Robotic Embodiment of Human-Like Motor Skills via Sim-to-Real Reinforcement Learning

A THESIS SUBMITTED TO THE FACULTY
OF THE UNIVERSITY OF MINNESOTA BY

LUIS GUZMAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF SCIENCE

Advisor: Nikolaos Papanikolopoulos

December 2021

Acknowledgment

I would like to thank my advisor, Nikolaos Papanikolopoulos, for his unwavering support and guidance. I am also grateful to my committee members, Nikolaos Papanikolopoulos, Vassilios Morellas and Hyun Soo Park, whose expertise and teaching was invaluable in this project and my future career. Finally, I would like to thank all the members of the Center for Distributed Robotics Laboratory for their help. This material is based upon work partially supported by the Minnesota Robotics Institute (MnRI), Honeywell, and the National Science Foundation through grants CNS-1439728, CNS-1531330, and CNS-1939033. USDA/NIFA has also supported this work through the grant 2020-67021-30755.

Abstract

State of the art methods continue to face difficulties automating many tasks, particularly those which require human-like dexterity. The proposed "Internet of Skills" enables robots to learn advanced skills from a small set of expert demonstrations, bridging the gap between human and robot abilities. In this work, I train Reinforcement Learning (RL) control policies for the tasks of hand following and block pushing. I build a sim-to-real pipeline and demonstrate these policies on a Kinova Gen3 robot. Lastly, I test a prototype system that allows an expert to control the Kinova robot using only their arm movements, captured using a Vicon motion tracking system. My results show that performance of state of the art RL methods could be improved through the use of demonstrations, and I build a shared representation of human and robot action that will enable robots to learn new skills from observing expert actions.

Contents

List of Tables

List of Figures

1 Introduction

Despite recent advances in machine learning, there is still an entire class of tasks that are too complex to automate with robots, but cannot be completed entirely by humans. State-of-the-art methods, for example, cannot always address the planning and robot control problems for unknown, cluttered, and uncertain environments, and working in hazardous environments may require a robot to be operated remotely by a human. These teleoperation schemes usually lack the dexterity of a human worker and the precision and repeatability of full robotic automation, thus combining the least desirable elements of humans and robots. This thesis is part of a proposed system, entitled the "Internet of Skills" (IoS), which promotes and enhances the notion of a *Tactile Internet* to enable humans and machines in dispersed geographical regions to operate as a unified entity, intelligently balancing both human and robot (i.e., bilateral) strengths and weaknesses.

Teaching robots new skills continues to be a challenging task. When presented with a new scenario, a robot must know *what* the goal is and also *how* to achieve that goal. Methods like Reinforcement Learning (RL) have shown success in solving only one of these problems. In RL, a robot agent is given a clear goal, and it must learn the optimal way of completing that goal. This works well for simple tasks that can easily be quantified but is less effective when goals are abstract or require significant future planning.

To address some of these issues, we can utilize human demonstrations to teach robots new skills. By directly observing the actions of the expert, the robot can learn the goal of a demonstrated skill. Additionally, when the robot is unsure how to act, it can refer to a cache of expert experience and choose to act similarly to how the expert would in that situation. Using expert demonstrations has been shown to improve convergence rates and performance, especially in high-dimensional tasks such as object manipulation [1]. Furthermore, using reinforcement learning algorithms simultaneously with expert demonstrations can produce a policy that exceeds the performance of "imperfect" expert demonstrations [2–5]. Possible benefits include higher precision than humanly-attainable and the filtering of hand-shaking when performing delicate manipulation tasks. Including expert demonstrations in the training process can significantly improve the time it takes the robot to learn new skills and teach robots to complete tasks that were previously too complex to be done autonomously.

In this thesis project, I create virtual environments and train Reinforcement Learning algorithms for the task of following the expert's hand location while avoiding obstacles in the workspace, a crucial prerequisite that will enable the IoS framework. I expand on this by mapping the expert's full arm movements to the robot, and demonstrate this prototype system on a physical Kinova Gen3 7DOF robot. Finally, I evaluate the performance of both the expert and various RL algorithms on a block pushing task, highlighting the performance gap that can be closed through the use of expert demonstrations. This work builds all the necessary data pipelines and provides initial research that will enable further work in utilizing demonstrations to train robots to complete new skills.

2 Related Work

Learning from Demonstrations: Forming the connection between human and robot actions requires a shared language where human-given commands can understood by any robot; however, human action

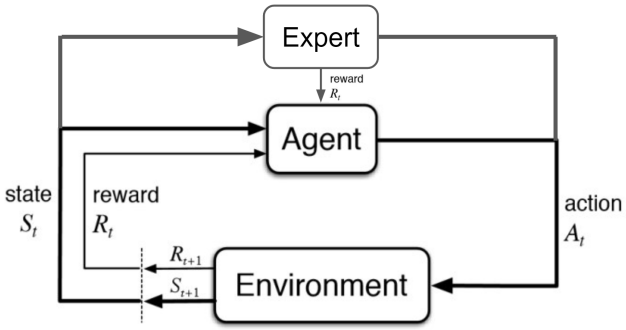


Figure 1: Learning from Demonstrations

is difficult to represent in an analytic form. Learning from demonstrations (LfD) is a technique for determining a robot agent's control policy from human (expert) demonstrations. It shares similarity with reinforcement learning, although in RL, a policy is learned through interaction and exploration in a training environment rather than from expert demonstrations. In both RL and LfD, the robot's surroundings are represented by a state S and the robot performs actions A. The same state and action space can be used to quantify the experience of a human demonstrator. A policy π maps the robot's observed surroundings to an action $\pi : S \rightarrow A$. The goal is to learn a policy that either optimizes a certain reward function (RL) or one that performs similar actions to an expert demonstrator (LfD).

Vecerik et al. show how demonstrations can be utilized to improve performance on various manipulation tasks like placing a peg inside of a hole [6]. Their method modifies the reinforcement learning algorithm DDPG by placing expert demonstrations inside of the replay buffer during training. As the agent is sampling trajectories to train the policy, some of the experience it receives is from the agent and some is from the expert. In this way, the expert can show the agent what "good" actions are, and the agent will converge behavior similar to the expert. They found that training from demonstrations yields higher performance and converges faster than state of the art reinforcement learning methods on the same task.

The collection of expert experience has often been the limiting factor for developing practical applications of this research. Because the expert must share the same state and action space as the robot, expert demonstrations are usually collected either through teleoperation or by kinesthetically manipulating the joints of the robot, a process which can become tedious when multiple demonstrations are required to teach a robot a single skill [6, 7]. Additionally, teleoperation and kinesthetic teaching do not realistically represent how a human may perform the skill, and these interfaces fail to capture the fluidity and nuance of certain gestures. In order for robots to learn human skills naturally, a more effective teaching interface must be developed.

