Learning to Optimally Dodge Punches: A Deep Reinforcement Learning Approach to Real-Time Robot Defense Strategies

Maximilian Drach

Introduction

Reinforcement learning (RL) has emerged as a powerful tool for solving complex control problems, particularly in the domain of robotics. In recent years, RL has been successfully applied to a variety of tasks, including motor control, navigation, and game-playing. One area of particular interest is the development of RL algorithms that can enable robots to effectively interact with their environment. This paper explores the use of Deep RL approaches to develop a model that can optimally dodge an incoming punch.

1.1 Motivation

There has been a lot of research into tracking body mechanics and motion synthesis with reinforcement learning. Most of this research has been used to improve game movements and has focused on the creation of more accurate moments or optimizing game winnings in virtual environments. This means there has been little research done into practical real-life scenarios where there isn't a perfect mapping of the environment. These theoretical environments are usually too complex to accurately measure in real life, such as the human joints placement and orientation in 3D space location, but don't factor in subtle factors, such as the frame rate of cameras, to train the RL algorithm. This gap means that many RL algorithms are too complex for real-life tracking and path planning, especially when trying to track a punch that takes around 100ms to complete its full trajectory.

1.2 Objective

The objective is to build a reinforcement learning environment and algorithm that can track a punch, optimally wait for the right time to dodge the punch and move in the right direction to dodge a punch. The reinforcement learning environment should closely simulate the conditions of a computer vision system, from the total tracking area to the target and punch location update speed. This means the end objective is a model that makes one big action movement to dodge the punch and stay put to conserve energy.

1.3 Related Work

Reinforcement Learning research previously focused on simulating captured motions [2,4]. These methods used a variety of motion capture sensors to build a 3D model of an environment by training off collected real data of people punching, kicking, and moving. These methods are time-consuming and expensive to reimplement, because of the specialized equipment needed to stimulate the environment. In recent times, the focus has shifted to improving virtual environment movement mechanics geared towards Virtual Reality. [3,6,7] These complex RL models are effective in simulating boxing decisions but require intricate knowledge about the environment variables, such as the exact 3D coordinate locations of both the opponent and the target extremities (limbs, feet, hips, hands). These models are also computationally expensive and not practical for real-world solutions with slow-moving mechanical objects.

The Deep Deterministic Policy Gradient (DDPG) algorithm has emerged as a notable approach for continuous control tasks [9]. DDPG operates as an off-policy actor-critic algorithm, designed to address the challenges posed by tasks with continuous action spaces. The fundamental architecture of the DDPG model comprises two key components: the actor-network and the critic-network. The actor-network takes the current state of the environment as input and produces a control signal, representing the chosen action. On the other hand, the critic-network is responsible for evaluating the action's effectiveness by taking both the current state and the action from the actor-network as input, ultimately outputting a value function. This value function serves as a critical measure, representing the expected reward associated with executing a specific action in a given state.

TD3 introduces key innovations to optimize continuous control tasks. One notable enhancement in TD3 is the adoption of twin critics, employing two separate critic networks to estimate the value function. This dual-critic architecture mitigates overestimation bias, providing more accurate and robust assessments of chosen actions' effectiveness. Additionally, TD3 introduces delayed updates, addressing the destabilizing effects associated with frequent updates in traditional DDPG algorithms. This delayed update mechanism contributes to stability and convergence, facilitating more effective learning in complex and dynamic environments. Finally, TD3 incorporates a target policy smoothing, adding a small amount of noise to selected actions during training. This technique promotes smoother and more continuous policy updates, enhancing the algorithm's robustness, particularly in scenarios where small perturbations in the action space can have a significant impact. [12]

2. Methodology

2.1 Environment

The simulated environment is a 2-dimensional grid with a size of 2x2 and corresponds to the total area a camera could track a punch (2x2m). There are two variables that interact within the environment: the robot (target) and the punch. The robot is represented by a central x-y coordinate location which represents the center of a circle with a .2 radius and models the approximate area of a human torso. The punch is also represented by a central x-y coordinate location which represents the center of a circle with a .07 radius and models the approximate area of a punching glove.

2.1.1 State Space

There are 10 input values in the space state.

$Target\ Location\ (x, y) \in [-10,10]$ → This crucial informs the actor-network of the present location of the robot and allows it to learn its next ideal action to move its location.

