Sim-to-real transfer of Reinforcement Learning policies for Tennis Wheelchair Robot

MS Robotics, Capstone Project

Senthil Kumar, Surya Prakash
MS Robotics, School of Interactive Computing
Georgia Institute of Technology
Atlanta, Georgia

Gombolay, Matthew
Assistant Professor, School of Interactive Computing
Georgia Institute of Technology
Atlanta, Georgia

***Abstract*—The field of robotics has witnessed a remarkable growth in the past decade and this progress can be largely attributed to the advancements in Reinforcement Learning (RL) based algorithms for planning and control. ESTHER (Experimental Sport Tennis Wheelchair Robot), is designed to play the game of tennis, which by nature requires precise trajectory estimation, shot selection, and dynamic control of racket swings. However, a significant challenge lies in developing control policies to test ESTHER in different environments for the tasks such as in-court navigation and executing strokes for ball interception. Therefore, the goal of this project is two-fold; (i) design a high-fidelity simulator to accurately capture the dynamics of the robot and train it on three diverse sets of tasks using reinforcement learning (RL), and (ii) perform sim-to-real transfer of the learned policies onto the physical robot. The code for this project is available here.**

***Index Terms*—Reinforcement Learning, Control, Tennis, Simulator**

I. Introduction

ESTHER [7] utilizes commercially available off-the-shelf hardware and is designed with portability in mind. The setup consists of a 7-DOF Barret WAM serial arm equipped with a tennis racket, which is mounted on a regulation Top End Pro Tennis Wheelchair. This research highlights several future challenges, including learning strokes from expert demonstration (LfD), integrating game knowledge to strategically return shots, and enabling effective human-robot collaboration for doubles play. Designing control policies to navigate the tennis court effectively and plan strokes appropriately is a complex task, which is where reinforcement learning (RL) comes into play.

A typical RL 2 agent interacts with its environment to learn a specific task by iterative adjustment of its behaviour policy based on the rewards obtained. In spite of their success in various Atari-environments [8], they require millions of episodes of training which can be tedious and time-consuming. Real world extension of this training paradigm can pose risks as the learning process may result in unintended trajectories that could potentially damage the surroundings and the robot.

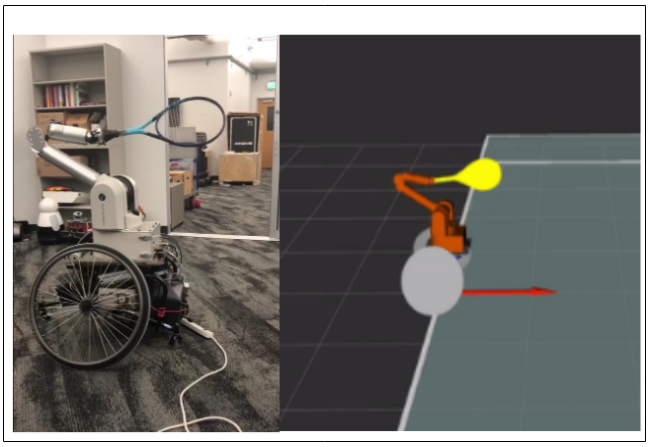

Fig. 1. Left: Wheelchair Tennis Robot (ESTHER). Right: Visualisation in RViZ

Therefore, our key contributions through this project can be summarized as follows:

- Develop a fully functioning physics simulator for safe & efficient design of control policies using RL based on Soft Actor-Critic (SAC) [16].
- Integrate the learned policy on the ESTHER and validate performance.

II. Related Work

A. Physics Simulators

Several simulators with high-level physics engines and realistic assets have been developed over the years. Google research [1] developed a reinforcement learning environment tailored for training agents to play football. This environment employs an advanced, physics-based 3D simulator, offering a realistic representation of the game dynamics. Robot Air Hockey [2] is a simulation environment that provides a platform for building air hockey agents in a competitive setting against each other in an entire game with support for both simulation and sim-to-real transfer of policies on the real robot. Iterative policy improvement techniques like [3] continuously improve the initial human player policy by cycling alternately