The Correspondence Problem: The goal of the IoS framework is to teach robots new skills using only the movements of the expert's body. This mode of teaching suffers from what is known as the correspondence problem—how can human actions be mapped to a representation that the agent can understand? In reinforcement learning, this usually equates to mapping from human to robot joint angles. Mohammad and Nishida solve the correspondence problem analytically for humanoid robots [8]. This is generally possible to do since the actuators of the robot line up closely with

the human body, and the mapping must only account for small differences in joint segment length and actuator placement. Learning this mapping on non-humanoid robots is more complicated and usually requires some form of neural network. For example, [9] uses recurrent neural networks (RNNs) with Long-Short Term Memory (LSTM) to map from human poses to robot joint angles. Lastly, some approaches try to avoid the correspondence problem altogether. Jackson et al. used virtual reality in order to avoid the need for a learned correspondence mapping [10], but only the expert's hand is tracked, missing information about the orientation of the rest of the expert's arm when completing the skill.

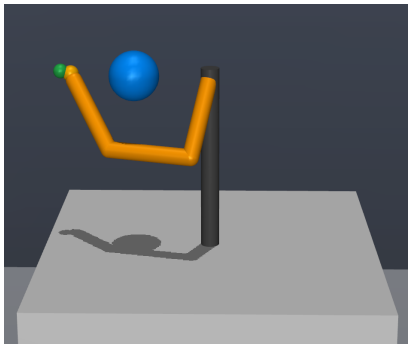
Reinforcement Learning: For the hand tracking task, I draw inspiration from Sangiovanni et al. [11], who uses reinforcement learning to solve inverse kinematics and obstacle avoidance. In this task, the robot is given a goal location, and it must learn how to position its arm to reach that location while avoiding an obstacle. Particularly, my choice of reward function is inspired by this work. They use a method known as normalized advantage functions (NAF), which is similar to deep Q-learning for continuous tasks. I utilize more recent RL algorithms that have been shown to have better convergence rates and performance than NAF. Furthermore, I propose significant modifications to the method, which improves agent performance and stability, and demonstrate the task on a physical Kinova robot.

Proximal Policy Optimization (PPO) [12] is an example of an on-policy reinforcement learning algorithm. It is characterized by a policy update equation that includes a clipping parameter, which ensures that no single policy update is too large and destroys the policy. Previous work has shown success when applying PPO to robot control tasks, such as object manipulation [13] and control of a humanoid robot [14]. PPO is robust to hyperparameters, so it requires very little tuning in order to get optimal performance.

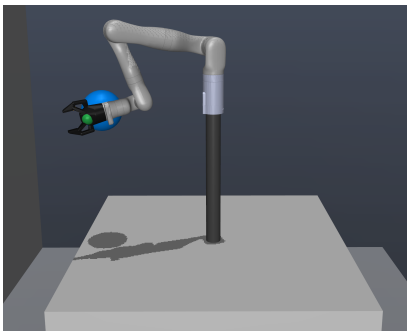
Twin delayed deep deterministic policy gradients (TD3) [15] is a modified version of the deep deterministic policy gradients (DDPG) [16] algorithm. It uses two critic networks in order to minimize the Q-function approximation error, and they delay the policy updates to every-other update of the critic network to further decrease the error. It features a higher sample efficiency than PPO since it is an off-policy algorithm and thus stores its experience in a replay buffer so that preferable trajectories can be reused for training. In general, off-policy algorithms are less stable than on-policy ones, so more hyperparameter tuning may be required.

Actor-critic using Kronecker-factored trust region (ACKTR) [17] and advantage actor-critic (A2C) [18] are two other examples of on-policy learning algorithms, and soft actor-critic (SAC) [19] and deep deterministic policy gradients (DDPG) [16] are additional off-policy algorithms that will appear later in this paper.

Experience Copy: When training reinforcement learning algorithms on large state or action spaces, a common technique is to use a method known as transfer learning or experience copy [20]. In this technique, a policy is trained on an easier subtask that may only require exploring a smaller subspace of the state space. Once the agent has mastered the easier task, it is trained on incrementally more challenging tasks. Sangiovanni et al. found this method to improve performance on the obstacle avoidance task, where they first trained the agent on stationary obstacles before training on a dynamic environment with moving obstacles [11].



(a) 5 DOF robot



(b) 7 DOF Kinova Gen3 Robot

Figure 2: The hand following virtual environments. The target point is shown in green, and the obstacle is in blue.

3 Method

3.1 Virtual Environments

All of the algorithms mentioned in this paper are trained in simulation before being demonstrated on a physical robot. OpenAI Gym [21] is an open-source toolkit for training RL algorithms within virtual environments. I built the virtual environments within this framework in order to provide a familiar interface that can be reused when testing multiple RL and LfD algorithms. OpenAI gym also supports the MuJoCo physics engine, which I will use for its superior simulation accuracy, including electro-mechanical responses of the robot actuators. MuJoCo is capable of using the Unified Robot Description Format (URDF) so that a multitude of robot configurations can easily be loaded into the simulation.

For initial testing, I created two simulated robots to be used in the Gym environments. The first robot is a 5 degree of freedom (DOF) manipulator, shown in Figure 2a. This robot is a modified version of the OpenAI Reacher environment and is a simplified model of a human arm, with a 3 DOF ball joint at the shoulder, and two single DOF hinge joints at the elbow and wrist. The second robot is a 7 DOF Kinova Gen3 Robot, shown in Figure 2b. Kinova supplies URDF models for their robots, and these have been converted to the MuJoCo XML format by the community [22].

3.2 Task 1: Hand following

The first task I examine is following the hand location of the expert, while avoiding obstacles in the workspace. Each environment (Figure 2) features an obstacle (in blue), and a target point (shown in green). The robot must learn to reach the target point, while avoiding the obstacle. This is a non-trivial task, since every goal location has infinitely many possible solutions for the joint angles of the robot.

3.2.1 Reward Function

For this application, I build on the reward function presented in Sangiovanni et al. [11].

$$r = c_1 R_T + c_2 R_A + c_3 R_O + c_4 R_X.$$

Large distances from the end effector to the target point are penalized with the R_T term. [11] proposes a Huber Loss function for this purpose.

$$R_T = L_\delta(d) = -\begin{cases} \frac{1}{2}d^2 & \text{for } |d| < \delta \\ \delta(|d| - \frac{1}{2}\delta) & \text{otherwise} \end{cases}$$

where d is the Euclidean distance between the target point and the end effector and δ is the Huber Loss parameter, which determines the regions of linear and quadratic loss.

R_A penalizes large actions (a) to reduce overshoot and encourage the manipulator to remain stationary after it has reached the target point.

$$R_A = -||a||^2.$$

During training, I disable collisions in the MuJoCo environment so that the manipulator can pass freely through the obstacles. This is to ensure continuity of the action space —with collisions enabled, a large action could result in zero movement, which would cause an incorrect policy update. If the manipulator passes through an obstacle, it receives a large negative reward in the form of R_O.

$$R_O = -\left(\frac{d_{\text{ref}}}{d_O + d_{\text{ref}}}\right)^p$$

where d_O is the minimum distance from the manipulator to the center of obstacle, and p sets the exponential decay rate. d_{ref} is a constant that determines how close the manipulator can get to the obstacle without incurring significant penalty. This value is set to be approximately the size of the obstacle or slightly larger.

