Reinforcement learning for non-prehensile manipulation: Transfer from simulation to physical system

Kendall Lowrey[1,2], Svetoslav Kolev[1], Jeremy Dao[1], Aravind Rajeswaran[1] and Emanuel Todorov[1,2]

***Abstract*— Reinforcement learning for continuous control has emerged as a promising methodology for training robot controllers. Most results however, have been limited to simulation, due to the need for a large number of samples and lack of automated-yet-safe data collection methods. Model-based reinforcement learning methods provide an avenue to circumvent these challenges, but the traditional concern has been the mismatch between the simulator and the real world. Here, we show that control policies learned in simulation can successfully transfer to a dynamic physical system, composed of three Phantom robots pushing an object to changing targets. Learning is done with a natural policy gradient method, applied to a carefully identified simulation model. The resulting policies, trained in simulation, work well on the physical system without additional training. In addition, we show that training with an ensemble of models makes the learned policies more robust to modeling errors, thus compensating for difficulties in system identification. The results are illustrated in the accompanying video.**

I. INTRODUCTION

Non-prehensile object manipulation remains a challenge for robotic control. In this work we focus on a particularly challenging system using three Phantom robots as fingers. These are haptic robots that are torque-controlled and have higher bandwidth than the fingers of existing robotic hands. In terms of speed and compliance (but not strength) they are close to the capabilities of the human hand. This makes them harder to control, especially in non-prehensile manipulation tasks where the specifics of each contact event and the balance of contact forces exerted on the object are very important and need to be considered by the controller in some form.

Here we develop a solution using Reinforcement Learning (RL) within the MuJoCo physics simulator [1]. For Reinforcement Learning we use a normalized natural policy gradient method [2], [3], [4]. While RL is in principle model-free, in practice it requires large amounts of data. In the absence of an automatic way to generate safe exploration controllers, most learning is thus done in simulation. Indeed the large majority of recent results in continuous RL have been obtained in simulation. These studies often propose to extend the corresponding methods to physical systems in future work, but the scarcity of such results indicates that 'sim-to-real' transfer is harder than it seems. The few successful applications to real robots have been in tasks involving position or velocity control that avoid some of the difficulty of control.

*This work was supported by the NSF.

[1] University of Washington, [2] Roboti LLC.

We use an accurate simulation model whose physics parameters have been carefully tuned via system identification [5]. System identification has long been studied in robotics, and it is reasonable to assume that the process are already in place. While true model-free RL may one day become feasible, we believe leveraging the capabilities of a physics simulator will always help speed up the learning process.

As with any controller developed in simulation, performance on the real system is likely to degrade due to modeling errors. To assess, as well as mitigate, the effects of these errors, we compare learning with respect to three different models: (i) the nominal model obtained from system identification; (ii) a modified model where the object mass is intentionally mis-identified; (iii) an ensemble of models where the mean is wrong but the variance is large enough to cover the nominal model. We find that (i) achieves the best performance as expected, but (iii) is also robust even though it is not as performant. This is consistent with our earlier observations using model ensembles in the context of trajectory optimization [6].

A. Related Work

There are many methods towards developing safe and robust robot controllers. Robot actions that involve dynamic motions require not only precise control execution, but also robust compensation when the action inevitably does not go according to plan–the physics of the real world are notoriously uncooperative. Control methods that depend on physical models, whether reduced and simplified models or not, are able to produce dynamic actions [7], [8], [9], [10], [11], [12]. They frequently rely on physics simulations for testing purposes, before usage on real hardware. This step is critical, as any modeling errors can significantly contribute to poor performance or even hardware damage [13]. Including uncertainty in the planning stage is one way to avoid this problem, and may also enable model learning simultaneously [14], [15], [16], [17], [18]. These model centric approaches offer strong performance expectations, but unless uncertainty or robustness is explicitly taken into account, may be brittle to external unknowns [19].

On the other hand, Reinforcement Learning offers a means to directly learn from the robot's experience [20], [21]. The difficulty, of course, is where the robot's experiences come from: as RL algorithms may need significant amounts of data, doing this on hardware may be infeasible [22]. Directly training on hardware has been feasible in some cases [23], [24], but domains with highly nonlinear dynamics will always require more data to sufficiently explore, human

demonstrations with imitation learning, and/or parameterized explorations [25], [26], [27]. Another common issue with learning in the real world is how to reset the state of the system, with some work being done [28]. For sensitive and delicate systems, the only safe place to perform learning is in simulation. Transferring to real hardware can take many approaches as well, either through adaptation [29], [30], or incorporating uncertainty [31], [32].

This work focuses on using a physics simulator to train policies for manipulation using reinforcement learning. As the manipulator is non-prehensile, we do not use any demonstrations or guide the policy search. To facilitate transfer to hardware, we also avoid the use of an estimator (i.e. the use of a model to predict state like a Kalman filter) by learning a function that directly converts from sensor values to motor torques. The policy is then transfered to the hardware for evaluation, and show that even for incorrect models used during training, useful policies are obtained by using an ensemble of models. Sections 2 and 3 detail the RL problem formulation and solution. Section 4 explains the hardware platform and details of the manipulation task are in section 5. Finally Section 6 contains the results and Section 7 the discussion.

