{K}}_{\mathrm{s}}$ signify matrices containing the value of kernel functions between state-action pairs or states, respectively. $\mathbf{k}_{\mathrm{sa}}(\mathbf{s},\mathbf{a})$ and $\mathbf{k}_{\mathrm{s}}(\mathbf{s})$ are vectors of such values, whereas $\boldsymbol{\alpha}$ is a vector of coefficients that is obtained through maximization of the dual (6).

B. Fitting non-parametric control policies

The desirability of state-action pairs is given by (5). However, (5) can only be evaluated for sampled state-action pairs, as we only know $\mathcal{R}^{\mathbf{a}}_{\mathbf{s}}$ at those points. To obtain a generalizing policy, cost-sensitive Gaussian Processes (GPs) are used, as introduced in the cost-regularized kernel regression (CrKR) algorithm [38]. This policy requires a weighting factor for each data-point, which in the bandit case [38] was given by the obtained reward. These weighting factors need to be transformed to be strictly positive, as discussed in [39].

In REPS, such a transformation naturally appears as a result of solving the constrained optimization problem [26, 37], as given in (5). The resulting weighting factors take into account not only the immediate reward, but also the cumulative long-term reward through the value function V. Because generally we are working with samples from reference distribution q, the weighting factors w should be normalized by q: $w(\mathbf{a},\mathbf{s}) = \pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s})/q(\mathbf{a},\mathbf{s})$. The parameter η, that specifies the transformation and determines how greedy the optimization is, is a Lagrangian parameter that is naturally set through optimization of the dual function (6). The resulting weighting factors are

$$w_j \leftarrow \exp(\delta(\mathbf{s}_j,\mathbf{a}_j,\boldsymbol{\alpha})/\eta), \quad (8)$$

and they define a diagonal cost matrix C with $C_{jj} = w_j^{-1}$. The corresponding cost-sensitive Gaussian process policy

$$\tilde{\pi}(\mathbf{a}|\mathbf{s}) = \mathcal{N}(\mu(\mathbf{s}), \sigma^2(\mathbf{s})), \;\; \mu(s) = \mathbf{k}_{\mathrm{s}}(\mathbf{s})^T(\mathbf{K}_{\mathrm{s}} + \lambda\mathbf{C})^{-1}\mathbf{A},$$
$$\sigma^2(\mathbf{s}) = k + \lambda - \mathbf{k}_{\mathrm{s}}(\mathbf{s})^T(\mathbf{K}_{\mathrm{s}} + \lambda\mathbf{C})^{-1}\mathbf{k}_{\mathrm{s}}(\mathbf{s}), \quad (9)$$

where $\mathbf{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_n]^T$ is a matrix containing all sampled actions, and λ is a regularization parameter that is set with free kernel parameters using weighted maximum likelihood. Algorithm 1 shows all steps of our algorithm.

Using all past data makes the learning algorithm very slow, but a simple forgetting mechanism that only retains data from the M most recent policies addresses this issue sufficiently. In this paper, we will use $M = 3$.

Algorithm 1 The NPREPS algorithm

Require: Initial explorative policy $\tilde{\pi}_0$
for $i = 1, \ldots,$ max_iteration **do**
  generate roll-outs according to $\tilde{\pi}_{i-1}$
  minimize kernel-based dual:
    $\eta^*, \boldsymbol{\alpha}^* \leftarrow \arg\min g(\eta, \boldsymbol{\alpha})$ Eq. 6
  calculate kernel-based Bellman errors:
    $\boldsymbol{\beta}_j \leftarrow (\mathbf{K}_{\mathrm{sa}} + \lambda\mathbf{I})^{-1}\mathbf{k}_{\mathrm{sa}}(\mathbf{s}_j,\mathbf{a}_j)$ Eq. 7
    $\delta_j \leftarrow \mathcal{R}_j + \boldsymbol{\alpha}^{*T}\left(\tilde{\mathbf{K}}_{\mathrm{s}}\boldsymbol{\beta}_j - \mathbf{k}_{\mathrm{s}}(\mathbf{s}_j)\right)$ Eq. 7
  calculate the weighting factors:
    $w_j \leftarrow \exp(\delta_j/\eta^*)$ Eq. 8
  determine cost-sensitive GP:
    $\tilde{\pi}_i(\mathbf{a}|\mathbf{s}) = \mathcal{N}(\mu(\mathbf{s};\mathbf{w}), \sigma^2(\mathbf{s};\mathbf{w}))$ Eq. 9
end for