Because collisions are disabled in MuJoCo, I also penalize behavior which would be impossible on a physical robot. R_X discourages behavior where the manipulator passes through itself to reach the target point by penalizing large joint angles.

$$R_X = \sum_{n=0}^{N} \min\left(0, \frac{\pi}{2} - \theta_n\right)$$

where N is the total number of hinge joints and θ_n is the angle of each joint.

3.2.2 Observation & Action Spaces

The observation (state) space defines what information the robot agent can access to determine its policy. At each timestep, the robot agent receives:

$$s_t = \{\ q,\ \dot{q},\ q_T,\ q_O,\ d_T,\ d_O\ \}$$

where q and $\dot{q}$ represent the robot's joint angles and velocities, q_T is the location of the target, and q_O is the location of the obstacle. Joint angles are represented as cosine/sine pairs (i.e. $q_i \rightarrow (\cos(q_i), \sin(q_i))$ in order to avoid discontinuities and bound them to $[-1, 1]$, which stabilizes training

and improves performance. For ball joints that use quaternions, raw quaternion values are used. d_T and d_O are the distances from the robot to the target and obstacle, respectively. These last two values were added to improve convergence times, as these are the actual values that the agent must optimize. Additional discussion and data that supports this choice of observation space are provided in the results.

In the case of a ball-type joint, the observation space gains an additional dimension, since ball joints are represented as 4-dimensional quaternions. This has the disadvantage of adding additional dimensionality that can slow down training, but it crucially avoids the problem of gimbal lock, which can cause exploding gradients during training.

The action space is comparatively simple, and is equal to the number of degrees of freedom of the robot. The network is trained to output motor control commands, so all error correction must be learned by the policy network.

Optimizing over this state space is a challenging task. For the 5 DOF robot with a ball-type shoulder joint, the observation space includes 20 continuous dimensions (6 joint angles, 6 joint velocities, 3 target coordinates, 3 obstacle coordinates, and 2 distances), and the action space includes 5 continuous dimensions. Since attempting to explore the entire state space would be impossible, state-of-the-art reinforcement learning algorithms and techniques to simplify this exploration problem are required.

3.2.3 Experience Copy

In order to limit the amount of exploration needed, I segment the training process into easier sub-tasks. In the first task, I ask the agent to reach stationary goal points that are near the obstacle. For high DOF robots, I can further limit this first sub-task to the $z = 0$ plane and include the z dimension later. By placing this constraint, I decrease the size of the observation space to explore, so training is quicker and more likely to converge.

Next, using *experience copy*, I load in the policy I trained on the first sub-task, and begin the training process on the next sub-task. The second sub-task is to trace out a variety of arc patterns. Ideally, all points that make-up the arc trajectory have been seen by the agent during the training of the first sub-task. Training on the second task makes minor improvements to the agent's ability to track moving objects.

3.3 Task 2: Block pushing

The second environment I tested tasks the robot with pushing a block from a randomly initialized starting location to a goal location. Figure 3 shows the block pushing environment. The block is shown in green and the goal location is the transparent red cube. The robot must first learn to contact the green block, then learn to slide it across the table to the goal location without overshooting. The robot is initialized at approximately shoulder-height in order to be consistent with the expert's demonstrations. I train various RL algorithms on this environment and compare the performance to the expert completing the same task.

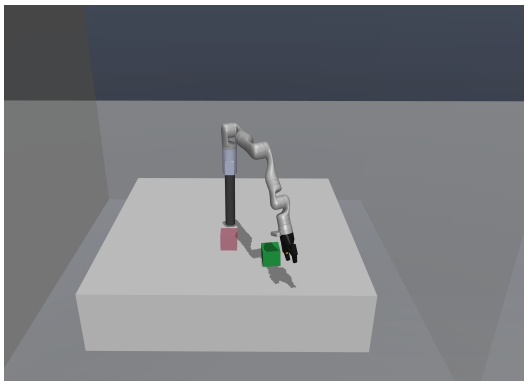


Figure 3: The block pushing environment

3.3.1 Reward Function

The block pushing task can be expressed with either sparse or dense reward functions. The sparse representation would give the agent zero reward until the block is within a specified radius of the goal, then the agent would receive a large positive reward. In the dense representation, the agent receives a negative reward at every time step that is proportional to the distance from the block to the goal location. The sparse representation is more challenging, but it is a more realistic representation of the task since in physical systems, it is not always possible to know the distances between objects without complicated vision systems or sensors. I chose the dense representation to increase the likelihood of convergence, but I also included sparse rewards to give preference to trajectories that sufficiently complete the goal.

The reward function for the block pushing task is

$$r = c_1 R_G + c_2 R_T$$

R_G quantifies the distance of the block from the goal location and provides a positive reward when it's within the goal radius.

$$R_G = \begin{cases} +\ 1/c_1 & \text{for } |d| < \delta \\ -\ \delta(|d| - \frac{1}{2}\delta) & \text{otherwise} \end{cases}$$

Using only R_G is not sufficiently dense since the agent must position its arm to contact the block before it can generate any improved rewards. In order to avoid the sparse reward problem, I also included the target loss R_T from the hand following task with the following modification: in the context of block pushing, d represents the distance from the agent's end effector to the block so that the agent is encouraged to just reach the block before it learns to push it.

3.3.2 Observation & Action Spaces

The observation space for the block pushing task is

$$s_t = \{\ q,\ \dot{q},\ q_B,\ q_G - q_B\ \}$$

where q and $\dot{q}$ represent the robot's joint angles and velocities, q_B is the location of the block, and q_G is the location of the goal. Similarly to hand tracking, $q_G - q_B$ is used instead of just q_G because that is the feature the agent must minimize, so including this explicitly prevents the policy network from having to learn this relationship. The same sinusoidal transformation is applied to revolute joint angles to bound them to $[-1, 1]$.

The action space is the same as the hand tracking environment and is equal to the number of degrees of freedom of the robot.

3.4 Training

All policy networks were trained using a machine running Ubuntu 20.04 with an Intel(R) Core(TM) i7-4770K CPU @ 3.90GHz, 16 GiB of memory, and a Nvidia GeForce RTX 3070 GPU. Initially, policy networks were trained to 10 million time steps, but I noticed very little improvement after 5 million steps. All figures and results are reported at 5 million time steps. In order to minimize the effect of stochastic policy training, networks were trained three times using different seed values. The reported results are the average of the three training instances.

For the implementations of PPO, ACKTR, and A2C, I modified the implementation of [23]. For TD3 and DDPG, I used [15]. For SAC, I used [24]. All networks are implemented using PyTorch.