II. PROBLEM FORMULATION

We model the control problem as a Markov decision process (MDP) in the episodic average reward setting, which is defined using the tuple: $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \rho_0, T\}$. $\mathcal{S} \subseteq \mathbb{R}^n$, $\mathcal{A} \subseteq \mathbb{R}^m$, and $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ are (continuous) set of states, set of actions, and the reward function respectively. $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the stochastic transition function; ρ_0 is the probability distribution over initial states; and T is the maximum episode length. We wish to solve for a stochastic policy of the form $\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, which optimizes the average reward accumulated over the episode. Formally, the performance of a policy is evaluated according to:

$$\eta(\pi) = \frac{1}{T}\, \mathbb{E}_{\pi,\mathcal{M}} \left[\sum_{t=1}^{T} r_t \right]. \tag{1}$$

In this finite horizon rollout setting, we define the value, Q, and advantage functions as follows:

$$V^{\pi}(s,t) = \mathbb{E}_{\pi,\mathcal{M}} \left[\sum_{t'=t}^{T} r_{t'} \right]$$

$$Q^{\pi}(s,a,t) = \mathbb{E}_{\mathcal{M}} \Big[\mathcal{R}(s,a) \Big] + \mathbb{E}_{s' \sim \mathcal{T}(s,a)} \Big[V^{\pi}(s', t+1) \Big]$$

$$A^{\pi}(s,a,t) = Q^{\pi}(s,a,t) - V^{\pi}(s,t)$$

We consider parametrized policies π_θ, and hence wish to optimize for the parameters (θ). Thus, we overload notation and use $\eta(\pi)$ and $\eta(\theta)$ interchangeably. In this work, we represent π_θ as a multivariate Gaussian with diagonal covariance. In our experiments, we use an affine policy as our function approximator, visualized in figure 6.

III. METHOD

A. Natural Policy Gradient

Policy gradient algorithms are a class of RL methods where the parameters of the policy are directly optimized typically using gradient based methods. Using the score function gradient estimator, the sample based estimate of the policy gradient can be derived to be: [33]:

$$\hat{g} = \frac{1}{NT} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^i | s_t^i) \hat{A}^{\pi}(s_t^i, a_t^i, t) \tag{2}$$

A straightforward gradient ascent using the above gradient estimate is the REINFORCE algorithm [33]. Gradient ascent with this direction is sub-optimal since it is not the steepest ascent direction in the metric of the parameter space [34]. Consequently, a local search approach that moves along the steepest ascent direction was proposed by Kakade [2] called the natural policy gradient. This has been expanded upon in subsequent works [3], [35], [36], [37], and forms a critical component in state of the art RL algorithms. Natural policy gradient is obtained by solving the following local optimization problem around iterate θ_k:

$$\begin{aligned} \underset{\theta}{\text{maximize}} \quad & g^T(\theta - \theta_k) \\ \text{subject to} \quad & (\theta - \theta_k)^T F_{\theta_k} (\theta - \theta_k) \leq \delta, \end{aligned} \tag{3}$$

where F_{θ_k} is the Fisher Information Metric at the current iterate θ_k. We apply a normalized gradient ascent procedure, which has been shown to further stabilize the training process [35], [36], [4]. This results in the following update rule:

$$\theta_{k+1} = \theta_k + \sqrt{\frac{\delta}{g^T F_{\theta_k}^{-1} g}}\; F_{\theta_k}^{-1} g. \tag{4}$$

The version of natural policy gradient outlined above was chosen for simplicity and ease of implementation. The natural gradient performs covariant updates by rescaling the parameter updates according to curvature information present in the Fisher matrix, thus behaving almost like a second order optimization method. Furthermore, due to the normalized gradient procedure, the gradient information is insensitive to linear rescaling of the reward function, improving training stability. For estimating the advantage function, we use the GAE procedure [38] and use a quadratic function approximator with all the terms in s for the baseline.

B. Distributed Processing

As the natural policy gradient algorithm is an on-policy method, all data is collected from the current policy. However, the NPG algorithm allows for the rollouts and most computation to be performed independently as only the gradient and the Fisher matrix need to be synced. Independent processes can compute the gradient and Fisher matrix, with a centralized server averaging these values and performing the matrix inversion and gradient step as in equation (4). The new policy is then shared with each worker. The total size of messages passed is proportional to the size of the Fisher Matrix used for the policy, and linear in the

number of worker nodes. Policies with many parameters may experience message passing overhead, but the trade-off is that each worker can perform as many rollouts during sample collection without changing the message size, encouraging more data gathering (which large policies require).

