Hit the Puck! - Teaching Robots to Play Air Hockey

Justin Yue[1] **An Thanh Dang**[1]
[1]UC-Irvine EECS-298

Abstract

In this project, we aim to utilize Reinforcement Learning to create a controller that allows a robot to play a game of air hockey. Our report details the progression of ideas and algorithms used to achieve the best current results on the problem. We initially experimented with classical policy gradient algorithms such as REINFORCE. Afterward, we found the need to progress towards Actor-Critic methods and eventually imitation learning procedures. We intend to discuss the role each algorithm plays in our finalized product, and its importance in other reinforcement learning-related tasks in robotics. Next, we talk about our experimental IK2PD model and future work to further this project.

1 Introduction

Problem The goal of this project is to train a robotic platform to perform complex tasks in simulation. The problem of deep reinforcement learning has been an increasingly popular field of study ever since the deep learning revolution. Deep Reinforcement Learning has found recent success in discrete platforms such as chess where the actions taken are discrete [2]. However, the problem becomes much harder for robotic platforms where the nature of the problem is not inherently discrete as in this survey [9]. There have been multiple ongoing studies in this field of applying reinforcement learning in robotics . Our approach seeks to shed light on this problem and greater insights as we achieve our goal of using reinforcement learning to control a robot's actions.

Software We will primarily use the Python programming language. This is due to its frequent use and support with the latest deep learning technology. Our deep learning framework of choice will be PyTorch because of its rich support with RL libraries. We will use the MushroomRL, Stable-Baselines, and OpenAI gym framework in order to implement all of the algorithms discussed in this report.

Policy Representation. In terms of formulation, we attempt to find θ to represent $p(a|s_t, \theta)$ where s_t is the current state/observation and θ is the parameters of a neural network. In most literature, $p(a|s_t, \theta)$ is referred to as the policy which is represented also as $\pi_\theta(a|s)$.

2 Dataset/Challenge

About Challenge. Reinforcement Learning is not inherently a supervised training process, so most Reinforcement Learning datasets come in the form of a simulation. The simulation that we will be using stems from the Air Hockey Challenge [12]. It is a single-robot arm whose end-effector is attached to an air hockey stick. There are multiple environments associated with our simulation. This is because the problem of reinforcement learning is vast and its chief problem is struggling to train a deep neural network to accomplish its tasks. The challenge consists of two environments. The first environment is hitting a still puck. The second environment is defending a moving puck.

Simulation Software. The challenge simulation builds itself on MuJoCo which is a popular multi-joint robotic simulator designed for robotic manipulation research [6]. Additionally, the challenge

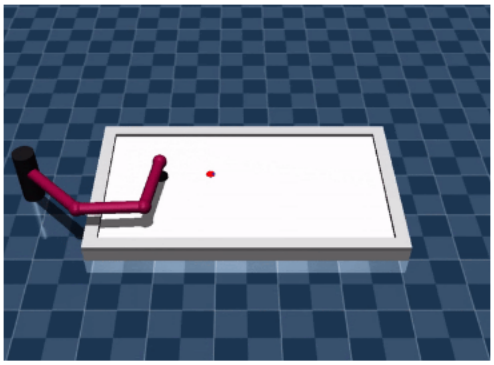

Figure 1: Showing robot simulation ran using our code with air hockey software.

focuses on reinforcement learning, so it builds software surounding the MuJoCo simulator in MushroomRL, a powerful reinforcement learning library, to enable easier integration of deep reinforcement learning algorithms for research [1]. Our objective is to train a network that controls a 3DoF planar robot provided by the challenge in order to accurately hit the puck based on the environments provided. We will effectively use one of the two simulation environments provided.

About Challenge Robot Our challenge robot utilizes a PD controller combined with a trajectory interpolator to send appropriate trajectories to the PD controller to follow. The equation is listed with τ_{cmd} being the command torque on the joints, q_d being the desired robot joint state, and q being the current measured robot joint state. This was published on the site, and the rest of the terms were left unspecified.

$$\tau_{cmd} = M(q)\ddot{q}_d + c(q, \dot{q}) + g(q) + K_p(q_d - q) + K_d(\dot{q}_d - \dot{q})$$

With the trajectory interpolator sending appropriate desired trajectories to the PD controller, our reinforcement learning algorithms are only allowed to interface with giving a desired final trajectory. The desired trajectory is interpolated and fed into the PD controller send a torque command to the robot motors. A limiter is used to help control the trajectory such that the robot will not go beyond its maximum positions or velocities. This is similar to what is used on industrial robots such as Universal Robots. Our observations will be our robot joint state in radians for each joint in its arm and the puck state in cartesian coordinates relative to the center of the air hockey table.

