

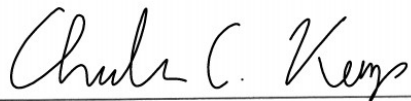
Investigating Sim-to-Real Transfer and Multi-Agent Learning in Assistive Gym

Holden C. Schaffer

Healthcare Robotics Lab, Georgia Institute of Technology

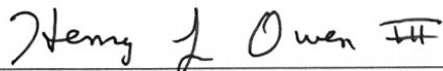
Faculty Advisor: Charles C. Kemp

Graduate Co-Mentor: Zackory Erickson



Charlie C. Kemp

Faculty Advisor



Henry L. Owen III

Second Faculty Reader

Investigating Sim-to-Real Transfer and Multi-Agent Learning in Assistive Gym

Holden Schaffer

Contents

Abstract	2
1 Transferring Policies from Assistive Gym to a PR2 for Drinking Assistance	3
1.1 Introduction	3
1.2 Literature Review	5
1.3 Methodology	7
1.4 Results and Discussion	10
1.5 Conclusion and Future Works	15
2 Multi-Agent Learning in Assistive Gym	16
2.1 Introduction	16
2.2 Literature Review	18
2.3 Methodology	21
2.4 Results and Discussion	24
2.5 Conclusion and Future Works	28

Abstract

As the world's population grows older on average and the number of available caregivers decreases, assistive robotics pose an opportunity for older adults or people with disabilities to continue receiving the care that they need. Recent work has shown tremendous progress in using deep reinforcement learning to teach robotic caregivers how to properly assist people in simulation, where robots can learn how to interact with humans in a safe, controlled manner. However, transferring what the robot has learned from simulation to reality continues to pose a challenge for assistive robotics, and a gap in the literature exists in finding techniques to overcome this challenge for this particular domain. The first part of this research uses an assistive simulation framework known as Assistive Gym and its simulated drinking environment to test various approaches to sim-to-real transfer for assistive robotics. The end result of this portion of the research is the identification of a series of baseline steps that are necessary to transfer the Drinking task in Assistive Gym to a physical PR2. Next, the avenues for future works are addressed by investigating a few potential modifications to the drinking task which could be implemented for a more successful transfer of policies. The second part of the research investigates how multi-agent learning could be implemented in Assistive Gym. This section implements multi-agent assistance for the bed-bathing environment, then tests the effectiveness of three different algorithms in order to gauge their effectiveness for solving this new multi-agent task. These algorithms include two variations of single-agent Proximal Policy Optimization modified for multi-agent use as well as Multi-Agent Deep Deterministic Policy Gradient. Finally, future works related to multi-agent assistance are discussed, namely choosing alternate implementations of MADDPG and investigating the dressing environment for its greater potential for cooperation between robots.

1 Transferring Policies from Assistive Gym to a PR2 for Drinking Assistance

1.1 Introduction

A 2015 report from the World Health Organization concluded that as the number of older adults needing care rises worldwide, the total number of caregivers for these individuals is decreasing [1]. As healthcare systems become increasingly strained, greater numbers of working adults must act as unpaid caregivers to their family members. The Congressional Budget Office estimated that unpaid older adult care in the United States in 2011 was worth \$234 billion in lost productivity [2]. This economic burden has been largely shouldered by family caregivers alone [3]. Beyond that, caregiving can challenge the mental and physical well-being of both the caregiver and the recipient. Caregivers frequently report struggling with isolation and unpredictable circumstances as they put their lives on hold to support another person [4]. On the other side, those receiving care often have difficulty adjusting to decreased autonomy and additional challenges posed by daily living. Assistive robotics presents an opportunity to alleviate some of these issues.

Assistive robotics refers to the branch of robotics that directly interacts with humans to help them accomplish certain tasks. For instance, an assistive robot might hold a glass of water to someone's mouth to help them eat or pull a gown over their shoulders to help them get dressed. In hospital settings, these robots might provide sponge baths for patients or lift someone's arm back onto their bed if they aren't able to move it themselves.

Deep Reinforcement Learning (DRL) has demonstrated remarkable progress in recent years in generating controllers for robots to complete such tasks [5, 6, 7, 8, 9]. During DRL training, a robot first takes a randomly generated action. After it takes this action, the robot is given a numerical reward value corresponding to the relative success of that action. For instance, if that action moved the robot closer towards completing the task, then the reward value for that action would be high. Eventually, through trial and error and a large number of repeated iterations, the robot can learn

which actions it should take at any given state in order to maximize reward. This pairing between states and actions is referred to as the robot's policy. When the robot has learned the best policy, it has learned how to complete the task.

However, this approach requires a very large number of training samples that can be both expensive and time-consuming to collect. For robots that must train through human interaction, this process can even be dangerous. For this reason, physical training of the robot is often replaced by training inside of a simulation. In a simulation, a robot is able to learn from its mistakes without running the risk of harming any humans, property, or itself. Furthermore, many iterations can be trained simultaneously to greatly expedite the training process.

In the past year, the Healthcare Robotics Lab at Georgia Tech created Assistive Gym, which is a series of environments and benchmarks that enable assistive robots to learn how to aid humans in a safe, simulated environment [10]. Notably, this research is unique in that it emphasizes a human-centric approach by simulating human needs and desires to be used as rewards for policy learning. For instance, applying too much physical force onto a human would cause the reward value for that action to dramatically decrease. In Assistive Gym, these human needs and desires are used to simulate tasks related to daily living such as dressing or feeding assistance. Twenty-four environments (six tasks implemented with four robots each) have been created in simulation. The next step of the research is to transfer the policies learned in these simulations to physical robots that are able to assist real people.

Unfortunately, training in simulation often does not transfer well to a real environment without careful data collection and environment manipulation [9]. The real world contains noise and physical variability such as actuator delays that are difficult to model [11]. Other factors that may be possible to model in theory cannot be practically modeled given modern constraints on computing power. The discrepancies between simulated environments and real environments are referred to as the Reality Gap [12].

The primary objective of this research is to explore how control policies for assistive robots learned in simulation can be transferred to real robots interacting with real people. To accomplish

this, this research aims to identify some of the key elements necessary in order to transfer a particular task achieved in simulation to reality with a PR2 robot, which is a humanoid research robot developed by Willow Garage. This task to be attempted is the drinking task implemented in Assistive Gym. Chosen for its relative simplicity, this task consists of a robot carefully holding a cup of water up to a person’s mouth to help them drink. Various transfer techniques will be iteratively incorporated into the drinking environment until transfer is successful, such as domain randomization [11, 13] and domain-specific system identification [9]. To provide a brief description of these two approaches, domain randomization involves randomizing the simulation environment so that learned policies are more resilient to environmental changes. On the other hand, system-identification involves identifying key elements of the real environment to incorporate into the simulation environment in an effort to reduce the gap between the two. Further details and examples of these approaches can be found below. Finally, the approaches which best transfer policy learned in simulation to reality will be documented for future integration into the Assistive Gym framework to lay a foundation for transferring future tasks. By integrating sim-to-real approaches in the Assistive Gym framework, it will become much easier for researchers and roboticists to use this framework to train physical robotic systems with real world healthcare applications. By further developing the potential of assistive healthcare robotics systems such as Assistive Gym, this research seeks to take steps towards alleviating the crisis in assistive care by exploring the potential of assistive healthcare robotics across both the simulated and the physical world.