$Previous\ Target\ Location\ (x, y) \in [-10,10]$→ This informs the actor-network of the previous location of the robot and trains the weights to minimize the amount of movement between the two locations.

$Punch\ Location\ (x, y) \in [-10,10]$→ Since the punch comes in a variety of directions this state informs the network about areas where the punch is coming from and trains the network to avoid the area.

$Punch\ Previous\ Location\ (x, y) \in [-10,10]$→ This variable tells the network about the previous location of the punch to help it infer the next position of where the punch will go,

$Punch\ Direction\ Vector\ (\hat{x}, \hat{y}) \in [-1,1]$→ This is a unit vector of the direction of the punch with respect to the location of the robot. This advises the network about the proper direction for the next action.

2.1.2 Action Space

$Action\ Direction\ Vector\ (\hat{x}, \hat{y}) \in [-.25, .25]$→The actor-network returns an x-y coordinate vector to instruct the target where to move on the grid. It is defined from -.25 to .25 because of the physical limitations of the robot's maximum available torque, this confines the action space to more limited movements of just 25cm or .25 in the virtual grid.

2.1.2 Transition Function

The Transition Function aims to simulate the mechanics and movement of the real-life punch-dodging robot. The simulated environment is a 2-dimensional grid with a size of 2x2 and corresponds to the total area a camera could track a punch (2x2m). There is a timestep update that is synced to the frame rate of the camera so that the observation space is only visible in select (1 / camera framerate) timesteps. This means that the punch may have big jumps between the previous location and makes a more realistic model of the real-world data input into the actor-network.

Initial Settings

Robot: The robot's initial position is randomly selected from a uniform distribution over the grid space to approximate its starting position near the center (formula below). It has a radius of .2 and the maximum movement in the x and y directions is [-.25, .25] for every timestep. The robot's initial location is then altered using a normal distribution to create stochasticity for the punch's targeting system.

$$pos_{target_{init}[x,y]} = [U(.5,1.5), U(.6,1.1)]$$
$$pos_{target_{init},stochastic\ [x,y]} = pos_{t_{init}[x,y]} + [N(0,.02), N(0,.04)]$$

Punch: The punch's initial position is randomly selected from a uniform distribution on the x-axis from [0,2] and starts on the zero-y-axis, simulating a punch from the front of the robot. The punch has a radius of .07 to simulate the approximate area of a boxing glove.

$$pos_{punch_{init}[x,y]} = [U[0,2], 0]$$

Movement Settings

Robot: The robot dynamically navigates its surroundings by updating its current location and integrating the input action vector, provided by the actor-network, into the XY coordinates.

$$pos_{target[x,y]} = pos_{target[x,y]} + vector_{action\ [x,y]}$$

Punch: The punch travels in a linear direction toward the robot's altered initial location with an acceleration of -24 m/s^2 and a velocity of 11.4 m/s. [8] It uses the timestep, synced with the framerate, to update the velocity value and the unit vector direction to guide the most direct path to the initial target location. This path is meant to simulate a real-life jab in the observation space.

$$dir_{punch \to target_{init}} = \frac{pos_{t_{inint}, stochastic} - pos_p}{\left|\left|pos_{t_{inint}, stochastic} - pos_p\right|\right|_2}$$

$$timestep = \frac{1}{camera\ frame\ rate}, \qquad vel = vel_{prev} + a_{punch} * timestep$$

$$pos_p = pos_p + dir_{p \to t_{init}} * vel * timestep$$

Ending Settings

Punch: The episode ends when the punch's y-coordinate is greater than the initial target location's y-coordinate because the punch has crossed the area where its intended target was located. Afterward, all the variables and locations are reset according to the *Initial Settings*.

2.2 Rewards

The reward function is designed to incentivize the robot to keep an ideal distance from the punch but also conserve the distance it moves to dodge the punch. Originally there were two reward functions: the distance between the robot and the punch and the amount of movement by the robot. During my original training, the robot would learn that the punch would not travel outside the boundaries therefore teaching the robot to move outside the boundaries and hide. To combat these issues, I created a boundary reward that would impose strong negative penalties for leaving the 2x2 grid.