3 Timeline of Progress

Acknowledgement. Before delving into the rest of the reports, we want to acknowledge some differences between our original proposal and this report. For example. our earlier goals included experiments on deep Q-networks (DQN), evaluate our trained policy on the 3 degree-of-freedom planar robot and its potential improvement with a 7 degree-of-freedom robot, and use hierarchical reinforcement learning to choose when to perform hitting or defending against a puck. However, we no longer could evaluate the 7 degree of freedom robot as the challenge's organizers postponed its release, and in the case of DQN, we found that discrete actions are not suitable for learning the continuous actions required of this challenge. And due to challenges in teaching the robot how to learn actions, much of our report focuses on imitation learning.

4 Policy Gradient Search Methods

In our project, we believe in exploring everything from the ground up. With this in mind, we start off with Policy Gradient methods which are one of the first algorithms explored before the new development of Actor-Critic methods. This was something explored much earlier in the 1900s [13].

About Policy Gradient. For brevity, we omit the derivations of many of the algorithm formulas and choose to focus on equations that show the problem we solve using the algorithms in MushroomRL. Intuitively, we can see the idea of deep reinforcement learning to be synthesizing a data-driven

Week	Achievement
2	Setup software with Air Hockey Simulation
3	Trained REINFORCE algorithm with a simple feedforward network.
4	Moved to Actor-Critic Algorithms and trained DDPG and PPO on a feedforward actor and critic network.
5	Moved to imitation learning. Trained Behavior Cloning for supervised training on baseline agents.
6	Moved to the DAgger algorithm which performs better than Behavior Cloning.
7	Implemented GAIL training process. Warm-started GAIL with Behavior Cloning.
8	Trained Soft Actor-Critic independent of imitation learning
9	Trained Soft Actor-Critic with imitation learning (GAIL and Behavior Clone). Additionally implemented the IK2PD prototype idea.
10	Finish up the project and add figures.

Figure 2: Showing timeline of what we did in our project.

controller instead of using classical methods. With this perspective, our data is chosen to be the trajectories taken from our simulation. Our deep neural network will take in an observation vector and output an action vector. We see $\pi_\theta(a|s)$ to be the neural network we attempt to learn. In order to train the neural network, we find the maximum of a reward function we set for ourselves. Without detailing the reward function yet, we have the following:

$$J_\theta(\pi_\theta) = E_{\tau\sim\pi_\theta}[R(\tau)]$$

Where J is the cost function, R is the reward function, τ is a trajectory which is the state and action of the robot from the start to end of a simulation (normally called an episode).

The probability representing the trajectory is defined as follows, and it is reduced using the probability chain rule.

$$p(\tau|\theta) = p_0(s_0)\prod_{t=0}^{T} p(s_{t+1}, a_t|s_t)$$

$$p(\tau|\theta) = p_0(s_0)\prod_{t=0}^{T} p(s_{t+1}|s_t)p(a_t|s_t)$$

Upon taking the derivative of the log-probability of τ, we will get the following

$$\nabla_\theta p(\tau|\theta) = \sum_{t=0}^{T} \nabla_\theta log\pi_\theta(a_t, s_t)$$

Using this, we can take the gradient of the cost function and get the following:

$$\nabla_\theta J_\theta(\pi_\theta) = E_{\tau\sim\pi_\theta}[\sum_{t=0}^{T} \nabla_\theta log\pi_\theta(a_t|s_t)R(\tau)]]$$

With this equation, we can calculate the gradient of the neural network using backpropagation through τ scaled by the reward R. All of the policy gradient equations follow this format with slight variation. Additionally, we follow the gradient update rule.

$$\theta_{k+1} = \alpha\nabla_\theta J_{\pi_\theta} + \theta_k$$

Setup. The algorithms we decided to use was REINFORCE[16], GPOMDP[10], and eNAC[3]. There are black box methods we trained such as PGPE[3] and Constrained REP[8]S. All of these algorithms are implemented and integrated into our training system using MushroomRL. We train using the Adam Optimizer [5].

