arXiv:2011.06332v1 [cs.RO] 12 Nov 2020

Joint Space Control via Deep Reinforcement Learning

Visak Kumar,[1] David Hoeller, Balakumar Sundaralingam, Jonathan Tremblay, Stan Birchfield
NVIDIA

***Abstract*— The dominant way to control a robot manipulator uses hand-crafted differential equations leveraging some form of inverse kinematics / dynamics. We propose a simple, versatile joint-level controller that dispenses with differential equations entirely. A deep neural network, trained via model-free reinforcement learning, is used to map from task space to joint space. Experiments show the method capable of achieving similar error to traditional methods, while greatly simplifying the process by automatically handling redundancy, joint limits, and acceleration / deceleration profiles. The basic technique is extended to avoid obstacles by augmenting the input to the network with information about the nearest obstacles. Results are shown both in simulation and on a real robot via sim-to-real transfer of the learned policy. We show that it is possible to achieve sub-centimeter accuracy, both in simulation and the real world, with a moderate amount of training.**

I. Introduction

Fundamental to robot control is the mapping from *task space* to *joint space*. With this mapping, a desired task described in the Cartesian world in which the robot moves can be translated into specific low-level commands to be sent to the joint motors to accomplish the task. The control law that maps Cartesian task space target poses to robot joint space commands is known as *operational space control* [1].

For decades, traditional approaches have leveraged analytic techniques based on differential equations to perform operational space control for reaching end-effector target poses [2]. The prebuilt controllers of commercially available robotic systems fall into this category. Such approaches, however, are unable to navigate in unstructured environments, because they cannot reason about constraints in joint space (*e.g.*, avoiding collision between the robot's links and the environment). This limitation greatly reduces the usefulness of pre-built operational space controllers. Recent methods [3]–[5] that integrate perception with control have shown success in reaching task space goals while avoiding dynamic obstacles. Such methods, however, require extensive robotics expertise to integrate into new manipulators, and they also require tuning for novel environments or tasks.

Reinforcement learning (RL) is a promising approach to replace analytic, hand-crafted solutions with learned policies. Most RL-based robotic systems learn in task space rather than in joint space [6]–[9]. By working in task space (*i.e.*, Cartesian end-effector space), the learning problem is greatly simplified, and the safe operation of the robot can be delegated to an analytic low-level controller that maps the RL actions to joint commands. Like the pre-built controllers mentioned above, however, such solutions cannot operate in unstructured environments in which collisions between the arm and environment require joint-level reasoning.

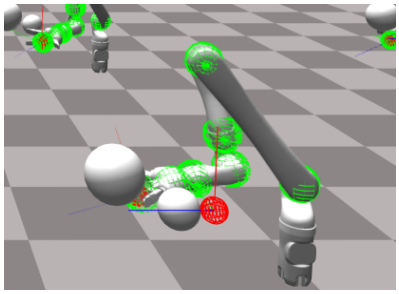


Fig. 1. Our deep RL-based JAiLeR system is able to learn a policy that enables a robot manipulator to reach any goal position (red sphere) in the workspace with sub-centimeter average error, while avoiding obstacles (silver spheres). The green points on the robot links are used for distance computations for collision avoidance.

In this paper, we present a Joint Action-space Learned Reacher (JAiLeR), a method that leverages recent advancements in model-free deep reinforcement learning to directly map task-space goals into joint-space commands. See Fig. 1. Redundancy, joint limits, and acceleration / deceleration profiles are automatically handled by learning directly from behavior in simulation, without the need to explicitly model the null space or manually specify constraints or profiles. Collision avoidance is incorporated into the approach by augmenting the network's input with distances between robot links and nearby surfaces, which allows the robot to avoid dynamic obstacles while still reaching target end-effector poses—without requiring scene-specific training. We show that JAiLeR can reach a large workspace with average error less than 1 cm, both in simulation and in reality.

II. Operational Space Control

For more than three decades, the dominant approach to controlling redundant robots has been operational space control [1]. In this section we provide background material and notation related to this approach.

A. Basics

A robot manipulator with n rotational joints is controlled by sending torques $\boldsymbol{\tau} \in \mathbb{R}^n$ to the motors. These torques can be computed via the inverse dynamics control law [10] as

$$\boldsymbol{\tau} = \boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}}_r + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}), \tag{1}$$

[1]Also affiliated with Georgia Tech. Work was performed while the first author was an intern with NVIDIA. The initial version of the approach was designed and implemented by the second author.

where $\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}} \in \mathbb{R}^n$ are the joint angles, velocities, and accelerations, respectively; $\boldsymbol{M}(\boldsymbol{q}), \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \in \mathbb{R}^{n \times n}$ and $\boldsymbol{g}(\boldsymbol{q}) \in \mathbb{R}^n$ are the manipulator's inertial matrix, the Coriolis forces, and gravitational force, respectively; and $\dot{\boldsymbol{q}}_r, \ddot{\boldsymbol{q}}_r \in \mathbb{R}^n$ are the reference velocity and acceleration, respectively. Despite the fact that Eq. (1) is an approximation that does not take into account the effects of joint friction or motor dynamics, this widely-used model has been found in practice to be sufficient for controlling manipulators.

For many manipulation tasks, we are interested in reaching a goal configuration $\boldsymbol{x}_d$ for the end-effector in Cartesian space (also known as *task space*, or *operational space*). This goal configuration can be a Cartesian position ($\boldsymbol{x}_d \in \mathbb{R}^3$), a full Cartesian pose ($\boldsymbol{x}_d \in \mathbb{SE}(3)$), or a position with orientation constraints. To solve this problem, the task space goal must be mapped to joint space. When the number of degrees-of-freedom is greater than the dimensionality of the task space, an infinite number of joint configurations will satisfy any particular goal configuration. This redundancy is often leveraged in manipulation tasks to satisfy other task requirements such as avoiding collisions.

