Optimal Control with Learned Local Models: Application to Dexterous Manipulation

Vikash Kumar[1], Emanuel Todorov[1], and Sergey Levine[2]

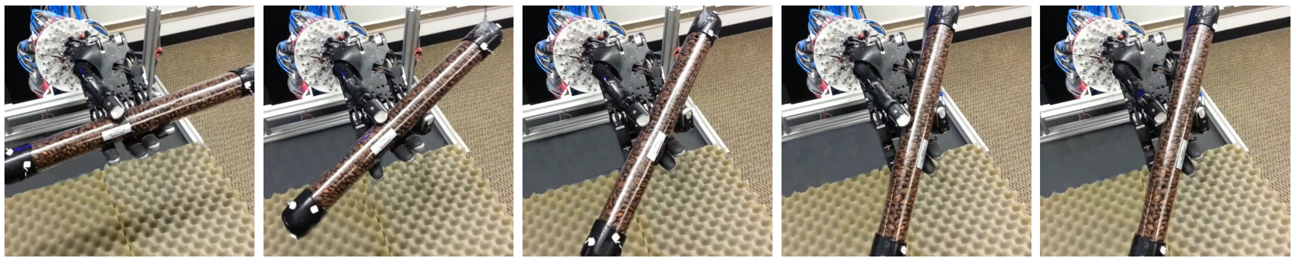


Fig. 1: Learned hand manipulation behavior involving clockwise rotation of the object

***Abstract*— We describe a method for learning dexterous manipulation skills with a pneumatically-actuated tendon-driven 24-DoF hand. The method combines iteratively refitted time-varying linear models with trajectory optimization, and can be seen as an instance of model-based reinforcement learning or as adaptive optimal control. Its appeal lies in the ability to handle challenging problems with surprisingly little data. We show that we can achieve sample-efficient learning of tasks that involve intermittent contact dynamics and under-actuation. Furthermore, we can control the hand directly at the level of the pneumatic valves, without the use of a prior model that describes the relationship between valve commands and joint torques. We compare results from learning in simulation and on the physical system. Even though the learned policies are local, they are able to control the system in the face of substantial variability in initial state.**

I. Introduction

Dexterous manipulation is among the most challenging control problems in robotics, and remains largely unsolved. This is due to a combination of factors including high dimensionality, intermittent contact dynamics, and under-actuation in the case of dynamic object manipulation. Here we describe our efforts to tackle this problem in a principled way. We do not rely on manually designed controllers. Instead we synthesize controllers automatically, by optimizing high-level cost functions. The resulting controllers are able to manipulate freely-moving objects; see Fig 1. While the present results are limited to learning local models and control policies, the performance we obtain, together with the small amount of data we need (around 60 trials) indicate that the approach can be extended with more global learning methods such as [1].

We use our Adroit platform [2], which is a ShadowHand skeleton augmented with high-performance pneumatic actuators. This system has a 100-dimensional continuous state space, including the positions and velocities of 24 joints, the pressures in 40 pneumatic cylinders, and the position and velocity of the object being manipulated.

Pneumatics have non-negligible time constants (around 20 ms in our system), which is why the cylinder pressures represent additional state variables, making it difficult to apply torque-control techniques. The system also has a 40-dimensional continuous control space – namely the commands to the proportional valves regulating the flow of compressed air to the cylinders. The cylinders act on the joints through tendons. The tendons do not introduce additional state variables (since we avoid slack via pre-tensioning) but nevertheless complicate the dynamics. Overall this is a daunting system to model, let alone control.

Depending on one's preference of terminology, our method can be classified as model-based Reinforcement Learning (RL), or as adaptive optimal control [3]. While RL aims to solve the same general problem as optimal control, its uniqueness comes from the emphasis on model-free learning in stochastic domains [4]. The idea of learning policies without having models still dominates RL, and forms the basis of the most remarkable success stories, both old [5] and new [6]. However RL with learned models has also been considered. Adaptive control on the other hand mostly focuses on learning the parameters of a model with predefined structure, essentially interleaving system identification with control [7].

Our approach here lies somewhere in between (to fix terminology, we call it RL in subsequent sections). We rely on a model, but that model does not have any informative predefined structure. Instead, it is a time-varying linear model learned from data, using a generic prior for regularization. Related ideas have been pursued previously [8], [1], [9]. Nevertheless, as with most approaches to automatic control and computational intelligence in general, the challenge is not only in formulating ideas but also in getting them to scale to hard problems – which is our main contribution here. In particular, we demonstrate scaling from a 14-dimensional state space in [9] to a 100-dimensional state space here. This

[1]Computer Science & Engineering, University of Washington; [2]Electrical Engineering & Computer Science, University of California, Berkeley. This work was supported by the US National Science Foundation & MAST Program, Army Research Office.