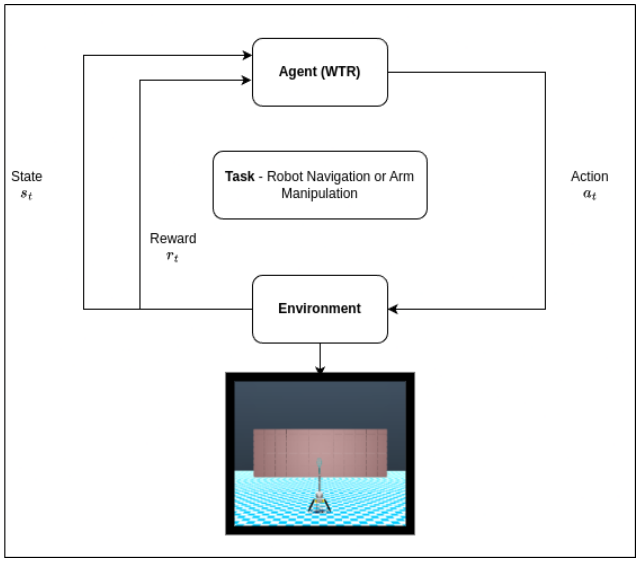

Fig. 2. RL setup

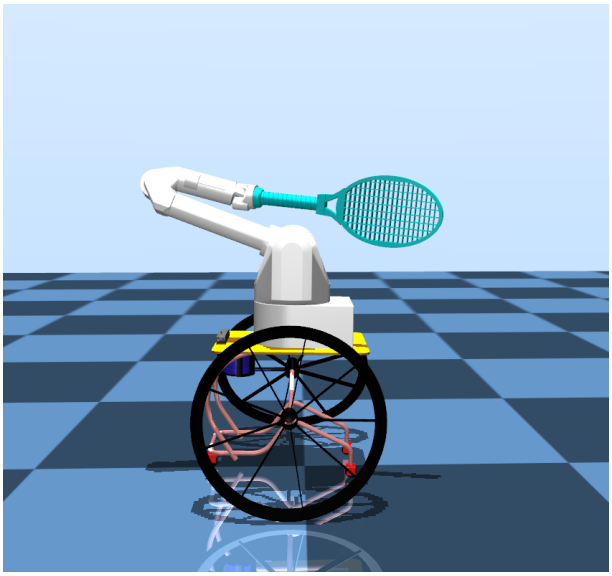

Fig. 3. Replica of ESTHER in MuJoCo

between the real world and simulation where in each iteration, both the human behavior model and the policy are refined.

B. Reinforcement Learning in Robotics

An imitation learning framework for robotic table-tennis system proposed by Ding et al [5] iteratively improves a network by training it on a goal-conditioned behavior aided by a self-supervised correction. Zhang et al [6] developed a system that teaches different tennis skills through the demonstrations of tennis play from broadcast videos in a hierarchical learning approach that combines low-level imitation with a high-level motion planning policy. The controllers produced by integrating these two policies, are capable of simulating character models engaged in a rally with distinct range of shots and other tennis skills. Shixiang Gu et al. [9] have demonstrated reductions in training durations by asynchronously parallelizing the RL algorithm across multiple robots on simulation, which eventually pool their policy updates. Jonas et.al. [10] have developed a sample-efficient, DDPG based RL algorithm for a KUKA Agilus Robotic arm, playing table tennis. Pneumatically actuated muscular robots were trained end-end to play table tennis using an Hybrid Sim and Real (HYSR) training procedure as proposed by Dieter et.al [11].

The remainder of this report is divided as follows: In Section III, we introduce the choice of simulator used and the governing design elements. In Section IV and V, design of different Gym [13] compatible RL environment will be described in detail. In the sections VI and VII, we will describe the results obtained and hint at the potential future directions with this project.

III. Simulator Design

We chose MuJoCo (Multi-Joint dynamics with Contact) as our physics engine to build our simulator and test different RL policies [4]. We designed our model by closely adhering to the kinematic and dynamic constraints of the physical robot, as shown in Fig. 3.

Position controllers were implemented to manipulate the arm, while *velocity controllers* were used for navigating the wheelchair to maintain consistency with the framework introduced in [7]. To ensure compatibility with the forward kinematic engine, a PD controller was utilized to convert position & velocity signals into torque input, with gain values determined based on the maximum linear and angular velocities of the arm joints and the wheelchair. STL files for different robot meshes were generated through Onshape, and a model tree was built as an XML.

Further details on the tuning of gain values for the PD controller and model design can be found at the code repository.