Grid Boundary Reward Function: The grid boundary is a simple function that checks if the x & y coordinates are between [.01,1.99] and heavily penalizes the reward if the robot coordinates are not within the boundary.

$$pos_{target} + action = pos_{target+1}$$

$$pos_{t+1,[x,y]} \notin [0,2] \rightarrow\ reward_{boundry} \mathrel{+}= -150$$

Distance Reward: The distance reward measure is based on the L2 Norm distance between the closest points on the circumferences of the punch and target circles.

$$dist_{ideal}, pos_{target}, pos_{punch}, radius_{punch}, radius_{target}$$

$$dir_{p \to t} = \frac{pos_{target} - pos_{punch}}{||pos_{target} - pos_{punch}||_2}, dir_{t \to p} = -dir_{p \to t}$$

$$dist_{t,p} = ||\left(pos_{target} + rad_t * dir_{t \to p}\right) - \left(pos_{punch} + rad_p * dir_{p \to t}\right)||_2$$

$$dist_{t,p} = dist_{t,p} - radius_{punch} - radius_{target}$$

$$reward_{orginal} -= \begin{cases} 10 * \left(dist_{t,p} - dist_{ideal}\right) \rightarrow \left(dist_{t,p} \geq dist_{ideal}\right) \\ 10 * \dfrac{1}{dist_{t,p}} \rightarrow \left(dist_{t,p} < dist_{ideal}\right) \end{cases}$$

In response to issues with learning stability caused by an initially defined continuous piecewise reward function that never yielded positive rewards, I introduced a refined formulation. The revised reward structure incorporates an interval-based definition of the ideal distance, utilizing a user-specified tolerance parameter. This modification facilitates enhanced learning by transforming the problem from predicting a precise real number to learning within a defined interval. To further stabilize training, positive rewards are now assigned to successful distance achievements, alleviating the prior instability. Additionally, I employ reward clipping to regulate negative rewards, mitigating the risk of exploding gradients. Finally, I added a negative reward when the punch makes a clear impact on the target. (See Collision Reward)

$$reward_{distance} += \begin{cases} -10 * \left(dist_{t,p} - dist_{ideal}\right) \rightarrow \left(dist_{t,p} > dist_{ideal} + dist_{tolerance}\right) \\ 10 \rightarrow \left(dist_{ideal} - dist_{tolerance} \leq dist_{t,p} \leq dist_{ideal} + dist_{tolerance}\right) \\ -10 * clip[\dfrac{1}{dist_{t,p}}, 0,5] \rightarrow \left(dist_{t,p} < dist_{ideal} - dist_{tolerance}\right) \end{cases}$$

Target Moment Reward Function: The target movement reward function is based on the L2 Norm distance traveled between the current state and the next state (ie the action). The function slightly penalizes the longitudinal direction (y-axis) to incentivize the robot to dodge attacks with movements on the x-axis. This prevents the robot from moving far forward, placing it in an easier spot to be attacked again or too far backward by cowering away from a punch.

$$pos_{target} + action = pos_{target+1}$$
$$reward_{movement} += -30 * |action_y| + -1 * |action_x|$$

Originally, I had a strict reward structure dictating the movement of the x-direction, where I would incentivize the target to move in the opposite direction of the x-component of the punch direction. Though empirical results, the action network would often be overfit to a specific x-direction thus leading to poor testing results as the model would always produce actions in the same x-direction irrespective of where the punch was generated.

Collision Reward Function: Since the timestep makes it so the punch and target can jump around the grid a lot, I had to create two different methods to detect that a collision would have occurred. The first method comes from the $reward_{distance}$ function: where if the distance between the target and punch is a negative number it means that a collision has occurred, because the punch will be inside the target circle. The next method is if the L2 distance between the closest points on the circumference of the previous punch location and the target location is within .05m. This would signify that the target would have gone straight into the punch trajectory and a collision would have occured.