Algorithm 1 RL with linear-Gaussian controllers

```
1: initialize $p(\mathbf{u}_t|\mathbf{x}_t)$
2: for iteration $k = 1$ to $K$ do
3:     run $p(\mathbf{u}_t|\mathbf{x}_t)$ to collect trajectory samples $\{\tau_i\}$
4:     fit dynamics $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ to $\{\tau_j\}$ using linear regression with GMM prior
5:     fit $p = \arg\min_p E_{p(\tau)}[\ell(\tau)]$ s.t. $D_{\mathrm{KL}}(p(\tau)\|\hat{p}(\tau)) \le \epsilon$
6: end for
```

is important in light of the curse of dimensionality. Indeed RL has been successfully applied to a range of robotic tasks [10], [11], [12], [13], however dimensionality and sample complexity have presented major challenges [14], [15].

The manipulation skills we learn are represented as time-varying linear-Gaussian controllers. These controllers are fundamentally trajectory-centric, but otherwise are extremely flexible, since they can represent any trajectory with any linear stabilization strategy. Since the controllers are time-varying, the overall learned control law is nonlinear, but is locally linear at each time step. These types of controllers have been employed previously for controlling lower-dimensional robotic arms [16], [9].

II. Reinforcement Learning with Local Linear Models

In this section, we describe the reinforcement learning algorithm (summarised in algorithm 1) that we use to control our pneumatically-driven five finger hand. The derivation in this section follows previous work [1], but we describe the algorithm in this section for completeness. The aim of the method is to learn a time-varying linear-Gaussian controller of the form $p(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t\mathbf{x}_t + \mathbf{k}_t, \mathbf{C}_t)$, where $\mathbf{x}_t$ and $\mathbf{u}_t$ are the state and action at time step t. The actions in our system correspond to the pneumatic valve's input voltage, while the state space is described in the preceding section. The aim of the algorithm is to minimize the expectation $E_{p(\tau)}[\ell(\tau)]$ over trajectories $\tau = \{\mathbf{x}_1, \mathbf{u}_1, \dots, \mathbf{x}_T, \mathbf{u}_T\}$, where $\ell(\tau) = \sum_{t=1}^{T} \ell(\mathbf{x}_t, \mathbf{u}_t)$ is the total cost, and the expectation is under $p(\tau) = p(\mathbf{x}_1)\prod_{t=1}^{T} p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)p(\mathbf{u}_t|\mathbf{x}_t)$, where $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ is the dynamics distribution.

A. Optimizing Linear-Gaussian Controllers

The simple structure of time-varying linear-Gaussian controllers admits a very efficient optimization procedure that works well even under unknown dynamics. The method is summarized in Algorithm 1. At each iteration, we run the current controller $p(\mathbf{u}_t|\mathbf{x}_t)$ on the robot to gather N samples ($N = 5$ in all of our experiments), then use these samples to fit time-varying linear-Gaussian dynamics of the form $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(f_{\mathbf{x}t}\mathbf{x}_t + f_{\mathbf{u}t}\mathbf{u}_t + f_{ct}, \mathbf{F}_t)$. This is done by using linear regression with a Gaussian mixture model prior, which makes it feasible to fit the dynamics even when the number of samples is much lower than the dimensionality of the system [1]. We also compute a second order expansion of the cost function around each of the samples, and average the expansions together to obtain a local approximate cost function of the form

$$\ell(\mathbf{x}_t, \mathbf{u}_t) \approx \frac{1}{2}[\mathbf{x}_t; \mathbf{u}_t]^{\mathrm{T}} \ell_{\mathbf{xu},\mathbf{xu}t}[\mathbf{x}_t; \mathbf{u}_t] + [\mathbf{x}_t; \mathbf{u}_t]^{\mathrm{T}} \ell_{\mathbf{xu}t} + \text{const}.$$

where subscripts denote derivatives, e.g. $\ell_{\mathbf{xu}t}$ is the gradient of ℓ with respect to $[\mathbf{x}_t; \mathbf{u}_t]$, while $\ell_{\mathbf{xu},\mathbf{xu}t}$ is the Hessian. The particular cost functions used in our experiments are described in the next section. When the cost function is quadratic and the dynamics are linear-Gaussian, the optimal time-varying linear-Gaussian controller of the form $p(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t\mathbf{x}_t + \mathbf{k}_t, \mathbf{C}_t)$ can be obtained by using the LQR method. This type of iterative approach can be thought of as a variant of iterative LQR [17], where the dynamics are fitted to data. Under this model of the dynamics and cost function, the optimal policy can be computed by recursively computing the quadratic Q-function and value function, starting with the last time step. These functions are given by

$$V(\mathbf{x}_t) = \frac{1}{2}\mathbf{x}_t^{\mathrm{T}} V_{\mathbf{x},\mathbf{x}t}\mathbf{x}_t + \mathbf{x}_t^{\mathrm{T}} V_{\mathbf{x}t} + \text{const}$$
$$Q(\mathbf{x}_t, \mathbf{u}_t) = \frac{1}{2}[\mathbf{x}_t;\mathbf{u}_t]^{\mathrm{T}} Q_{\mathbf{xu},\mathbf{xu}t}[\mathbf{x}_t;\mathbf{u}_t] + [\mathbf{x}_t;\mathbf{u}_t]^{\mathrm{T}} Q_{\mathbf{xu}t} + \text{const}$$