1.2 Literature Review

Recently, DRL has found some successful uses in robotics research. When compared to the analytical models hand-tuned by humans, reinforcement learning models demonstrate remarkable improvements both in task efficiency and the development of new learned behaviors [11]. The research discussed in this paper uses an algorithm known as Proximal Policy Optimization (PPO), which was developed by OpenAI researchers in 2017 [3]. PPO alternates between sampling data through interaction with the environment and optimizing a ”surrogate” objective function using

stochastic gradient ascent. This new objective function is given as:

$$L^{CLIP}(\theta) = \hat{E}_t[\min(r_t(\theta)\hat{A}, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t)]$$

where θ is the policy parameter, \hat{E}_t is the empirical expectation over timesteps, r_t is the ratio of the probability under the new and old policies, \hat{A}_t is the estimated advantage at time t, and ϵ is a hyperparameter [3]. Aside from this brief description, further exploration of algorithm choice and advantages will not be discussed at length, as it does not play a major role in sim-to-real transfer.

To this end, there are several instances of successful sim-to-real transfer in recent years [11, 6, 9, 13, 14, 15, 16]. One common thread between each success is their use of domain-specific approaches to solving the problem. For instance, many take advantage of a technique known as system identification (sysID), which involves injecting key domain-specific parameters into the simulation. For instance, measuring friction values of a gown used by a physical robot to dress a patient would be important to inject into the simulation. Some researchers take this a step further by splitting system identification of the hardware into two stages: pre-training (pre-sysID) and post-training (post-sysID) [9]. During pre-sysID, the researchers did not attempt to measure the true value of physical hardware parameters but instead approximated that range of values the parameters could hold. Next, they trained a number of machine learning models conditioned on transformed random parameters values. Finally, during post-sysID, they used Bayesian optimization to choose parameter values which optimize policy performance on the physical robot.

Another common approach in attempting to solve sim-to-real transfer is to use a technique known as domain randomization. This technique involves randomizing or adding noise into the simulation in some fashion so that learned policies are robust enough to transfer to a different environment [17, 13]. By randomizing the simulation, the policy generated learns to adapt to variability, which is one important component of transferring to the physical world. Of course, many studies employ both sysID and domain-randomization to maximize the chance of successful transfer. Aside from these two techniques, various researchers have also developed solutions that are even more

domain-specific, such as using Randomized-to-Canonical adaptation networks for image data [18] and unique system design choices for grasping [15].

Aside from sim-to-real, many robotics tasks have been solved by training in the physical world, such as tossing arbitrary objects and in-hand manipulation [5, 7]. Significant efforts have been concentrated in the domains of grasping, walking, and batting [19]. However, a gap exists in learning policies in the domain of assistive robotics. In the past, the Healthcare Robotics Lab at Georgia Tech has successfully used deep haptic model predictive control [20] and multidimensional capacitive sensing [21] to achieve robot-assisted dressing and bathing. To improve performance, a potential next step would be completing these tasks using deep reinforcement learning. Notably, assistive healthcare robotics tasks differ fundamentally from traditional robotic tasks such as grasping objects because the former must directly interact with humans. To minimize risk, human preferences must be simulated and incorporated into every step of the process. Thus, a unique blend of strategies must be incorporated and developed in order to successfully transfer these policies to a physical robot. The goal of this research is to fill this gap in the literature by building towards sim-to-real transfer in assistive robotics.

1.3 Methodology

The methodology for this paper consisted of three primary steps. First, the parameters of the simulation environment and real world environment were tested and synchronized. Next, an initial policy was trained on the Drinking PR2 simulation environment in Assistive Gym. Finally, varying strategies of sim-to-real transfer learning were iteratively attempted and implemented based on observed levels of success. Details on each of these three steps can be found in the following sections.

Section 1: Synchronizing Simulation and Real World Parameters

In order to transfer anything from the simulation to the physical environment, it was discovered that there are a few essential parameters which must be made consistent between the two environ-

ments. The first item requiring conversion was the coordinate system referenced by the robot in the real world versus simulation. The physical robot used a local coordinate system whose origin was the center of the robot, so this needed to be synchronized with the virtual robot’s coordinate system. Next, in simulation, observations returned from the robot are normalized by the model before training. To maintain consistency, the same normalization formula must be manually applied to the physical robot’s observations before inputting them into the model. The formula used for this normalization (RunningMeanStd) is as follows:

$$obs = \max(-10, \min((obs - \bar{x}_o) / \sqrt{(\sigma_o^2 + \epsilon)}, 10))$$

where obs are the observations, \bar{x}_o is the running mean stored in the model, σ_o^2 is the running variance stored in the model, and epsilon is a constant defined as 1e-8.

Section 2: Train PPO Policy in Assistive Gym

Once these parameters were synchronized, an initial policy was then trained in simulation. In this context, the term ‘policy’ refers to the robot’s learned mapping between state of the environment (observations), and how the robot should move given those observations (actions). This research uses a custom implementation of OpenAI’s Proximal Policy Optimization baseline implementation to train policies, located in Assistive Gym [10]. This algorithm was chosen for its previous successes in generating policies for Assistive Gym tasks, as documented in [10]. However, the choice of reinforcement learning algorithm is largely inconsequential for the purposes of this research. In theory, any choice of algorithm that provides good results and can operate in a continuous environment could be chosen for transfer - choice of algorithm is not the focus of this research. For detailed instructions on training policies in Assistive Gym, please refer to the Assistive Gym Wiki articles linked in the repository.

At the time of research, Assistive Gym used Python 3.6.3 and a custom implementation of Bullet3 physics, which is also linked in the Assistive Gym repository. Bullet3 Physics is an open-source

Physics SDK with python bindings which supports real-time collision detection and multi-physics simulation for robotics. While choice of physics engine may play a role in sim-to-real effectiveness, this topic will not be addressed in this paper. As far as the training algorithm, PPO hyperparameters were left identical to the ones chosen for the original Assistive Gym Paper [10] due to proven effectiveness and a desire to maintain consistency. The algorithm was allowed to run for ten million timesteps in order to generate the policy. Each episode was 200 timesteps long, meaning that the robot was able to take 200 actions and observe 200 observations for each attempt to solve the task. After each episode, the environment and robot was reset back to the initial position. As the training episodes progress, the robot learns which actions are good and which are bad until it develops an effective policy on how to complete the task. Similar to before, 10 million iterations were chosen because this number proved to be large enough for the robot to have time to develop an effective policy, while also being small enough to train in a reasonable timeframe.