V. Experiments

For each trial in our experiments, we initialize the policy, and then go through ten iterations of gathering data with the current policy, and using this data to improve the policy according to Alg. 1. After these ten iterations, we evaluate how well the policy generalizes to different rollable (i.e., cylindrical) objects. Each of these steps will be explained in the following sub-sections.

A. Hand-coded baseline policy

As standard of comparison, we hand-coded a rough policy, defined as follows:

$$\mathbf{a}^T = \begin{cases} [0.00, -0.02] & \text{if } |\mathbf{x}_r|_1 > t_{\text{max}} \\ [0.02, -0.02] & \text{else if } |\mathbf{x}_l|_1 > t_{\text{min}} \text{ and } |\mathbf{x}_r|_1 > t_{\text{min}} \\ [0.02, \quad 0.01] & \text{else if } |\mathbf{x}_l|_1 < t_{\text{min}} \\ [0.01, \quad 0.02] & \text{otherwise,} \end{cases}$$

where $\mathbf{a}$, $\mathbf{x}_r$ and $\mathbf{x}_l$ are actions and pressure sensor values defined previously, and t_{max} and t_{max} are minimum and maximum sensor thresholds (set to 100 and 40 in our experiments). The first two cases move the object in the desired direction, the latter two increase pressure when needed.

B. Policy initializations

The system needs to be initialized with a starting policy. A completely random (e.g. Gaussian) policy is possible, but it might require many roll-outs to sufficiently cover the relevant state-space. Therefore, we use the rough hand-coded policy with additive Gaussian noise with a standard deviation of 80% of the action limit as the initial exploration policy. Although the hand-coded policy succeeds in exploring the relevant parts of the state-space, it is sub-optimal in terms of speed and coordination. We expect the final learned policy to outperform it (i.e., obtain higher average rewards).

C. Using finite roll-out lengths

The non-parametric REPS algorithm is designed for infinite-horizon settings. To work with finite episodes, a state-independent reset probability can be used that resets the system to an initial state [26]. If this is done, as a result the length of individual roll-outs follows a geometric distribution. However, the disadvantage is that this leads to a high variance in roll-out length, and so makes it hard to evaluate learning progress and compare across conditions.

To address this problem, in this paper, we represent the geometric distribution using 10 samples chosen at the $5^{\text{th}}, 15^{\text{th}}, \ldots, 95^{\text{th}}$ percentile of the distribution, leading to a roll-out length of 50 steps on average.

D. Evaluation

When training with a single object, even open-loop controllers could obtain high rewards by over-fitting to the object. In order to ensure the learned controller uses tactile sensing to adapt to the object, we use two objects of different dimensions in the training stage (shown in Fig. 3). We always switch objects after every roll-out. After performing ten roll-outs, the policy is updated according to Algorithm 1.

Fig. 3: The objects used in the experiment. The wooden cylinders on the left were used for training; the four objects on the rights were used to evaluate skill generalization.

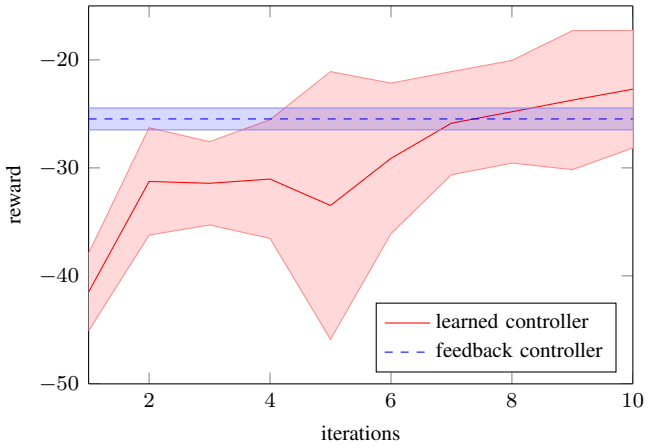


Fig. 4: Average reward and standard deviation across four independent learning trials. The feedback controller has been hand-tuned to successfully complete the tasks, matching its performance shows learning is successful. Each iteration represents approximately 500 time steps (50 s.) of robot data.