We can express them with the following recurrence:

$$\begin{aligned} Q_{\mathbf{xu},\mathbf{xu}t} &= \ell_{\mathbf{xu},\mathbf{xu}t} + f_{\mathbf{xu}t}^{\mathrm{T}} V_{\mathbf{x},\mathbf{x}t+1} f_{\mathbf{xu}t} \\ Q_{\mathbf{xu}t} &= \ell_{\mathbf{xu}t} + f_{\mathbf{xu}t}^{\mathrm{T}} V_{\mathbf{x}t+1} \\ V_{\mathbf{x},\mathbf{x}t} &= Q_{\mathbf{x},\mathbf{x}t} - Q_{\mathbf{u},\mathbf{x}t}^{\mathrm{T}} Q_{\mathbf{u},\mathbf{u}t}^{-1} Q_{\mathbf{u},\mathbf{x}t} \\ V_{\mathbf{x}t} &= Q_{\mathbf{x}t} - Q_{\mathbf{u},\mathbf{x}t}^{\mathrm{T}} Q_{\mathbf{u},\mathbf{u}t}^{-1} Q_{\mathbf{u}t}, \end{aligned}$$

which allows us to compute the optimal control law as $g(\mathbf{x}_t) = \hat{\mathbf{u}}_t + \mathbf{k}_t + \mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t)$, where $\mathbf{K}_t = -Q_{\mathbf{u},\mathbf{u}t}^{-1}Q_{\mathbf{u},\mathbf{x}t}$ and $\mathbf{k}_t = -Q_{\mathbf{u},\mathbf{u}t}^{-1}Q_{\mathbf{u}t}$. If we consider $p(\tau)$ to be the trajectory distribution formed by the deterministic control law $g(\mathbf{x}_t)$ and the stochastic dynamics $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$, LQR can be shown to optimize the standard objective

$$\min_{g(\mathbf{x}_t)} \sum_{t=1}^{T} E_{p(\mathbf{x}_t,\mathbf{u}_t)}[\ell(\mathbf{x}_t, \mathbf{u}_t)]. \tag{1}$$

However, we can also form a time-varying linear-Gaussian controller $p(\mathbf{u}_t|\mathbf{x}_t)$, and optimize the following objective:

$$\min_{p(\mathbf{u}_t|\mathbf{x}_t)} \sum_{t=1}^{T} E_{p(\mathbf{x}_t,\mathbf{u}_t)}[\ell(\mathbf{x}_t, \mathbf{u}_t)] - \mathcal{H}(p(\mathbf{u}_t|\mathbf{x}_t)).$$

As shown in previous work [18], this objective is in fact optimized by setting $p(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t\mathbf{x}_t + \mathbf{k}_t, \mathbf{C}_t)$, where $\mathbf{C}_t = Q_{\mathbf{u},\mathbf{u}t}^{-1}$. While we ultimately aim to minimize the standard controller objective in Equation (1), this maximum entropy formulation will be a useful intermediate step for a practical learning algorithm trained with fitted time-varying linear dynamics.

B. KL-Constrained Optimization

In order for this learning method to produce good results, it is important to bound the change in the controller $p(\mathbf{u}_t|\mathbf{x}_t)$ at each iteration. The standard iterative LQR method can change the controller drastically at each iteration, which can cause it to visit parts of the state space where the fitted dynamics are arbitrarily incorrect, leading to divergence. Furthermore, due to the non-deterministic nature of the real world domains, line search based methods can get misguided leading to unreliable progress.