We train our deep networks using the following reward function. Remember τ represents $s_0, s_1, s_2...a_0, a_1, a_2$ where s is robot state and a is action. s should also have both robot joint state r_t and puck state o_t.

$$R(\tau) = \sum_{t=0}^{T} K(r_t)_x - o_{tx} + (K(r_{t+1}) - o_{(t+1)x}) + C(r_t)$$

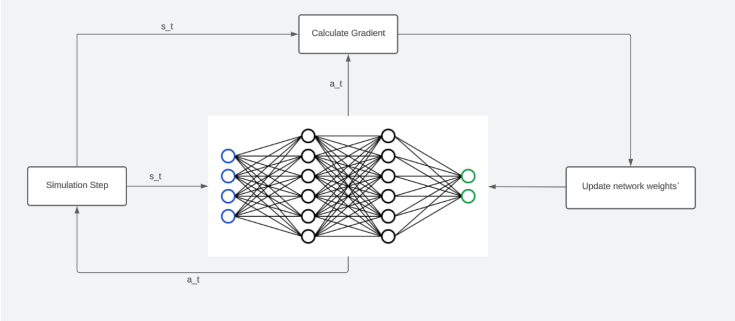

Figure 3: Showing our training pipeline for policy gradient. s_t is the state, and a_t is the action.

In this function, R is the reward function, K is the forward kinematics of the robot state (to be in the same coordinate as the puck), and C is the constraint function which is -9000 if violated, and 0 if not violated. Our C function is just a list of constraints that [12] lists. We define the x in the subscript to be taking only the "x" component of the vector described.

Results. From Figure 4, there is no noticeable trend in how each policy gradient search method is learning the hitting and defending actions. Relying more on visual observations of the training process, we found black box optimization methods to be more erratic than the robots that used white box optimization methods, which is plausible due to black box optimization methods having much less information e.g. no information to a loss function's gradients, so to achieve comparable performance would require more training episodes. Even then, the robots using white box optimization methods struggled to hit or defend against a puck as it was clear that its actions were random. More specifically, the robot arm tended to "snake" around the air hockey board.

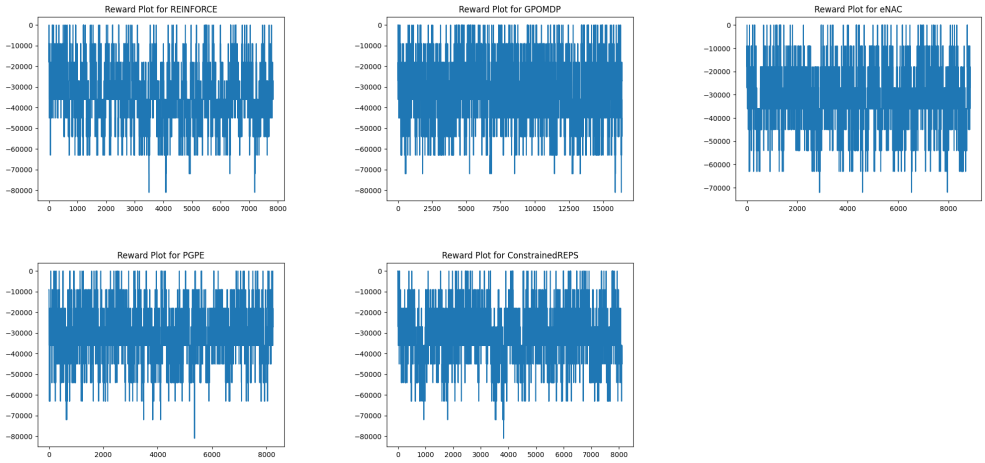

Figure 4: Reward Function Policy Gradient Plots - First 3 are white box optimization methods while the rest are black box optimization methods.

5 Actor Critic Policy

About Actor-Critic With the underfitting of our policy gradient framework, we saw the need to improve our training pipeline using newer technology developed recently in Reinforcement Learning. This led us to utilize Actor-Critic methods. Before Actor-Critic, we had our actor-only algorithms that were trained. Our policy gradient framework can be classified under actor-only. Critic-only a method that uses the Bellman equations and dynamic programming aspect of reinforcement learning. With continuous states, there is much difficulty in training critic only networks. However, the

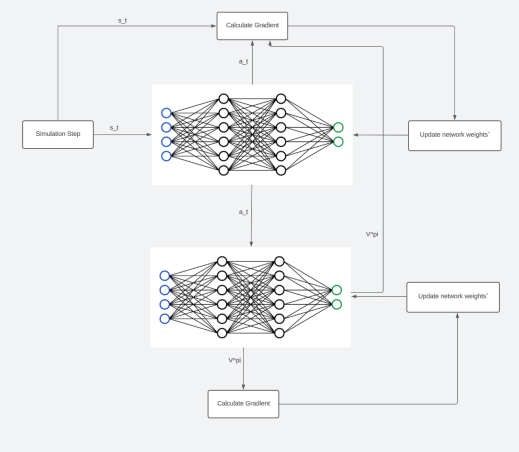

Figure 5: Showing our training pipeline for Actor-Critic supported by MushroomRL.

introduction of Actor-Critic networks changed the hierarchy of algorithms utilized which is the most used framework for robotic systems interested in using Reinforcement Learning [15].