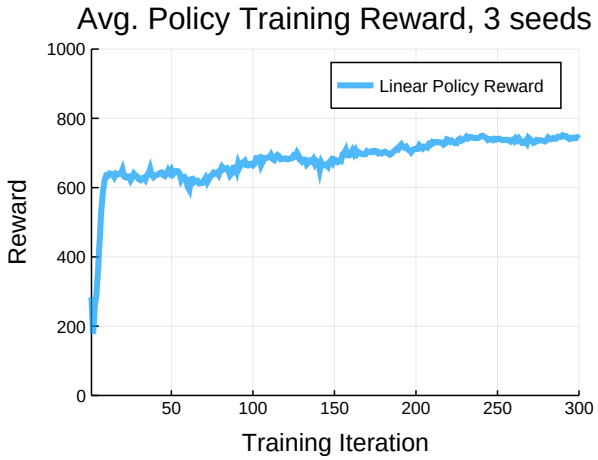

Fig. 1. Learning curve of our linear/affine policy. We show here the curve for the policy trained with the correct mass as a representative curve.

We implemented our RL code and interfaced with the MuJoCo simulator with the Julia programming language [39]. The built-in multi-processing and multi-node capabilities of Julia facilitated this distributed algorithm's performance; we are able to train a linear policy on this task in less than 3 minutes on a 4 node cluster with Intel i7-3930k processors.

Algorithm 1 Distributed Natural Policy Gradient

Initialize policy parameters to θ_0
for $k = 1$ **to** K **do**
  Distribute Policy and Value function parameters.
  for $w = 1$ **to** $N_{workers}$ **do**
    Collect trajectories $\{\tau^{(1)}, \ldots \tau^{(N)}\}$ by rolling out the stochastic policy $\pi(\cdot; \theta_k)$.
    Compute $\nabla_\theta \log \pi(a_t|s_t; \theta_k)$ for each (s, a) pair along trajectories sampled in iteration k.
    Compute advantages A_k^π based on trajectories in iteration k and approximate value function V_{k-1}^π.
    Compute policy gradient according to eq. (2).
    Compute the Fisher matrix (4).
    Return Fisher Matrix, gradient, and value function parameters to central server.
  end for
  Average Fisher Matrix gradient, and perform gradient ascent (5)
  Update parameters of value function.
end for

IV. HARDWARE AND PHYSICS SIMULATION

A. System Overview

We use our Phantom Manipulation Platform as our hardware testbed. It consists of three Phantom Haptic Devices,

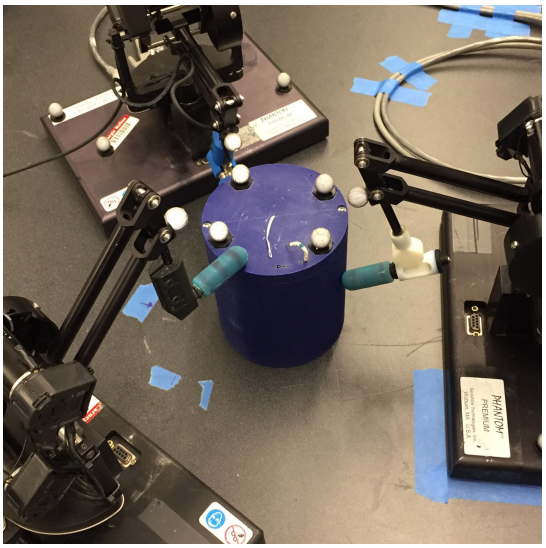

Fig. 2. Phantom Manipulation Platform.

each acting as a robotic finger. Each haptic device is a 3-DOF cable driven system shown in figure 2. The actuation is done with Maxon motors (Model RE 25 #118743), three per Phantom. Despite the low gear reduction ratio, they are able to achieve 8.5N instantaneous force and 0.6N continuous force at the middle of their range of motion, with very low friction for the entire range of motion.

The three robots are coupled together to act as one manipulator. Each robot's end effector was equipped with a silicon covered fingertip to enable friction and reliable grasping of objects. The softness of the silicon coating was an additional challenge in both contact modeling and robust policy learning. For this work, we had the robots manipulating a 3D printed cylinder with a height of 14cm and diameter of 11cm, and a mass of 0.34kg.

The soft contacts, combined with the direct torque control and high power-to-weight ratio–leading to high acceleration–make this platform particularly difficult to control. Systems with more mass and natural damping in their joints naturally move more slowly and smoothly; this is not the case here. Being able to operate in this space, however, allows for the potential for high performance, dynamic manipulation, and the benefits that come with torque based control. However, this requires that we operate our robot controller at 2kHz to successfully close the loop.

B. Sensing

As we wish to learn control policies that map from observations to controls, the choice of observations are critical to successful learning. Each Phantom is equipped with 3 optical encoders at a resolution of 5K steps per radian. We use a low-pass filter to compute the joint velocities. We also rely on a Vicon motion capture system, which gives us position data at 240Hz for the object we are manipulating. While being quite precise ($0.1mm$ error), the overall accuracy is significantly worse ($< 1mm$) due, in part, to imperfect object models and camera calibrations. While the Phantoms' position sensors

are noiseless, they often have small biases due to imperfect calibration. One Phantom robot is equipped with an ATI Nano17 3-axis force/torque sensor. This data is not used during training or in any learned controller, but used as a means of hardware / simulation comparison described in a later section. The entire system is simulated for policy training in the MuJoCo physics engine [1].