3.5 Full Arm Following Prototype

Lastly, I built a prototype system that allows the expert to fully control a Kinova Gen3 robot using motion tracking data of their arm. This prototype extends the hand-following task, since the robot follows all joint angles of the expert rather than just the hand location. Expert trajectories are recorded using a Vicon motion tracking system. Reflective trackers are placed on the expert's right arm and torso, according to the layout of the plug-in gait upper limb model [25]. This tracker placement is standard for human motion capture applications, and an example image is shown in Figure 4. The expert's joint angles are calculated from the Vicon tracker positions, then these angles are directly mapped to the actuators on the Kinova robot. This direct mapping is only possible

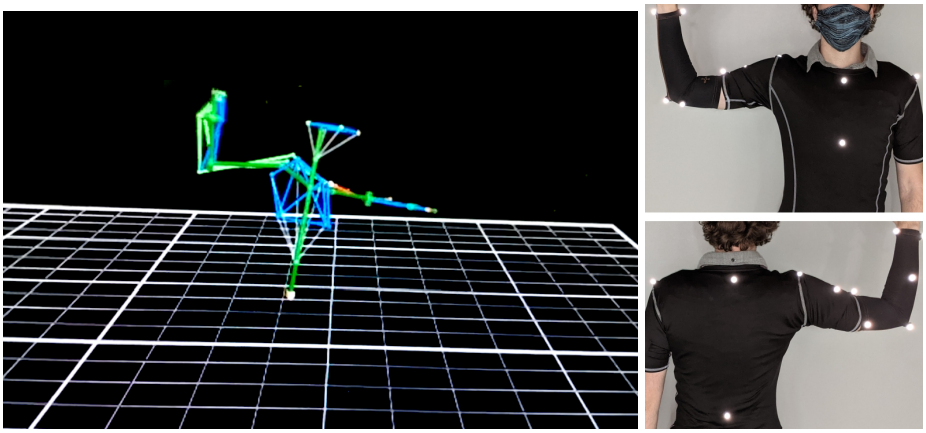


Figure 4: View of the motion tracking data and tracker placement. The expert's entire right arm and torso are tracked

because the Kinova robot has the same number of degrees of freedom as the human arm and the physical dimensions of the Kinova arm closely match the expert's[1]. If either of these requirements are not met, a more complicated mapping must be learned through recurrent neural networks (RNNs), for example. In order to match the 5DOF robot used in initial experiments, only five of the seven degrees of freedom are mapped, representing every possible movement other than wrist rotation and abduction. The full calculation which maps expert joint angles to the Vicon robot is given in the Appendix.

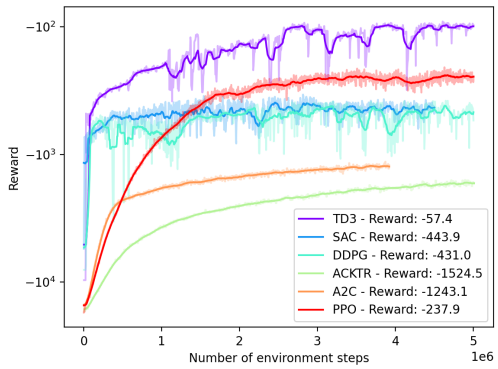
I implemented a sim-to-real pipeline for the Kinova Gen3 robot so that both simulated and physical robots could be controlled using this system. The Kinova robot requires 1 kHz control feedback in order to prevent jerky motion, so low-level control must be handled by the embedded device on-board the Kinova robot [26]. My control interface sends the robot joint velocities, which are proportional to the error between the desired and current joint angles. The joint velocity commands and joint feedback are communicated using Kinova's Robot Operating System (ROS) interface.

3.6 Experimental Setup

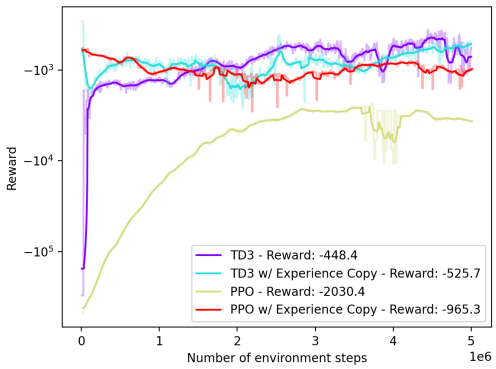
In order to evaluate expert performance on the block pushing task, I gathered a dataset of 30 expert demonstrations. The target block is also tracked using the Vicon system, so its position can be used to calculate the expert's reward at every time step. Expert performance is averaged over the 30 trials, using a variety of block starting locations.

After comparing performance on the two simulated environments, I offer two physical trials that demonstrate the sim-to-real pipeline. Both the hand following policy and the arm following prototype are compared. I demonstrate the two control methodologies on the block pushing task and on a pick-and-place task, which shows the feasibility and benefits of using such a system for teleoperated control. The coordinate systems of the Vicon system, mujoco simulation and physical demonstration environment are all aligned so that movements can be accurately quantified across all three environments.

[1]The expert's arm lengths are 0.35m shoulder to elbow and 0.30m elbow to wrist. The Kinova robot measures 0.42m and 0.31m for the same joint segments, respectively. The Kinova robot's last link is 0.32m from wrist to end effector, which is much longer than the experts wrist to finger length. This mismatch was avoided by using a custom fabricated push stick, which effectively extends the expert's last joint segment length to match the Kinova robot.



(a) Comparison of RL algorithms on the stationary goal environment



(b) Comparison of PPO and TD3 experience copy on the trajectory following environment

Figure 5: Results on the hand following environments.

4 Results

4.1 Task 1: Hand following

The results of six different RL algorithms on the stationary goal environment are shown in Figure 5a. TD3 and PPO were the top performing algorithms, with SAC and DDPG tied for third. The off-policy algorithms (TD3, SAC, and DDPG) converged very quickly, then made incremental improvements for the remainder of the training time. In contrast, the on-policy algorithms took much longer to converge but they experienced a more stable training process (i.e., the reward value nearly monotonically increases). Since these values are averaged over three trials, TD3 and DDPG appear to be more stable than their single trials, where I often observed reward fluctuations up to an order of magnitude in size.

In Figure 5b I compare the performance of PPO and TD3 when using experience copy. The copied policies are compared against policies that are trained directly on the trajectory following environment. In this test, the sample efficiency of TD3 is particularly noticeable. Any advantage that is gained by using experience copy is quickly negated by "vanilla" TD3's fast convergence to near-optimal values. PPO, on the other hand, showed a 26% performance gain when using experience copy. The additional reward was enough to achieve comparable performance to TD3 on the trajectory-following task.

The PPO implementation is parallelized over 8 environment instances, so PPO took 45 minutes to train, whereas TD3 was single-threaded and took over 30 hours to train for the same number of time steps. However, one advantage of TD3 is its sample efficiency, and as shown in Figure 5a, it reaches its maximum reward at around two million time steps. The corresponding training training time is 7.5 hours, which is still much longer than it takes for PPO to converge.