The fundamental relationship between joint and task space velocities is $\dot{\boldsymbol{x}} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}$, where $\boldsymbol{J}(\boldsymbol{q})$ is the kinematic Jacobian of the manipulator. Differentiating this equation yields the corresponding relationship between accelerations in the two spaces:

$$\ddot{\boldsymbol{x}} = \boldsymbol{J}\ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}\dot{\boldsymbol{q}}, \tag{2}$$

where the dependency of $\boldsymbol{J}$ on $\boldsymbol{q}$ has been omitted for brevity.

Given a goal position, velocity, and/or acceleration in task space, we can use the equations above to compute the corresponding quantities in joint space. Then, given a desired position $\boldsymbol{q}_d$, velocity $\dot{\boldsymbol{q}}_d$, and acceleration $\ddot{\boldsymbol{q}}_d$ in joint space, the reference acceleration can be computed as

$$\ddot{\boldsymbol{q}}_r = k_p(\boldsymbol{q}_d - \boldsymbol{q}) + k_d(\dot{\boldsymbol{q}}_d - \dot{\boldsymbol{q}}) + \ddot{\boldsymbol{q}}_d, \tag{3}$$

where $k_p, k_d \in \mathbb{R}$ are the proportional and derivative gains, respectively.[1] The gain k_p, also known as the "stiffness gain", penalizes any deviation from the desired joint position $\boldsymbol{q}_d$; whereas k_d, often referred to as the "damping gain", dampens any sudden movement of the robot from its desired velocity $\dot{\boldsymbol{q}}_d$.

Since the reference acceleration is premultiplied by the robot's inertial matrix in Eq. (1), any error in the robot dynamics model will affect the robot's ability to reach the target. To alleviate this issue in real robotic systems, the control law obtained by combining Eqs. (1) and (3) is often modified to remove the premultiplication by $\boldsymbol{M}$ from the proportional and derivative terms:

$$\boldsymbol{\tau} = \boldsymbol{M}\ddot{\boldsymbol{q}}_d + \boldsymbol{C}\dot{\boldsymbol{q}}_d + \boldsymbol{g} + k_p(\boldsymbol{q}_d - \boldsymbol{q}) + k_d(\dot{\boldsymbol{q}}_d - \dot{\boldsymbol{q}}), \tag{4}$$

where for notational brevity the dependency of $\boldsymbol{M}$, $\boldsymbol{C}$, and $\boldsymbol{g}$ on the joint angles and velocities has been omitted.

[1] In practice, these gains are often diagonal matrices, to allow different scales for different joints.

B. Controllers

Nakanishi *et al.* [2] analyze the eight most common ways, in the context of operational space control (OSC), to map task space goals to joint torques to move the robot's end-effector to follow a particular Cartesian trajectory. The two best performing methods from their work are as follows.

1) Velocity-based controller (OSC-V).[2] Given a target position $\boldsymbol{x}_d$ and velocity $\dot{\boldsymbol{x}}_d$ in Cartesian task space, this controller first computes a reference velocity $\dot{\boldsymbol{x}}_r$ using a proportional-derivative (PD) controller:

$$\dot{\boldsymbol{x}}_r = \kappa_p(\boldsymbol{x}_d - \boldsymbol{x}) + \dot{\boldsymbol{x}}_d, \tag{5}$$

where κ_p is the stiffness gain in task space.

The reference joint velocity $\dot{\boldsymbol{q}}_r$ is then computed as

$$\dot{\boldsymbol{q}}_r = \boldsymbol{J}^+\dot{\boldsymbol{x}}_r - \alpha(\boldsymbol{I} - \boldsymbol{J}^+\boldsymbol{J})\boldsymbol{f}, \tag{6}$$

where $\boldsymbol{J}^+(\boldsymbol{q})$ is the Moore-Penrose pseudoinverse of the Jacobian; and $\boldsymbol{f} \in \mathbb{R}^n$ is the null space force, which allows for additional task goals such as avoiding collisions and avoiding singularity configurations of the robot.

To follow this reference joint velocity $\dot{\boldsymbol{q}}_r$, along with the reference joint acceleration $\ddot{\boldsymbol{q}}_r$ (obtained by differentiating the reference velocity), the controller computes

$$\boldsymbol{\tau} = \boldsymbol{M}\ddot{\boldsymbol{q}}_r + \boldsymbol{C}\dot{\boldsymbol{q}}_r + \boldsymbol{g} + k_d(\dot{\boldsymbol{q}}_r - \dot{\boldsymbol{q}}), \tag{7}$$

which is similar to Eq. (4) except that the velocity is not integrated to obtain a reference position (thus $k_p = 0$).

2) Simplified acceleration-based controller (OSC-A).[3] Given a target position $\boldsymbol{x}_d$, velocity $\dot{\boldsymbol{x}}_d$, and acceleration $\ddot{\boldsymbol{x}}_d$, this controller first computes the reference acceleration in task space as

$$\ddot{\boldsymbol{x}}_r = \kappa_p(\boldsymbol{x}_d - \boldsymbol{x}) + \kappa_d(\dot{\boldsymbol{x}}_d - \dot{\boldsymbol{x}}) + \ddot{\boldsymbol{x}}_d. \tag{8}$$

The reference joint acceleration is then obtained by rearranging Eq. (2):

$$\ddot{\boldsymbol{q}}_r = \boldsymbol{J}^+(\ddot{\boldsymbol{x}}_r - \dot{\boldsymbol{J}}\dot{\boldsymbol{q}}), \tag{9}$$

assuming that the actual joint velocity well approximates the reference joint velocity, $\dot{\boldsymbol{q}} \approx \dot{\boldsymbol{q}}_r$.