To address these issues, we solve the following optimization problem at each iteration:

$$\min_{p(\mathbf{u}_t|\mathbf{x}_t)} E_{p(\tau)}[\ell(\tau)] \text{ s.t. } D_{\mathrm{KL}}(p(\tau)\|\hat{p}(\tau)) \leq \epsilon,$$

where $\hat{p}(\tau)$ is the trajectory distribution induced by the previous controller. Using KL-divergence constraints for controller optimization has been proposed in a number of prior works [19], [20], [21]. In the case of linear-Gaussian controllers, a simple modification to the LQR algorithm described above can be used to solve this constrained problem. Recall that the trajectory distributions are given by $p(\tau) = p(\mathbf{x}_1)\prod_{t=1}^{T} p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)p(\mathbf{u}_t|\mathbf{x}_t)$. Since the dynamics of the new and old trajectory distributions are assumed to be the same, the KL-divergence is given by

$$D_{\mathrm{KL}}(p(\tau)\|\hat{p}(\tau)) = \sum_{t=1}^{T} E_{p(\mathbf{x}_t,\mathbf{u}_t)}[\log \hat{p}(\mathbf{u}_t|\mathbf{x}_t)] - \mathcal{H}(p),$$

and the Lagrangian of the constrained optimization problem is given by

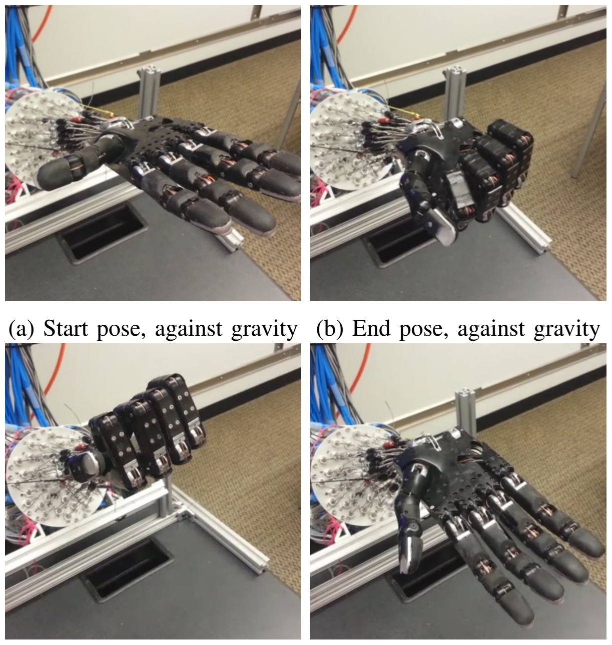
$$\mathcal{L}_{\text{traj}}(p,\eta) = E_p[\ell(\tau)] + \eta[D_{\mathrm{KL}}(p(\tau)\|\hat{p}(\tau)) - \epsilon] =$$
$$\left[\sum_t E_{p(\mathbf{x}_t,\mathbf{u}_t)}[\ell(\mathbf{x}_t,\mathbf{u}_t) - \eta \log \hat{p}(\mathbf{u}_t|\mathbf{x}_t)]\right] - \eta\mathcal{H}(p(\tau)) - \eta\epsilon.$$

The constrained optimization can be solved with dual gradient descent [22], where we alternate between minimizing the Lagrangian with respect to the primal variables, which are the parameters of p, and taking a subgradient step on the Lagrange multiplier η. The optimization with respect to p can be performed efficiently using the LQG algorithm, by observing that the Lagrangian is simply the expectation of a quantity that does not depend on p and an entropy term. As described above, LQR can be used to solve maximum entropy control problems where the objective consists of a term that does not depend on p, and another term that encourages high entropy. We can convert the Lagrangian primal minimization into a problem of the form

$$\min_{p(\mathbf{u}_t|\mathbf{x}_t)} \sum_{t=1}^{T} E_{p(\mathbf{x}_t,\mathbf{u}_t)}[\tilde{\ell}(\mathbf{x}_t,\mathbf{u}_t)] - \mathcal{H}(p(\mathbf{u}_t|\mathbf{x}_t))$$

by using the cost $\tilde{\ell}(\mathbf{x}_t,\mathbf{u}_t) = \frac{1}{\eta}\ell(\mathbf{x}_t,\mathbf{u}_t) - \log \hat{p}(\mathbf{u}_t|\mathbf{x}_t)$. This objective is simply obtained by dividing the Lagrangian by η. Since there is only one dual variable, dual gradient descent typically converges very quickly, usually in under 10 iterations, and because LQR is a very efficient trajectory optimization method, the entire procedure can be implemented to run very quickly.

We initialize $p(\mathbf{u}_t|\mathbf{x}_t)$ with a fixed covariance $\mathbf{C}_t$ and zero mean, to produce random actuation on the first iteration. The Gaussian noise used to sample from $p(\mathbf{u}_t|\mathbf{x}_t)$ is generated in advance and smoothed with a Gaussian kernel with a standard deviation of two time steps, in order to produce more temporally coherent noise.