In total, our control policy has an observational input of 36 dimensions with 9 actuator outputs– the 9 positions and 9 velocities of the Phantoms are converted to 15 positions and 15 velocities for modeling purposes due to the parallel linkages. We additionally use 3 positions for both the manipulated object and the tracked goal, with 9 outputs for the 3 actuators per Phantom. Velocity observations are not used for the object as this would require state estimation that we have deliberately avoided.

C. System Identification

System identification of model parameters was performed in our prior work [5], but modeling errors are difficult to eliminate completely. For system ID we collected various behaviors with the robots, ranging from effector motion in free space to infer intrinsic robot parameters, to manipulation examples such as touching, pushing and sliding between the end effector and the object to infer contact parameters. The resulting data is fed into the joint system ID and state estimation optimization procedure. As explained in [5], state estimation is needed when doing system ID in order to eliminate the small amounts of noise and biases that our sensors produce.

The recorded behaviors are represented as a list of sensor readings $\mathbf{S} = \{s_1, s_2, \ldots, s_n\}$ and motor torques $\mathbf{U} = \{u_1, u_2, \ldots, u_n\}$. State estimation means finding a trajectory of states $\mathbf{Q} = \{q_1, q_2, \ldots, q_n\}$. Each state is a vector $q_i = (\theta_1, \ldots, \theta_{k'}, x, y, z, q_w, q_x, q_y, q_z)$, representing joint angles and object position. We also perform system ID which is finding the set of parameters $\mathbf{P}$, which include coefficients of friction, contact softness, damping coefficients, link inertias and others. We then pose system ID and estimation joint problem as the following optimization problem:

$$\min_{\mathbf{P},\mathbf{Q}} \sum_{i=1\ldots n} \|\hat{\tau}_i - u_i\|^{*_1} + \sum_{i=1\ldots n} \|\hat{s}_i - s_i\|^{*_2}$$

where $\hat{\tau}_i$ (predicted control signal) and $\hat{s}_i$ (predicted sensor outputs) are computed by the inverse dynamics generative model of MuJoCo: $(\hat{\tau}_i, \hat{s}_i) = \text{mjinverse}(q_{i-1}, q_i, q_{i+1})$. The optimization problem is solved via Gauss-Newton method [5].

V. TASK & EXPERIMENTS

In this section we first describe the manipulation task used to evaluate learned policy performance, then describe the practical considerations involved in using the NPG algorithm in this work. Finally, we describe two experiments evaluating learned policy performance in both simulation and on hardware.

A. Task Description

We use the NPG algorithm to learn a pushing task. The goal is to reduce the distance of the object, in this case the cylinder, to a target position as much as possible. This manipulation task requires that contacts can be made and broken multiple times during a pushing movement. As there are no state constraints involved in this RL algorithm, we cannot guarantee that the object will reach the target location (the object can be pushed into an un-reachable location). We feel that this is an acceptable trade-off if we can achieve more robust control over a wider state space.

For these tasks we model the bases of each Phantom as fixed, arrayed roughly equilateral around the object being manipulated–this is to achieve closure around the object. We do not enforce a precise location for the bases to make the manipulation tasks more challenging and expect them to shift during operation regardless.

B. Training Considerations

Policy training is the process of discovering which actions the controller should take in which state to achieve a good reward score, as such, it has implications for how well-performing the final policy is. Training structured informs the policy of good behavior, but is contrasted with the time required to craft the reward function. In this task's case, we use a very simple reward structure. In addition to the primary reward of reducing the distance between the object and the goal location, we provide the reward function with terms to reduce the distance between each finger tip and the object. This kind of hint term is common in both reinforcement learning and trajectory optimization. There is also a control cost, a, that penalizes using too much torque. The entire reward function at time t is as follows:

$$R_t(s) = 1 - 3||O_{xy} - G_{xy}|| - \sum_{i}^{3} ||f_i - O_{xy}|| - 0.1a^2 \quad (5)$$

Where O_{xy} is the current position on the xy-plane of the object, G_{xy} is the goal position, and f_i is the Phantom end effector position.

The initial state of each trajectory rollout is with the Phantom robots at randomized joint positions, deliberately not contacting the cylinder object. The cylinder location is kept at the origin on the xy-plane, but the desired goal location is set to uniformly random point within a circle of 12 cm diameter around the origin. To have more diverse initializations and to encourage robust policies, new initial states have a chance of starting at some state from one of the previous iteration's trajectories, provided the previous trajectory had a high reward. If the initial state was a continuation of the previous trajectory, the target location was again randomized: performing well previously only gives an initial state, and the policy needs to learn to push the object to a new goal. This is similar to a procedure outlined in [40] for training interactive and robust policies.

Finally, to further encourage robust behavior, we vary the location of the base of the Phantom robots by adding

Fig. 3. In these plots, we seed our simulator with the state measured from the hardware system. We use the simulator to calculate the instantaneous forces measured by a simulated force sensor, per time-step of real world data, and compare it to the data collected from hardware. These plots are in the frame of the contact, with the force along the normal axis being greatest. Note that the Y-axis of the Normal Axis Torque plot is different from the other torque plots.