IV. Environment Design

To test the functionalities and stability of the simulator, the model was tested on four different tasks with the objective of minimizing the positional error with respect to a goal. Currently, we provide complete support (position/velocity control and torque control), for two environments which can be found in the repository mentioned above and these are designed to independently test the ESTHER on 2D navigation and 3D reaching tasks.

A. Navigation Task

1) Problem Description: In this experiment, the objective is to teach the ESTHER to *navigate* to an arbitrarily sampled goal position in a constrained 2D plane while respecting the controller dynamics. Fig. 6 shows the environment, which is a 16 x 16 space consisting of the robot and a goal position spawned at random at the end of every episode. Episodes will be terminated if the robot goes out of bounds or if the

total time steps have elapsed.

2) Formulation as a RL problem: Following a typical RL convention, we define our problem as a Markov Decision Process (MDP) with the states, actions, rewards and the transition probability functions represented as a tuple (S, A, p, r). The robot at each step receives a reward r_t and a new state s_t based on the current and the past actions in the environment. The state, $s_t \subseteq S \in \mathbb{R}^9$ can be defined as, $s_t = \{d_t^g, \Delta\theta_t, \phi_t^b, v_t^b, \omega_t^b\}$, where d_t^g is the 2D euclidean distance between the robot base and the goal, $\Delta\theta_t$ is the relative orientation between the robot and the goal, ϕ_t^b is the global yaw, $v_t^b \in \mathbb{R}^3$ & $\omega_t^b \in \mathbb{R}^3$ are the linear and angular velocities of the robot base at time t. We relax the motion planning constraints as suggested by [14] thereby allowing the robot to reach the goal in both forward and backward configurations. For our experiments, we used the velocity control and hence the action space A would comprise the change in angular velocity required $(\Delta\omega_1, \Delta\omega_2)$ required to navigate the robot.

3) Kinematic Constraints: Respecting the limitations of the real robot, these constraints are inculcated into our calculations so as to maintain consistency for sim-to-real transfer. The action space A is confined between $[-1, 1]$ and the resultant velocity is clamped based on the following constraints.

$$|\Delta v_t^b| \leqslant a_{max} \cdot dt, \text{ and } |\Delta\omega_t^b| \leqslant \alpha_{max} \cdot dt,$$

4) Reward Shaping: To encourage motion towards the goal, we define a dense reward function d_t^{rew} which offers rewards based on how close the robot is to its destination and penalises if it moves away from the goal in subsequent time steps. We also provide a control cost, $ctrl_t^{rew}$ to restrict drastic change in the base velocities. The agent receives a final reward $term^{rew}$ if it reaches the goal within the stipulated number of time steps t_{max}, and if the episode is not terminated prematurely due to violation of environment constraints. The following definitions are,

$$term^{rew} = \begin{cases} 120, & \text{if } d_t^g < \epsilon \text{ and } t < t_{max} \\ -100 & \text{if } d_t^g > \epsilon \text{ and } t >= t_{max} \\ 0, & \text{otherwise} \end{cases}$$

$$d_t^{rew} = c_{dist} \cdot (d_t^g - d_{t-1}^g),$$

$$ctrl_t^{rew} = c_{ctrl} \cdot (||\Delta v_t^b|| + ||\Delta\omega_t^b||),$$

$$r_t = d_t^{rew} - ctrl_t^{rew} + term^{rew}$$

5) Setup: For our experiments, we set linear acceleration a_{max} and angular acceleration α_{max} as 2.5 m/s^2 and 1 rad/s^2 respectively with the interval between each time step as 8 ms. The robot was deemed successful if it remained within a vicinity of 0.5 m, denoted by ϵ, around the goal. The other scaling parameters, c_{dist} and c_{ctrl} were respectively fixed as 0.01 & 500 with each episode lasting for a total of 1000 time steps.

