Policy Transfer via Modularity

Ignasi Clavera*[1], David Held*[1] and Pieter Abbeel[1,2,3]

***Abstract*— Non-prehensile manipulation, such as pushing, is an important function for robots to move objects and is sometimes preferred as an alternative to grasping. However, due to unknown frictional forces, pushing has been proven a difficult task for robots. We explore the use of reinforcement learning to train a robot to robustly push an object. In order to deal with the sample complexity of training such a method, we train the pushing policy in simulation and then transfer this policy to the real world. In order to ease the transfer from simulation, we propose to use modularity to separate the learned policy from the raw inputs and outputs; rather than training "end-to-end," we decompose our system into modules and train only a subset of these modules in simulation. We further demonstrate that we can incorporate prior knowledge about the task into the state space and the reward function to speed up convergence. Finally, we introduce "reward guiding" to modify the reward function and further reduce the training time. We demonstrate, in both simulation and real-world experiments, that such an approach can be used to reliably push an object from many initial positions and orientations.**

I. Introduction

Non-prehensile manipulation, such as pushing, can be used by robots to move or to rearrange objects. Compared to grasping, pushing may be easier or even necessary in certain cases. For example, pushing is especially important for moving objects that are too big or too heavy to grasp [1]. Pushing can also be used to move multiple objects at once, such as when clearing space on a cluttered table, in which the robot can sweep multiple objects out of the way in a single stroke [2]. Additionally, if a robot needs to grasp an object, it may need to first push the object to a more ideal position or orientation before grasping [3], [4], [5].

However, pushing is a challenging task for robots due to frictional forces that are difficult to model [6], [7], [8], [9]. Many of the assumptions that are commonly made for analyzing pushing motions have been shown to not always hold in practice. For example, the frictional forces involved can be non-uniform, time-varying, anisotropic, and deviating in other ways from the assumptions that are usually made in pushing models [10].

In this work, we explore the use of reinforcement learning for robot pushing tasks. Reinforcement learning has shown to be a powerful technique for environments with unknown dynamics or for tasks with complex dynamics that are difficult to explicitly optimize over. Reinforcement learning has been used for tasks such as learning to swing a bat, perform locomotion, stand up from a sitting position, drive a car, and many other tasks [11], [12], [13], [14], [15], [16].

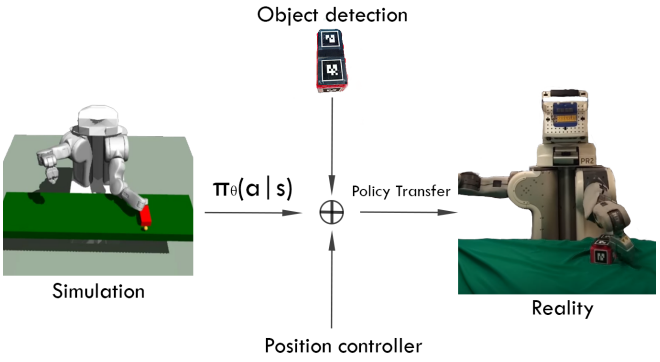

Fig. 1: After training a policy to perform a pushing task in the simulator, we transfer the policy directly to the real world. We decompose the system into the modules of a pose estimator, a policy, and a position controller. This decomposition leads to a robust transfer of the policy from simulation to the real world.

However, reinforcement learning methods, especially those using a neural network to represent the policy, can often be difficult to use in the real world due to the large number of samples required for learning. This is especially true for deep policy gradient methods; such methods have demonstrated impressive results in simulation, but their use in the real world is limited by their large sample complexity [17], [13], [14], [16]. Based on the results that have been demonstrated in simulation, we are motivated to explore whether such methods can also be made to work in the real world.

We propose an approach for dealing with the sample complexity by training a policy in simulation, where many training samples can be quickly generated. We then transfer the trained policy into the real world. However, such an approach is challenging due to mismatch between the simulation and the real world: images in simulation may differ from images in the real world, and the system dynamics in simulation can differ from the dynamics in the real world.

We show that deep reinforcement learning methods can be trained in simulation and then transferred to the real world by using the principle of modularity. In contrast to most deep reinforcement learning approaches which train the system end-to-end (e.g. from pixels to torques) [13], [16], [18], we break the problem into multiple separate pieces. Our system has a module that maps from image inputs to object pose, from object pose to target joint positions, and from target

*The first two authors contributed equally to this work

[1] Department of Electrical Engineering and Computer Science, UC Berkeley, iclavera@berkeley.edu davheld@eecs.berkeley.edu

[2] OpenAI

[3] International Computer Science Institute (ICSI) pabbeel@berkeley.edu

joint positions to motor torques. As shown in Figure 1, we train only the middle module in simulation and then transfer this module to the real-world. The surrounding modules are designed in both simulation and in the real-world to enable ease of transfer.

Further, we show that we can incorporate our *prior knowledge* about the task into the system through modifications to the state space and to the reward function. This is again in contrast to the traditional deep reinforcement learning approach in which minimal prior knowledge of the robot task is incorporated into the learning procedure [13], [14], [15], [16].