(a) Start pose, against gravity (b) End pose, against gravity

(c) Start pose, with gravity (d) End pose, with gravity

Fig. 2: Positioning task

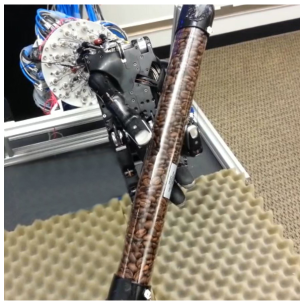
III. System and task description

Modularity and the ease of switching robotic platforms formed the overarching philosophy of our system design. The training can either happen locally (on the machine controlling the robot) or remotely (if more computational power is needed). The learning algorithm (algorithm 1) has no dependency on the selected robotic platform except for step 3, where the policies are shipped to the robotic platform for evaluation and the resulting trajectories are collected. For the present work, the system was deployed locally on a 12 cores 3.47GHz Intel(R) Xeon(R) processor with 12GB memory running Windows x64. Learning and evaluated was studied for two different platforms detailed below.

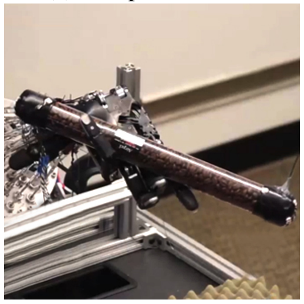
A. Hardware platform

The Adroit platform is described in detail in [2]. Here we summarize the features relevant to the present context. As already noted, the hand has 24 joints and 40 Airpel cylinders acting on the joints via tendons. Each cylinder is supplied with compressed air via a high-performance Festo valve. The cylinders are fitted with solid-state pressure sensors. The pressures together with the joint positions and velocities (sensed by potentiometers in each joint) are provided as state variables to our controller.

The manipulation task also involves an object – which is a long tube filled with coffee beans, inspired by earlier work on grasping [23]. The object is fitted with PhaseSpace active infrared markers on each end. The markers are used to estimate the object position and velocity (both linear and angular) which are also provided as state variables. Since all


(a) End pose, learned

(b) End pose, human


(c) End pose, learned

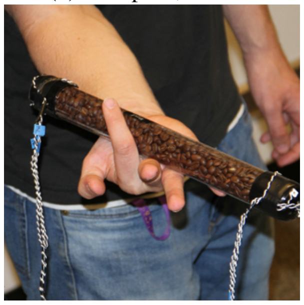

(d) End pose, human

Fig. 3: Object rotation task: end poses for the two rotation directions (a-b: clockwise, c-d: anticlockwise), comparing the learned controller to the movement of a human who has not seen the robot perform the task.

our sensors have relatively low noise, we apply a minimal amount of filtering before sending the sensor data to the controller.

B. Simulation platform

We model the entire system, including the air dynamics, tendon actuation and hand-object interactions, in the MuJoCo simulator we have developed [24]. Here MuJoCo is used as a MEX file called from MATLAB. Simulating a 5 s trajectory at 2 ms timestep takes around 0.7 s of CPU time, or around 0.3 ms of CPU time per simulation step. This includes evaluating the feedback control law (which needs to be interpolated because the trajectory optimizer uses 50 ms time steps) and advancing the physics simulation.

Having a fast simulator enables us to prototype and quickly evaluate candidate learning algorithms and cost function designs, before testing them on the hardware. Apart from being able to run much faster than real-time, the simulator can automatically reset itself to a specified initial state (which needs to be done manually on the hardware platform).

Ideally, the learning on the hardware platform should be seeded from the results of the software platform. This however is hard in practice due to (a) the difficulty in aligning the state space of the two platforms, (b) the non-deterministic nature of the real world. State space alignment requires precise system identification and sensor calibration which otherwise are not necessary, as our algorithm can learn the local state space information directly from the raw sensor values.

TABLE I: Different tasks variations learned

Task	Platform	# different task variations learned
Hand posit-ioning	Hardware	5 (move assisted by gravity) 2 (move against gravity)
	Simulated	5 (move assisted by gravity) 3 (move against gravity)
Object manipu-lation	Hardware + an elongated object (Fig:3)	2 ({clockwise & anti-clockwise} object rotations along vertical)
	Simulated + 4 object variations	13 ({clockwise, anti-clockwise, clockwise then anti-clockwise} object rotation along vertical 8 ({clockwise, anti-clockwise} object rotation along horizontal)

C. Tasks and cost functions

Various tasks we trained for (detailed in table I and the accompanying video) can be classified into two broad categories

1) Hand behaviors: We started with simple positioning tasks: moving the hand to a specified pose from a given initial pose. We arranged the pair of poses such that in one task-set the finger motions were helped by gravity, and in another task-set they had to overcome gravity; see Figure 2. Note that for a system of this complexity even achieving a desired pose can be challenging, especially since the tendon actuators are in agonist-antagonist pairs and the forces have to balance to maintain posture.