Section 3: Iteratively Implement and Test Various Sim-To-Real Techniques

Upon running the policy on the physical robot, it was noted that the primary inconsistency between the simulation and real world environment was the execution of the action returned by the trained model. In other words, given the same action, the robot in simulation and the physical robot would move in slightly different ways. For instance, suppose the robot in simulation generated a series of actions that had the effect of moving its end effector forty-five degrees to the left and five inches up. If this same sequences of actions were fed to the physical robot, it was noted that the net result was something entirely different and not recognizable as the same movement of the simulated robot.

There were three primary strategies attempted to resolve this issue:

1. Evaluation of four alternative movement strategies:
 - (a) Record desired end-effector position in action and use inverse kinematics to calculate desired joint angles during runtime.
 - (b) Record changes to the target position during runtime. Use inverse kinematics to move

- the robot end-effector towards the new target position, calculated by the current position plus the offset.
- (c) Record desired joint angles in action and move the robot to those joint angles during runtime.
 - (d) Record change in joint angles in action and modify robot current joint angles at runtime.
2. Perform gains adjustments so that the robot's movement speed in simulation matched the physical robot.
 - (a) Option 1: Adjust delays until both are reaching approximately the same percentage of the desired target
 - (b) Option 2: Add delays to both so that both reach their target destination
 3. Inject Domain Randomization by adding random noise values equal to a magnitude roughly 0.05 to observations in the simulation.

After each modification, another policy had to be retrained to account for the additional modifications. After that, the success of the policy was evaluated on the physical robot by evaluating the average reward at the end of each episode of 200 timesteps. Note that the 'reward' measures the overall effectiveness of the robot in completing the task in the given number of timesteps. For instance, one component of the reward was the distance from the end effector to the desired mouth position. Once all the different modifications had been attempted, the most beneficial strategy was chosen and integrated into Assistive Gym. This is discussed in the following section.

1.4 Results and Discussion

At the onset of experimentation, it became clear that standardizing the coordinate systems and joint angles between the simulation and real world environment would be an important component to properly test and ensure consistency between before proceeding with other techniques. While the

problems posed by desynchronous coordinate frames and joint angles are not unique to environments involving humans, they do become more pronounced. For instance, if a robot is simply learning how to pick up an object, a slight difference in reference frames may only result in a robot picking up the object in a suboptimal grip. On the other hand, when the robot is interacting with humans, this might instead cause the robot to poke someone's eye instead of wash around their eye. For this reason, for sim-to-real transfer in human-assistive environments, it is critical to spend adequate time ensuring that coordinates systems are identical, joint positions have the same magnitudes, observations are normalized properly, etc. If this is done incorrectly, it becomes very difficult for more advanced transfer techniques to demonstrate any success or progress, as the environment and simulation are fundamentally mismatched.

After this research had ensured consistent environments, it then became time to evaluate four different movement strategies to ensure that the robot taking a particular action in simulation would result in the same movement as the physical robot when given the same action. As noted in the methodology, this proved to be the primary inconsistency found between the simulation and real world environment. It is also worth noting that the viability of these different strategies may vary depending on the choice of robot and control framework, so the method discovered to be the most effective for this environment may not necessarily be the best method for all environments.

The first movement strategy tested was a strategy in which the desired end-effector position of the robot was stored directly as the action. During runtime, inverse kinematics (IK) was used to generate joint angles which the robot would move to. This strategy proved useful for ensuring that coordinates were consistent between the simulation and real environment, but unfortunately proved problematic in other ways. Because the joint poses needed to be calculated during the runtime of the policy, the physical robot and simulated robot had to calculate the IK joint angles separately. This means that their respective joint positions naturally differed depending on the solution returned by the IK solver. For instance, given the same end-effector position, the robot in simulation may choose to move its elbow to the right, while the robot in the real world might choose to move its elbow to the left. Both might be technically correct if they result in the same

end-effector position, but this change in joint position could have unintended consequences for a robot helping a human. For instance, this might result in the physical robot elbowing the person in the face while the simulated robot does not. For this reason, this movement strategy was deemed to not be the best strategy for this task.

The second movement strategy tested was similar to the first in that it also relied on keeping track of end-effector positions. However, instead of recording the desired end-effector positions directly, this strategy involved recording the desired change in the target end-effector position in the action at every iteration. During runtime, the robot added this desired change to its current end-effector target position, then used IK to solve for the joint angles needed to move its end-effector to that new target position. Unfortunately, this strategy proved even more unsuccessful than the first. Not only did this approach suffer from the same joint angle inconsistencies, but it also suffered from major drift between the physical and simulated robot's movements. By only storing the changes in the target position rather than the target position itself, any small differences between the achieved end-effector position of the simulated and physical robot would quickly compound until large differences in movement emerged. In other words, if one robot didn't move its end effector to the right position, that error would never be corrected. Instead, it would keep modifying the wrong position in the same way as it would modify the correct position, which would lead to very different movements between the two robots as more errors accumulated over time.

The third movement strategy tested involved recording the desired joint angles in the action, and simply moving the robot to those joint angles during runtime. This provided much more consistent movement patterns compared to storing end-effector positions directly due to decreased reliance on IK solvers during runtime. Despite being more successful, this strategy was still not flawless. When only new joint angles are considered, the fact that the robot in simulation may be able to move faster than the robot in the physical world becomes a problem. If the returned joint angles are unreachable for one or more of the robots given a certain amount of movement time, the situation arises where the simulated robot and physical robot movements may become out of sync. Furthermore, it is possible that the trained policy may learn to return joint angles that overshoot the real desired

position, knowing that the robot in simulation would not actually be able to reach it in the timestep provided. Again, this becomes a problem if the robot in simulation and the robot in the physical world do not have perfectly synchronized speeds, motor forces, and gains - a feat almost impossible to practically accomplish. Thus, it became necessary to examine a fourth approach that would take the magnitude of the change in joint angles into account.

Finally, the last and most successful movement strategy tested involved recording changes in joint angles in the action and adding that change in joint angles to the robot's current joint angles during runtime. This approach maintained the consistency of focusing on joint angles rather than end-effector position, but also enabled the fine-tuning of the possible magnitudes of the changes in joint angles to ensure that both the robot in simulation and the physical robot could reach the same position during a given timestep. In addition, this approach enabled fine-tuning to disallow magnitudes of change that were too small or too large. If the changes in joint angles were very small, then that might result in jerky, unpleasant motion as the robot would be able to reach its target location too quickly and would come to a full stop. While this may be fine for other environments, this jerky motion during assistive environments with tasks such drinking assistance is more likely to result in human discomfort. Thus, due to a combination of the benefits provided by this approach, this fourth movement strategy was chosen for use in further experiments. As a bonus to this approach, due to the nature of recording the change in joint angles, it became unnecessary to do a complex synchronization efforts to synchronize the gains between the simulation and physical robot, as both robots could measure their joint angles directly to ensure that the proper movement was performed. However, this synchronization was mentioned in the methodology because doing so might be necessary for other researchers to replicate the results on their own system if one of the prior movement strategies was found to be more successful for their robot and environment.