$$reward_{collision} += -20 * clip\left[\frac{1}{dist_{t,p}}, 0,5\right] \rightarrow dist_{t,p} < 0$$

$$reward_{collision} \mathrel{+}= -100 \rightarrow dist_{t,p-prev} \leq .05$$

2.3 RL Algorithm & Training

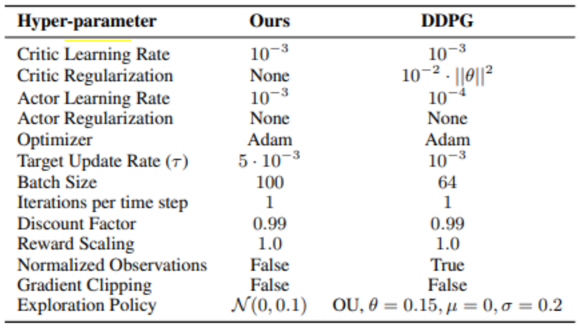
2.3.1 Algorithm

Background: The initial choice of algorithm was the Deep Deterministic Policy Gradient (DDPG) algorithm. However, during experimentation, issues with convergence to a stable value were encountered. It was suspected that the reward structure might be causing the problem. Simplifying the reward structure and increasing the number of input states help the model coverage much better.

Transition to Twin Delayed DDPG (TD3): As an alternative solution to the DDPG, the Twin Delayed DDPG (TD3) algorithm was adopted. The adoption of TD3 was motivated by its three key enhancements: Clipped Double-Q Learning, Delayed Policy Updates, and Target Policy Smoothing. The target policy smoothing and delayed policy smoothing were key advantages that led me to use TD3. The policy being learned in the actor-network had to be carefully generalized due to the stochastic nature of the punch generation. The less frequent policy updates help mitigate volatility in training, providing a stabilizing effect on the learning process. This noise is added to the target policy, making it harder for the policy to exploit errors in the Q-function by introducing smoothness along with changes in action.

2.3.2 Model Specification

The TD3 and DDPG model specifications are taken from the TD3 paper since changing hyperparameters can cause policy instability and reduced learning effectiveness. Without sufficient computational resources to test different model architectures, I decided to keep the same hyperparameters from the paper for both the critic and actor network. [12] Figure 1 shows the hyperparameters used the in TD3 paper.



Hyper-parameter	**Ours**	**DDPG**
Critic Learning Rate	10^{-3}	10^{-3}
Critic Regularization	None	$10^{-2} \cdot \|\|\theta\|\|^2$
Actor Learning Rate	10^{-3}	10^{-4}
Actor Regularization	None	None
Optimizer	Adam	Adam
Target Update Rate (τ)	$5 \cdot 10^{-3}$	10^{-3}
Batch Size	100	64
Iterations per time step	1	1
Discount Factor	0.99	0.99
Reward Scaling	1.0	1.0
Normalized Observations	False	True
Gradient Clipping	False	False
Exploration Policy	$\mathcal{N}(0, 0.1)$	OU, $\theta = 0.15, \mu = 0, \sigma = 0.2$

Figure 1

2.3.3 Model Training

Building on the recommendation from the DDPG and TD3 paper, I tested both models with Ornstein-Uhlenbeck (OU) noise and Normal noise to my action vector. The OU noise induces temporally correlated fluctuations in the noise, meaning that consecutive noise values are correlated. This helps provide a smoother and more continuous exploration signal, helping strike a better balance between exploration and exploitation during training. It also helps the agent from stalling in local minima and allows the agent to escape from suboptimal solutions more easily. The Normal noise is uncorrelated allowing for a more random noise to the action.

I would evaluate a model's average step rewards over a 5000-step period and if the average reward over 100 evaluation periods had not gone up then I would save the best model into a folder. If the model didn't converge, I then created an option to reload the best model from the previous evaluation periods and retrain to find the best new model over a new set of episodes. This could lead to overfitting, but usually, the model would converge especially being trained on such a large dataset of at least 500,000 steps.

Finally, I trained 4 different models: DDPG with OU noise, DDPG with Normal Noise, TD3 with OU Noise, and TD3 with Normal Noise. I compared the average reward results between the models, while also visually testing each model to see if overfitting had occurred within the training.

3. Results

Looking at Figures 2-5, we can see that all the algorithms bounce around a fair bit throughout the steps. The DDPG OU model clearly has trouble with convergence as evidenced by its plot, this would coincide with my issues in early training where the DDPG OU model would have trouble converging to a point. As expected with time-correlated noise, the OU Noise models are characterized by longer regions low variance in average episode reward, and large variance regions as evidenced in Figure 3 & Figure 4.

The only big difference between the two models was that the TD3 typically was a little bit more stable in its average episode rewards. It was surprising to see that the DDPG models did so well compared to the TD3 models.