Additionally, when viewing the control policies in simulation, I noticed that TD3 produced irregular trajectories, whereas PPO tended to follow smooth arcs. This is quantified in Figure 6, where I compare the distances to the target of the two policies. Although TD3 accrues a better

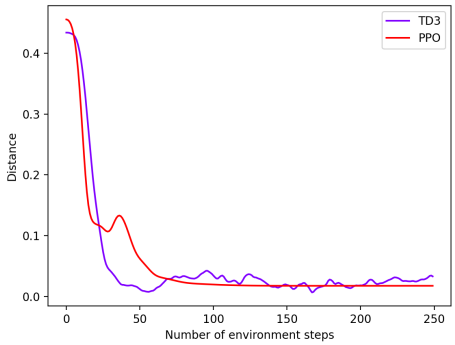
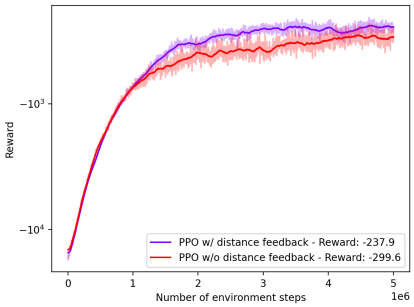
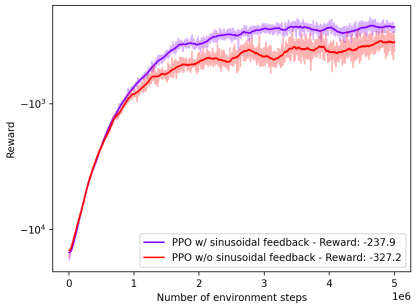


Figure 6: Distances between the robot fingertip and the target for both the PPO and TD3 policy networks. PPO is found to be much more stable, despite having a lower average reward than TD3.



(a) Reward comparison of the additional distance information



(b) Reward comparison of the sinusoidal joint angle representation

Figure 7: Comparison showing performance increase of two of the proposed modifications to the observation space.

reward, PPO shows preferable behavior by holding the fingertip stationary at the target point. This gives major preference to choosing PPO for manipulator control tasks due to its comparably safe and predicable trajectories.

In Figure 7, I provide some experiments that demonstrate the performance gains of the method described in Section 3.2.2. Figure 7a considers the inclusion of the additional distance information (d_T and d_O) in the observation space. Including this additional information showed a 20.6% increase in the asymptotic reward. Similarly, Figure 7b shows that applying the sinusoidal transformation to the raw joint angles improved performance by 27.3%. These improvements to the method in [11] can be applied to future LfD tasks to improve stability and performance.

A analysis of the evironment hyperparameters is given in the Appendix in Figure 11. Since changing these values affects the calculated reward of a given policy, I chose to evaluate them on average distances to the obstacle and the target. A small target distance and large obstacle distance are desired. I note that c_1 and c_3 had the smallest impact on performance simply because the magnitude of R_T and R_O is smaller than the other values. After training the control policy using the optimal hyperparameters, I observed more robust obstacle avoidance, where the average obstacle

distance rose from 4.97 cm to 11.83 cm. The target distance was not significantly affected, rising from 7.18 cm to 8.54 cm, which is within the margin of error I recorded for multiple trials.

As shown in Table 1, both the 5 DOF and 7 DOF Kinova robots were able to successfully reach the goal points within 13 cm average error. The 5DOF robot performed better than the Kinova robot, likely because the shoulder ball joint offers far more flexibility than the three real shoulder actuators on the Kinova robot. The values in Table 1 include cases where the goal point is inside of the obstacle, where I observed the robot to correctly avoid the obstacle while maintaining the closest possible distance to the goal point. This is an important feature of this system, since it demonstrates robust obstacle avoidance, even if the human operator is unaware of an obstacle and instructs the robot to collide with it. These results demonstrate a successful method for using real-world trajectories to operate a robot using reinforcement learning-based control policies, a necessary prerequisite for using demonstrations to train robots to complete more complex tasks.

Robot Policy	Error (cm)
Kinova	15.2
5DOF	11.2

Table 1: Comparison of target point error between different robots

4.2 Task 2: Block pushing

Figure 8 shows the performance of TD3, PPO, and the expert on the block pushing environment. TD3 was able to produce a policy that could successfully push the block to the goal around half of the time. I observed that most of the failures of TD3 were due to the robot overshooting the goal, so it's possible that adding an additional term to the reward function that prioritizes slow block movement could improve performance.

PPO, on the other hand, was not able to produce a successful policy. The agent was not able to learn how to contact the block, so the reward value never increased significantly. I believe this

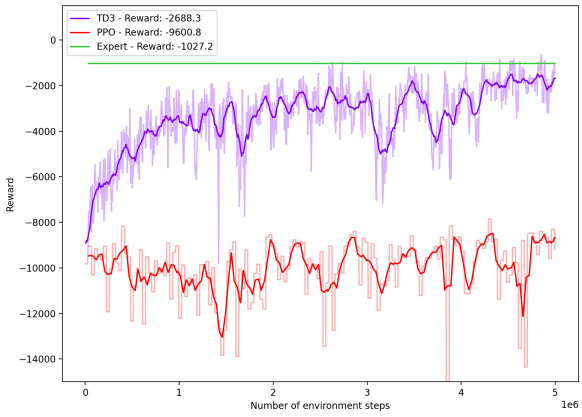


Figure 8: Training results on the block pushing environment

is because there are infinitely many ways to reach the object, and PPO struggles in this underconditioned scenario. Whereas the hand following environment had an additional obstacle constraint that gave preference to one specific configuration, in the block pushing environment, the constraining factor is how well the configuration pushes the block towards the goal location. This is a much more challenging constraint to learn, and the result is that subsequent policy updates destroy any progress that is made from successful trajectories. Since on-policy learning throws away successful trajectories after a single training update, the successful trajectories make up a negligible portion of the training dataset, and training data ends up being mostly noise.

Using the trajectories from the expert dataset, I evaluated the expert's performance using the same reward function as the agent. The expert performed 67.0% better than TD3, demonstrating the gap in performance that learning from demonstrations could help fill. The expert successfully completed the task 100 percent of the time, but still received a negative reward from every time step before the target was in the goal location.

4.3 Physical Trials & Prototype

Figures 9 and 10 show physical trials on a Kinova Gen3 robot. The images on the left show the raw expert locations and the locations of a wooden block. The center images show the arm following prototype, where the robot must follow all joint angles of the expert. The images on the right demonstrate the PPO hand following policy trained in section 4.1.

The first trial (Figure 9) focuses on the block pushing task. This trial is different from the task in section 4.2 in that the robot is completely teleoperated by the expert using Vicon data, and it has no knowledge of the block or goal locations. The environments are aligned so that the block and goal have the same locations as the expert environment. The sequence of images demonstrates how both the hand following policy and full arm following prototype are able to push the block to the goal location. The arm following prototype correctly matches the joint angles of the expert demonstration, with the correct elbow orientation. The hand following policy is only constrained to the expert's hand path, so it exhibits some learned orientation of the elbow, which is also able to complete the task.

Figure 10 demonstrates both control policies in a more challenging scenario. In this trial, the expert picks up the block and places it inside of a shallow box. Since the expert trajectories do not currently gather any data to operate the robot's gripper, the gripper is controlled manually. Both methods were able to successfully maneuver to the starting location and pick up the block. When lifting the block, the arm following prototype holds the block naturally, similar to the expert. The hand following policy positions its elbow even higher than the object, which is not an ideal configuration since it nearly reaches a singularity in the elbow joint of the manipulator. In the last row of images, the arm following prototype successfully places the block inside of the box and the hand following policy drops the block from around 20cm above the box. This is due to the hand location nearly reaching the edge of the workspace, where the hand following is not as accurate.