Once the proper movement strategy had been identified, domain randomization was injected into the environment by adding random noise values equal to a magnitude roughly 0.05 to observations in the simulation. This magnitude was chosen as it was believed to approximate the average amount of noise present in the sensor data. However, injecting this randomization did not appear

to affect the performance of the robot either positively nor negatively in completing the specified task. Thus, it was ultimately discarded in the end.

After implementing the above combination of changes to the simulated learning environment, it was discovered that the trained policies were robust and consistent enough that the physical robot was now able to consistently replay the actions generated by any simplistic policy learned in simulation. To be more precise, the physical robot could accurately replay the actions generated by any environment in which the only information required to complete the task was robot's joint angles and no extraneous sensor data. When a policy was trained on the drinking environment with a static, known mouth position, the physical robot could hold a glass of water and maneuver it to an imaginary person's mouth in nearly exactly the same manner as the simulated robot. By the standards set in the introduction, this meant that this research had accomplished the first step in its goal. It had now successfully synchronized its simulated and physical environment to such an extent that a physical robot could follow a policy trained in simulation to net the same result as a simulated robot, as long as the environment was simplistic enough. It also identified the key elements necessary to evaluate before any assistive environment could be transferred from simulation to reality.

However, there were still some essential components of an assistive task that needed to be implemented before this research could be called a complete success. Because the robot is interacting with people, a drinking task involves more than moving a glass to an imaginary target and changing its orientation so that water slowly pours out. The robot must also be able to perceive and respond to the human's movements, as well as generate its own mouth position during runtime by using sensor data to identify the mouth position of the human.

Unfortunately, due to external global factors beyond the control of this experiment, these extensions were unable to be completed in the desired timeframe of this paper. These possible extensions and intended implementations will be discussed in the following section.

1.5 Conclusion and Future Works

Overall, this paper serves to identify and document a few essential steps needed to transfer the Drinking task in Assistive Gym from the simulation environment to a physical PR2 robot. Namely, these steps are the standardization of world-space, action generation, and action deployment (movement) from the simulated robot to the physical robot. After these baselines are successfully established, the next step of this research would be to integrate domain-specific components which contain extra essential information necessary for the robot to complete more complicated tasks. For instance, this research assumed a static, known mouth position for the Drinking task. In the real world, the robot would need to use computer vision models to be able to identify dynamic mouth position at runtime so that it could assist an actual person with drinking. A key component of future works for this research would be integrating these domain-specific techniques into Assistive Gym so that tasks such as Drinking, Feeding and Dressing can eventually be performed by a physical robot.

Finally, there are a few other clear extensions of this research, mainly relating to exploring alternative characteristics of simulated environments which may enhance their capacities for sim-to-real transfer. For one, it would be beneficial to conduct an analysis of different physics engines in terms of their readiness for sim-to-real transfer attempts. While Bullet3 works well for Assistive Gym purposes, it would be interesting to see if other physics engines make transferring tasks easier or harder. Next, it would be useful to explore and document which alternative reinforcement learning algorithms are most suited to sim-to-real transfer. Finally, it would be useful to evaluate the capacity for Assistive Gym to utilize simulated multi-robot systems and algorithms to solve more complicated assistive tasks. For instance, for dressing assistance, it might be more beneficial to have two smaller robots help dress a person by having one robot hold each sleeve, rather than having one robot attempt to do both. After that's been evaluated, the next logical step would be to convert these environments to physical environments as well, building off of the framework explored by this research. By further developing the potential of assistive healthcare robotics systems such as Assistive Gym, these future works would take steps towards alleviating the crisis in

assistive care by exploring the potential of assistive healthcare robotics across both the simulated and the physical world.

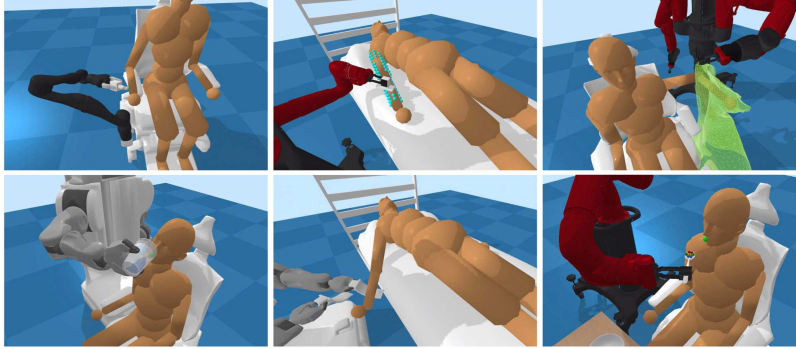
2 Multi-Agent Learning in Assistive Gym

2.1 Introduction

The field of assistive robotics demonstrates significant potential in improving the lives of many struggling older adults or persons with disabilities. However, there are many tasks related to daily living that pose challenges for assistive robotics operating with only a single robot. Sometimes, this challenge is a matter of efficiency. If an older adult or person with disabilities needed robotic assistance with a particular task which involved some sort of discomfort, perhaps the robot might take too long to accomplish the task on its own. Alternatively, perhaps the task is too complicated for a single robot to accomplish with any reasonable accuracy at all. In these instances, it would be useful to investigate how multiple robots could cooperate in order to help humans complete simpler tasks with greater efficiency or advanced tasks with greater accuracy.

The objective of this research is twofold. First, this research seeks to investigate designing and implementing a multi-robot environment in Assistive Gym. Second, this research seeks to test various algorithmic approaches to learning policies for this multi-robot system to determine which is more successful.

There are six environments currently implemented in Assistive Gym, shown in the figure below. These are, from top to bottom and left to right: Itch-scratching, Bed-bathing, Dressing, Drinking, Arm-lifting, and Feeding. For more information on the details of each task, see [10].

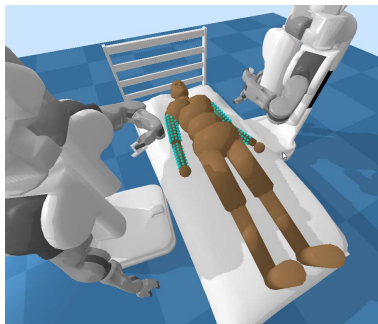


Of the six, Dressing poses the greatest potential for benefiting from multi-agent assistance. It's easy to imagine how two robots can cooperate to help pull a gown over a person by grabbing each sleeve at the same time. However, since the Dressing environment must render a soft-body cloth at every timesteps, adding additional robots made it too computationally expensive to run given the limited timeframe of the research. In Assistive Gym, the single largest resource burden shared between all tasks is the act of loading the complex robot models after every reset of the environment. As the number of robots increases, this computational burdens also increases by the same factor. This factor, in conjunction with the computational requirements of simulating a gown, made the Dressing task unreasonable to consider for this research.