We perform two kinds of evaluation. The learned policies at each stage of learning are compared to the hand-coded feedback policy. Furthermore, the learned policies are executed without exploration noise to grasp various novel objects to evaluate the generalizability of the learned controller.

These objects are shown in Fig. 3, and they differ in properties such as diameter, surface texture, and weight (two of the containers are full, whereas the toy fire extinguisher and the pill bottle are empty). The policy, however, is only optimized for the training objects. As such, this evaluation shows the robustness margin of the learned policy; but the policy cannot generally be expected to perform optimally for objects that are not similar to the objects it has been trained on. For that reason, we only used cylindrical objects.

We perform four independent trials where we learn a policy from scratch and evaluate it on the various objects. We also perform four independent trials with the feedback policy as comparison standard.

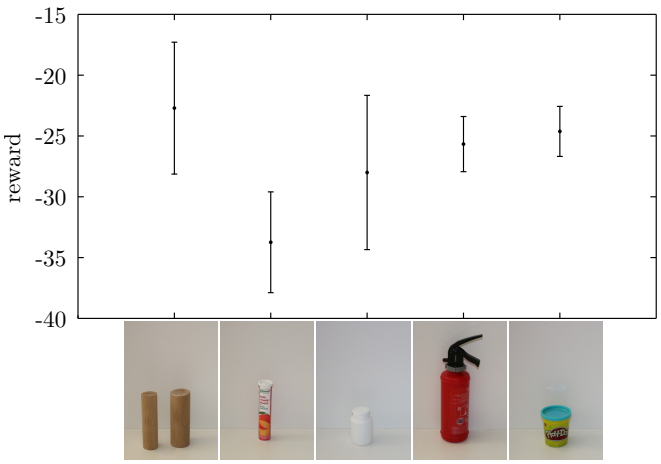


Fig. 5: Average reward and standard deviation obtained by the trained policies on four novel objects. Except for the vitamin container, which is hard to grasp due to its small diameter and low friction, we obtain performance close to the performance on the original training objects (shown by the leftmost bar).

VI. Results

The performance during the course of learning is shown in Fig. 4. This figure illustrates how the starting policy obtains quite poor results, which the learning algorithm successfully improves. The final policy, on average, obtains better rewards than the feedback policy. We observed from the robot's behavior, that the learned policy executes the desired movement quickly, but incurs the risk of slipping away from the object, leading to occasional very low rewards and, therefore, more variable rewards than the hand-coded feedback policy. From this graph, we see that we can successfully learn policies that are competitive with manually-designed policies.

The results for generalization over different objects are shown in Fig. 5. Here, we see that performance on three of the objects was comparable (if slightly worse) to the performance on the two wooden cylinders the policy was trained on. The performance on the container of vitamin pills, however, is markedly worse. This object is smaller than the objects that the policy was optimized for and has low surface friction. As the robot has not been trained to deal with these properties, this results in the robot losing grips much easier and consequentially obtaining lower rewards.

A closer analysis of the sensor data revealed an effect that made the problem harder. After applying a force in the direction of, or away from, the fingernail, the subsequent sensor behavior changed. Since this behavior is hard to predict and might make subsequent sensor readings ambiguous, this represents a substantial difficulty for learning algorithms. Nevertheless, the NPREPS algorithm was able to learn competitive policies for this task.

VII. Discussion and Conclusion

In this paper, we have employed a reinforcement learning method to learn a tactile skill on a compliant, under-actuated robot hand. We have shown, first of all, that such a technique is feasible for finding controllers for dexterous manipulation task even for an underactuated hand with unknown dynamics and kinematics. The learned policies obtain a higher average reward than a hand-coded feedback policy, although the learned policy is more risk-seeking and obtains a higher reward variance. We have also shown that the policies learned on two simple training objects can be generalized to novel objects with, in most cases, only a small loss in performance.