The running cost was

$$\ell(\mathbf{x}_t, \mathbf{u}_t) = ||q_t - q^*||^2 + 0.001||\mathbf{u}_t||^2$$

where $\mathbf{x} = (q, \dot{q}, a)$. Here q denotes the vector of hand joint angles, a the vector of cylinder pressures, and $\mathbf{u}_t$ the vector of valve command signals. At the final time we used

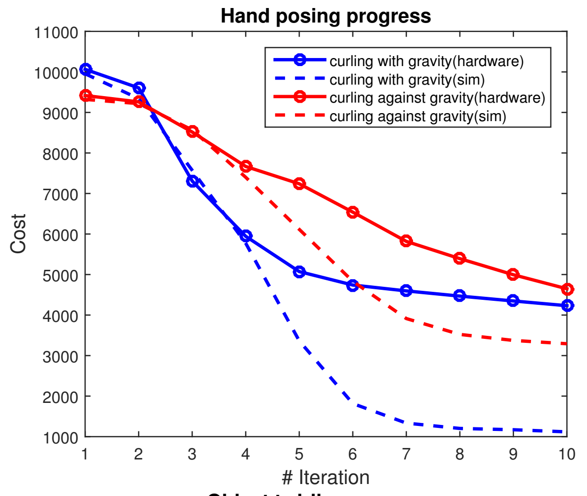
$$\ell(\mathbf{x}_t, \mathbf{u}_t) = 10||q_t - q^*||^2$$

2) Object manipulation behaviours: The manipulation tasks we focused on require in-hand rotation of elongated objects. We chose this task because it involves intermittent contacts with multiple fingers and is quite dynamic, while at the same time having a certain amount of intrinsic stability. We studied different variations (table I) of this task with different objects: rotation clockwise (figure 1), rotation counter-clockwise, rotation clockwise followed by rotation counter-clockwise, and rotation clockwise without using the wrist joint (to encourage finger oriented maneuvers) – which was physically locked in that condition. Figure 3 illustrates the start and end poses and object configurations in the task learned on the Adroit hardware platform.

Here the cost function included an extra term for desired object position and orientation. The relative magnitudes of the different cost terms were as follows: hand pose: 0.01, object position: 1, object orientation: 10, control: 0.001. The final cost was scaled by a factor of 2 relative to the running cost, and did not include the control term.

IV. Results

The controller was trained as described above. Once the order of cost parameters were properly established, minimal parameter tuning was required. Training consisted of around



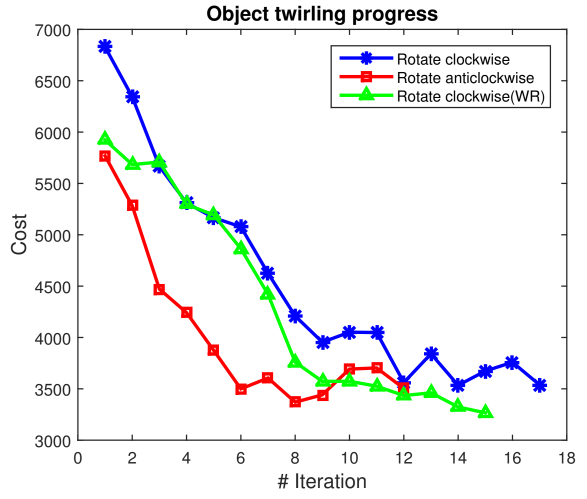


Fig. 4: Learning curves for the positioning (top) and manipulation (bottom) tasks. [1]

15 iterations. In each iteration we performed 5 movements with different instantiations of the exploration noise in the controls. The progress of training as well as the final performance is illustrated in the video accompanying the submission, and in the figure at the beginning of the paper.

Here we quantify the performance and the robustness to noise. Figure 4 shows how the total cost for the movement (as measured by the cost functions defined above) decreased over iterations of the algorithm. The solid curves are data from the physical system. Note that in all tasks and task variations we observe very rapid convergence. Surprisingly, the manipulation task which is much harder from a control viewpoint takes about the same number of iterations to learn.

In the positioning task we also performed a systematic comparison between learning in the physical system and learning in simulation. Performance early in training was comparable, but eventually the algorithm was able to find better policies in simulation. Although it is not shown in the figure, training on simulation platform happens a lot faster, because the robot can only run in real-time while the simulated platform runs faster than real-time, and because resetting between repetitions needs to be done manually on the robot.