Finally, we formulate a new approach for varying the reward function over iterations of the optimization, which we refer to as "reward guiding." We show that our approach leads to faster convergence (compared to other approaches such as reward shaping) without modifying the optimal policy asymptotically. Our simulation and real-world experiments illustrate the success of our approach for robot pushing tasks.

II. Related Work

Non-prehensile Manipulation. Pushing, a form of non-prehensile manipulation, has been studied in robotics for many years. An early analysis showed that one can compute the direction of rotation of the pushed object, based on the direction of the friction cone and the pushing force compared to the location of the center of friction [3]. Others have modeled frictional forces using the notion of a limit surface [6], [7], [8], [9].

However, all these approaches make a number of assumptions about the friction that recent experiments have shown do not always hold in practice [10]. For example, a recent study has shown that, for certain materials, the friction of an object can vary greatly over the object surface. The friction can also vary over time (it becomes lower as the object is repeatedly pushed and the surface is smoothed) and it can vary based on the object speed and the direction of pushing [10]. This study showed that most of the models that are normally used for estimating friction, such as the principle of maximum-power inequality [6] and the ellipsoidal approximation of a limit surface [8], do not always hold in practice, resulting in pushing motions that differ from those predicted by these models.

Learning a Dynamics Model. Some previous efforts have been made to deal with the uncertainties in friction by learning how to push objects [19], [20], [21], [22], [23], [18]. These approaches typically involve first learning a dynamics model for how an object will respond when pushed, and then choosing actions based on this dynamics model. However, as discussed above, the dynamics model for pushing can itself be fairly complex, sometimes involving a non-uniform friction distribution over the object surface [10]. Thus, learning an accurate dynamics model is itself a challenge, and once the dynamics model is learned, finding an optimal policy for pushing can be similarly complex. In practice, a number of approximations are usually made to the dynamics model to make it easier to learn and to optimize over (such as assuming linear-Gaussian dynamics as a function of the state [18]), but these approximations might lead to a reduced task accuracy [24]. In contrast, we explore whether we can directly learn a policy for pushing using reinforcement learning.

Policy Transfer. Due to the long training times required to learn complex policies using reinforcement learning algorithms, we explore whether we can train such policies in simulation and then transfer the trained policies to the real world. Most of previous work for policy transfer involves policies that do not involve object interaction [25], [26], [27], [28]. We explore whether we can transfer policies that use non-prehensile manipulation (i.e. pushing).

Furthermore, many transfer learning techniques require further training in the real-world to adapt the policy to the real-world dynamics [29]. However, such an approach does not work well when training complex policies represented by neural networks trained using policy gradient reinforcement learning algorithms, which require many samples for the policy to converge. We explore whether we can train policies in simulation and transfer them to the real-world directly, without further time-consuming fine-tuning required.

III. Methods

A. Overview

Our approach to object pushing aims at training a policy in simulation and transferring it to the real-world. The task that the robot has to perform consists of pushing a block placed on a table, starting from any initial position and orientation, to a certain goal position (for this work, we are not concerned with the final orientation of the block).

In order to train the policy quickly in simulation, we explore how we can incorporate prior knowledge into the learning algorithm by modifying the input state and the reward function. This is in contrast to many current approaches for deep reinforcement learning which attempt to train the system end-to-end, from pixels to torques, without any prior knowledge of the task. Additionally, we explore using the idea of modularity to enable the policy to easily transfer from simulation to the real-world. These ideas will be discussed in further detail below.

B. Modularity

Rather than learn a policy end-to-end, from pixels to torques, we use the idea of modularity to decompose the policy into multiple pieces, as shown in Figure 2. Each piece can be designed separately and then composed together to obtain the entire system. We show that decomposing the policy in this manner allows us to easily transfer a policy trained in simulation to the real world.

1) Image to Object Pose: The input to our system is an image containing the object to be pushed. In order to push this object correctly, we need to obtain the object position. To achieve this, we use AR tags, which can be easily identified and used to determine the object pose. An alternative approach would be to train an object pose

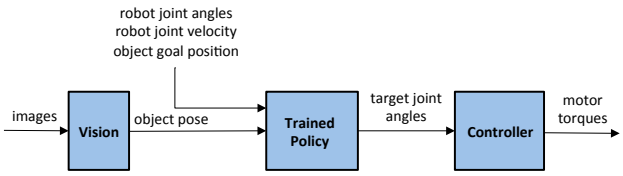

Fig. 2: Our system for robot pushing. The center module ("Trained Policy") is trained in simulation and transferred to the real world. The other modules are designed to create the desired inputs and outputs for ease of transfer.

detector, using one of various image algorithms for such tasks [30], [31], [32], [33]. When we train the policy in simulation, we directly use the ground-truth object position, obtained via our simulator.

Separating the pose estimation from the rest of the system is crucial for transferring policies from simulation to the real world. The images rendered in simulation have a very different appearance than the images from the real world. Although some efforts have been made to increase the appearance variation of the simulated images [34], such an approach is still somewhat fragile, as differences between the simulated and real-world images can still lead to unexpected behavior.