To follow this reference acceleration, the controller computes

$$\boldsymbol{\tau} = \boldsymbol{M}\ddot{\boldsymbol{q}}_r + \boldsymbol{C}\dot{\boldsymbol{q}} + \boldsymbol{g} - (\boldsymbol{I} - \boldsymbol{J}^+\boldsymbol{J})(k_d\dot{\boldsymbol{q}} + \alpha\boldsymbol{f}), \tag{10}$$

which is similar to Eq. (7) except that the null space is not premultiplied by $\boldsymbol{M}$, and the null space includes an extra term to dampen the torques based on the velocities.

III. Learning to control

The goal of this work is to replace operational space controllers, such as those presented in the previous section, with a learned controller. Given a large amount of training data, a neural network with sufficient capacity, and a reasonably efficient training algorithm, we will show that a controller can be learned to perform the mapping between

[2] This controller is #2 in [2], described in §3.1.2 of that paper.

[3] This controller is #5 in [2], described in §3.2.3 of that paper.

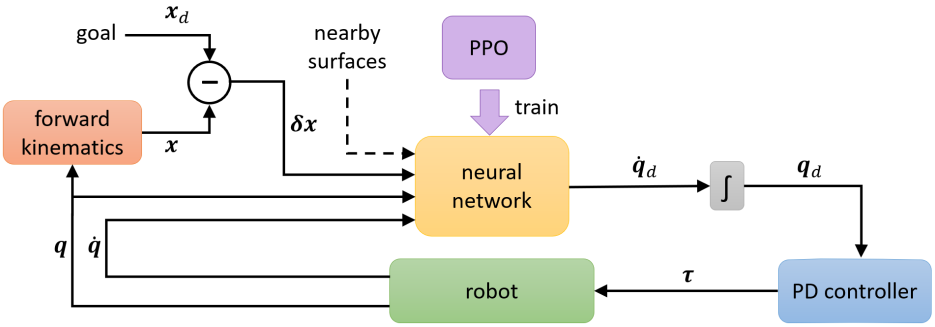


Fig. 2. Overview of the proposed JAiLeR approach. The goal is provided externally (at run time) or by curriculum learning (during training). The robot can be either simulated or real. To avoid clutter, the inputs ($\delta\boldsymbol{x}$, $\boldsymbol{q}$, and $\dot{\boldsymbol{q}}$) to PPO used to train the network are not shown. Information about nearby surfaces may optionally be fed as input to the neural network, for obstacle avoidance.

task and joint spaces, with comparable accuracy to hand-crafted controllers. Moreover, a learned controller provides several advantages over traditional approaches, namely, that redundancy, joint limits, and acceleration / deceleration profiles are automatically learned from behavior in simulation during training.

In our approach, a neural network executes a policy $\pi_\theta : \boldsymbol{s}_t \mapsto \boldsymbol{a}_t$ that maps the current state $\boldsymbol{s}_t$ to an action $\boldsymbol{a}_t$, where θ are the parameters of the policy (*i.e.,* the network weights). During training, the network learns to maximize the expected total reward, $\arg\max_\theta \mathbb{E}\left[\sum_{t=1}^{T} r(\boldsymbol{s}_t, \boldsymbol{a}_t)\right]$, where $r : \boldsymbol{s}_t, \boldsymbol{a}_t \mapsto \mathbb{R}$ is the reward function, and T is the episode length.

An overview of our system is illustrated in Fig. 2. The end effector error $\delta\boldsymbol{x} = \boldsymbol{x}_d - \boldsymbol{x}$ in task space is fed to the network, along with the robot's current joint angles $\boldsymbol{q}$ and velocities $\dot{\boldsymbol{q}}$, as well as (optionally) information about nearby surfaces. The network outputs the desired joint velocities $\dot{\boldsymbol{q}}_d$, which are then integrated to obtain the desired angles $\boldsymbol{q}_d$. These are fed to a proportional-derivative (PD) controller, which outputs the torques to control the robot.

A. State and action representation

The choice of state and action space is crucial. One possibility is to represent both states and actions in operational space, as done in [6]–[9]. In this case the policy outputs $\boldsymbol{x}_d$, $\dot{\boldsymbol{x}}_d$, or $\ddot{\boldsymbol{x}}_d$, which is then leveraged by an analytic controller such as from Eq. (7) or Eq. (10) to compute the joint torques. The problem with this approach is that the policy cannot directly account for task constraints in manifolds other than the end-effector's Cartesian task space. In particular, it cannot handle collision avoidance of links other than the end effector. This is a major limitation for deploying such reinforcement learning policies in unstructured environments.

In contrast, we propose to learn the mapping from task space to joint space, as shown in the figure. That is, the network maps $\delta\boldsymbol{x}$ to $\dot{\boldsymbol{q}}_d$, with the current joint angles and velocities providing context to the network. We choose to output desired velocities, since the integration to desired positions provides robustness to noise, and since position-based control yields better transferability from simulation to reality since it abstracts away the dynamics. The desired joint positions are fed to a proportional-derivative (PD) controller:

$$\boldsymbol{\tau} = \boxed{\boldsymbol{C}\dot{\boldsymbol{q}} + \boldsymbol{g}} + k_p(\boldsymbol{q}_r - \boldsymbol{q}) + k_d(\dot{\boldsymbol{q}}_r - \dot{\boldsymbol{q}}), \qquad (11)$$

where the robot dynamics (contained in the terms $\boldsymbol{C}\dot{\boldsymbol{q}}+\boldsymbol{g}$) are a black box inaccessible to the user. In other words, the robot internally computes the torques from the input $k_p(\cdot) + k_d(\cdot)$, where $\boldsymbol{q}_r = \boldsymbol{q}_d = \int \dot{\boldsymbol{q}}_d$, and $\dot{\boldsymbol{q}}_r = \boldsymbol{0}$.