In one case of the generalization trials, the learned policy did not perform well. This was probably because the vitamin container's properties make it inherently different to manipulate (larger curvature with lower surface friction) and the policy has not been optimized for this particular object. We expect that re-training for new objects would enable learning policies to handle such objects, as well as objects of non-cylindrical shape.

We believe that the focus on learning a reactive feedback policy that directly depends on the tactile sensor data was one of the elements that made generalization successful. If we had learned trajectories, it would not be straightforward to adapt to different situations.

Compared to many of the existing methods for planning in-hand manipulations, our method has the advantage that we can apply it to an under-actuated, compliant hand without analytic models. The disadvantage, however, is that learning a policy requires system interaction. Our experiments show, that generalization to roughly similar objects is successful, so that once a policy is learned it can be generalized.

There are a couple of issues we would like to address in future work. By adapting the reward function to prefer lower-risk policies, e.g. by increasing the importance of maintaining grasp pressure, we want to learn controllers with lower variance. We also want to extend the method to learn grasps without an initial demonstration policy that generalizes to a wide range of initial object positions. Generalization over initial positions and learning without demonstration policy will require more data, and hence we will work on approximations that make our method applicable to large data sets.

References

[1] R. Ma and A. Dollar, "On dexterity and dexterous manipulation," in *ICAR*, 2011.

[2] L. Han and J. Trinkle, "Dextrous manipulation by rolling and finger gaiting," in *ICRA*, vol. 1, 1998.

[3] Z. Doulgeri and L. Droukas, "On rolling contact motion by robotic fingers via prescribed performance control," in *ICRA*, 2013.

[4] A. Bicchi and R. Sorrentino, "Dexterous manipulation through rolling," in *ICRA*, vol. 1, 1995.

[5] M. Cherif and K. Gupta, "Planning quasi-static fingertip manipulations for reconfiguring objects," *Trans. Robotics and Automation*, vol. 15, no. 5, 1999.

[6] K. Hertkorn, M. Roa, and C. Borst, "Planning in-hand object manipulation with multifingered hands considering task constraints," in *ICRA*, 2013.

[7] Y. Bai and K. Liu, "Dexterous manipulation using both palm and fingers," in *ICRA*, 2014.

[8] I. Mordatch, Z. Popović, and E. Todorov, "Contact-invariant optimization for hand manipulation," in *Symp. Computer Animation*, 2012.

[9] R. Deimel and O. Brock, "A novel type of compliant, underactuated robotic hand for dexterous grasping," in *RSS*, 2014.

[10] L. Jentoft, Q. Wan, and R. Howe, "Limits to compliance and the role of tactile sensing in grasping," in *ICRA*, 2014.

[11] P. Payeur, C. Pasca, A.-M. Cretu, and E. M. Petriu, "Intelligent haptic sensory system for robotic manipulation," *Trans. Instrumentation and Measurement*, vol. 54, no. 4, pp. 1583–1592, 2005.

[12] Y. Chebotar, O. Kroemer, and J. Peters, "Learning robot tactile sensing for object manipulation," in *IROS*, 2014.

[13] M. Li, Y. Bekiroglu, D. Kragic, and A. Billard, "Learning of grasp adaptation through experience and tactile sensing," in *IROS*, 2014.

[14] K. Tahara, S. Arimoto, and M. Yoshida, "Dynamic object manipulation using a virtual frame by a triple soft-fingered robotic hand," in *ICRA*, 2010.

[15] H. Maekawa, K. Tanie, and K. Komoriya, "Tactile sensor based manipulation of an unknown object by a multifingered hand with rolling contact," in *ICRA*, vol. 1, 1995.

[16] M. Li, H. Yin, K. Tahara, and A. Billard, "Learning object-level impedance control for robust grasping and dexterous manipulation," in *ICRA*, 2014.

[17] Q. Li, M. Meier, R. Haschke, H. Ritter, and B. Bolder, "Rotary object dexterous manipulation in hand: a feedback-based method," *Int. J. Mechatronics and Automation*, vol. 3, no. 1, pp. 36–47, 2013.