Gaussian noise before each rollout. The standard deviation of this noise is 0.5cm. In this way an ensemble of models is used in discovering robust behavior, similar to [41]. As discussed above, we expect to have imprecisely measured their base locations and for each base to potentially shift during operation. We examine this effect more closely as one of our experiments.

C. Experiments

We devised two experiments to explore the validity of the Natural Policy Gradient reinforcement learning algorithm to discover robust policies for difficult robotic manipulation tasks.

First, we collect runtime data of positions, velocities, and force-torque measurements from a sensor equipped Phantom from the hardware using the best performing controller we have learned. We use this hardware data (positions and calculated velocities of the system) to seed our simulator, where instantaneous forces are calculated using a simulated force-torque sensor. This data is compared to force-torque data collected from the hardware. Instantaneous force differences highlight the inaccuracies between a model in simulation and data in the real world that eventually lead to divergence.

We also compare rollouts in simulation that have been seeded with data collected with hardware. This compares the policies behavior, not the system's, as we wish to examine the performance of the policy in both hardware and simulation. Ideally, the simulation environment matches closely to the hardware environment as an indication that system identification as well as any state estimations have been performed correctly. Secondly, we would like to compare the behavior of the learned control policy. From the perspective of task completion, the similarity or divergence of sim and real is less important as long as the robot completes the tasks satisfactorily. Said another way, poor system identification or sim/real divergence matters less if the robot gets the job done.

In these experiments, the target location is set by the user in moving a second tracked object above the cylinders used in our experiments. This data was recorded and used to collect the above datasets for analysis.

As a second experiment direction, we show how the effects of model ensembles during training affect robustness and feasibility. To do this, we explicitly vary the mass of the object being manipulated. The object (cylinder) was measured to be 0.34kg in mass, therefore, we train a policy with the mass set to this value. The objects mass was chosen to be modified due to the very visible effects an incorrect mass would have on performance. We train two additional policies, both with a mass of 0.4kg (approximately 20% more mass). One of the additional policies is trained with an approximated ensemble: we add Gaussian noise to the object mass parameter with standard deviation of 0.03 (30 grams). All three policies are evaluated in simulation with a correctly measured object mass, and in the real world with our 0.34kg cylinder.

To evaluate the policies, we calculate a path for the target to follow. The path is a spiral from the origin moving outward until it achieves a radius from the origin of 4 cm, at which it changes to a circular path and makes a full rotation, still at a radius of 4cm. This takes 4 seconds to complete. This path was programmatically set in both the simulator and on the real hardware to be consistent. This object ideally follows this trajectory path, as it presents a very visible means to explore policy performance.

VI. Results

The results for the two experiments are presented as follows, with additional discussion in the next section.

A. Simulation vs Hardware

We show comparisons between calculated forces and torques in simulation and hardware in figure 3. Our simulated values closely match the sensed hardware values, but are at times different. This is most likely due to the discretization of hardware sensors for the joint positions and velocities not being as precise as simulation expects, resulting in the calculation of instantaneous forces being different. While MuJoCo can represent soft contacts, the parameters defining