Alternatively, we could output desired positions, accelerations, or torques. There are pros and cons to each of these approaches, and we have tried each of them with varying degrees of success. We have settled on position-based controllers because, although they yield slightly less smooth behavior than the others, they are much more easily transferred to the real robot. Moreover, they do not require direct access to the motor torques, in contrast to the controllers of Eq. (7) or Eq. (10), which can only be used on robotic platforms that provide such direct access.

To facilitate obstacle avoidance, the input can be augmented with information about the closest objects. Specifically, let $\mathcal{L}_i$ be the (possibly infinite) set of points on the surface of the i^{th} link, and let $\mathcal{P}$ be the (possibly infinite) set of points on the surface of all objects in the world. Let $\boldsymbol{p}_i \in \mathcal{P}$ and $\boldsymbol{\ell}_i \in \mathcal{L}_i$ be the (possibly non-unique) closest points in the two sets, *i.e.*,

$$\|\boldsymbol{p}_i - \boldsymbol{\ell}_i\| = \min_{\boldsymbol{p}\in\mathcal{P}} \min_{\boldsymbol{\ell}\in\mathcal{L}_i} \|\boldsymbol{p} - \boldsymbol{\ell}\|. \qquad (12)$$

The network receives as input, in addition to the quantities already mentioned, the vector for each link connecting these closest points, *i.e.*, $\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_n$, where $\boldsymbol{d}_i \equiv \boldsymbol{p}_i - \boldsymbol{\ell}_i$, and n is the number of links. In practice, we simplify the above by considering, for each link i, only the points $\mathcal{L}_i$ on a sphere attached to the link, as shown in Fig. 1.

This approach to capturing information about obstacles is advantageous because the network does not need to be retrained when the environment changes. In addition, the network is independent of the scene parameterization. Alternatively, of course, we could extend the neural network with additional layers (such as PointNet [11], [12]) to automatically process the scene rather than relying on a separate

traditional geometric computation as described in Eq. (12). We leave such exploration for future work.

To summarize, the network receives an $(m + 2n)$-dimensional input vector $\boldsymbol{s} = \left[\delta\boldsymbol{x}^\top, \boldsymbol{q}^\top, \dot{\boldsymbol{q}}^\top\right]^\top$ without obstacle avoidance, or an $(m+5n)$-dimensional input vector $\boldsymbol{s} = \left[\delta\boldsymbol{x}^\top, \boldsymbol{q}^\top, \dot{\boldsymbol{q}}^\top, \boldsymbol{d}_1^\top, \ldots, \boldsymbol{d}_n^\top\right]^\top$ with obstacle avoidance. The value m is determined by the state representation: $m = 3$ in the case of position-only, *i.e.*, $\boldsymbol{x} \in \mathbb{R}^3$; $m = 6$ or 7 in the case of position and orientation, *i.e.*, $\boldsymbol{x} \in \mathbb{SE}(3)$, depending upon whether quaternions are used; and so forth. Either way, the action is an n-dimensional output vector $\boldsymbol{a} = \dot{\boldsymbol{q}}_d$.

B. Reward

The reward used to train the network is straightforward:

$$r(\boldsymbol{s}, \boldsymbol{a}) = \exp(-\lambda_{err}\|\delta\boldsymbol{x}\|^2) - \lambda_{\mathit{eff}}\|\ddot{\boldsymbol{q}}\| - \lambda_{obs}\sum_{i=1}^{n}\psi_i, \quad (13)$$

where the first term encourages the end effector error $\delta\boldsymbol{x}$ to go to zero, the second term encourages solutions that require minimal effort, and the third term penalizes the robot for getting too close to an obstacle. The obstacle avoidance penalties are given by

$$\psi_i = \max\left(0, 1 - \|\boldsymbol{d}_i\|/d_{max}\right), \quad (14)$$

where $d_{max} = 5$ cm is the maximum distance that incurs a penalty. We set the relative weights accordingly: $\lambda_{err} = 20$, $\lambda_{\mathit{eff}} = 0.005$, and $\lambda_{obs} = 0.1$.

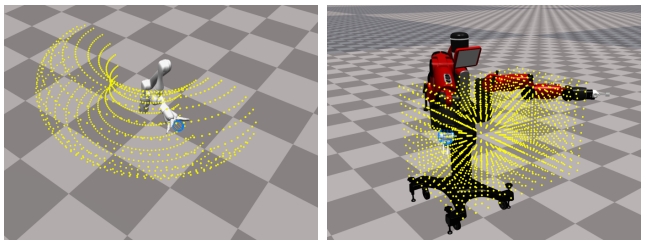
When the network is trying to reach a position in 3D space, $\delta\boldsymbol{x}$ is simply the Euclidean distance between the current position and the goal position. When the network is trying to reach a specific position and orientation, $\delta\boldsymbol{x}$ in Eq. (13) is the sum of three Euclidean distances, using three points defined along orthogonal axes from the goal, as described in [13], [14].