These physical demonstrations show how an expert can teleoperate a robot using only their own movement and a motion capture system. This offers a far more intuitive method for controlling robots than traditional teleoperation systems and can be utilized when specific expertise must be

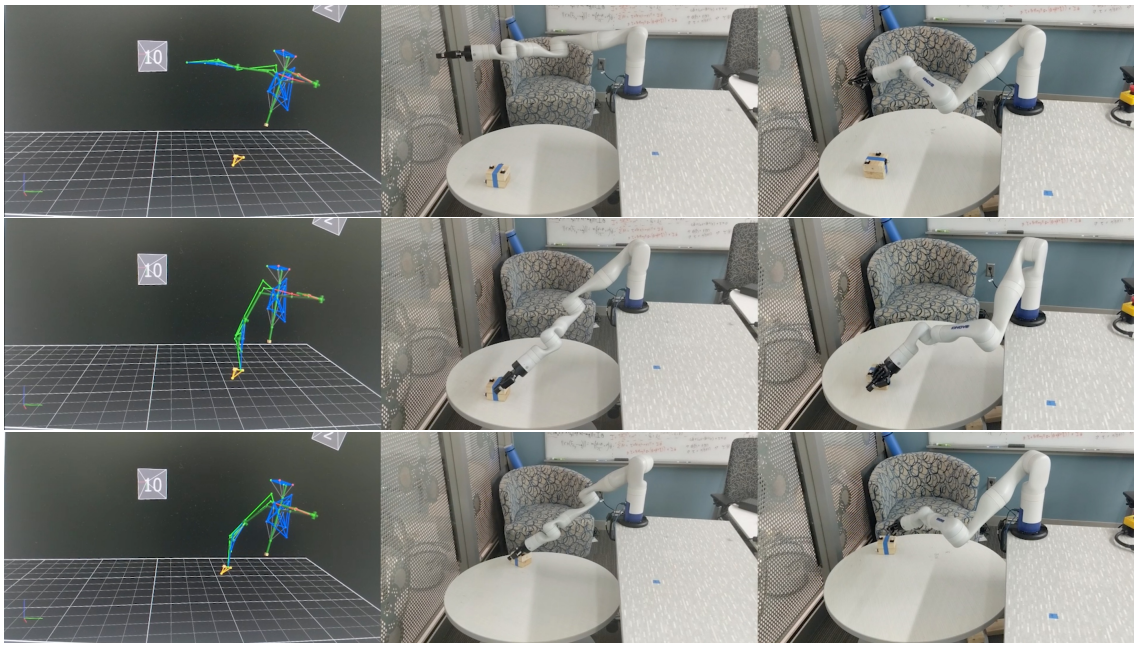


Figure 9: Demonstration of the Kinova robot completing the block pushing task (left to right: expert demonstration, direct joint mapping, learned hand following)

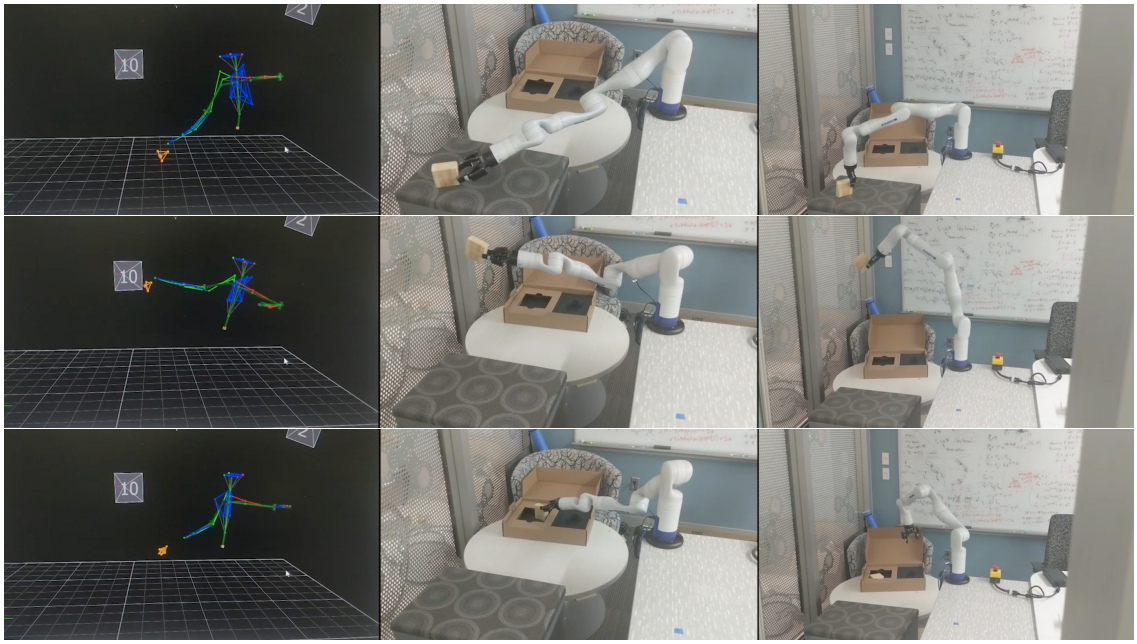


Figure 10: Demonstration of the Kinova robot completing the pick and place task (left to right: expert demonstration, direct joint mapping, learned hand following)

quickly replicated on a robot (e.g. in tele-health or manufacturing). Since the expert and agent share the same state and action space, this prototype system can be utilized in future research to teach the agent new skills autonomously from demonstrations.

5 Conclusion

In this work, I integrated human and robot action using motion capture and reinforcement learning. Both direct joint mapping and learned hand following are implemented on a Kinova Gen3 robot, and I demonstrate successful teleoperation on block pushing and pick and place tasks. Multiple state of the art RL algorithms are compared to the performance of the expert, highlighting the ability for human demonstrations to aid in training. Although PPO was preferred for the hand following task, only TD3 was able to successfully complete the block pushing task, which is a trade-off that must be considered in future research. This work provides foundational insight and builds the systems necessary for developing an Internet of Skills that will enable robots to learn complex tasks from human demonstrations.

6 Future work

Future goals for this project are to utilize the expert demonstrations during training, similar to [6]. Doing so would improve convergence times and allow the robot agent to complete more complicated skills. There are many imitation learning techniques such as generative adversarial imitation learning [27] that could be leveraged in order to develop new LfD methods that offer better performance and more stable training than current methods.

Next, I propose the use of recurrent neural networks (RNNs) to form a mapping from human to robot joint angles. Robots that have significantly different actuator layout than a human arm will not be able to directly interpret the actions that the human experts take in the environment. This constraint led me to choose the Kinova 7DOF robot for my initial testing, since the human-robot mapping is nearly identity. Learning a joint angle mapping with RNNs would allow for any robot phenotype to learn from human demonstrations, even if the number of actuators or physical layout varies greatly from a human's.