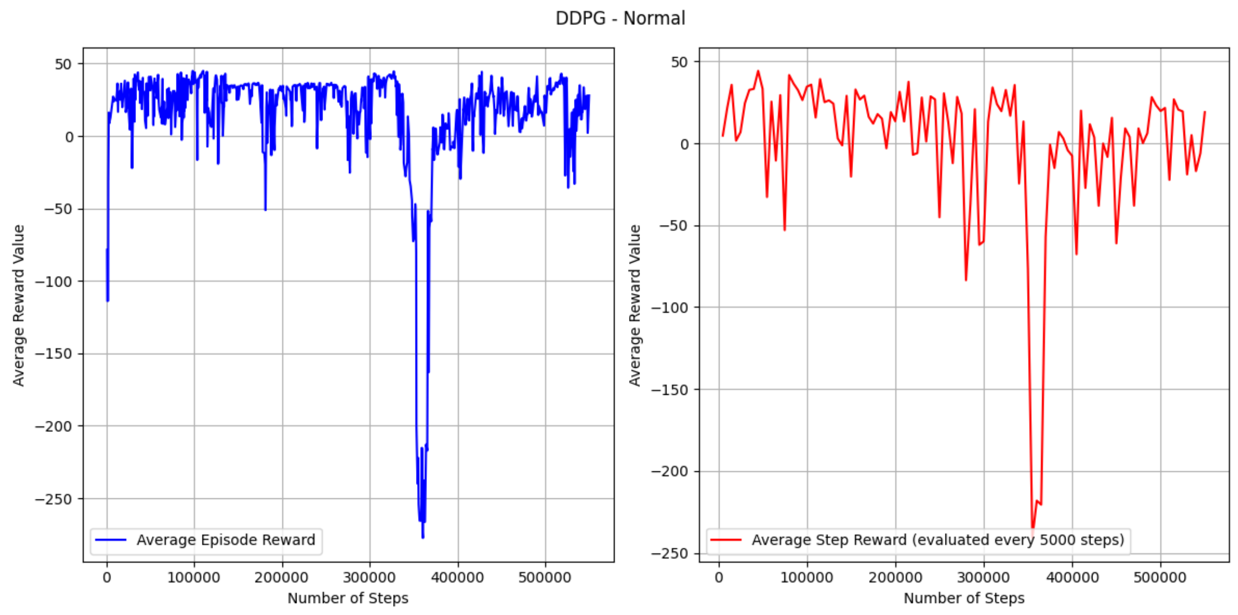


Figure 2

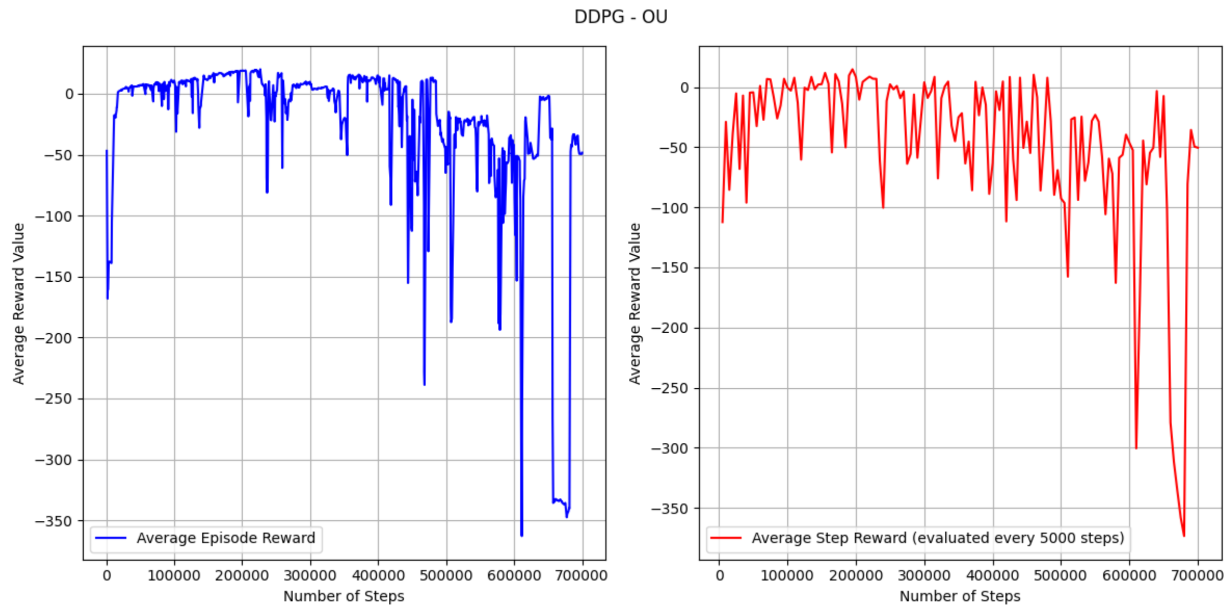


Figure 3

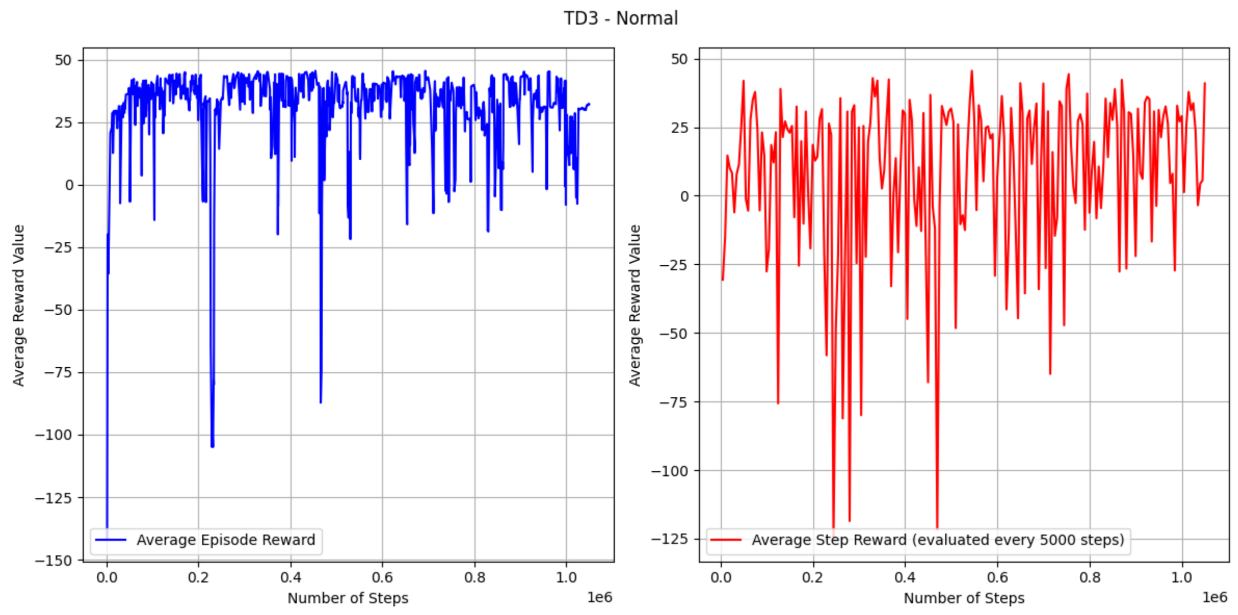


Figure 4

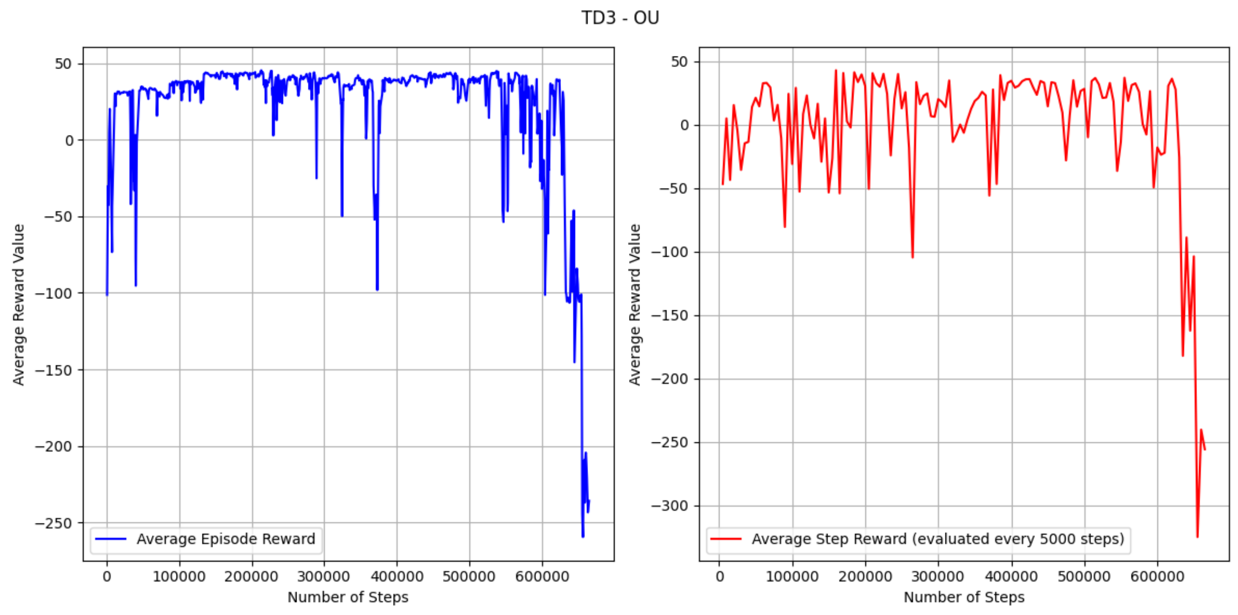


Figure 5

While TD3 Normal appears to have a decent reward score, looking at Figures 6 and 7 the visual trajectories it is apparent that there is overfitting in the model. The target always goes to