2) Object Pose to Robot Joint Angles: Next, we learn a policy that maps from the object pose to the target robot position. We learn this policy entirely in simulation. The policy is trained using a reinforcement learning algorithm (TRPO with GAE) [13], [14], which allows the robot to learn a fairly complex policy for object pushing. However, because this policy is separated from the raw image inputs and the raw torque outputs by the surrounding modules (see Figure 2), the policy easily transfers from simulation to the real-world without any fine-tuning required.

3) Robot Joint Angles to Robot Torque: Rather than outputting the raw torques, the policy that we train in simulation outputs the target robot joint angles. We then use a PD controller to map from the target position to the robot torques. A similar approach has been used for reaching, leaning, balancing, and other tasks that do not involve object interaction [25], [26], [27], [28]. We demonstrate that such an approach can also be used for tasks that involve object manipulation, specifically non-prehensile manipulation (pushing).

C. Input State

A common approach for deep reinforcement learning is to train a policy using the minimal information necessary to perform the task. For example, the minimal information required for the robot pushing task is the robot joint angles and velocities; the object position, orientation, and velocity; and the goal position (i.e. the location to which the robot must push the object). Using only this information and sufficient training time, the robot is able to learn a policy to successively achieve the task.

However, for many robotics tasks, the algorithm designer might have prior knowledge about how the task should be performed. For example, in the case of object pushing, it is clear that the robot must first move its arm towards the object, and then it must move the object towards the goal. We thus simplify the task faced by the learning algorithm by adding additional features to our state-space: the three-dimensional vector from the robot end-effector to the object, and the three-dimensional vector from the object to the goal position, for a total of six additional features (See Figure 3). We show that adding these additional features significantly speeds up the time required to learn an optimal policy.

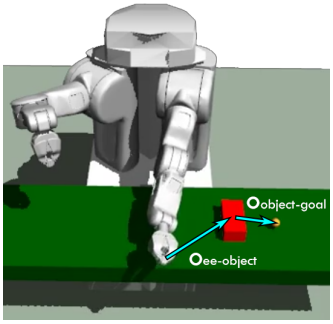

Fig. 3: We add additional inputs to our observation based on the vector from the end-effector to the object, $o_{ee-object}$, and the vector from the object to the goal, $o_{object-goal}$.

D. Reward Function

The goal of the robot is to push the object to the goal location. Thus, the minimal reward function necessary to complete this task is the negative distance from the object to the goal. However, the robot will have a difficult time learning a policy under this reward function, since most random actions that the robot takes will not have any affect on this reward (since most random actions will not cause the robot to interact with the object, and hence will not affect the distance between the object and the goal).

Thus, in order to decrease the training time, we add additional terms to the reward function to guide the policy in a direction to maximize the objective. For example, we add a term to the reward corresponding to the negative distance from the robot end-effector to the object, to encourage the robot to move its end-effector near to the object. We also add a term based on the angle between the end-effector, the object, and the goal, to encourage the robot to move its end-effector such that these three points lie on a straight line; specifically, we add $cos(o_{ee-object}, o_{object-goal})$, where $o_{ee-object}$ is the vector from the robot end-effector to the object and $o_{object-goal}$ is the vector from the object to the goal.

E. Reward Guiding

Modifying the reward function by adding additional terms can help to decrease the time to convergence, but it has the downside of modifying the learned policy. It has been shown that, in order for the optimal policy to be unchanged, then the reward function can only be modified in the following

way:

$$r(s,a,s') = r_0(s,a,s') + \gamma\phi(s') - \phi(s) \tag{1}$$

where $r(s,a,s')$ is the new reward function when transitioning from state s to state s' after taking action a, $r_0(s,a,s')$ is the original reward function, γ is the discount factor, and $\phi(s)$ is a real-valued function over states [35].

Although such a modification has been shown to not affect the optimal policy, unfortunately it also does not always speed up the training time as much as one would like. The reason for this is simple: the effect of modifying the reward function in this way is equivalent to simply subtracting a baseline [36]. We can see this if we accumulate the reward function to compute the return. The original, unmodified return is given by

$$R_0 = \sum_{k=0}^{\infty} \gamma^k r_{0,k} \tag{2}$$

where $r_{0,k}$ is the unmodified reward r_0 received on the kth timestep. Then the modified return is given by [36]

$$\begin{aligned} R' &= \sum_{k=0}^{\infty} \gamma^k (r_{0,k} + \gamma\phi(s_{k+1}) - \phi(s_k)) && (3)\\ &= \sum_{k=0}^{\infty} \gamma^j r_{0,k} + \sum_{k=0}^{\infty} \gamma^{k+1}\phi(s_{k+1}) - \sum_{k=0}^{\infty} \gamma^k \phi(s_k)) && (4)\\ &= R_0 + \sum_{i=1}^{\infty} \gamma^k \phi(s_k) - \left(\phi(s_0) + \sum_{k=1}^{\infty} \gamma^k \phi(s_k)\right) && (5)\\ &= R_0 - \phi(s_0) && (6)\\ & && (7) \end{aligned}$$

where s_k is the state encountered at the kth timestep. Thus the return is only changed by subtracting a baseline which is a function of the initial state s_0. It has been shown that subtracting a baseline, especially an optimal baseline, can help reduce the variance of the estimator [37], [12]. However, this is unfortunately a somewhat restricted way to modify the reward and we show that this can lead to slower convergence than might be possible.