Lastly, I would like to incorporate gripping data into the demonstration dataset so that the robot can learn when to open or close its end effector. Initially, the robot could infer that when an object is lifted, the gripper must be closed, and when it is set down, the gripper must be open. Later, CNNs could be trained to predict whether the expert is gripping an object, similar to [28], which would give more reliable gripper data.

References

[1] T. L. Paine, C. Gulcehre, B. Shahriari, M. Denil, M. Hoffman, H. Soyer, R. Tanburn, S. Kapturowski, N. Rabinowitz, D. Williams, G. Barth-Maron, Z. Wang, N. de Freitas, and W. Team, "Making efficient use of demonstrations to solve hard exploration problems," 2019.

[2] V. G. Goecks, G. M. Gremillion, V. J. Lawhern, J. Valasek, and N. R. Waytowich, "Integrating behavior cloning and reinforcement learning for improved performance in dense and sparse reward environments," in *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, ser. AAMAS '20. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2020, p. 465–473.

[3] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband, G. Dulac-Arnold, J. Agapiou, J. Leibo, and A. Gruslys, "Deep q-learning from demonstrations," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, Apr. 2018. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/11757

[4] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Overcoming exploration in reinforcement learning with demonstrations," *CoRR*, vol. abs/1709.10089, 2017. [Online]. Available: http://arxiv.org/abs/1709.10089

[5] B. Kang, Z. Jie, and J. Feng, "Policy optimization with demonstrations," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 2469–2478. [Online]. Available: https://proceedings.mlr.press/v80/kang18a.html

[6] M. Vecerík, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. A. Riedmiller, "Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards," *CoRR*, vol. abs/1707.08817, 2017. [Online]. Available: http://arxiv.org/abs/1707.08817

[7] J. Schulman, J. Ho, C. Lee, and P. Abbeel, *Learning from Demonstrations Through the Use of Non-rigid Registration.* Cham: Springer International Publishing, 2016, pp. 339–354. [Online]. Available: https://doi.org/10.1007/978-3-319-28872-7_20

[8] Y. Mohammad and T. Nishida, "Tackling the correspondence problem," in *Active Media Technology*, T. Yoshida, G. Kou, A. Skowron, J. Cao, H. Hacid, and N. Zhong, Eds. Cham: Springer International Publishing, 2013, pp. 84–95.

[9] Z. Al-Qurashi and B. D. Ziebart, "Recurrent neural networks for hierarchically mapping human-robot poses," in *2020 Fourth IEEE International Conference on Robotic Computing (IRC)*, 2020, pp. 63–70.

[10] A. Jackson, B. D. Northcutt, and G. Sukthankar, "The benefits of immersive demonstrations for teaching robots," in *2019 14th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2019, pp. 326–334.

[11] B. Sangiovanni, A. Rendiniello, G. P. Incremona, A. Ferrara, and M. Piastra, "Deep reinforcement learning for collision avoidance of robotic manipulators," in *2018 European Control Conference (ECC)*, 2018, pp. 2063–2068.

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[13] A. A. Shahid, L. Roveda, D. Piga, and F. Braghin, "Learning continuous control actions for robotic grasping with reinforcement learning," in *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2020, pp. 4066–4072.

[14] L. C. Melo, D. C. Melo, and M. R. Maximo, "Learning humanoid robot running motions with symmetry incentive through proximal policy optimization," *Journal of Intelligent & Robotic Systems*, vol. 102, no. 3, pp. 1–15, 2021.

[15] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," *CoRR*, vol. abs/1802.09477, 2018. [Online]. Available: http://arxiv.org/abs/1802.09477

[16] 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," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016. [Online]. Available: http://arxiv.org/abs/1509.02971

[17] Y. Wu, E. Mansimov, S. Liao, R. B. Grosse, and J. Ba, "Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation," *CoRR*, vol. abs/1708.05144, 2017. [Online]. Available: http://arxiv.org/abs/1708.05144

[18] 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," *CoRR*, vol. abs/1602.01783, 2016. [Online]. Available: http://arxiv.org/abs/1602.01783

[19] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *CoRR*, vol. abs/1801.01290, 2018. [Online]. Available: http://arxiv.org/abs/1801.01290

[20] Z. Zhu, K. Lin, and J. Zhou, "Transfer learning in deep reinforcement learning: A survey," *CoRR*, vol. abs/2009.07888, 2020. [Online]. Available: https://arxiv.org/abs/2009.07888

[21] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[22] V. Zhang, "Gen3 mujoco," https://github.com/vincentzhang/gen3-mujoco, *GitHub repository*, 2019.

[23] I. Kostrikov, "Pytorch implementations of reinforcement learning algorithms," https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail, *GitHub repository*, 2018.

[24] R. Yang, "torchrl," https://github.com/RchalYang/torchrl, *GitHub repository*, 2021.

[25] J. C. Perry, J. M. Powell, and J. Rosen, "Isotropy of an upper limb exoskeleton and the kinematics and dynamics of the human arm," *Applied Bionics and Biomechanics*, vol. 6, no. 2, pp. 175–191, July 2009. [Online]. Available: https://doi.org/10.1080/11762320902920575

[26] KinovaRobotics, "ros_kortex," https://github.com/Kinovarobotics/ros_kortex, *GitHub repository*, 2021.

[27] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper/2016/file/cc7e2b878868cbae992d1fb743995d8f-Paper.pdf

[28] D. Shan, J. Geng, M. Shu, and D. F. Fouhey, "Understanding human hands in contact at internet scale," *CoRR*, vol. abs/2006.06669, 2020. [Online]. Available: https://arxiv.org/abs/2006.06669

Appendix A - Vicon to Kinova joint mapping

The Vicon Nexus software is able to compute joint angles using the plug-in gait model; however, the Kinova robot's actuator configuration requires that joint angles are calculated in a specific order so I needed to calculate the joint angles from the raw position data rather than use the angles calculated by Vicon's software. To find the joint angles in a format that could be mapped to the Kinova robot, I first convert positions to quaternions, then from quaternions to Euler angles. All rotations are defined as being relative to the "zero position," shown in Figure 4. The shoulder tracker is defined as the origin of the entire coordinate frame.