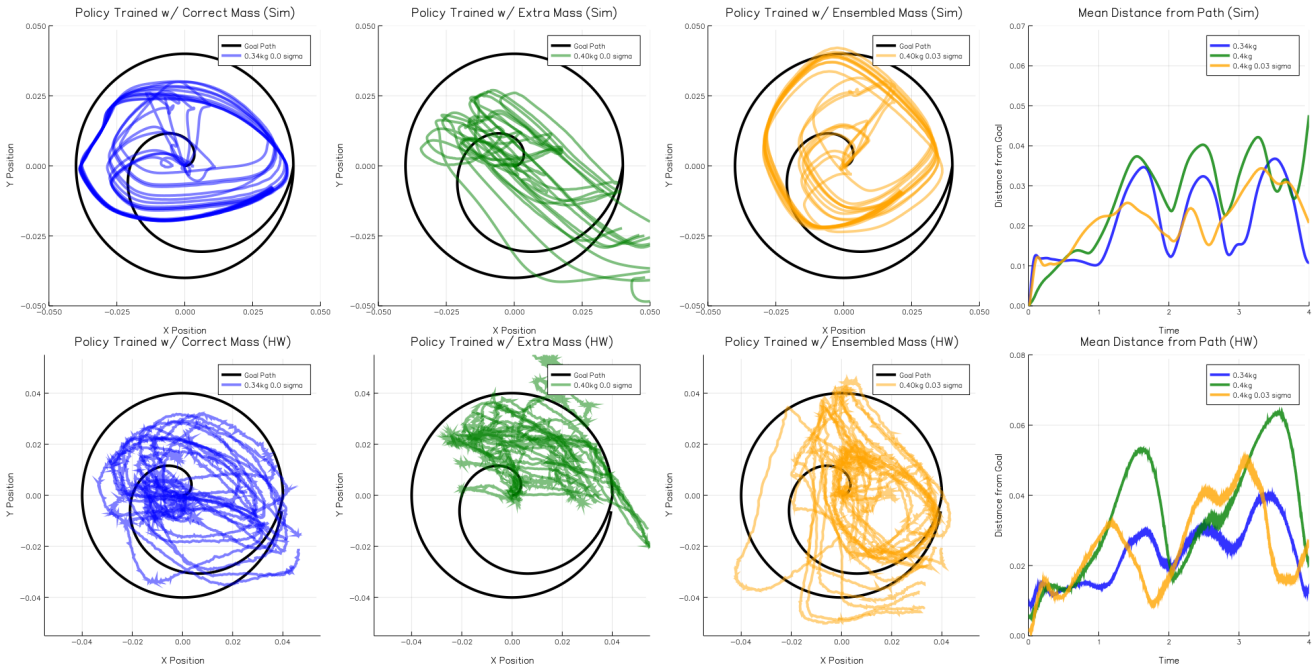

Fig. 4. 10 rollouts are performed where the target position of the object is the path that spirals from the center outward (in black) and then performs a full circular sweep (the plots represent a top-down view). We compare three differently trained policies: one where the mass of the object cylinder is 0.34kg, one where the mass is increased by 20 percent (to 0.4kg), and finally, we train a policy with the incorrect mass, but add model noise (at standard deviation 0.03) during training to create an ensemble. We evaluate these policies on the correct (0.34kg) mass in both simulation and on hardware. In both, the policy trained with the incorrect mass does not track the goal path. We also calculate the per time-step error from the goal path, averaged from all 10 rollouts (right-most plots).

them were not identified accurately. Critically, we can see that when contact is not being made, the sensors, simulated and hardware, are in agreement.

We find that our learned controllers are still able to perform well at task completion, despite differences between simulation and hardware. We can see in figure 5 that for the correct policy (learned with a correct model), when we perform a rollout in simulation based on hardware data, the simulated rollout is close to the data collected from hardware. The policy performance in simulation is close to the policy performance in hardware. This is not the case for the incorrect (learned with a wrong model) and ensemble policies, where the simulated rollout is different from the hardware data. We presume that because the incorrect policy has over-fit to the incorrect model, it is taking advantage of that model's parameters to perform the task. As such, that over-fitting could have different effects when evaluated on hardware or in the correctly simulated model. Despite the correct simulation being similar to hardware, the controller's behavior could cause divergence on whatever remaining small parameter differences. The ensemble policy, as expected, lies somewhere between the correct and incorrect policy.

B. Training with Ensembles

We find that training policies with model ensembles to be particularly helpful. Despite being given a very incorrect mass of the object, the policy trained with the ensemble performed very well (figure 4). In addition to performing well in simulation, we found it to perform nearly as well as the correctly trained policy in hardware. This mirrors our comparison in the previous section, where the correct policy performs comparably in both hardware and simulation, with the other policies less so. This is important to note given the poor performance of the incorrect policy: this task's training is indeed sensitive to this model parameter. The implication of the ensemble approach is not just that it can overcome poor or incorrect modeling, but can provide a safe initial policy to collect valuable data to improve the model.

TABLE I
AVERAGE DISTANCE FROM TARGET, 10 ROLLOUTS

	Sim	Hardware
0.34kg Policy (correct)	2.1cm	2.33cm
0.4kg Policy (incorrect)	2.65cm	3.4cm
0.4kg Policy ensemble	2.15cm	2.52cm

VII. DISCUSSION

Our results suggest two interesting observations. It is not easy to discover robust control policies for challenging real-world tasks, but with the appropriate consideration and techniques, successful control can provide a useful baseline for additional control opportunities. Simulation can provide a safe backdrop to test and develop none-intuitive controllers (see figure 6). This controller was developed for a robotic system without an intermediate controller such as PID, and