C. Learning procedure

We train the network using PPO [15], a recent model-free deep reinforcement learning algorithm. In our system, the network has 3 layers, each with 128 neurons with `tanh` activation for the first two layers. The network is trained using curriculum learning [16]–[23], where the entire workspace is divided into a sequence of regions $\mathcal{R}_1 \subset \mathcal{R}_2 \subset \cdots \subset \mathcal{R}_k$, where $\mathcal{R}_k$ is the workspace, and k is the number of regions in the curriculum. Starting with $i = 1$, the network is trained on region $\mathcal{R}_i$, periodically measuring the average error err_{avg}, until $err_{avg} < th$, at which point training proceeds to region $\mathcal{R}_{i+1}$, and so forth; where $th = 1$ cm is a threshold. Note that each region completely contains the previous region, to prevent catastrophic forgetting. We found curriculum learning to be crucial for getting good results.

IV. Experiments

In this section we evaluate our JAiLeR system in both simulation and in the real world.



(a) Jaco workspace (b) Baxter workspace

Fig. 3. Workspace for Jaco and Baxter reaching experiments. Due to Jaco's rotational symmetry, the policy can reach the entire 360° space (not shown), with a simple offset trick at runtime.

TABLE I

Jaco reaching experiment in simulation for Region $\mathcal{R}_4$.

	no curriculum	JAiLeR	OSC-V	OSC-A
success@1.0 cm	6.6%	96.5%	53.1%	97.9%
average error	2.9 cm	0.4 cm	0.8 cm	0.4 cm

A. Reaching policy

We used the procedure described above to train a reaching policy in simulation using a Kinova Jaco 6-DoF robot arm. Although it is straightforward to incorporate orientation constraints into the method, all experiments in this paper were conducted without regard for orientation. Using the simulation system developed by Liang et al. [24], we trained 40 robots in parallel, which greatly sped up the training time. On an NVIDIA Titan Xp machine, the policy required about two hours to train from scratch.

The workspace is defined as a torus with major radius 45 cm and minor radius 30 cm, centered at the base of the robot, as shown in Fig. 3a. For curriculum learning, the workspace is divided into 4 regions, each a superset of the previous. Each region $\mathcal{R}_i$ is a partial torus, that is, a surface of revolution in which the rotation angle is less than 360 degrees. All four regions share their major radius. The minor radii are given by 15, 20, 30, and 30 cm, respectively; and the rotation angles are given by $\pi/4$, $\pi/3$, $\pi/3$, and $\pi/2$. Only the final $\mathcal{R}_4$ is shown in the figure.

The trained policy was tested in simulation by randomly sampling start and goal positions within the $\mathcal{R}_4$ workspace (using a uniform distribution). Results are shown in Tab. I. Our JAiLeR system successfully reached within 1.0 cm of 96.5% of the goal positions, with an average of 0.4 cm average error. This is comparable to the best version of OSC, and significantly better than without curriculum learning.

It is instructive to compare these results with those of Lewis et al. [25]. Whereas their procedure uses a fixed start configuration and samples only goals, we sample start-goal pairs from within the workspace, so that our system is able to reach from any start $\boldsymbol{x}_s \in \mathcal{R}_k$ to any goal $\boldsymbol{x}_g \in \mathcal{R}_k$ within the workspace. Whereas their system terminates as soon as the end effector reaches within an ϵ-ball of the goal, we require the end effector to settle down to zero velocity before measuring error. And whereas we achieve an average

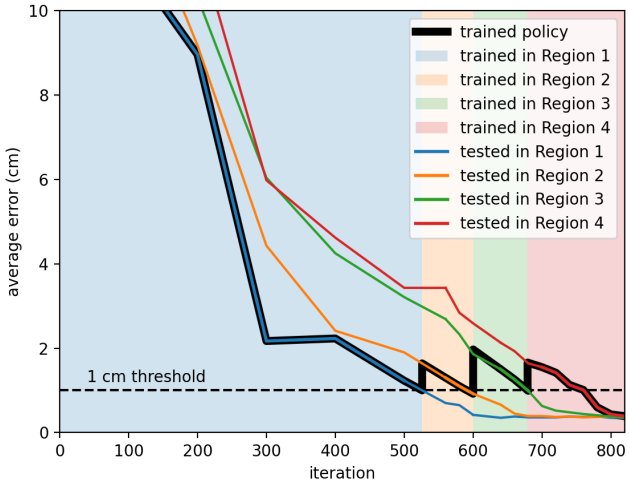


Fig. 4. Curriculum learning through four regions $\mathcal{R}_i \subset \mathcal{R}_{i+1}$, $i = 1, 2, 3$, where $\mathcal{R}_i$ is Region i. The network shows good generalization from smaller training sets to larger enclosing test sets.

error of 0.4 cm over a large workspace, their system exhibits errors greater than $\epsilon = 10$ cm for more than half of the sampled goals, for moderately-sized workspaces, even when only half the joints are used.

Curriculum learning is shown in Fig. 4. Training the network involves progressing through a sequence of four curriculum stages with increasingly larger regions. Switching to the next region occurs when the average error reaches a threshold of 1 cm. Note from the figure that the method shows surprisingly good generalization when the policy trained on a smaller region is tested on a larger enclosing region. Moreover, as the policy improves on the training region, it also improves on the larger enclosing regions.

B. Obstacle avoidance

The network was augmented with inputs regarding nearby obstacles, as described earlier. We used transfer learning to initialize the weights with values from the (non-obstacle-aware) network trained in the previous subsection. During obstacle-aware training, a virtual sphere with radius 8 cm was placed randomly within the workspace so that it remained at least 5 cm from both the start and goal. Otherwise, training proceeded exactly as before, with the same curriculum. The network learned to avoid obstacles, with a moderate impact on accuracy. When tested with $n_{test} = 2$ randomly placed spheres, the policy avoids collision 91% of the time, achieving an average endpoint error of 0.9 cm. Due to our geometric-based formulation, the policy trained on a single sphere automatically generalizes to other shapes. Fig. 5 shows the Jaco reaching for a target while avoiding obstacles. We offer these results to show the potential of the proposed approach to incorporate sensory feedback at run time.