The next best choice of task is Bed-bathing, which was the task chosen for this research. Like the Dressing environment, it's easy to see how two robots could work together to bathe a person at the same time, with each robot bathing each side of a person. Perhaps, if one robot cannot reach a part of a person's arm, the other robot could even cooperate by reaching over to reach the areas where one robot could not. It would be interesting to see if any of this cooperation behavior would emerge from training. In addition, multi-agent assistance for bed-bathing has real-world implications, as a hospital might want to employ two bed-bathing robots instead of one robot so that each patient is able to go through the bathing process as fast as possible. This would not only improve the efficiency of the hospital, but would also minimize the time each patient must spend in the uncomfortable bathing process.

Thus, multi-agent bed-bathing was created as a new environment for the focus of this research. This new environment consists of two robots standing on either side of a person laying down in a

bed. In each of the robot's right end-effectors, there is a small tool used to represent a cloth or a sponge. On either arm of the human, little spheres spawn on the surface of the arm to represent dirt or areas that the robot should wash. When a robot's tool comes into contact with any dot, the dot disappears, and both robots are given the same numerical reward. If any robot applies too much force on a person, both robots are given a negative reward. Since the reward function is shared between both robots, the two robots have the shared goal of wiping down both arms and minimizing the force felt by the person.



After the multi-agent environment was created, it became time to examine different algorithms and approaches to multi-agent reinforcement learning which might be useful to compare within this new environment. These topics will be examined in the below Literature Review.

2.2 Literature Review

Multi-agent reinforcement learning has been used to address problems in many domains, spanning from robotics to distribution control, telecommunications, and economics [22]. A key challenge in multi-agent reinforcement learning is to translate the success of deep learning on single-agent reinforcement learning to the multi-agent setting [23]. One technique that has achieved widespread success in the single agent use cases is known as Deep Q-learning [24]. To briefly summarize this approach, in reinforcement learning, Q-values are values that estimate the long-term expected future reward of taking a particular action at a given state. During the training process, agents learn these Q values and use them to determine which action should be taken at any given state. While not all multi-agent reinforcement learning algorithms use Q-learning, the majority follow a similar

process of using Q-values to dictate the actions that an agent takes. As seen later, one key element of designing multi-agent algorithms is modifying how much information each agent has access to during the Q-value learning process.

In order to learn Q-values, the agent must learn by taking a series of actions and recording the long-term reward values associated with those actions. In order to select which action to take, there are two methods: On-policy algorithms, and Off-policy algorithms [25]. Put simply, on-policy algorithms learn by keeping track of a central policy, then choosing the action during training based on that central policy. On the other hand, off-policy algorithms will take some sort of greedy approach to action selection and often rely on a replay buffer, which is a list previous experiences of the robot that it can continue to learn from. Q-learning is technically off-policy, since it uses stored Q-values to dictate actions rather than relying on a central policy.

In recent years, one on-policy algorithm known as Proximal Policy Optimization [26] has demonstrated some successes over Deep Q-Learning for single-agent environments. Single-agent Proximal Policy Optimization, which was the primary algorithm used in Assistive Gym [10], is an example of an on-policy algorithm that uses Q-values to keep track of a central policy. As this algorithm was already detailed in Part I of this thesis, it will not be discussed in detail for in this literature review. To briefly reiterate, PPO is a policy-gradient algorithm that uses batch-updates to update a central policy. This central policy is a mapping from states to optimal actions which the agent uses during the training and execution processes.

One popular variation of Deep Q-Learning for multi-agent systems is known as Independent Q-Learning [27]. During Independent Q-learning, each agent acts alone with no explicit cooperation with other agents and treats the other agents as part of the environment. In other words, each agent i has a Q-function Q_i which only takes in state data and actions for that robot. In mathematical notation, $Q_i(s_i, a_i) \rightarrow \mathbb{R}$. Despite its relatively simple approach, this strategy has proven to be successful for many multi-agent applications [28].

Even though this research did not test Independent Q-Learning, this algorithm is mentioned here because later in the paper, this research will investigate two variations of transforming single-

agent PPO for multiple agents, with one variation based on Independent Q-learning and the other based on a more cooperative form where both agents share a single Q-value function. Details of these variations will be discussed in the methodology section.

One issue with Independent Q-Learning is that the use of a replay buffer often causes instability, since the nonstationarity introduced by Independent Q-Learning which generated the experience data may no longer reflect the current dynamics of the system [29]. However, in general, it would be useful to use an experience replay to stabilize training and improve sample efficiency. One such algorithm which takes advantage of experience replays for multi-agent learning is an algorithm known as MADDPG, or Multi-Agent Deep Deterministic Policy Gradient [30].

MADDPG, much like single-agent PPO, is a policy gradient algorithm. However, unlike PPO, MADDPG keeps track of a combined experience replay buffer that stores the state information and actions from all agents. This information is used in order to create unique Q functions for each agent, where each Q-function takes in all the observations and actions from all other agents. In mathematical notations, every agent i has $Q_i(s_1, a_1 \dots s_i, a_i \dots s_n, a_n) \rightarrow \mathbb{R}$, where there are n agents total. These Q-functions are used to generate a unique policy for each agent, similarly to PPO. Thus, at runtime, each agent doesn't have to share any information between one another as it did during training, but instead relies entirely on its policy to generate actions. In the original paper, this algorithm demonstrated remarkable improvement over algorithms, including Deep Q-Learning.

One central gap of current literature is that most algorithms and implementations are tested in gamified environments similar to Atari games [24], rather than robotics environments. This includes many of the algorithms discussion above, including MADDPG, which was tested in custom game-like Waterworld environments. It is important to test some of these implementations in more computationally complex environments like robotics in order to ensure the robustness of algorithm implementations for different applications. The process of testing different variations of Multi-agent PPO and MADDPG for robotics applications is described in the following section.

2.3 Methodology

Before the different algorithms could be tested, a multi-agent bed-bathing environment first had to be constructed in Assistive Gym. This research used an August 21st, 2020 version of Assistive Gym V1. The multi-agent bed-bathing environment was fundamentally a mirrored version of the single-agent environment, with one caveat. Normally, before the robot attempts any task in Assistive Gym, the robot is placed in an optimal pose according to a computationally expensive pose optimization algorithm. However, with two robots, it became necessary to replace this expensive pose optimization with simple pose randomization instead. During creation of the environment, each robot is placed in a random location within a 20 x 40 cm box a fixed distance to the left and right of the human model. In addition, the base orientation of the robot is turned randomly by -20 to +20 degrees. These randomizations were added in order to make the trained policies more robust, as they will learn to adapt to variations in the robot's poses during runtime.

Another important note was that the two robots shared the same joint reward function, meaning that both robots received the same reward in all cases. If one robot cleaned off a bit of the human's arm, both robots were rewarded equally. This shared reward function was hypothesized to encourage cooperation, and to see if any emergent behaviors would develop because of it.