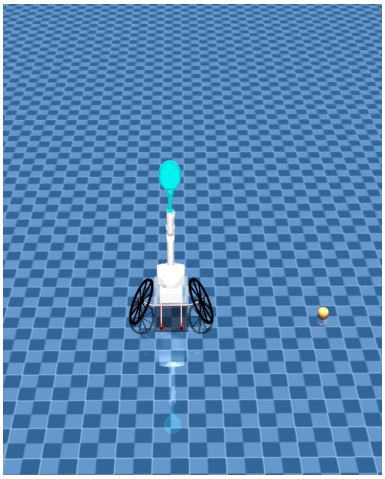

Fig. 4. ESTHER setup for the task of navigation and reach

B. Reaching Task

1) Problem Description: To test the capabilities of the arm, we define our next task where the objective is to enable the WAM arm on the ESTHER to *reach* an arbitrarily placed 3D goal position within the robot's work envelope without navigating the robot. Ideally, the policy learned should reflect the inverse kinematic mapping from joint accelerations to joint torques. Similar to the previous task, episodes will be terminated if the time steps have elapsed.

2) Formulation as a RL problem: Analogous to the previous task, we define our problem as an MDP. For this task, the state, $s_t \subseteq S \in \mathbb{R}^{27}$ can be defined as, $s_t = \{\theta_t^{1:7}, \dot{\theta}_t^{1:7}, \ddot{\theta}_t^{1:7}, e_t, x_t^g\}$, where $\theta_t^{1:7}$ denote the joint angles in generalised coordinates, and their time derivatives represented by $\dot{\theta}_t^{1:7}$ & $\ddot{\theta}_t^{1:7}$. The last two elements in the state vector denote the 3D coordinates of the end effector and the goal position in the world frame. Position controllers were used in this task and hence the action space A represent the change in the joint positions $\Delta\theta_t^{1:7}$ to manipulate the robot.

3) Kinematic Constraints: The constraints on the joints were applied on the robot in the simulation and like the previous task, the action space A is confined between $[-1, 1]$ and then rescaled before applying the signal to the actuators. The arm attempts to reach a point randomly chosen within its work envelope, which resembles a hemisphere, following the convention of spherical coordinates.

4) Reward Shaping: We employed a similar dense reward function d_t^{rew} to encourage the arm to reach the 3D target while respecting its motion constraints. A control cost $ctrl_t^{rew}$ is provided to discourage drastic changes in the joint

velocities. The agent receives a terminal reward $term^{rew}$ if it reaches the goal within the stipulated number of time steps. The definition for the $term^{rew}$ and d_t^{rew} is same as the navigation task with the exception that d_t^g here denotes the 3D euclidean distance between the goal and the end-effector. The control cost is defined as,

$$ctrl_t^{rew} = c_{ctrl} \cdot \sum_{i=0}^{7} \Delta\dot{\theta}_t^i,$$

where $\Delta\theta$ is the change in joint velocities between between subsequent steps.

$$r_t = d_t^{rew} - ctrl_t^{rew} + term^{rew}$$

5) Setup: In this task , the interval between each time step was still fixed as 8 ms. Success is determined if the arm reaches the goal within a vicinity of 8 mm, denoted by ϵ. The other scaling parameters, similar to earlier task, c_{dist} and c_{ctrl} were respectively fixed as 0.01 & 500 with each episode lasting for a total of 1000 time steps.

C. Stroke Task

1) Problem Description: For our last experiment, we aimed to teach the robot how to execute a basic tennis swing, where the ball is projected from an arbitrary position on a solid wall with the base being static. To make things easier, our initial goal was for the robot to make contact with the ball by understanding its movement dynamics, eventually extending the task to perform a stroke. Episodes will be terminated if the robot fails to execute the swing, i.e if the ball goes beyond the robot arm or if the total number of time steps have elapsed.

2) Formulation as a RL problem: Learning the ball trajectory for performing a tennis stroke requires a lot of trials and hence longer training before convergence. Studies from [10] and [11] have pointed out the effectiveness of feeding the 3D ball coordinates at only at the point of interception by extrapolating the trajectory from the point of launch using kinematics. We employed this strategy to estimate the point of interception by solving the following differential equation.

$$\dot{v} = -k_D||v|| + k_M\omega \times v - g, \tag{1}$$

where v is the velocity of the ball along the direction of projection (+x in the simulator), k_D is the coefficient of drag, k_M is the Magnus coefficient and g is the acceleration due to gravity.

The state vector in this case, $s_t \subseteq S \in \mathbb{R}^{20}$ can be defined as, $s_t = \{b_t, \dot{b}_t, r_t, \dot{r}_t\}$, where $b_t, \dot{b}_t$ is the 3D ball position and velocity at the point of interception, $r_t, \dot{r}_t$ represents the position and velocity of the robot arm at time t in generalised coordinates. For this experiments, we worked with a torque controller and hence the action space would comprise joint torques to actuate the arm to touch the ball. Therefore, an action sampled at time step t would be given by, $\tau_t^{1:7}$.