The formulation is fairly quick to comprehend.

$$V^{\pi}(s) = E_{\tau \sim \pi_\theta}[R(\tau)|s_0 = s]$$

This V is the value function of the bellman equation dynamic programming iteration. We include this into our formulation as follows:

$$J_\theta(\pi_\theta) = E_{\tau \sim \pi_\theta}[\sum_{t=0}^{T}(R_{t+1} + \gamma V^{\pi_\theta}(s_{t+1}))log\pi_\theta(a_t|s_t)]]$$

In this case, we make sure to optimize not only θ but the parameters of the neural network for our critic network defined as ϕ.

$$\nabla_\theta J_\theta(\pi_\theta) = E_{\tau \sim \pi_\theta}[\sum_{t=0}^{T}(R_{t+1} + \gamma V_\phi(s_{t+1}) - V_\phi(s_t))\nabla_\theta log\pi_\theta(a_t|s_t)]]$$

$$J(\phi) = \frac{1}{2}(R_{t+1} + \gamma V_\phi(s_{t+1}) - V_\phi(s_t))^2$$
$$\nabla_\phi J(\phi) = R_{t+1} + \gamma \nabla_\phi V_\phi(s_{t+1}) - \nabla_\phi V_\phi(s_t)$$

In many of the algorithms we use that go under actor-critic, there is slight engineering and variation on the calculation of the $J(\phi)$ and $J(\theta)$.

Results. Using the same reward function formulated as before, we run our experiments. With the training pipeline setup, we run our experiment on DDPG and PPO. We evaluate the results and found that our results were very similar to the Policy Gradient methods. This means that it was not the complexity of our architecture and training process that was the problem. We saw through the same reward plots and visualization that our networks had the same behavior. This is most likely due to the initialization of the network weights which led us to think about imitation learning.

6 Imitation Learning

Idea. With all of our troubles in training a neural network with our reinforcement learning pipeline, we decided to explore other avenues. The motivation behind imitation learning was that we could not find a good design of the reward function to single-handedly train a neural network. We concluded that the rewards that needed to be desigend would be too complex for us to model easily in a function. One way of looking at this is seeing how optimal control algorithms like model predictive control work. Those algorithms take in an observation and run an NP-Hard algorithm (optimization solving)

in order to output the best action. There is no reward that could possibly formulate the complexity of understanding how the algorithms did that. Instead, we opted to use supervised training by collecting a dataset of the baseline agent provided by the air hockey challenge creators in order to allow our neural network to learn these complex reward features. The idea is to transfer learn these details back into our regular training pipeline.

Software We utilized another library called Imitation which bases itself on the stable-baselines RL framework. In order to work between multiple software libraries, a lot of effort was put into writing software interfaces between each RL framework and library.