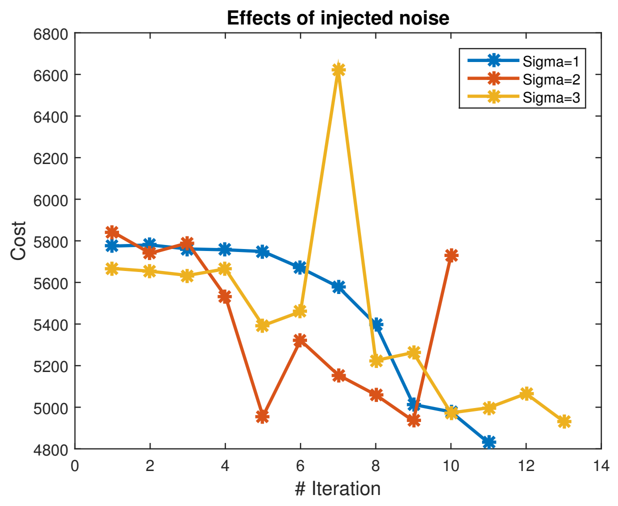


Fig. 5: Effect of noise smoothing on learning. Sigma=1 (width of the Gaussian kernel used for smoothing noise), takes a slow start but maintains a constant progress. Higher sigma favors steep decent but it fails to maintain the progress as it is unable to successfully maintain the stability of the object being manipulated and ends up dropping it. The algorithm incurs a huge cost penalty and restarts its decent from there. [1]

We further investigated the effects of exploration noise magnitude injected during training. Figure 5 shows that for a relatively small amount of noise performance decreases monotonically. As we increase the noise magnitude, sometimes we see faster improvement early on but the behavior of the algorithm is no longer monotonic. These are data on the Adroit hardware platform.

Finally, we used the simulation platform to investigate robustness to perturbations more quantitatively, in the manipulation task. We wanted to quantify how robust our controllers are to changes in initial state (recall that the controllers are local). Furthermore, we wanted to see if training with noisy initial states, in addition to exploration noise injected in the controls, will result in more robust controllers. Naïvely adding initial state noise at each iteration of the algorithm (Algorithm 1) severely hindered the overall progress. However, adding initial state noise after the policy was partially learned (iteration ≥ 10 in our case) resulted in much more robust controllers.

The results of these simulations are shown in Figure 6. We plot the orientation of the object around the vertical axis as a function of time. The black curve is the unperturbed trajectory. As expected, noise injected in the initial state makes the movements more variable, especially for the controller that was trained without such noise. Adding initial state noise during training substantially improved the ability of the controller to suppress perturbations in initial state. Overall, we were surprised at how much noise we could add (up to 20 % of the range of each state variable) without the hand dropping the object, in the case of the controller trained

[1] At each iteration, the current controller $p(\mathbf{u}_t|\mathbf{x}_t)$ is deployed on the robot to gather N samples ($N = 5$ in all of our experiments).

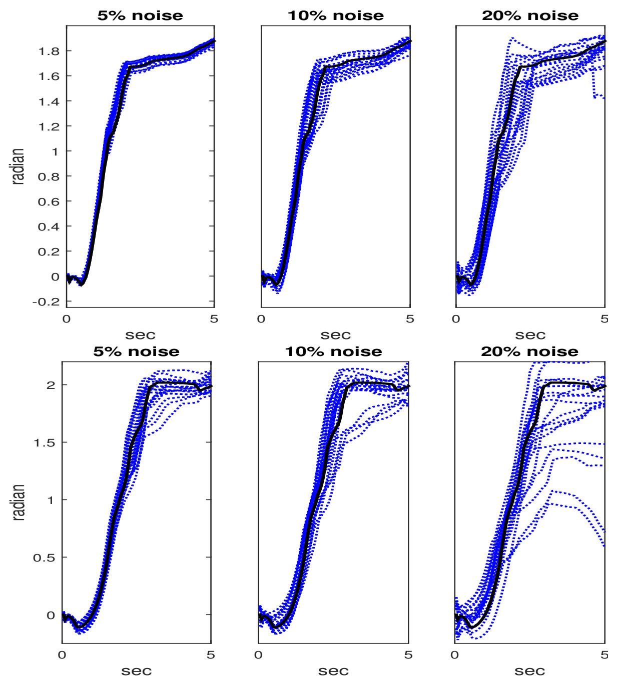


Fig. 6: Robustness to noise in initial state. Each column corresponds to a different noise level: 5, 10, 20 % of the range of each state variable. The top row is a controller trained with noise in the initial state. The bottom row is a controller trained with the same initial state (no noise) for all trials.

with noise. The controller trained without noise dropped the object in 4 out of 20 test trials. Thus injecting some noise in the initial state (around 2.5 %) helps improve robustness. Of course on the real robot we cannot avoid injecting such noise, because exact repositioning is very difficult.

V. Discussion and Future Work

We demonstrated learning-based control of a complex, high-dimensional, pneumatically-driven hand. Our results show simple tasks, such as reaching a target pose, as well as dynamic manipulation behaviors that involve repositioning a freely-moving cylindrical object. Aside from the high-level objective encoded in the cost function, the learning algorithm does not use domain knowledge about the task or the hardware, learning a low-level valve control strategy for the pneumatic actuators entirely from scratch. The experiments show that effective manipulation strategies can be automatically discovered in this way.