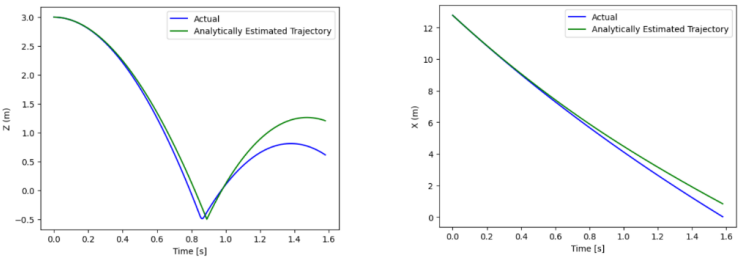

Fig. 5. Errors in position estimates along z and x direction

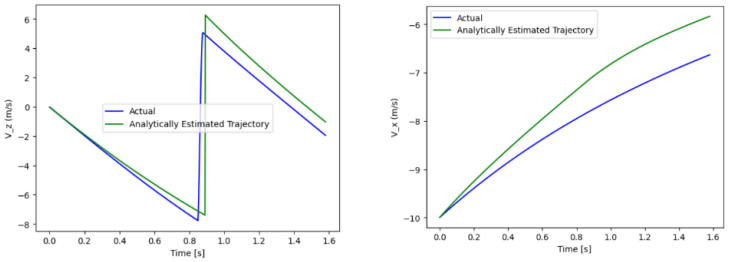

Fig. 6. Errors in velocity estimates along z and x direction

3) Analytical Ball Trajectory Prediction: To estimate the trajectory of the ball, we utilized the fourth-order Runge-Kutta method to solve equation 1 over a 4-second window with a step size of 2 meters. This enabled us to determine the ball's position at the interception point. The interception point in our case is defined as the highest point reached by the ball in its trajectory following its initial bounce. We simplified the bounce model by reversing the velocity vector along the z-direction and keeping the coefficient of restitution of the ball constant at 0.8. The drag coefficient k_D was fixed as 0.04 through experiments, and k_M was set to 0 for the sake of simplicity. We assessed the accuracy of our estimation across various projection velocities ranging from 1 to 20 m/s. The results showed an average positional error of **0.478** in the x-direction and **0.473** in the z-direction, along with an average velocity error of **1.03** and **0.005** in the z and x-directions, respectively.

4) Reward Shaping: We utilized a method similar to the one outlined in [10] to structure our rewards, incorporating equations from our previous tasks. The robot receives a terminal reward upon reaching near the ball and a distance-based incentive for its attempts to approach the ball as closely as possible.

5) Setup: The dynamic constraints on the robot arm were applied similarly to the previous tasks. Additionally, the scaling parameters c_{dist} and c_{ctrl} were set at 0.01 and 500, respectively, with each episode lasting for a total of 1000 time steps.