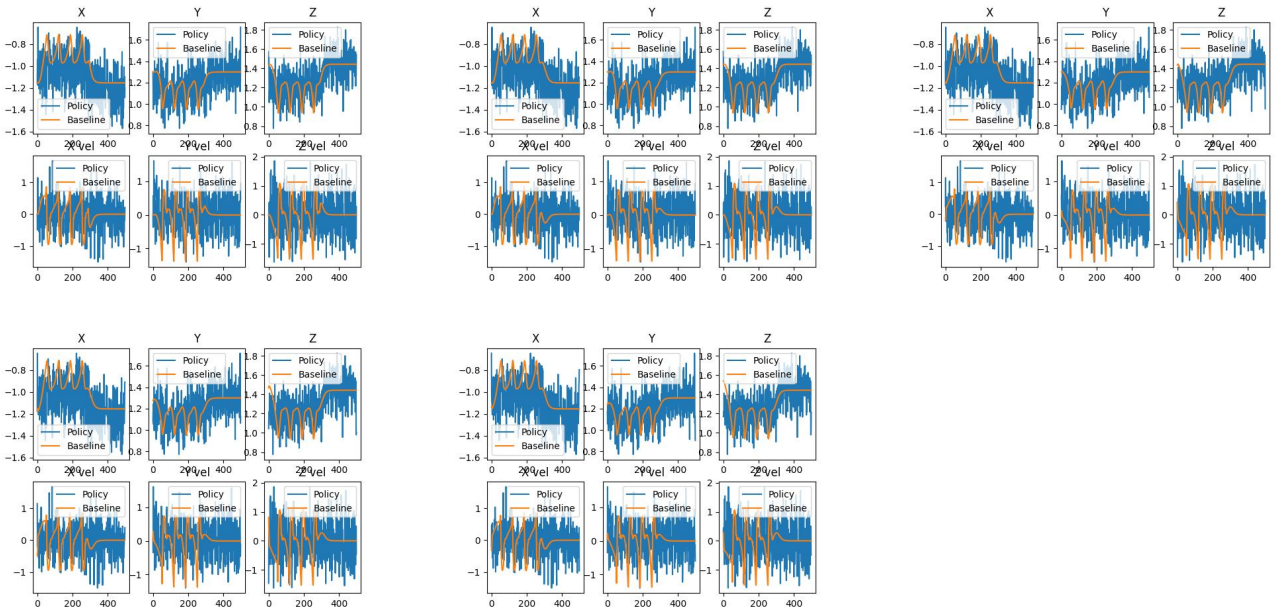

Figure 6: Plot of policy's actions(blue) vs baseline agent's actions (orange) across 5 episodes. This policy is trained using DAgger with regularization and removal of the robot pose.

6.1 Behavior Cloning (BC) and DAgger

Behavior Clone. Behavior cloning[7] is a simple supervised learning process that collects data of (s_t, a_t) state action pairs and runs supervised training by enforcing π_θ to follow the state action pairs. Behavior cloning did not work that well which is expected. This is because Behavior Cloning's supervised training process assumes that the datapoints are all independent and identically distributed (iid) which is used for most supervised processes in deep learning. This is not true for RL state action pairs as they are part of a markov decision process. This means for each trajectory, each state depends on the previous state.

DAgger. DAgger[14] is a more advanced type of imitation learning that combats this problem. Like behavior cloning, it iteratively trains a policy on a dataset from the expert . But it also runs the currently training policy to gather observations, query the baseline agent for good actions, and appends these new actions to the dataset. This distinguishes DAgger as an on-policy, where the policy being trained may take the wrong action but will be corrected by the baseline agent. Additionally, DAgger improves upon behavior cloning by aggregating data that is more likely to be seen by the trained policy in the simulation; this also requires querying the expert online.

DAgger had similar performance compared to behavior cloning at first, but then we implemented regularization and excluded the robot pose from states s_t because we speculated that the robot's position might be noisy to the policy in learning to hit the puck's position. As shown in Figure 6, regularization in conjunction with removing the robot position from the observation does improve our policies' performance.

The baseline agent is shown to reach a steady state at some point during each episode whereas without these changes, it continues to keep retracting, then attempting to hit the puck to the end of the episode. This suggests that the policy π_θ has learned the hitting and defend actions better than before since the baseline agent believes it is done.

6.2 Generative Adversarial Imitation Learning (GAIL)

Even with DAgger, our performance was still not satisfactory. We decided to utilize generative adversarial networks for imitation learning [11]. The training process involves the usage of a generator and dscriminator. The generator is our policy. The discriminator tries to determine whether our policy is fake or the original baseline. This training process is a primal-dual optimization procedure that resembles a minimax game. Our policy will fight with the discriminator to eventually improve itself to be indistinguishable from the original baseline agent.

Optimization Note Unfortunately like GANs, GAIL also suffers from the phenomena known as mode collapse; the generator chooses an action(s) that tricks the discriminator reliably enough that the generator learns to always choose those actions. Of course, this tactic does not generalize well to different types of scenarios. To mitigate mode collapse, we opted to use DAgger to warm start GAIL's training process first.

Results. The result of GAIL was much better than DAgger. It seemed that the noisiness of the policies actions over time is much smaller compared to the actions of the DAgger trained network. While the results are definitely better, it still cannot accomplish the task of getting a robotic arm to hit the puck as we would want. Although it moved somewhat like our baseline agent, we can assume that it has learned the complex reward features and we can perform transfer learning now with regular Reinforcement Learning.