Instead, we set the reward-shaping discount factor from Equation 1 to $\gamma = \gamma(i)$, in which the discount factor can vary over the iterations of the optimization. Let γ_M be the discount factor from the MDP. We then increase the reward-shaping discount factor over later iterations of the optimization as:

$$\gamma(i) = \gamma_M(1 - \exp(-i/\tau)) \tag{8}$$

where $\gamma(0) = 0$ and $\gamma(i) \to \gamma_M$ as $i \to \infty$. Thus, at convergence, $\gamma(i) \to \gamma_M$ and our time-varying reward shaping method approaches the original reward shaping formulation from Eq. 1, and thus the optimal policy at convergence is unchanged (the proof from the original reward shaping formulation [35] applies at convergence). However, by varying $\gamma(i)$ we are able to achieve a more flexible form of reward shaping that leads to faster convergence. We name this approach "reward guiding".

In this formulation, the discount factor $\gamma(i)$ can be viewed as a variance reduction parameter. A similar perspective of the discount factor is taken in previous work [14], in which the discount factor of the MDP is viewed as a variance reduction parameter compared to the undiscounted MDP.

IV. Implementation Details

Our neural network maps from the input state to target robot joint positions, which are defined as changes in angle from the current joint positions. The network is defined by 3 hidden layers followed by the output layer. Each hidden layer has 64 nodes, with tanh non-linearities between each hidden layer. The network is implemented in Theano [38]. For the reinforcement learning algorithm, we use TRPO [13] and Generalized Advantage Estimation (GAE) [14] with a linear baseline using the implementation from rllab [17]. We use the default parameters from TRPO in rllab for our optimization, and we train the policy for 15,000 iterations with a batch size of 50,000 timesteps and 150 timesteps per episode. We use a discount factor of γ = 0.95, a KL-divergence constraint of δ_{KL} = 0.01, and a λ = 0.98 from GAE. We train the policy in simulation using the Mujoco [39] simulator.

We use a position controller to map the target joint angles output by the network to motor torques. For ensuring good transfer from simulation to reality, we found that it is important that the controller gains are set low enough such that the controller does not overshoot. In simulation, we allow enough time between actions to allow the controller reach the target position with an error smaller than 0.01 radians.

On the real PR2, we detect the position of the object using the Robot Operating System (ROS) package `ar_track_alvar`. For the position controller, we use the integrated position controller of the PR2, from the ROS package `pr2_controller_manager`.

V. Results

The experiments were designed to evaluate the impact of our contributions, both of how to learn the policy learning in simulation and how to transfer the policy from simulation to reality. In particular we seek to answer the following questions:

- Can we incorporate prior task knowledge into the observation space or reward function to increase the sampling efficiency or improve task performance?
- Can reward guiding improve the learning time without asymptotically modifying the MDP?
- How does a learning-based approach to non-prehensile manipulation compare to that of a hard-coded baseline?
- Can modularity be used to transfer a policy from simulation to the real-world while maintaining a similar level of performance?

We analyze our method on simulation in Sections V-A, V-B, and V-C. And We show the results on real world in Section V-D. In addition to the below experiments, videos of our results are also available online[1].

[1] `https://goo.gl/fehPWw`

A. Simulation Training

1) Prior knowledge in the state space: We analyze the effect of modifying the state space on the training time. In a typical deep reinforcement learning setup, the system only receives the minimal state space as input. In this case, that would include the robot joint angles and velocities; the object position, orientation, and velocity; and the goal position to which the robot must push the object. We call this minimal observation o_{base}.

However, we show that we can augment the state space to improve the convergence rate. We define $o_{ee-object}$ as the 3-dimensional vector pointing from the robot end-effector to the object, and we define $o_{object-goal}$ as the 3-dimensional vector pointing from the object to the goal. Figure 4 shows that adding these terms to the state space reduces the training time needed for the robot to learn to perform the task.

Fig. 4: We use prior knowledge about the task to modify the state space. Our results shown that adding prior knowledge can lead to faster convergence compared to just using the original state space.

2) Prior knowledge in the reward function: Next, we analyze the effect of modifying the reward function has on training time. We explore the contribution of each of the following reward terms:

(i) $r_{obj,goal} = -d(obj, goal)$
(ii) $r_{angle} = cos(o_{ee-object}, o_{object-goal})$
(iii) $r_{ee,obj} = -d(ee, obj)$

The first term, $-d(obj, goal)$, measures the negative distance between the object and the goal (the position to which the object must be pushed). This term must be included in order to encourage the robot to push the object towards the goal.