After the environment had been constructed, it then became time to train three different policies, using the three different algorithms. These algorithms are Double PPO, Single PPO, and MADDPG:

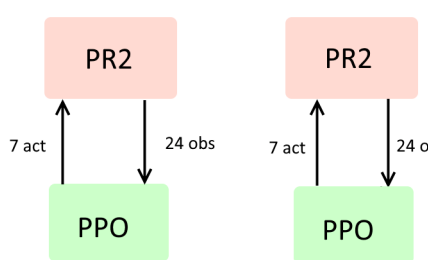


Figure 1: Double PPO

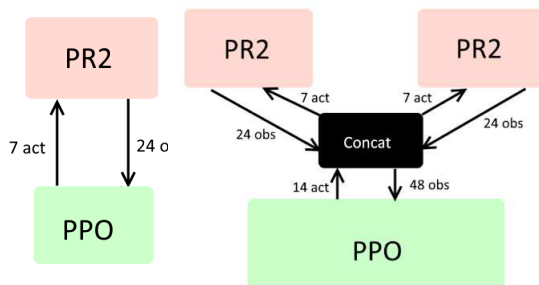


Figure 2: Single PPO

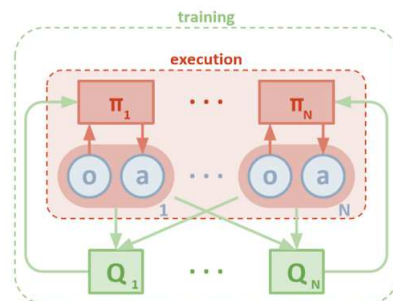


Figure 3: MADDPG, from [30]

In the diagram, one can see the basic layout of each algorithm. Double PPO, as the name would

suggest, involves training two PPO models, one for each robot. These robots have no explicit interaction beyond a shared reward function, so each instance of PPO is identical to the single-agent PPO version. Single PPO, on the other hand, involves both agents sharing the same single-agent PPO model. At each timestep, each agent’s observations are combined before input into the PPO model, and the PPO policy outputs the actions for both robots simultaneously. This effectively creates a single robotic ”brain” controlling both robots as if each robot were part of the same body. Even if this might be unrealistic in practice, this approach was chosen because it would be interesting to see if the vast increase in state space caused by joining the observations and actions of each robot would inhibit the ability of the model to generate an effective policy in a reasonable amount of time. Finally, the last diagram is of MADDPG, which was described in the literature review of this paper.

One important key differences between the three algorithms is the calculation of the Q values. Q values functions are functions which take in state information and actions, then output an estimate of the long-term expected reward of taking that action at a particular state. This Q value function is used to train the policy, which is a model that takes in observations from the robot and returns the optimal action to work towards completing the task. For Single PPO, there is a single Q function which takes in all the observations from both robots, and all the actions from both robots to calculate an estimate of expected reward. Again, this can be thought of as both robots having the same central ’brain’, which treats each robot as components of the same system. In mathematical notation, both agents share a central Q function, $Q(s_1, a_1 \dots s_i, a_i \dots s_n, a_n) \rightarrow \mathbb{R}$. For Double PPO, each robot i has its own Q function, Q_i and Q_i only takes in observations and actions from that particular robot: $Q_i(s_i, a_i) \rightarrow \mathbb{R}$. Each robot also has its own policy, which operate completely independently and with zero knowledge of what the other robot is doing. Below is a table summarizing the key differences between each algorithm.

	Single PPO	Double PPO	MADDPG
Q Inputs During Training	$Q(o_1, o_2, a_1, a_2)$	$Q_1(o_1, a_1)$ $Q_2(o_2, a_2)$	$Q_1(o_1, o_2, a_1, a_2)$ $Q_2(o_1, o_2, a_1, a_2)$
On/Off Policy	On-policy	On-policy	Off-policy, replay buffer
Robot Information during Execution	Omniscient robots	Robots act independently	Robots act independently

It is important to note is that the implementation chosen was not created by the original authors of the paper. This implementation was found on Github, linked [31]. It was later discovered that OpenAI has an implementation of MADDPG, which might provide an alternative MADDPG algorithm implementation should this research be replicated in the future. However, using a different implementation does illustrate some potential pitfalls of implementing multi-agent algorithms for robotics applications, so it's useful to examine for this research.

A copy of each learned policy was saved every 100,000 timesteps so that statistics could be generated later. Each iteration consisted of 200 timesteps, and one iteration was one attempt by the robot to solve the task. The environment was reset after every iteration. PPO hyperparameters were kept constant to the hyperparameters described in the first part of this paper.

For MADDPG, the the replay buffer consisted of 1000000 tuples, the batch-size was 1000, and the policy began training after 100 episodes. One very important note was that the implementation chosen for MADDPG appeared to contain a significant memory leak during training. After roughly one million timesteps, the model class would become over a gigabyte in size and performance would slow to a crawl. One attempted fix was to save the components of the MADDPG algorithm class during runtime and attempt to inject them back into a new instance of the MADDPG class to continue training the policy. However, this appeared to cause significant performance issues, as is documented later. Future research would use a different implementation of MADDPG to avoid these issues, most likely one released by OpenAI such as [32].

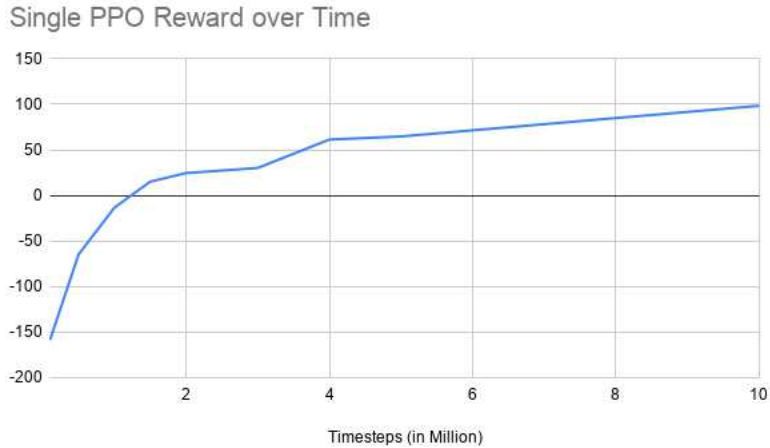
Finally, after a these policies had been trained, each policy was tested one-hundred times, and the resulting total rewards was recorded.

2.4 Results and Discussion

Before the overall effectiveness of each algorithm is explored, it is important to look at the computational performance of each algorithm. On an AWS EC2 instance consisting of 8 virtual CPUs, the Single PPO algorithm was the fastest, training at consistently at a slightly faster 72,283 timesteps per hour. Double PPO training ran only slightly slower than Single PPO at a consistent average rate of 71,827 timesteps per hour. Finally, MADDPG implementation ran the slowest, decreasing linearly from approximately 29,411 timesteps per hour to 17,391 timesteps per hour.