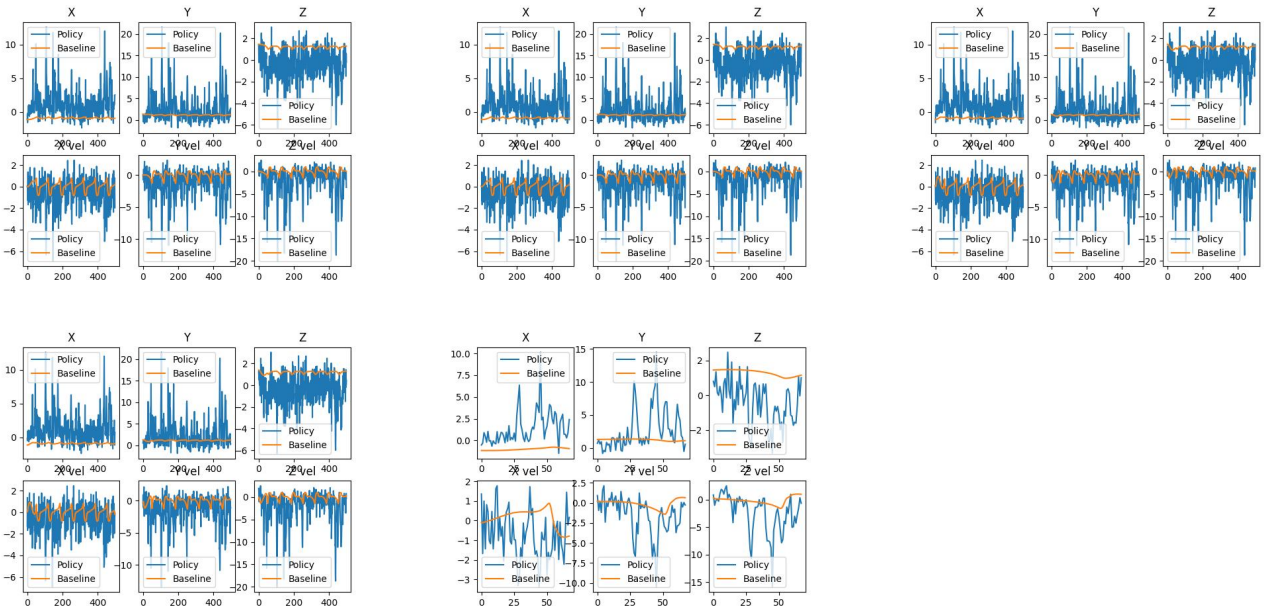

Figure 7: Plot of policy's actions(blue) vs baseline agent's actions (orange) across 5 episodes. This policy is trained using GAIL.

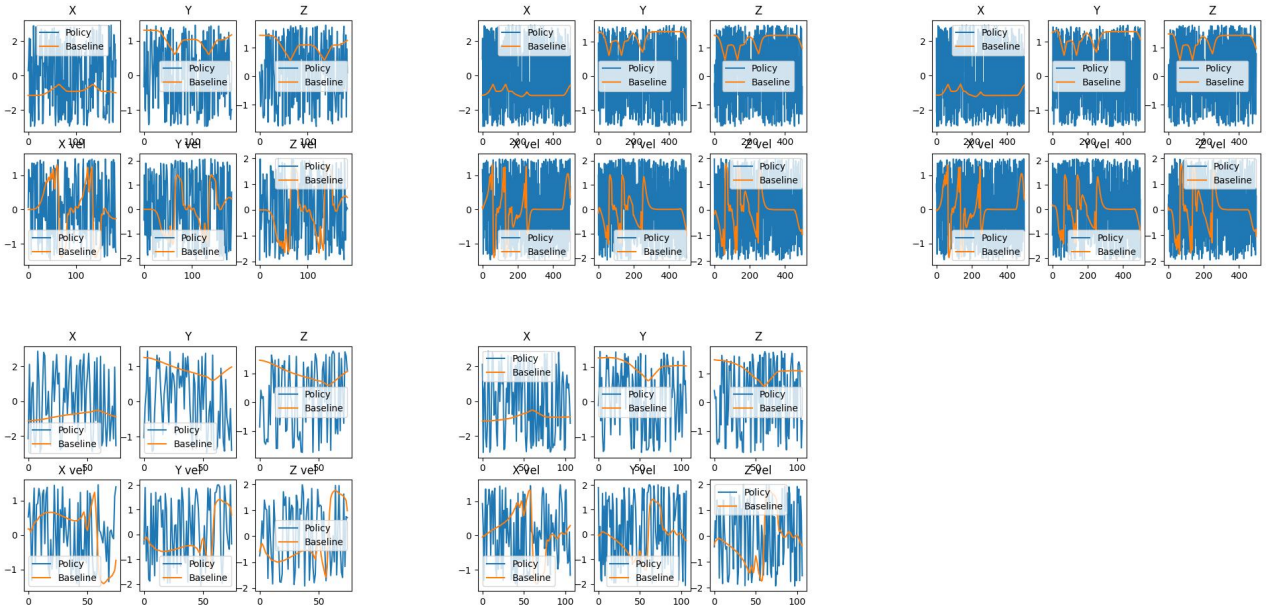

Figure 8: Plot of policy's actions(blue) vs baseline agent's actions (orange) across 5 episodes. This policy is trained using SAC over 1000 episodes.