The second term, $cos(o_{ee-object}, o_{object-goal})$, encourages the robot's end-effector to approach the object from the appropriate angle. The term $o_{ee-object}$ is the vector from the robot end-effector to the object, and $o_{object-goal}$ is the vector from the object to the goal. By measuring the cosine between these vectors, we encourage the end-effector, the object, and the goal to lie along a straight line. However, the weight on this term is relatively small because this term is only intended to guide the robot end-effector to the approximately correct position. For accurate pushing, the robot may occasionally need to move its end-effector to a different location that deviates from this line.

The third term, $-d(ee, obj)$, measures the negative distance from the robot end-effector to the object. This term will encourage the robot end-effector to move near the object. If the robot is not given this term, then it will initially perform actions that do not interact with the object. The robot would then not see any change to the distance between the block and the goal, and the robot would not receive any feedback that allows it to discover whether its actions are useful. By adding this term, the robot's initial actions are guided towards the object. However, the weight on this term is again small because this is not the primary objective; the robot has to move the end-effector around, and occasionally away from the block, in order to re-position and push the block in different directions.

We explore the effect of each of these terms by investigating the following reward functions:

(i) $r_1 = r_{obj,goal}$
(ii) $r_2 = r_{obj,goal} + w_1 r_{angle}$
(iii) $r_3 = r_{obj,goal} + w_2 r_{ee,obj}$
(iv) $r_4 = r_{obj,goal} + w_1 r_{angle} + w_2 r_{ee,obj}$

where w_1 and w_2 are weighting factors on each of the reward terms.

The results are shown in Figure 5. In the plot, we have named r_1 = baseline, r_2 = baseline + angle, r_3 = baseline + ee-distance, and r_4 = baseline + angle + ee-distance. We can see that, if only the minimum reward function is specified, as in r_1, then the robot makes very little progress in learning, since as mentioned, the robot's initial random actions will not usually affect the distance from the block to the goal. Adding in only the angle term (as in r_2) also does not have much effect. If we encourage the robot to move its end-effector towards the object (as in r_3), then we see a significant improvement in the training time. Combining all three terms leads to the best convergence; once the robot is encouraged to move towards the block, encouraging the robot to move and place its end-effector at the correct angle further helps guide the robot towards the correct pushing behavior.

Fig. 5: We use prior knowledge about the task to modify the reward function. Our results show that adding prior knowledge can lead to significantly faster convergence compared to just using the original reward function.

B. Reward guiding

Finally, we analyze the increase in sample efficiency of using our proposed annealing reward shaping, which we refer to as "reward guiding," instead of the conventional reward shaping method from Ng, *et. al.* [35].

For guiding we use the reward $r_{obj,goal}$, and $\phi(s) = w_1 r_{angle} + w_2 r_{ee,obj}$ as potential function. This results in a faster learning (as show in section V-A.2), and at the same time we optimize the original objective function asymptotically. Then our reward can be written as:

$$r_{guiding} = r_{obj,goal}(s') + \gamma(i)[w_1\, r_{angle} + w_2\, r_{ee,obj}](s') - [w_1\, r_{angle} + w_2\, r_{ee,obj}](s)$$

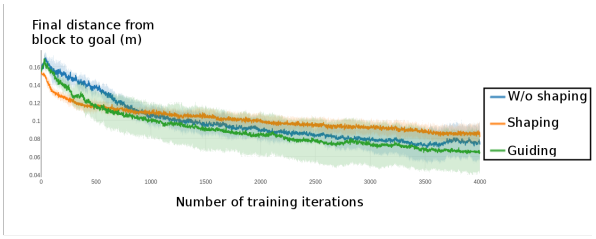

Fig. 6: We measure the performance over iterations of reward guiding, shaping, [35] and learning without shaping (i.e. just adding extra reward terms). Our results show that reward guiding can lead to faster convergence compared to just using reward shaping or learning without shaping.

Figure 6 shows the comparison between using reward guiding, reward shaping, and modifying the reward without proper shaping by adding additional reward terms (as in r_4 from the previous section).

The results show that reward guiding has the best performance. Reward guiding accomplishes the same final distance from the block to the goal as shaping after 40% of the total number iterations. Ng, *et. al.* [35] proved that reward shaping does not affect the optimal policy. Similarly, reward guiding is guaranteed to have the same final optimal policy, but simply adding extra reward terms (as in r_4) would not.

C. Baseline Comparison

In this section, we show the need for learning-based methods for non-prehensile manipulation, especially in domains where the robot needs to push objects from arbitrary initial positions and orientations. To demonstrate the need for learning in such scenarios, we compared our performance to that of a non-learning based ("hard-coded") baseline procedure. In our "hard-coded baseline", at the beginning of each episode, a gripper is placed on the opposite side of the object from the goal. The gripper then pushes the object along the vector that goes from the center of the object to the goal until the distance from the object to the goal stops decreasing.