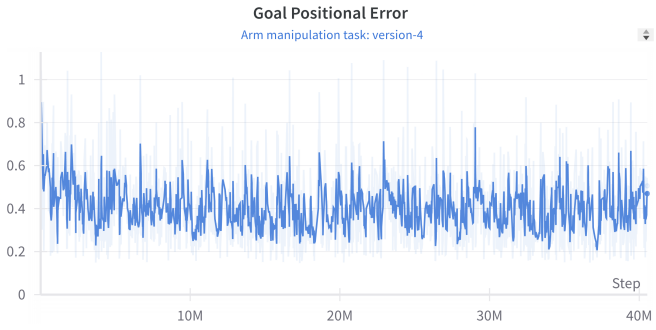

Fig. 7. Goal Positional Error for the reach task

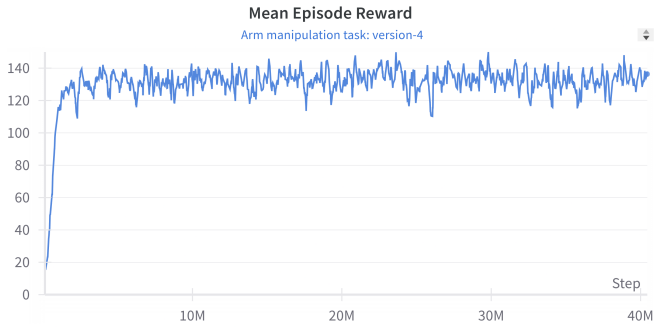

Fig. 8. Rewards for the reach task

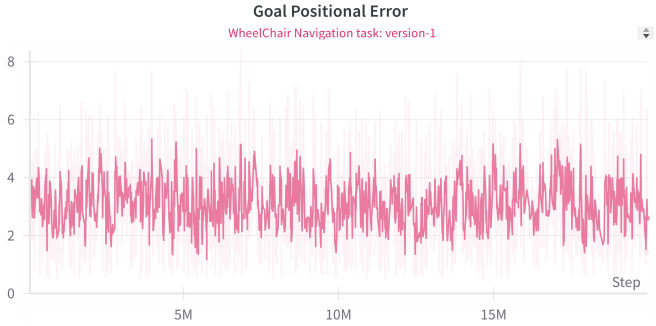

Fig. 9. Goal Positional Error for the navigation task

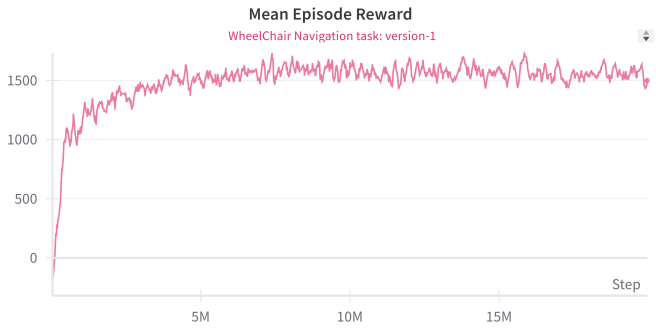

Fig. 10. Rewards for the navigation task

V. Results and Inference

We wrapped our environments using Gym [13] and conducted our simulation using policies trained from the stable baseline repository [17]. Off-policy methods were shown to be effective in terms of sample efficiency and training convergence for robotics related tasks. Despite the effectiveness of DDPG [15] in these tasks, we found SAC [16] to work well owing to its objective of maximizing the long-term entropy thereby encouraging exploration of the state space. As far as the model is concerned, we use a two-layered perceptron for both the actor and critic with a learning rate of 3×10^{-4}, with batch size of 256 was used.

In the reach task, we achieved a mean goal positional error of 0.35m and a maximum episode reward of 140, with a success rate of 84% as illustrated in Figure 7. In contrast, for the navigation task, we observed a mean goal positional error of 3.2m and a success rate of 60%.

Moving to the stroke environment, we only achieved a success rate of 1% and hence further investigation is needed into the environment implementation and reward functions. We suspected that, the issue could be with the ball interception point estimation and to make sure with our hypothesis we trained an end-end model by feeding in the ball coordinates at every time step. Despite observing a slight improvement in the success rate to 1%, the change was insufficient to uphold our theory.

VI. Discussion

One noteworthy finding is that we attained a 100% success rate for both the navigation and reaching tasks when utilizing a torque controller. This can be attributed to MuJoCo's forward kinematics engine to compute the required joint accelerations directly from torque inputs. Therefore, when using other stimuli such as position or velocity, precise tuning of a PD controller becomes necessary to achieve the desired success rate.

With this finding, we tried concatenating the observation and action spaces of the navigation and the reach tasks and trained the robot in an end-to-end fashion to make it reach an arbitrary 3D point in the environment. The remaining objectives were carried over from the previous tasks, and the model underwent training for approximately 16M steps. However, for the task of reaching the goal, the model did not reach convergence within this initial training period. Consequently, we opted to extend the training duration to 40M steps.

We were able to obtain a success rate of **15%** and this can be reasoned that a better navigation policy is needed for the reach policy to perform better. This is because the navigation policy enables the robot to achieve a good proximity to the goal along the x-y plane which will help the arm to reach the point in the z-y plane and vice-versa.