While the linear-Gaussian controllers we employ offer considerable flexibility and closed-loop control, they are inherently limited in their ability to generalize to new situations, since a time-varying linear strategy may not be effective when the initial state distribution is more diverse. Previous work has addressed this issue by combining multiple linear-Gaussian controllers into a single nonlinear policy, using methods such as guided policy search [9] and trajectory-based dynamic programming [25]. Applying these methods to our manipulation tasks could allow us to train more generalizable manipulation skills, and also bring in additional sensory modalities, such as haptics and vision, as described in recent work on guided policy search [26].

References

[1] S. Levine and P. Abbeel, “Learning neural network policies with guided policy search under unknown dynamics,” in *Advances in Neural Information Processing Systems (NIPS)*, 2014.

[2] Z. X. V. Kumar and E. Todorov, “Fast, strong and compliant pneumatic actuation for dexterous tendon-driven hands,” in *International Conference on Robotics and Automation (ICRA)*, 2013.

[3] R. Bellman and R. Kalaba, “A mathematical theory of adaptive control processes,” *Proceedings of the National Academy of Sciences*, vol. 8, no. 8, pp. 1288–1290, 1959.

[4] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[5] G. Tesauro, “Td-gammon, a self-teaching backgammon program, achieves master-level play,” *Neural computation*, vol. 6, no. 2, pp. 215–219, 1994.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[7] K. J. Åström and B. Wittenmark, *Adaptive control*. Courier Corporation, 2013.

[8] D. Mitrovic, S. Klanke, and S. Vijayakumar, “Adaptive optimal feedback control with learned internal dynamics models,” in *From Motor Learning to Interaction Learning in Robots*, 2010, vol. 264, pp. 65–84.

[9] S. Levine, N. Wagener, and P. Abbeel, “Learning contact-rich manipulation skills with guided policy search,” in *International Conference on Robotics and Automation (ICRA)*, 2015.

[10] R. Tedrake, T. Zhang, and H. Seung, “Stochastic policy gradient reinforcement learning on a simple 3d biped,” in *International Conference on Intelligent Robots and Systems (IROS)*, 2004.

[11] J. Kober, E. Oztop, and J. Peters, “Reinforcement learning to adjust robot movements to new situations,” in *Robotics: Science and Systems (RSS)*, 2010.

[12] P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal, “Learning and generalization of motor skills by learning from demonstration,” in *International Conference on Robotics and Automation (ICRA)*, 2009.

[13] M. Deisenroth, C. Rasmussen, and D. Fox, “Learning to control a low-cost manipulator using data-efficient reinforcement learning,” in *Robotics: Science and Systems (RSS)*, 2011.

[14] J. Kober, J. A. Bagnell, and J. Peters, “Reinforcement learning in robotics: A survey,” *International Journal of Robotic Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[15] M. Deisenroth, G. Neumann, and J. Peters, “A survey on policy search for robotics,” *Foundations and Trends in Robotics*, vol. 2, no. 1-2, pp. 1–142, 2013.

[16] R. Lioutikov, A. Paraschos, G. Neumann, and J. Peters, “Sample-based information-theoretic stochastic optimal control,” in *International Conference on Robotics and Automation (ICRA)*, 2014.

[17] W. Li and E. Todorov, “Iterative linear quadratic regulator design for nonlinear biological movement systems,” in *ICINCO (1)*, 2004, pp. 222–229.

[18] S. Levine and V. Koltun, “Guided policy search,” in *International Conference on Machine Learning (ICML)*, 2013.

[19] J. A. Bagnell and J. Schneider, “Covariant policy search,” in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

[20] J. Peters and S. Schaal, “Reinforcement learning of motor skills with policy gradients,” *Neural Networks*, vol. 21, no. 4, pp. 682–697, 2008.

[21] J. Peters, K. Mülling, and Y. Altün, “Relative entropy policy search,” in *AAAI Conference on Artificial Intelligence*, 2010.

[22] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004.

[23] J. R. Amend Jr, E. Brown, N. Rodenberg, H. M. Jaeger, and H. Lipson, “A positive pressure universal gripper based on the jamming of granular material,” *Robotics, IEEE Transactions on*, vol. 28, no. 2, pp. 341–350, 2012.

[24] E. Todorov, T. Erez, and Y. Tassa, “Mujoco: A physics engine for model-based control,” in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 5026–5033.

[25] C. Atkeson and J. Morimoto, “Nonparametric representation of policies and value functions: A trajectory-based approach,” in *Advances in Neural Information Processing Systems (NIPS)*, 2002.

[26] S. Levine, C. Finn, T. Darrell, and P. Abbeel, “End-to-end training of deep visuomotor policies,” *arXiv preprint arXiv:1504.00702*, 2015.