We compare the performance of our learning-based approach to that of the baseline, testing both methods in simulation (the real-world experiments of our method are desribed in Section V-D). In these experiments, the goal is fixed, and the object is placed in every point of a grid of size 0.4 m $\times$ 0.25 m, with a resolution of 0.01 m, and the object orientation is sampled from a random uniform distribution on $[-\pi, \pi]$.

Figure 7 shows the final distance between the object and the goal from each starting position. In it we can see that our method is robust to varying initial positions. On the other hand, the hard-coded baseline has much worse accuracy for some initial object positions. A histogram of the final distances between the block and the goal can be found in Figure 8. The poor performance displayed by the baseline is due to the variability in the environment, specifically in the orientation of the object relative to the goal position.

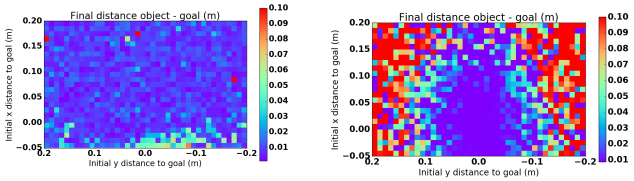

Fig. 7: Heatmap of the final distance between the object and the goal. The axes denote the relative initial position of the object with respect the goal. The final distance is thresholded to a maximum value of 0.1 m. Red indicates that the final distance from the object to the goal is higher than 0.1m, whereas purple indicates a 0 distance. **Left:** Our method. **Right:** Baseline. (Best viewed in color).

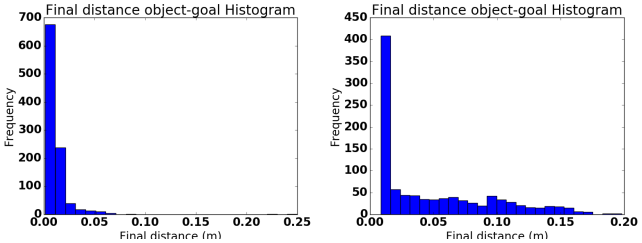

Fig. 8: Histogram of the final distance between the object and the goal. Thresholded to a maximum value of 0.25 m. **Left:** Our method. **Right:** Baseline.

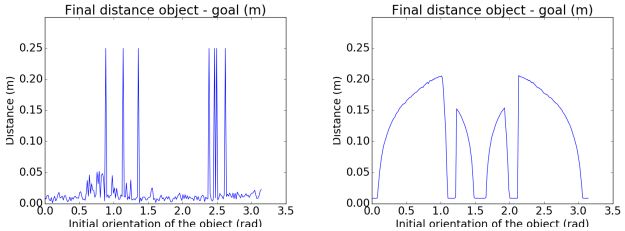

Fig. 9: Final distance between the object and the goal with respect the orientation of the object in the plane of the table. **Left:** Our method. **Right:** Baseline.

Figure 9 shows the final distance between the block and the goal as a function of the initial orientation of the object. The orientation is defined as the angle between longest side of the block and the x-axis.

The baseline fails to push the object for some initial orientations since the gripper slides off the side of the block instead of pushing it closer to goal. In contrast to our learning-based approach, the baseline is unable to recover from such situations. The sharp decrease in the final distance around 1.1 radians is due to the orientation at which the gripper starts pushing against the corner or against the shortest side of the block. To view some examples of the baseline failure cases, see the online video[2].

In contrast, our learned policy is robust to changes of

[2] https://goo.gl/fehPWw

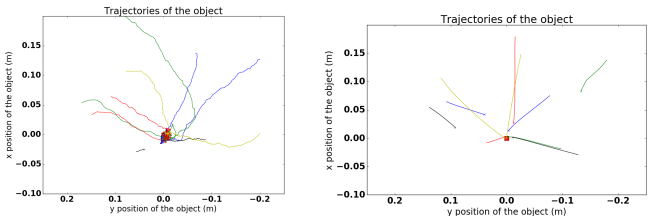

Fig. 10: Trajectories of the center of the object while being pushed. The axes denote the relative position of the object with respect to the goal. The red square identifies the goal, which is located at the origin. **Left:** Our method. **Right:** Baseline. (Best viewed in color).

orientation, as seen in Figure 9 (left). The learned policy is able to perform complex pushing trajectories, correcting the trajectory and recovering the object if it has pushed the block too far. A visualization of the pushing trajectories produced by our method, compared to those of the baseline, are shown in Figure 10.

D. Real-World Pushing

In this section we show how modularity leads to a good performance when transferring the policy from simulation to real world.

The experiments with the real PR2 were implemented with the same set up as described in the former section: the goal is placed in a fixed position and the object is randomly sampled on the table within a 1000 cm^2 area. Each episode is terminated after 300 timesteps or when the distance from the object to the goal is less than 2 cm (whichever happens first). We sampled 20 initial object positions to test the performance of our approach.