[18] K. Hsiao, S. Chitta, M. Ciocarlie, and E. Jones, "Contact-reactive grasping of objects with partial shape information," in *IROS*, 2010.

[19] J. Laaksonen, E. Nikandrova, and V. Kyrki, "Probabilistic sensor-based grasping," in *IROS*, 2012.

[20] T. Takahashi, T. Tsuboi, T. Kishida, Y. Kawanami, S. Shimizu, M. Iribe, T. Fukushima, and M. Fujita, "Adaptive grasping by multi fingered hand with tactile sensor based on robust force and position control," in *ICRA*, 2008.

[21] H. Dang and P. K. Allen, "Stable grasping under pose uncertainty using tactile feedback," *Autonomous Robots*, vol. 36, no. 4, pp. 309–330, 2014.

[22] P. Pastor, M. Kalakrishnan, L. Righetti, and S. Schaal, "Towards associative skill memories," in *Humanoids*, 2012.

[23] H. Zhang and N. Chen, "Control of contact via tactile sensing," *Trans. Robotics and Automation*, vol. 16, no. 5, pp. 482–495, 2000.

[24] J. Romano, K. Hsiao, G. Niemeyer, S. Chitta, and K. Kuchenbecker, "Human-inspired robotic grasp control with tactile sensing," *Trans. Robotics*, vol. 27, no. 6, pp. 1067–1079, 2011.

[25] Y. Bekiroglu, J. Laaksonen, J. A. Jorgensen, V. Kyrki, and D. Kragic, "Assessing grasp stability based on learning and haptic data," *Trans. Robotics*, vol. 27, no. 3, pp. 616–629, 2011.

[26] H. van Hoof, J. Peters, and G. Neumann, "Learning of non-parametric control policies with high-dimensional state features," in *AIStats*, 2015.

[27] R. Ozawa, S. Arimoto, S. Nakamura, and J.-H. Bae, "Control of an object with parallel surfaces by a pair of finger robots without object sensing," *Trans. Robotics*, vol. 21, no. 5, pp. 965–976, 2005.

[28] K. Tahara, S. Arimoto, M. Sekimoto, M. Yoshida, and Z.-W. Luo, "On iterative learning control for simultaneous force/position trajectory tracking by using a 5 d.o.f. robotic thumb under non-holonomic rolling constraints," in *ICRA*, 2008, pp. 2611–2616.

[29] O. Brock, D. Katz, and Y. Pyuro, "Learning to manipulate articulated objects in unstructured environments using a grounded relational representation," *RSS*, 2009.

[30] T. Lampe and M. Riedmiller, "Acquiring visual servoing reaching and grasping skills using neural reinforcement learning," in *IJCNN*, 2013.

[31] O. Kroemer, R. Detry, J. Piater, and J. Peters, "Combining active learning and reactive control for robot grasping," *Robotics and Autonomous Systems*, no. 9, pp. 1105–1116, 2010.

[32] O. Kroemer, C. Daniel, G. Neumann, H. van Hoof, and J. Peters, "Towards learning hierarchical skills for multi-phase manipulation tasks," in *ICRA*, 2015.

[33] M. Deisenroth, C. Rasmussen, and D. Fox, "Learning to control a low-cost manipulator using data-efficient reinforcement learning," in *RSS*, 2011.

[34] S. Levine, N. Wagener, and P. Abbeel, "Learning contact-rich manipulation skills with guided policy search," in *ICRA*, 2015.

[35] P. Pastor, M. Kalakrishnan, S. Chitta, E. Theodorou, and S. Schaal, "Skill learning and task outcome prediction for manipulation," in *ICRA*, 2011.

[36] M. Kalakrishnan, L. Righetti, P. Pastor, and S. Schaal, "Learning force control policies for compliant manipulation," in *IROS*, 2011.

[37] J. Peters, K. Muelling, and Y. Altun, "Relative entropy policy search," in *AAAI Conference on Artificial Intelligence*, 2010.

[38] J. Kober, E. Oztop, and J. Peters, "Reinforcement learning to adjust robot movements to new situations," in *IJCAI*, 2011.

[39] J. Peters and S. Schaal, "Reinforcement learning by reward-weighted regression for operational space control," in *ICML*, 2007.