First, I calculate the unit vectors from shoulder to elbow, elbow to wrist, and wrist to finger ($\hat{u}_{\text{sho-elb}}, \hat{u}_{\text{elb-wri}}, \hat{u}_{\text{wri-fin}}$). For the shoulder movement, the quaternion from the zero position ($\hat{x}$) to the current position ($\hat{u}_{\text{sho-elb}}$) can be found with

$$\begin{aligned} a &= \hat{u}_{\text{sho-elb}} \times \hat{x} \\ w &= 1 + \hat{u}_{\text{sho-elb}} \cdot \hat{x} \\ q_{\text{sho}} &= [a_x, a_y, a_z, w] \end{aligned}$$

and then normalizing the result $\hat{q}_{\text{sho}} = q_{\text{sho}}/|q_{\text{sho}}|$. The first two actuator angles (shoulder horizontal abduction and vertical flexion) can be found by converting $\hat{q}_{\text{sho}}$ to Euler angles in the intrinsic ZYX ordering.

$$[\theta_Z, \theta_Y, \theta_X] = \texttt{QUAT\_TO\_EULER}(\hat{q}_{\text{sho}}, \texttt{'ZYX'})$$

The Z axis rotation is actuator 1, and the Y axis rotation is actuator 2.

$$\begin{aligned} \theta_1 &= \theta_Z \\ \theta_2 &= \theta_Y \end{aligned}$$

The X rotation represents actuator 3 (shoulder rotation), but this value cannot be uniquely determined using only $\hat{u}_{\text{sho-elb}}$. Actuator 3 is calculated by setting the X rotation to zero, then calculating the true value using $\hat{u}_{\text{elb-wri}}$:

$$\begin{aligned} R_0 &= \texttt{EULER\_TO\_MATRIX}([\theta_Z, \theta_Y, 0], \texttt{'ZYX'}) \\ \hat{z}_{\text{sho}} &= R_0 \hat{z} \\ \theta_3 &= \cos^{-1}(\hat{u}_{\text{elb-wri}} \cdot \hat{z}_{\text{sho}}) \end{aligned}$$

where R_0 is the rotation matrix of $\hat{q}_{\text{sho}}$ with the X Euler angle set to zero.

Actuators 4 and 6[2] are comparatively simple to calculate as the angles between two vectors.

$$\theta_4 = \cos^{-1}(\hat{u}_{\text{sho-elb}} \cdot \hat{u}_{\text{elb-wri}})$$
$$\theta_6 = \cos^{-1}(\hat{u}_{\text{elb-wri}} \cdot \hat{u}_{\text{wri-fin}})$$

In order to incorporate the wrist rotation and abduction, the same process as above for calculating shoulder angles can be used. This is left to future work, since these degrees of freedom were not necessary for the demonstrations in this paper.

Appendix B - Hyperparameters

Parameter	Optimal Value	Estimated Value*
c_1	1500	1000
c_2	10	10
c_3	200	60
c_4	0.01	0.01
δ	0.005	0.01
p	6	8
d_{ref}	0.03	0.03

Table 2: Hand Following Hyperparameters.
*Estimated values were used for generating Figures 5-7.

Parameter	Value
c_1	1000
c_2	100
δ	0.06

Table 3: Block Pushing Hyperparameters.

[2]Actuator 5 is wrist abduction and is not currently implemented

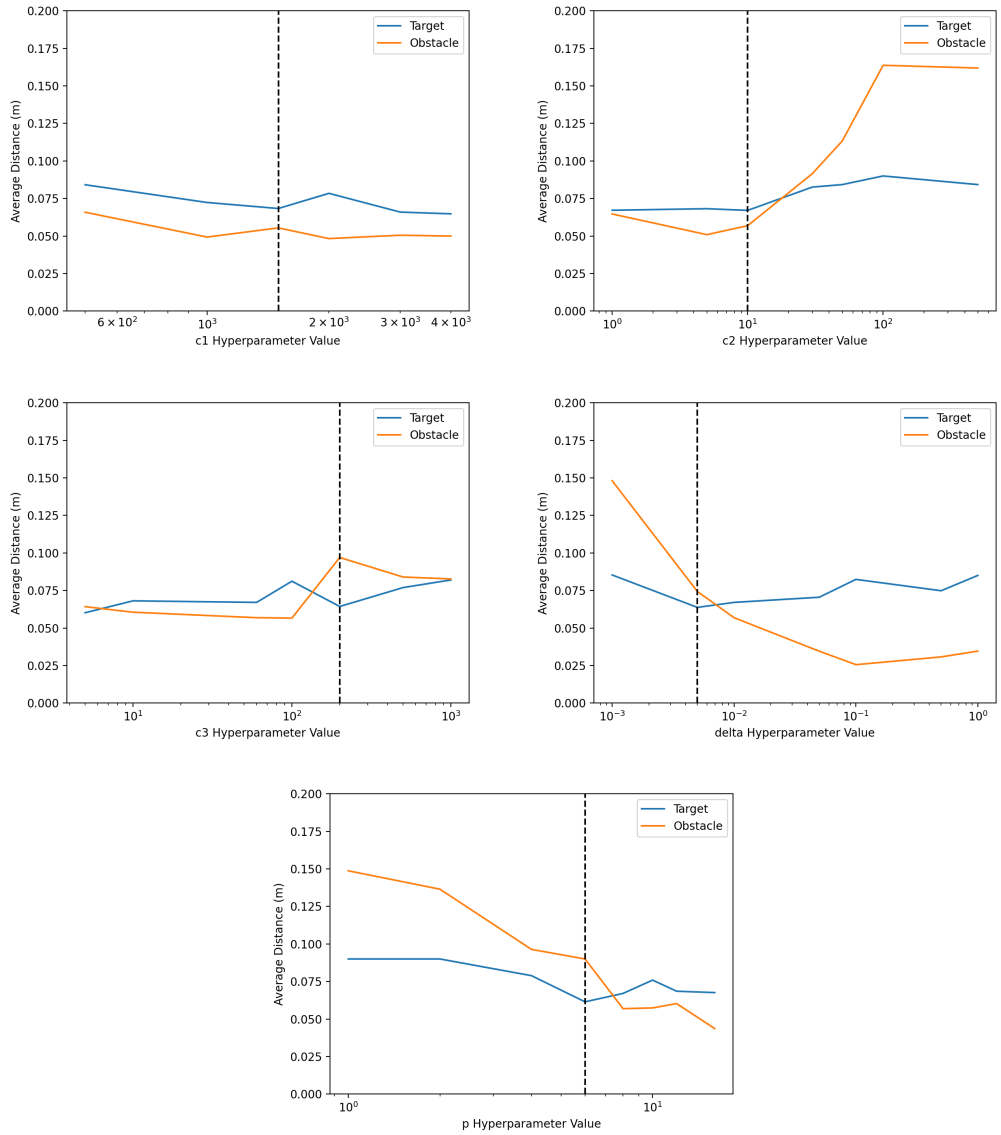


Figure 11: Performance comparison of hand following hyperparameter choices. Optimal values are marked with the black dashed line and are given in Table 5. c_4 and d_{ref} are chosen empirically since there is no quantifiable selection method.

Parameter	Value
Learning rate	$3 * 10^{-4}$
Entropy coefficient	0.01
Value loss coefficient	0.5
Epochs per update	10
Num mini batch	1
Discount Factor (γ)	0.99
GAE Discount Factor (γ_{GAE})	0.95
Clip	0.2

Table 4: PPO Hyperparameters

Parameter	Value
Start Timesteps	$25 * 10^3$
Evaluation Frequency	$5 * 10^3$
Noise STD	0.1
Batch Size	256
Discount Factor (γ)	0.99
Tau τ	0.005
Policy Frequency	2

Table 5: TD3 Hyperparameters