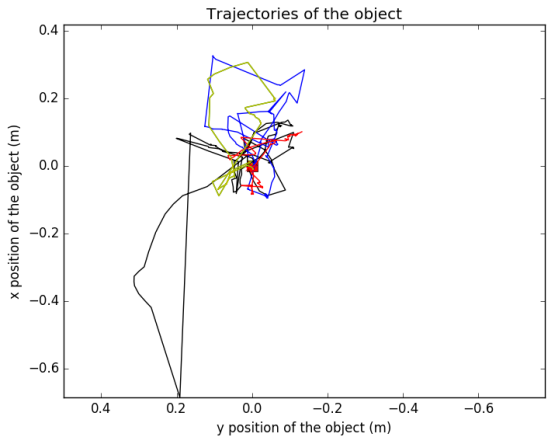

Fig. 11: Trajectories of the center of the object while being pushed in the real world. The axes denote the relative initial position of the object with respect the goal (located at the origin). The red square identifies the goal. (Best viewed in color).

The block in simulation and real-world are similar in size; but the mass distribution is different, the simulated block is solid, whereas the real one is hollow. The friction between

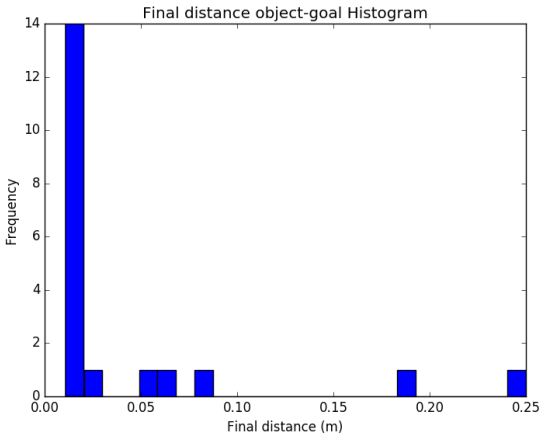

Fig. 12: Histogram of the final distance between the object and the goal. Thresholded to a maximum value of 0.25 m.

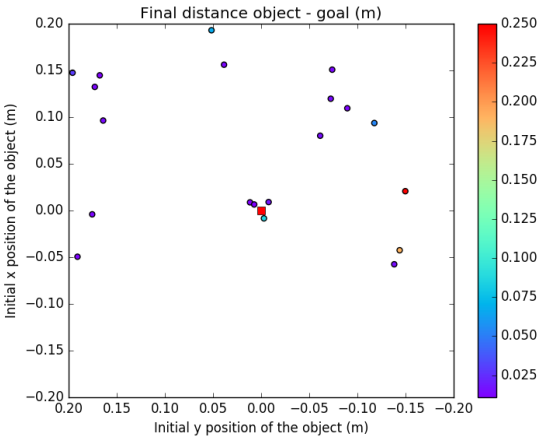

Fig. 13: Scatter plot of the final distance between the object and the goal. The axis denote the relative initial position of the object with respect the goal. Thresholded to a maximum value of 0.1 m. Red indicates that the final distance from the object to the goal is higher than 0.1m, whereas purple indicates a 0 distance. (Best viewed in color).

the block and the table and between the block and the gripper are also different between simulation and the real world. This leads to different dynamics of the object and different resulting block trajectories, as shown in Figure 11. Nevertheless, our method is capable of performing the task (defined as at the final time step being within 2cm of the goal) in 70% of the episodes, as shown in Figure 12. Figure 13 shows the final distance of the object for each of the 20 sampled initial positions, demonstrating that our method succeeds from varying initial block positions.

The failing cases, in which the block does not reach within 2 cm of the goal, were due to a poor estimation of the object position or orientation. The AR tags were detected from a single kinect camera attached to the robot's head. In some cases the robot arm occludes the AR tags, which led to an incorrect estimate of the object position. We mitigate this problem by estimating the position of the object using the

Mujoco simulator whenever the AR tags are not visible.

VI. Conclusion

In this paper, we present a transfer method based on the concept of modularity. We decouple the policy from the image inputs and the torque outputs. Our policy is thus more robust for transfer by using modularity to alleviate the mismatch problem between simulation and reality. We test our method with non-prehensible manipulation, specifically pushing. Our results show strong performance in reality that is comparable to our performance in simulation.

Furthermore, we present the concept of "reward guiding" which does not modify the asymptotic optimal policy and leads to a faster learning than reward shaping [35] or adding a linear combination of other reward terms. We also show the effects in performance and sample efficiency of adding prior knowledge of the task in the observation state and reward function.

References

[1] M. Dogar and S. Srinivasa, "A framework for push-grasping in clutter," *Robotics: Science and systems VII*, vol. 1, 2011.

[2] A. Cosgun, T. Hermans, V. Emeli, and M. Stilman, "Push planning for object placement on cluttered table surfaces," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, 2011, pp. 4627–4632.

[3] M. T. Mason, "Mechanics and planning of manipulator pushing operations," *The International Journal of Robotics Research*, vol. 5, no. 3, pp. 53–71, 1986.

[4] G. Lee, T. Lozano-Pérez, and L. P. Kaelbling, "Hierarchical planning for multi-contact non-prehensile manipulation," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 2015, pp. 264–271.