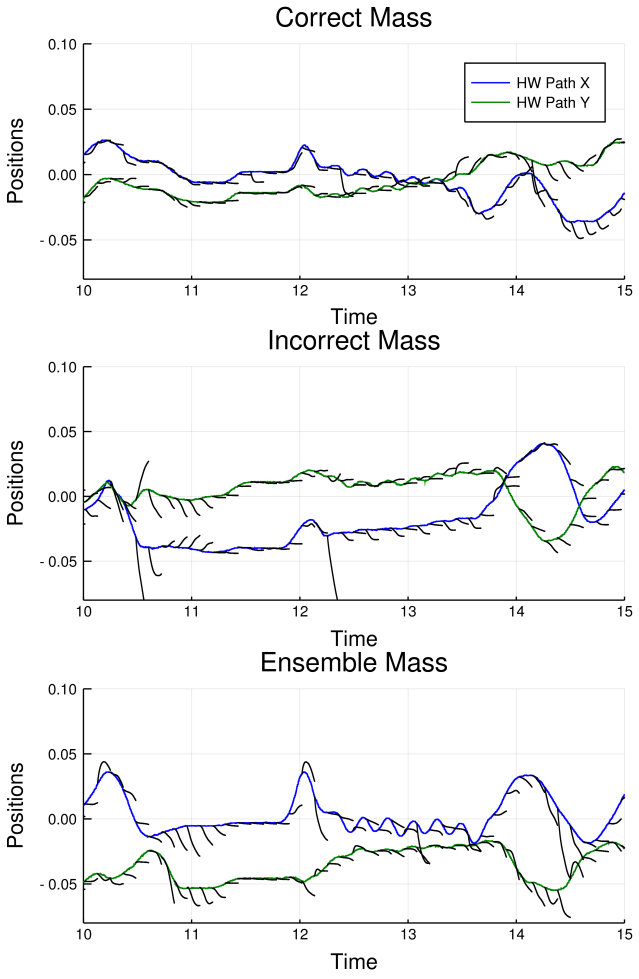

Fig. 5. We show the effects of different controllers. We seed our simulator with the hardware state, and, in simulation, perform a rollout of 200 time-steps – about 0.1 seconds. The correct policy (trained with measured mass), has rollouts that closely match the measured hardware state data. The incorrect policy (trained with an incorrect mass), performs differently in simulation. The remaining ensemble policy performs better than the incorrect one; this demonstrates that a 'safe' policy can be learned to at least begin an iterative data collection process. While it could be expected that the policies perform similarly in both simulation and hardware, we see that it is not the case here. A policy trained on an incorrect model would over-fit to the incorrect model, and changing to one of two different models (i.e. simulation or hardware) can have un-intuitive effects.

without human demonstrations to shape the behavior. We also eschewed the use of a state estimator during training and run-time as this would add additional modeling reliance and complexity.

Additionally, we show how simulated ensemble methods provide two major benefits. Firstly, it can partially make up for incorrectly measured / identified model parameters. This benefit should be obvious: it can be difficult to measure model parameters affecting nonlinear physical phenomena. Additionally, training in an ensemble has the added benefit of allowing for more conservative policies to enable appropriate data collection for actual model improvement. A natural extension of this observation would be full model adaptation using a technique such as EPopt [42].

Model adaptation suggests an obvious separation between

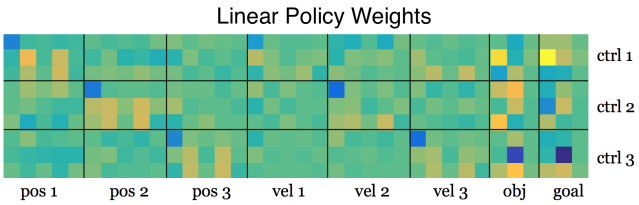

Fig. 6. We render the learned linear policy by combining the matrix and bias vector. A distinct pattern can be seen in the three negative weights above input dimensions 1-15, and 16-30. These correspond to the control outputs for the each of three three Phantom's turret joint; as each robot is sensitive, the policy outputs a negative gain to avoid large control forces. Additionally, we can see that the first and second Phantom contributes primarily the object's X location, while the third Phantom handles the Y location. Otherwise, the learned policy is not intuitive; the learned values between columns 1-30 seemingly have no pattern other than what was mentioned above.

model-based and model-free RL methods. Leveraging a model in simulation can provide a useful policy to begin robot operation. Very dynamic behaviors may not be suited to RL on hardware itself, without significant human imposed safety constraints, that both take time to develop and may not account for all use cases. Given that most robots are manufactured using modern techniques, a model to be used in simulation is very likely to exist: this should be used.

A. *Future Work*

Additional future work would be to push the capabilities of this automated policy learning method. This hardware platform acts as a representation of in-hand manipulation; lifting, rotating, rolling, throwing, are all very dynamic behaviors that would be better suited to initial exploration in simulation before attempting on hardware.

REFERENCES

[1] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," pp. 5026–5033, 2012.

[2] S. Kakade, "A natural policy gradient," in *NIPS*, 2001.

[3] J. Peters and S. Schaal, "Natural actor-critic," *Neurocomputing*, vol. 71, pp. 1180–1190, 2007.

[4] A. Rajeswaran, K. Lowrey, E. Todorov, and S. Kakade, "Towards generalization and simplicity in continuous control," *arXiv preprint arXiv:1703.02660*, 2017.

[5] S. Kolev and E. Todorov, "Physically consistent state estimation and system identification for contacts," in *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on*. IEEE, 2015, pp. 1036–1043.

[6] I. Mordatch, K. Lowrey, and E.Todorov, "Ensemble-CIO: Full-body dynamic motion planning that transfers to physical humanoids," in *IROS*, 2015.

[7] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 4906–4913.

[8] I. Mordatch, E. Todorov, and Z. Popović, "Discovery of complex behaviors through contact-invariant optimization," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 4, p. 43, 2012.

[9] Y. Tassa, N. Mansard, and E. Todorov, "Control-limited differential dynamic programming," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 1168–1175.

[10] S. Feng, E. Whitman, X. Xinjilefu, and C. G. Atkeson, "Optimization-based full body control for the darpa robotics challenge," *Journal of Field Robotics*, vol. 32, no. 2, pp. 293–312, 2015.

[11] J. Koenemann, A. Del Prete, Y. Tassa, E. Todorov, O. Stasse, M. Bennewitz, and N. Mansard, "Whole-body model-predictive control applied to the hrp-2 humanoid," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 2015, pp. 3346–3351.