One reason that the performance for MADDPG might have been lower than PPO was that the MADDPG implementation was not parallelized between multiple environments, but the PPO algorithm was. In general, it was noted that many multi-agent learning algorithms such as MADDPG were not tested within a robotics context, but rather simplified game environments. For example, the MADDPG paper [30] tested their algorithm on the WaterWorld environment, as mentioned in the literature review. On the other hand, the OpenAI PPO implementation was developed with robotics in mind. Since robotics tasks are generally more computationally complex than gamified environments, the increases computational demands of multi-agent robotics means that less researchers will test with these more expensive systems as a result. This also amplifies any computational implementation errors such as memory leaks which might be missed during testing of the algorithm.

As far as the effectiveness of the training, first Single PPO will be examined. Below is a graph and chart of the overall mean reward after two-hundred timesteps for the single PPO policy after a certain number of training iterations:



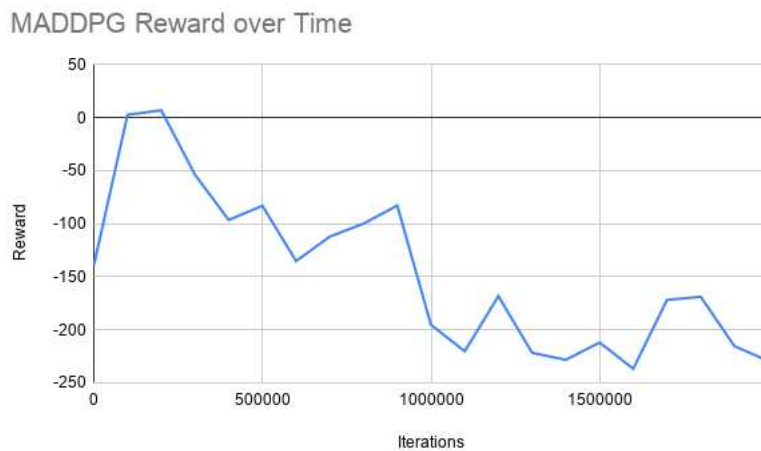
Single PPO starts at -158 average reward after 100,000 timesteps, but quickly increases to cross into a positive threshold after around 1.25 million timesteps. Afterwards, it continues to steadily increase, with the speed of increase only gradually decreasing with time until it reaches a final mean reward of 98.48 after ten million timesteps. These results might be hard to contextualize on their own, so next Double PPO will be analyzed for comparison. Below is the chart and line graph of the mean reward after two hundred timesteps for Double PPO, similar to the first algorithm:



Double PPO started at a slightly higher -118 mean reward after 100,000 timesteps, but very quickly crosses into a positive threshold at around 400,000 timesteps. This means that Double PPO was able to cross into a positive threshold roughly 3.125 faster than Single PPO. Afterwards, Double PPO continues to increase over time, but begins to level out after 5 million timesteps and 118 mean

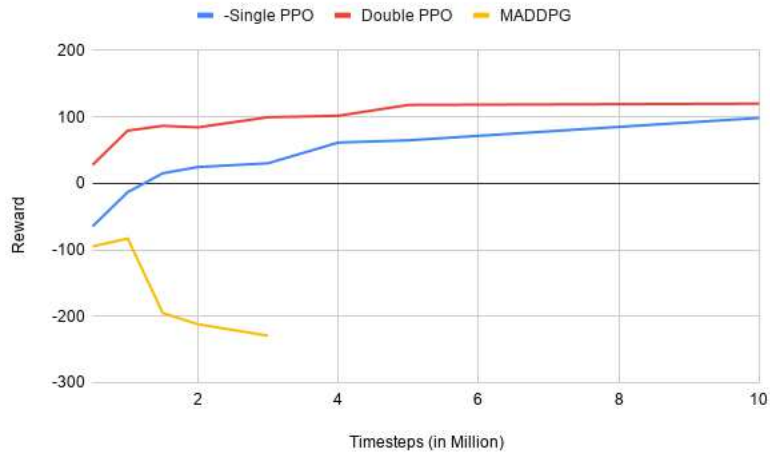
reward. The net reward achieved by Double PPO is overall higher than single PPO, although the rate of increase after 10 million timesteps is much lower than single PPO. This decrease in the speed of learning makes sense, as the state space of the single PPO algorithm is exponentially higher than Double PPO, so Double PPO should learn much faster.

Finally, it is time to examine MADDPG. Unfortunately, the implementation of MADDPG performed very poorly in comparison to Double and Single PPO. Below is the associated chart and graph:



Despite this algorithm outperforming the other two algorithms after 100,000 timesteps (-94 average), this performance actually began to get worse after 300,000 timesteps. After roughly 1 million timesteps, the MADDPG algorithm filesize became so bloated due to the memory leak that it required a restart. Unfortunately, after the restart and attempted re-injection of the model, performance dramatically decreased and never recovered for the rest of the training iterations. In addition, since the algorithm took significantly longer to run than PPO, only two million timesteps were able to be trained in the same time interval. That being said, it remains unclear whether training for more iterations would actually result in an improvement in performance or not. Overall, the MADDPG algorithm was the poorest running, and poorest performing algorithm of the three. However, it remains unclear whether this deficiency was a result of the implementation challenges or deficiencies of the MADDPG algorithm itself. Future research should test a different implementation to get a better gauge for the algorithm's performance overall. Below is a compounded line

graph where the three algorithms can be compared directly:



This chart simply reinforces the comparisons presented earlier. Double PPO was able to learn a quicker, more effective policy on average than single PPO, most likely due to its smaller state space. However, if one extrapolates the data to more than 10 million timesteps, it would appear that Single PPO could surpass Double PPO in mean reward eventually if the rate of increase does not dramatically slow down.

As far as the emergence of cooperative behaviors, it was noted that no significantly cooperative behavior developed with any algorithm. Neither robot was ever able to wipe down the entire arm in the given amount of timesteps, so it never became necessary for one to attempt to assist the other robot by reaching over the person. Perhaps this could be remedied by allowing more than two-hundred timesteps to complete the tasks in future experiments. In addition, it would appear that sharing observations between the robots did not significantly aid one robot's ability to complete their portion of the task, as independent robots were able to perform perfectly reasonably. To summarize, Double PPO performed the best short-term, Single PPO performed the second-best short term with the greatest potential for long-term success, and MADDPG performed the worst in all aspects.

2.5 Conclusion and Future Works

Overall, this research showed that for the Bed-bathing task of Assistive Gym, multi-agent assistance is a feasible alternative to single assistance, and does not require more advanced algorithms than slight modifications of single-agent Proximal Policy Optimization. It also illustrates that some of the implementations, especially those tested on gamified systems such as the implementation of MADDPG used, might require more testing in robotic contexts to ensure that the implementations are robust enough to withstand environments with greater computational resources required. Finally, it demonstrated the potential in using a singular single-agent deep reinforcement learning algorithm to complete multi-agent tasks in Assistive Gym, since the increase in state space does not appear to dramatically hinder the development of effective policies. In fact, given more time, this approach could even exceed each robot having their own policy in terms of overall effectiveness and mean reward.