C. Sim-to-Real transfer

To validate sim-to-real, we trained a reaching policy for the Baxter robot using the procedure described earlier. The workspace for the Baxter was defined as a $30 \times 40 \times 30$ cm

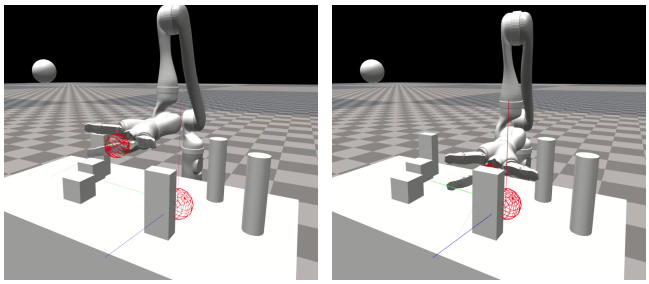


Fig. 5. By feeding the network additional inputs (vectors pointing from links to the nearest obstacles), the policy learns to avoid obstacles. This is *without* scene-specific training, because the policy is trained only on a spherical obstacle. The Jaco robot is reaching for a target (large red sphere).

region, shown in Fig. 3b. The curriculum followed a similar strategy as before. Although the Baxter is capable of position- or velocity-based control, we observed that the latter is noisy and unstable, due to noise in the encoders and other effects. As a result, we do not send velocities directly to the robot, rather relying solely upon position-based control.

The policy trained only in simulation works on the real robot. The only change we had to make was to decrease the proportional gain in the controller to reduce oscillations, and to apply an exponential filter to smooth the encoder readings. The reference position that we feed to Eq. (11) is computed from the desired velocity as $\boldsymbol{q}_r = \boldsymbol{q} + \lambda_1 \delta_t \dot{\boldsymbol{q}}_d$, where $\lambda_1 = 0.5$, and $\delta_t = 0.01$ sec is the temporal sampling interval.

To test the policy on the real robot, we sampled goal positions sequentially (within the workspace described above) so that the robot never had to reset to a start position. Rather, it simply moved sequentially through the goal positions, aiming for the next goal after the previous one was reached. Over this workspace, the real Baxter achieved an average error of 0.9 cm, with a standard deviation of 0.6 cm. Note, for comparison, that Rupert et al. [26], [27] achieved 1 to 2.5 cm steady-state error applying a traditional controller to the Baxter robot. Our JAiLeR approach thus achieves results comparable to traditional controllers.

Fig. 6 shows a plot of the step response as the robot moved from a start position to a goal position more than 30 cm away. Notice the relatively smooth behavior, and moderate amount of overshoot and oscillation. Similarly, Fig. 7 shows one example of the robot moving between two sequential goal positions.

A compelling advantage of the proposed JAiLeR approach is that learning on the robot comes essentially for free. That is, the same code that we use for training in simulation can be applied to the real robot. This fine-tuning procedure can be used to reduce the errors even further. Note that this is contrary to popular wisdom, which says that model-free reinforcement learning should not be applied to a real robot due to sample inefficiency. In our case, however, because the policy has already been trained in simulation, fine-tuning on the real robot is safe and practical. We fine-tuned the policy on the Baxter continuously for several hours by running PPO, sampling goal positions sequentially in the manner described

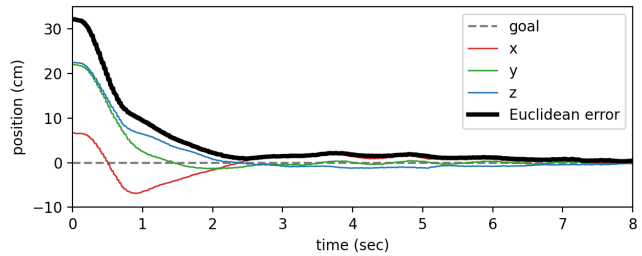
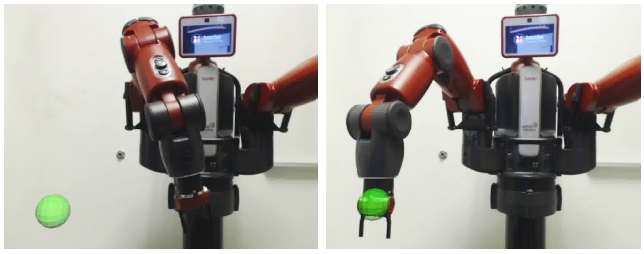


Fig. 6. Step response of the real Baxter robot reaching for a goal, showing the speed, accuracy, smoothness, relatively small overshoot, and stability of our learned policy. This policy was not finetuned in the real world and comes directly from simulation.



start position end position

Fig. 7. Real Baxter robot reaching a sequential goal position (green sphere), successfully overlapping the wrist with the goal, and achieving zero velocity.

above. We used the same training code as in simulation, except that the penalty term for acceleration was decreased by a factor of two to encourage the robot to reach the target. Fine-tuning proceeded for up to 4000 network updates (iterations). With this procedure, the error was reduced even further, yielding an average error of 0.7 cm, with a standard deviation of 0.7 cm.