Figure 9: Showing complete learning pipeline by the end of our project.

7 Soft Actor Critic Policy (SAC)

Idea Our goal is to eventually take the parameters learned during one of the imitation learning methods and build off that training. Since our earlier training with actor critic policies wasn't impressive, we also opted to use an improved version called Soft Actor Critic policies (SAC). SAC policies differ from regular ones by trying to compute an optimal policy that maximizes long-term expected reward and the entropy of the policy. This idea encourages the policy to explore states not seen before and learn them.

Results By itself, we trained SAC over 1000 to 5000 episodes, but we found that training SAC less episodes resulted in better performance. More episodes instead caused an overfitting effect where the robot arm would only reach out stiffly instead of learning actions akin to the baseline agent.

Finally, we attempted transfer learning where we trained 2 actor critic policies with DAgger and GAIL respectively before and reloaded their weights back into a SAC training pipeline. Unfortunately, the results were not good since in both training processes, the robot arm would venture outside the air hockey table. The current explanation for this result is that despite our actor critic policies were trained well by DAgger and GAIL, SAC may have forced the policies to explore the states outside of the environment's constraints.

8 Inverse Kinematics to PD Control (IK2PD)

Idea As mentioned earlier in this report, agents output $2x3$ arrays that specifies the intended x, y, and z positions and their respective velocities. However, these are relative to the robot and not in

world coordinates; meanwhile, the puck that the robot is trying to hit is in the world coordinates. Consequently, our training process involves trying to drawing an action relative to the robot that is somehow equivalent to the puck's position in world coordinates.

This type of problem is NP-hard that our policies are trying to learn and can be circumvented with the usage of kinematics. While drawing actions from the expert policy during imitation learning, we can convert its positions to be in world space instead, so the policy will learn which world position to hit the puck. Then, when evaluating the trained policy, we can run an inverse kinematics to make sure the trained policy's output is relative to the robot arm, which is what the air hockey environment expects, and the transformed output will guaranteed to be equivalent to the puck's position in world coordinate space. This is the core idea behind IK2PD[4].

Results We use the challenge's provided forward and inverse kinematic functions. On the plots, the robot arm movement's oscillations can still be observed, and there doesn't exist a noticeable improvement in how DAgger approximates the baseline agent's actions. When watching the robot arm in action, there does seem to be a larger overfitting effect compared to regular DAgger training; specifically, the policy π_θ tends to aim only towards the middle of the air hockey table. Future work would include increasing the number of minimum episodes per rollout as this would introduce the policy to more novel states that it could learn.