There are a few clear future extensions of this research. The first and most obvious is to test a different implementation of MADDPG to identify if an alternate implementation would avoid some of the noted performance issues documented in this research. This would be useful for determining if the problems noted in this experiment in terms of performance were implementation specific versus algorithm specific. Next, alternative multi-agent reinforcement learning algorithms could be tested against the ones presented in this research to get a clearer picture in the effectiveness of multi-agent reinforcement learning algorithms for assistive robotics in general. Finally, it would be important to attempt to generate similar results to this experiment for the Dressing environment, since that environment poses much greater potential for robot cooperation than the Bed-bathing environment.

References

- [1] Organization WH. World Report on Aging and Health; 2015.

- [2] Office USCB. Methods for analysis of the financing and use of long-term services and support: Supplemental material for rising demand for long-term services and supports for elderly people; 2013.
- [3] Schulz RM, Eden JF. Families Caring for an Aging America; 2016. .
- [4] Milliken A, Mahoney EK, Mahoney KJ, Mignosa K, Rodriguez I, Cuchetti C, et al. "I'm just trying to cope for both of us": Challenges and supports of family caregivers in participant-directed programs. *Journal of gerontological social work*. 2019;62 2:149–171.
- [5] Zeng A, Song S, Lee J, Rodriguez A, Funkhouser TA. TossingBot: Learning to Throw Arbitrary Objects with Residual Physics. *CoRR*. 2019;abs/1903.11239. Available from: <http://arxiv.org/abs/1903.11239>.
- [6] OpenAI, Andrychowicz M, Baker B, Chociej M, Józefowicz R, McGrew B, et al. Learning Dexterous In-Hand Manipulation. *CoRR*. 2018;Available from: <http://arxiv.org/abs/1808.00177>.
- [7] Kalashnikov D, Irpan A, Pastor P, Ibarz J, Herzog A, Jang E, et al. Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation. In: *CoRL*; 2018. .
- [8] Mahmood AR, Korenkevych D, Vasan G, Ma W, Bergstra J. Benchmarking Reinforcement Learning Algorithms on Real-World Robots. In: *CoRL*; 2018. .
- [9] Yu W, Kumar VCV, Turk G, Liu C. Sim-to-Real Transfer for Biped Locomotion. *ArXiv*. 2019;abs/1903.01390.
- [10] Erickson Z, Gangaram V, Kapusta A, Liu CK, Kemp CC. Assistive Gym: A Physics Simulation Framework for Assistive Robotics. *arXiv preprint arXiv*:. 2019;.
- [11] Lee J, Dosovitskiy A, Bellicoso D, Tsounis V, Koltun V, Hutter M. Learning agile and dynamic motor skills for legged robots. *Science Robotics*. 2019;4.

- [12] Neunert M, Boaventura T, Buchli J. Why off-the-shelf physics simulators fail in evaluating feedback controller performance - a case study for quadrupedal robots; 2016.
- [13] Peng XB, Andrychowicz M, Zaremba W, Abbeel P. Sim-to-Real Transfer of Robotic Control with Dynamics Randomization. 2018 IEEE International Conference on Robotics and Automation (ICRA). 2017;p. 1–8.
- [14] Tan J, Zhang T, Coumans E, Iscen A, Bai Y, Hafner D, et al. Sim-to-Real: Learning Agile Locomotion For Quadruped Robots. ArXiv. 2018;abs/1804.10332.
- [15] Yan M, Frosio I, Tyree S, Kautz J. Sim-to-Real Transfer of Accurate Grasping with Eye-In-Hand Observations and Continuous Control. ArXiv. 2017;abs/1712.03303.
- [16] Vandesompele A, Urbain G, Mahmud H, wyffels F, Dambre J. Body Randomization Reduces the Sim-to-Real Gap for Compliant Quadruped Locomotion. *Frontiers in Neurobotics*. 2019;13:9. Available from: <https://www.frontiersin.org/article/10.3389/fnbot.2019.00009>.
- [17] Vuong QV, Vikram S, Su H, Gao S, Christensen HI. How to pick the domain randomization parameters for sim-to-real transfer of reinforcement learning policies? ArXiv. 2019;abs/1903.11774.
- [18] James S, Wohlhart P, Kalakrishnan M, Kalashnikov D, Irpan A, Ibarz J, et al. Sim-to-Real via Sim-to-Sim: Data-efficient Robotic Grasping via Randomized-to-Canonical Adaptation Networks. ArXiv. 2018;abs/1812.07252.
- [19] Fabisch A, Petzoldt C, Otto M, Kirchner F. A Survey of Behavior Learning Applications in Robotics - State of the Art and Perspectives. ArXiv. 2019;abs/1906.01868.

- [20] Erickson ZM, Collier M, Kapusta A, Kemp CC. Tracking Human Pose During Robot-Assisted Dressing Using Single-Axis Capacitive Proximity Sensing. *IEEE Robotics and Automation Letters*. 2017;3:2245–2252.
- [21] Erickson ZM, Clever HM, Gangaram V, Turk G, Liu CK, Kemp CC. Multidimensional Capacitive Sensing for Robot-Assisted Dressing and Bathing. 2019 IEEE 16th International Conference on Rehabilitation Robotics (ICORR). 2019;p. 224–231.
- [22] Buşoniu L, Babuška R, De Schutter B. In: Srinivasan D, Jain LC, editors. *Multi-agent Reinforcement Learning: An Overview*. Berlin, Heidelberg: Springer Berlin Heidelberg; 2010. p. 183–221. Available from: https://doi.org/10.1007/978-3-642-14435-6_7.
- [23] Busoniu L, Babuska R, De Schutter B. A Comprehensive Survey of Multiagent Reinforcement Learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*. 2008;38(2):156–172.
- [24] Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, et al.. *Playing Atari with Deep Reinforcement Learning*; 2013.
- [25] Stone MA. *On-Policy vs. Off-Policy Updates for Deep Reinforcement Learning*; 2016. .
- [26] Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. *Proximal Policy Optimization Algorithms*. *ArXiv*. 2017;abs/1707.06347.
- [27] Tan M. *Multi-Agent Reinforcement Learning: Independent versus Cooperative Agents*. In: *ICML*; 1993. .
- [28] Matignon L, Laurent G, Fort-Piat N. Independent reinforcement learners in cooperative Markov games: A survey regarding coordination problems. *The Knowledge Engineering Review*. 2012 03;27:1 – 31.
- [29] Foerster J, Nardelli N, Farquhar G, Afouras T, Torr PHS, Kohli P, et al.. *Stabilising Experience Replay for Deep Multi-Agent Reinforcement Learning*; 2018.

- [30] Lowe R, Wu Y, Tamar A, Harb J, Abbeel P, Mordatch I. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments; 2020.
- [31] xuehy. <https://github.com/xuehy/pytorch-maddpg>. 2018;.
- [32] OpenAI. <https://github.com/openai/maddpg>. 2018;.