VII. Conclusion and Future Work

In this research, a physics simulator was developed to facilitate the sim-to-real transfer of RL policies, and several

experiments were conducted to test it in a real robotic environment. The model was trained on three different environments, and its performance was reported. The navigation and reach tasks performed really well when trained in isolation, but when combined, they did not achieve the desired success rate. To tackle this, we propose a hierarchical reinforcement learning paradigm wherein a high-level policy learns to choose between the tasks of navigation and manipulation, thereby enabling it to reach the 3D goal. This is left as future work and will be investigated further alongside the stroke environment.

Once a good policy for 3D navigation is established, the next goal would be to deploy this on the real robot and test it on a tennis court to verify the sim-to-real accuracy. Secondly, we would like to use this policy alongside the stroke environment to return a randomly spawned ball.

REFERENCES

[1] Karol Kurach, Anton Raichuk, Piotr Stańczyk, Michał Zajac, Olivier Bachem, Lasse Espeholt, Carlos Riquelme, Damien Vincent, Marcin Michalski, Olivier Bousquet, & Sylvain Gelly. (2020). Google Research Football: A Novel Reinforcement Learning Environment.

[2] Liu, P., Tateo, D., Bou-Ammar, H., & Peters, J. (2021, September). Efficient and reactive planning for high speed robot air hockey. In 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 586-593). IEEE.

[3] Saminda Abeyruwan, Laura Graesser, David B. D'Ambrosio, Avi Singh, Anish Shankar, Alex Bewley, Deepali Jain, Krzysztof Choromanski, & Pannag R. Sanketi. (2022). i-Sim2Real: Reinforcement Learning of Robotic Policies in Tight Human-Robot Interaction Loops.

[4] Todorov, E., Erez, T., & Tassa, Y. (2012). MuJoCo: A physics engine for model-based control. In 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (pp. 5026–5033).

[5] Tianli Ding, Laura Graesser, Saminda Abeyruwan, David B. D'Ambrosio, Anish Shankar, Pierre Sermanet, Pannag R. Sanketi, & Corey Lynch. (2022). GoalsEye: Learning High Speed Precision Table Tennis on a Physical Robot.

[6] Zhang, H., Yuan, Y., Makoviychuk, V., Guo, Y., Fidler, S., Peng, X., & Fatahalian, K. (Year is required!). Learning Physically Simulated Tennis Skills from Broadcast Videos. ACM Trans. Graph..

[7] Zulfiqar Zaidi, Daniel Martin, Nathaniel Belles, Viacheslav Zakharov, Arjun Krishna, Kin Man Lee, Peter Wagstaff, Sumedh Naik, Matthew Sklar, Sugju Choi, Yoshiki Kakehi, Ruturaj Patil, Divya Mallemadugula, Florian Pesce, Peter Wilson, Wendell Hom, Matan Diamond, Bryan Zhao, Nina Moorman, Rohan Paleja, Letian Chen, Esmaeil Seraj, & Matthew Gombolay. (2023). Athletic Mobile Manipulator System for Robotic Wheelchair Tennis.

[8] MG Bellemare, Y Naddaf, J Veness, and M Bowling. "The arcade learning environment: An evaluation platform for general agents." Journal of Artificial Intelligence Research (2012).

[9] Shixiang Gu and Ethan Holly and Timothy Lillicrap and Sergey Levine, "Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates," 2016.

[10] Jonas Tebbe, Lukas Krauch, Yapeng Gao, & Andreas Zell. (2024). Sample-efficient Reinforcement Learning in Robotic Table Tennis.

[11] Dieter Büchler, Simon Guist, Roberto Calandra, Vincent Berenz, Bernhard Schölkopf, & Jan Peters. (2020). Learning to Play Table Tennis From Scratch using Muscular Robots.

[12] Qingyu Xiao, Zulfiqar Zaidi, Matthew Gombolay. (2024). Multi-Camera Asynchronous Ball Localization and Trajectory Prediction with Factor Graphs and Human Poses.

[13] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, Wojciech Zaremba. (2016). OpenAI Gym.

[14] Leonid Butyrev, Thorsten Edelhäußer, Christopher Mutschler. (2019). Deep Reinforcement Learning for Motion Planning of Mobile Robots.

[15] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, Daan Wierstra. (2019). Continuous control with deep reinforcement learning.

[16] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, Sergey Levine. (2019). Soft Actor-Critic Algorithms and Applications.

[17] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, Noah Dormann (2021). Stable-Baselines3: Reliable Reinforcement Learning Implementations. Journal of Machine Learning Research, 22(268), 1-8.