[5] M. R. Dogar and S. S. Srinivasa, "Push-grasping with dexterous hands: Mechanics and a method," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. IEEE, 2010, pp. 2123–2130.

[6] S. Goyal, A. Ruina, and J. Papadopoulos, "Planar sliding with dry friction part 1. limit surface and moment function," *Wear*, vol. 143, no. 2, pp. 307–330, 1991.

[7] R. D. Howe and M. R. Cutkosky, "Practical force-motion models for sliding manipulation," *The International Journal of Robotics Research*, vol. 15, no. 6, pp. 557–572, 1996.

[8] S. H. Lee and M. Cutkosky, "Fixture planning with friction," *Journal of Engineering for Industry*, vol. 113, no. 3, pp. 320–327, 1991.

[9] K. M. Lynch, H. Maekawa, and K. Tanie, "Manipulation and active sensing by pushing using tactile feedback." in *IROS*, 1992, pp. 416–421.

[10] K.-T. Yu, M. Bauza, N. Fazeli, and A. Rodriguez, "More than a million ways to be pushed. a high-fidelity experimental dataset of planar pushing," in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, 2016, pp. 30–37.

[11] R. Tedrake, T. Zhang, and H. Seung, "Learning to walk in 20 min," in *Proceedings of the Yale workshop on adaptive and learning systems*, 2005, pp. 10–22.

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

[13] J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz, "Trust region policy optimization." in *ICML*, 2015, pp. 1889–1897.

[14] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[15] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[16] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, 2016.

[17] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, 2016.

[18] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, vol. 17, no. 39, pp. 1–40, 2016.

[19] M. Salganicoff, G. Metta, A. Oddera, and G. Sandini, *A vision-based learning method for pushing manipulation*. University of Pennsylvania, 1993.

[20] T. Meriçli, M. Veloso, and H. L. Akın, "Push-manipulation of complex passive mobile objects using experimentally acquired motion models," *Autonomous Robots*, vol. 38, no. 3, pp. 317–329, 2015.

[21] M. Lau, J. Mitani, and T. Igarashi, "Automatic learning of pushing strategy for delivery of irregular-shaped objects," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 3733–3738.

[22] S. Walker and J. K. Salisbury, "Pushing using learned manipulation maps," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*. IEEE, 2008, pp. 3808–3813.

[23] J. Zhou, R. Paolini, J. A. Bagnell, and M. T. Mason, "A convex polynomial force-motion model for planar sliding: Identification and application," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016, pp. 372–377.

[24] Y. Chebotar, M. Kalakrishnan, A. Yahya, A. Li, S. Schaal, and S. Levine, "Path integral guided policy search," 2017 (In press).

[25] A. A. Rusu, M. Vecerik, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell, "Sim-to-real robot learning from pixels with progressive nets," *arXiv preprint arXiv:1610.04286*, 2016.

[26] P. Christiano, Z. Shah, I. Mordatch, J. Schneider, T. Blackwell, J. Tobin, P. Abbeel, and W. Zaremba, "Transfer from simulation to real world through learning deep inverse dynamics model," *arXiv preprint arXiv:1610.03518*, 2016.

[27] 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.

[28] M. Cutler and J. P. How, "Autonomous drifting using simulation-aided reinforcement learning," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016, pp. 5442–5448.

[29] 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.

[30] S. Gupta, P. A. Arbeláez, R. B. Girshick, and J. Malik, "Aligning 3D models to RGB-D images of cluttered scenes," in *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[31] A. Krull, F. Michel, E. Brachmann, S. Gumhold, S. Ihrke, e. D. Rother, Carsten", I. Reid, H. Saito, and M.-H. Yang, *6-DOF Model Based Tracking via Object Coordinate Regression*. Cham: Springer International Publishing, 2015, pp. 384–399.

[32] E. Brachmann, A. Krull, F. Michel, S. Gumhold, J. Shotton, and C. Rother, *Learning 6D Object Pose Estimation Using 3D Object Coordinates*. Cham: Springer International Publishing, 2014, pp. 536–551.

[33] E. Brachmann, F. Michel, A. Krull, M. Y. Yang, S. Gumhold, and C. Rother, "Uncertainty-driven 6d pose estimation of objects and scenes from a single rgb image," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 3364–3372.

[34] F. Sadeghi and S. Levine, "$(CAD)^2$RL: Real singel-image flight without a singel real image," *arXiv preprint arXiv:1611.04201*, 2016.

[35] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *ICML*, vol. 99, 1999, pp. 278–287.

[36] J. Asmuth, M. L. Littman, and R. Zinkov, "Potential-based shaping in model-based reinforcement learning." in *AAAI*, 2008, pp. 604–609.

[37] E. Greensmith, P. L. Bartlett, and J. Baxter, "Variance reduction techniques for gradient estimates in reinforcement learning," *Journal of Machine Learning Research*, vol. 5, no. Nov, pp. 1471–1530, 2004.

[38] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016. [Online]. Available: http://arxiv.org/abs/1605.02688

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