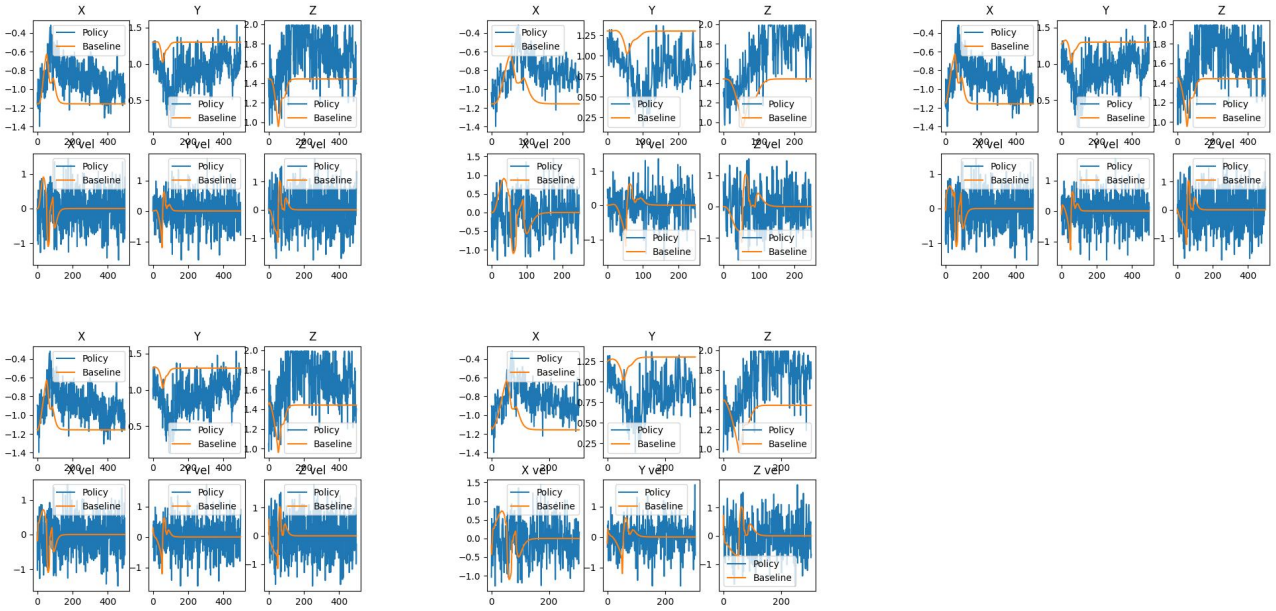

Figure 10: Plot of policy's actions(orange) vs baseline agent's actions (blue) across 5 episodes. This policy is trained using DAgger but also incorporates IK2PD into the training.

9 Conclusion

Overall, we learned much about reinforcement learning concepts. The experience we had showed the difference between simpler tasks such as training an AI to play chess versus teaching a robot arm to play air hockey. From the experiments we ran, we observed how deep reinforcement learning methods improve over the classical ones and also found why reinforcement learning is a difficult field. It requires a solid background and intuition of many topics that include classical control theory, statistics, machine learning, and so on.

10 Future Works

Moving forward, we would like to revise our reward function to punish violations more vigorously while encouraging good behavior. We still believe that SAC and IK2PD are viable solutions to improve our robot arm's performance and will continue to work on them. To improve our SAC pipeline, one idea is to train in increments to determine when the policy begins exploring states that violate its constraints; this would help provide insight on how to improve the reward function for training. Future work for IK2PD would include increasing the number of minimum episodes per rollout as this would introduce the policy to more novel states that it could learn.

References

[1] Andrea Bonarini Marcello Restelli Jan Peters Carlo D'Eramo, Davide Tateo. Mushroomrl: Simplifying reinforcement learning research. In *In proceedings with the Journal of Machine Learning Research 2022*, 2020.

[2] Julian Schrittwieser Ioannis Antonoglou Matthew Lai Arthur Guez Marc Lanctot David Silver, Thomas Hubert. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. In *Jounal of Science*, 2018.

[3] Peters J. 2013. Deisenroth M. P., Neumann G. A survey of policy search on robotics. 2013.

[4] Novalio Daratha Ruvita Faurina Agus Suandi Sulistyaningsih Denavit Hartenberg, Indra Agustian. Robot manipulator control with inverse kinematics pd-pseudoinverse jacobian and forward kinematics. In *In proceedings with the Jurnal Elektroniksa dan Telekomunikasi*, 2021.

[5] Jimmy Lei Ba Diedrek P. Kingma. Adam: A method for stochastic optimization. In *In proceedings with the International Conference on Learning Representations*, 2015.

[6] Yuval Tassa Emanual Todorov, Tom Erez. Mujoco: A physics engine for model-based control. In *In proceedings with IEEE/RSG International Conference on Intelligent Robots and Systems*, 2012.

[7] Peter Stone Faraz Torabi, Garrett Warnell. Behavior cloning from observation. In *In proceeds with the International Joint Conference on Arficial Intelligence*, 2018.

[8] Yasemin Altun Jan Peters, Katharina Mulling. Crelative policy entropy search. In *In proceedings with the Twenty-Fourth AAAI Conference on Artificial Intelligence*.

[9] Jans Peter Jens Kober, J. Andrew Bagnell. Reinforcement learning in robotics: A survey. In *International Journal of Robotics Research*, 2013.

[10] Peter L. Bartlett Jonathan Baxter. Infinite horizon policy-gradient estimation. In *In proceedings with the Journal Of Artificial Intelligence Research, Volume 15*, 2001.

[11] Stefano Ermon Jonathan Ho. Generative adversarial imitation learning. 2016.

[12] Jonas Gunster Dong Chen Ziyuan Liu Haitham Bou Ammar Davide Tateo Jan Peters Puze Liu, Niklas Funk. Robot air hockey challenge 2023. `https://air-hockey-challenge.robot-learning.net/home`, 2023.

[13] Satinder Singh Yishay Mansour Richard Sutton, David McAllester. Policy gradient methods for reinforcement learning with function approximation. In *In proceedings with the Conference of Neural Information Processing Systems*, 1999.

[14] J. Andrew Bagnell Stephane Ross, Geoffrey J. Gordon. A reduction of imitation learning and structured prediction to no-regret online learning. In *In proceedings with the Internal Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.

[15] John n. Tsitsiklis Vijay R. Konda. Actor-critic algorithms. In *In proceedings with the Conference of Neural Information Processing Systems*, 1999.

[16] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *In proceedings with The Springer International Series in Engineering and Computer Science*, 1990.