In addition, an early version of our system was transferred to a real Kiova Jaco robot and used to control the robot to grab bottles on a table and pour them, using prespecified waypoints. Another policy was transferred to the Kinova Gen3 robot, achieving 0.5 cm average accuracy in a relatively large workspace.[4]

V. Relationship to Previous Work

Using a neural network to control a robot arm is not new. Several researchers explored this idea for a small number (2–3) of joints since the early 1990s. Martinetz *et al.* [28] learned visuomotor control (inverse kinematics) of a simulated 3-DoF robot arm with a pair of simulated cameras for sensing. The network achieved a positioning error of 0.3% of the workspace dimensions, which matches our results. DeMers *et al.* [29] learned inverse kinematics for a 3-DoF manipulator in 3D space using a combination of unsupervised and supervised learning; the approach only works for non-redundant manipulators. Smagt *et al.* [30] learned to control 2 joints of a 6-DoF robot arm to hover above an object using a camera-in-hand, with only tens of trials. Sanger [31] proposed "trajectory extension learning"—a curriculum learning strategy—to train a neural network to learn inverse dynamics to follow specific trajectory. The technique was applied to a 2-DoF real arm and a 3-DoF simulated arm.

In the late 2000s, Peters and Schaal [32]–[34] learned joint space control to track Cartesian targets by learning piecewise linear regions. Their approach is able to learn a specific trajectory, but is not capable of reaching arbitrary goals within the workspace. Similarly, other researchers have explored learning in the context of specific tasks [35], [36].

Recently, Martín-Martín *et al.* [8] explored different action representations for manipulation. Their study shows the difficulty of learning in joint space, where their path-following policy is able to reach within 5 cm of a fixed sequence of four Cartesian points with an accuracy of about 70%. In an approach similar to ours but for a different task, Hwangbo *et al.* [37] use deep RL in simulation to learn policies that transfer to the real world, for quadruped walking.

A closely related problem is that of learning the inverse kinematics of the robot, then using a joint controller to reach the inverse mapped joint configuration. Several researchers have explored this idea [38], [39]. The work of D'Souza *et al.* [40] is most closely related to ours, in that they approximate the nonlinear mapping with local piecewise linear models, and they learn incrementally. In our case, the deep neural network automatically learns such local models, and curriculum learning is used to learn incrementally.

Despite the promise of deep RL, most recent approaches in this area are limited to operating in end-effector space to execute learned manipulation tasks [9], [41]–[44]. Lower-level control of the robot joints is then achieved by traditional operational space controllers. This choice of state space prevents such approaches from performing tasks that involve avoiding arm contact, such as reaching in clutter.

As discussed earlier, Lewis *et al.* [25] study a problem similar to ours. They use a deep RL algorithm (DDPG) to learn to reach any goal position within a workspace. Their work differs in several respects, namely, that their start position is fixed, and success is declared as soon as the end effector reaches within a radius of the goal, without requiring it to stop at the goal, similar to standard RL benchmarks [45]. Even so, their average error is about an order of magnitude worse than ours (approximately 10 cm versus 1 cm).

More recently, Aumjaud *et al.* [46] study the ability of deep RL algorithms (including PPO) to solve the reaching problem. With a fixed start and goal, they find that most algorithms are able to successfully achieve an error less than 5 cm in simulation, and about half the algorithms are successful in the real world. Performance, however, drops precipitously when the goal position is randomized (even when the start remains fixed), with most algorithms (including PPO) achieving less than 10% success at a threshold of 5 cm (the best algorithm achieves 75% success). Additionally, all algorithms struggle to achieve any measurable success either in simulation or the real world at a threshold of 1 cm. In contrast with these results, we show that deep RL is able,

[4]The workspace was 120° of an annular cylinder with height 0.5 m and inner/outer radii of 0.45 and 0.85 m, respectively.

via curriculum learning, to learn a general-purpose reaching controller (from a random start to a random goal) with high accuracy (less than 1 cm error) within a large workspace, both in simulation and in reality.

VI. Conclusion

We presented JAiLeR, a deep RL-based approach to controlling a robot manipulator. Training in simulation, we showed that this simple approach is able to achieve reaching accuracy comparable to that of classical techniques over a large workspace, in both simulation and reality. Advantages of the approach include automatic handling of redundancy, joint limits, and acceleration / deceleration profiles. Moreover, fine-tuning directly on the real robot is feasible, thanks to transfer learning. Orientation constraints and obstacle avoidance can be incorporated by augmenting the input to the network, although more research is needed to validate these ideas. We hope these encouraging results inspire further work in this area.

References

[1] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation," *IEEE Journal on Robotics and Automation*, vol. 3, no. 1, pp. 43–53, 1987.

[2] J. Nakanishi, R. Cory, M. Mistry, J. Peters, and S. Schaal, "Operational space control: A theoretical and empirical comparison," *The International Journal of Robotics Research*, vol. 27, no. 6, pp. 737–757, 2008.

[3] N. D. Ratliff, J. Issac, D. Kappler, S. Birchfield, and D. Fox, "Riemannian motion policies," in *arXiv:1801.02854*, 2018.

[4] C.-A. Cheng, M. Mukadam, J. Issac, S. Birchfield, D. Fox, B. Boots, and N. Ratliff, "RMPflow: A computational graph for automatic motion policy generation," in *International Workshop on the Algorithmic Foundations of Robotics*, 2018, pp. 441–457.

[5] D. Kappler, F. Meier, J. Issac, J. Mainprice, C. G. Cifuentes, M. Wüthrich, V. Berenz, S. Schaal, N. Ratliff, and J. Bohg, "Real-time perception meets reactive motion generation," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1864–1871, 2018.

[6] L. Shao, T. Migimatsu, Q. Zhang, K. Yang, and J. Bohg, "Concept2Robot: Learning Manipulation Concepts from Instructions and Human Demonstrations," in *Proceedings of Robotics: Science and Systems (RSS)*, 2020.

[7] T. Migimatsu and J. Bohg, "Object-centric task and motion planning in dynamic environments," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 844–851, 2020.