[12] M. Raibert, K. Blankespoor, G. Nelson, and R. Playter, "Bigdog, the rough-terrain quadruped robot," *IFAC Proceedings Volumes*, vol. 41, no. 2, pp. 10 822–10 825, 2008.

[13] D. Nguyen-Tuong and J. Peters, "Model learning for robot control: a survey," *Cognitive processing*, vol. 12, no. 4, pp. 319–340, 2011.

[14] I. Mordatch, N. Mishra, C. Eppner, and P. Abbeel, "Combining model-based policy search with online model learning for control of physical humanoids," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016, pp. 242–248.

[15] Y. Pan and E. A. Theodorou, "Data-driven differential dynamic programming using gaussian processes," in *American Control Conference (ACC), 2015*. IEEE, 2015, pp. 4467–4472.

[16] J. M. Wang, D. J. Fleet, and A. Hertzmann, "Optimizing walking controllers for uncertain inputs and environments," in *ACM Transactions on Graphics (TOG)*, vol. 29, no. 4. ACM, 2010, p. 73.

[17] G. Lee, S. S. Srinivasa, and M. T. Mason, "Gp-ilqg: Data-driven robust optimal control for uncertain nonlinear dynamical systems," *arXiv preprint arXiv:1705.05344*, 2017.

[18] S. Ross and J. A. Bagnell, "Agnostic system identification for model-based reinforcement learning," *arXiv preprint arXiv:1203.1007*, 2012.

[19] M. Johnson, B. Shrewsbury, S. Bertrand, T. Wu, D. Duran, M. Floyd, P. Abeles, D. Stephen, N. Mertins, A. Lesman, *et al.*, "Team ihmc's lessons learned from the darpa robotics challenge trials," *Journal of Field Robotics*, vol. 32, no. 2, pp. 192–208, 2015.

[20] J. Peters, S. Vijayakumar, and S. Schaal, "Reinforcement learning for humanoid robotics," in *Proceedings of the third IEEE-RAS international conference on humanoid robots*, 2003, pp. 1–20.

[21] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[22] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[23] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 3389–3396.

[24] M. Deisenroth and C. E. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *Proceedings of the 28th International Conference on machine learning (ICML-11)*, 2011, pp. 465–472.

[25] Y. Chebotar, M. Kalakrishnan, A. Yahya, A. Li, S. Schaal, and S. Levine, "Path integral guided policy search," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 3381–3388.

[26] A. Yahya, A. Li, M. Kalakrishnan, Y. Chebotar, and S. Levine, "Collective robot reinforcement learning with distributed asynchronous guided policy search," *arXiv preprint arXiv:1610.00673*, 2016.

[27] J. Kober and J. R. Peters, "Policy search for motor primitives in robotics," in *Advances in neural information processing systems*, 2009, pp. 849–856.

[28] W. Montgomery, A. Ajay, C. Finn, P. Abbeel, and S. Levine, "Reset-free guided policy search: efficient deep reinforcement learning with stochastic initial states," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 3373–3380.

[29] I. Menache, S. Mannor, and N. Shimkin, "Basis function adaptation in temporal difference reinforcement learning," *Annals of Operations Research*, vol. 134, no. 1, pp. 215–238, 2005.

[30] S. Barrett, M. E. Taylor, and P. Stone, "Transfer learning for reinforcement learning on a physical robot," in *Ninth International Conference on Autonomous Agents and Multiagent Systems-Adaptive Learning Agents Workshop (AAMAS-ALA)*, 2010.

[31] M. Cutler, T. J. Walsh, and J. P. How, "Reinforcement learning with multi-fidelity simulators," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 3888–3895.

[32] P. Abbeel, M. Quigley, and A. Y. Ng, "Using inaccurate models in reinforcement learning," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 1–8.

[33] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3, pp. 229–256, 1992.

[34] S. Amari, "Natural gradient works efficiently in learning," *Neural Computation*, vol. 10, pp. 251–276, 1998.

[35] J. Peters, "Machine learning of motor skills for robotics," *PhD Dissertation, University of Southern California*, 2007.

[36] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, "Trust region policy optimization," in *ICML*, 2015.

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

[38] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *ICLR*, 2016.

[39] J. Bezanson, S. Karpinski, V. B. Shah, and A. Edelman, "Julia: A fast dynamic language for technical computing," *arXiv preprint arXiv:1209.5145*, 2012.

[40] I. Mordatch, K. Lowrey, G. Andrew, Z. Popovic, and E. V. Todorov, "Interactive control of diverse complex characters with neural networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 3132–3140.

[41] I. Mordatch, K. Lowrey, and E. Todorov, "Ensemble-cio: Full-body dynamic motion planning that transfers to physical humanoids," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 2015, pp. 5307–5314.

[42] A. Rajeswaran, S. Ghotra, S. Levine, and B. Ravindran, "Epopt: Learning robust neural network policies using model ensembles," *arXiv preprint arXiv:1610.01283*, 2016.