[8] R. Martín-Martín, M. A. Lee, R. Gardner, S. Savarese, J. Bohg, and A. Garg, "Variable impedance control in end-effector space: An action space for reinforcement learning in contact-rich tasks," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 1010–1017.

[9] M. A. Lee, Y. Zhu, K. Srinivasan, P. Shah, S. Savarese, L. Fei-Fei, A. Garg, and J. Bohg, "Making sense of vision and touch: Self-supervised learning of multimodal representations for contact-rich tasks," in *ICRA*, 2019.

[10] C. C. de Wit, B. Siciliano, and G. Bastin, *Theory of robot control*. Springer Science & Business Media, 2012.

[11] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," in *CVPR*, 2017.

[12] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "PointNet++: Deep hierarchical feature learning on point sets in a metric space," in *NIPS*, 2017.

[13] A. Molchanov, T. Chen, W. Honig, J. A. Preiss, N. Ayanian, and G. S. Sukhatme, "Sim-to-(multi)-real: Transfer of low-level robust control policies to multiple quadrotors," in *IROS*, 2019.

[14] Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li, "On the continuity of rotation representations in neural networks," in *CVPR*, 2019.

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

[16] P. Oudeyer, F. Kaplan, and V. V. Hafner, "Intrinsic motivation systems for autonomous mental development," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 2, pp. 265–286, 2007.

[17] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *ICML*, 2009.

[18] K. A. Krueger and P. Dayan, "Flexible shaping: How learning in small steps helps," *Cognition*, vol. 110, no. 3, pp. 380–394, 2009.

[19] M. P. Kumar, B. Packer, and D. Koller, "Self-paced learning for latent variable models," in *NIPS*, 2010.

[20] Y. J. Lee and K. Grauman, "Learning the easy things first: Self-paced visual category discovery," in *CVPR*, 2011.

[21] C. Florensa, D. Held, M. Wulfmeier, M. Zhang, and P. Abbeel, "Reverse curriculum generation for reinforcement learning," in *CoRL*, 2017.

[22] G. Hacohen and D. Weinshall, "On the power of curriculum learning in training deep networks," in *ICML*, 2019.

[23] T. Matiisen, A. Oliver, T. Cohen, and J. Schulman, "Teacher–student curriculum learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 9, pp. 3732–3740, 2020.

[24] J. Liang, V. Makoviychuk, A. Handa, N. Chentanez, M. Macklin, and D. Fox, "GPU-accelerated robotics simulation for distributed reinforcement learning," in *CoRL*, 2018.

[25] W. C. Lewis II, M. Moll, and L. E. Kavraki, "How much do unstated problem constraints limit deep robotic reinforcement learning?" in *Rice University Technical Report TR19-01*, 2019.

[26] L. Rupert, P. Hyatt, and M. D. Killpack, "Comparing model predictive control and input shaping for improved response of low-impedance robots," in *Humanoids*, 2015.

[27] J. S. Terry, L. Rupert, and M. D. Killpack, "Comparison of linearized dynamic robot manipulator models for model predictive control," in *Humanoids*, 2017.

[28] T. M. Martinetz, H. J. Ritter, and K. J. Schulten, "Three-dimensional neural net for learning visuomotor coordination of a robot arm," *IEEE Transactions on Neural Networks*, vol. 1, no. 1, pp. 131–136, 1990.

[29] D. DeMers and K. Kreutz-Delgado, "Learning global direct inverse kinematics," in *NIPS*, 1992.

[30] P. P. van der Smagt and B. J. A. Kröse, "A real-time learning neural robot controller," in *ICANN*, 1991.

[31] T. D. Sanger, "Neural network learning control of robot manipulators using gradually increasing task difficulty," *IEEE Transactions on Robotics and Automation*, vol. 10, no. 3, pp. 323–333, 1994.

[32] J. Peters and S. Schaal, "Learning operational space control," in *RSS*, 2006.

[33] ——, "Reinforcement learning for operational space control," in *ICRA*, 2007.

[34] ——, "Learning to control in operational space," *International Journal of Robotics Research (IJRR)*, vol. 27, no. 2, 2008.

[35] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research (JMLR)*, 2016.

[36] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," in *ICRA*, 2019.

[37] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, Jan. 2019.

[38] B. Bócsi, D. Nguyen-Tuong, L. Csató, B. Schölkopf, and J. Peters, "Learning inverse kinematics with structured prediction," in *IROS*, 2011.

[39] D. Kubus, R. Rayyes, and J. Steil, "Learning forward and inverse kinematics maps efficiently," in *IROS*, 2018.

[40] A. D'Souza, S. Vijayakumar, and S. Schaal, "Learning inverse kinematics," in *IROS*, 2001.

[41] F. Sadeghi, A. Toshev, E. Jang, and S. Levine, "Sim2Real view invariant visual servoing by recurrent control," in *CVPR*, 2018.

[42] L. Shao, T. Migimatsu, and J. Bohg, "Learning to scaffold the development of robotic manipulation skills," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 5671–5677.

[43] M. A. Lee, C. Florensa, J. Tremblay, N. Ratliff, A. Garg, F. Ramos, and D. Fox, "Guided uncertainty-aware policy optimization: Combining learning and model-based strategies for sample-efficient policy learning," *arXiv preprint arXiv:2005.10872*, 2020.

[44] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine,

"QT-Opt: Scalable deep reinforcement learning for vision-based robotic manipulation," in *CoRL*, 2018.

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

[46] P. Aumjaud, D. McAuliffe, F. J. R. Lera, and P. Cardif, "Reinforcement learning experiments and benchmark for solving robotic reaching tasks," in *arXiv:2011.05782*, 2020.