the left on the x-axis regardless of the direction the punch is coming from.

The TD3 OU might have slight overfitting as shown by its decision to go outside the bounds in Figure 8. However, if you sample other trajectories with TD3 OU model it also shows it going into different x-axis directions and hence the uncertainty of around the overfitting.

Neither of the DDPG models appears to suffer from overfitting and generalize well, especially the DDPG OU model which shown in Figure 12 has made high-quality adjustments to its trajectory given the location of the punch. This is a surprising result given the supposed drawbacks of DDPG was it rougher policy updates.

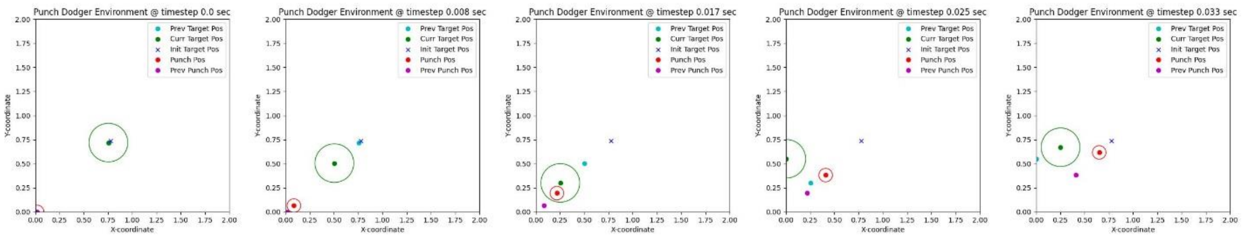


Figure 6: TD3 Normal

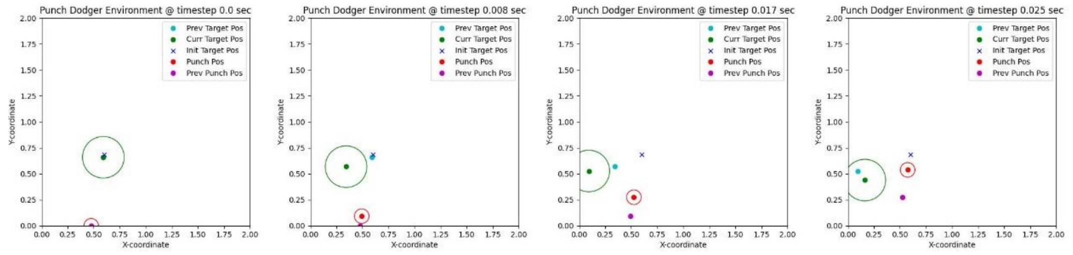


Figure 7: TD3 Normal

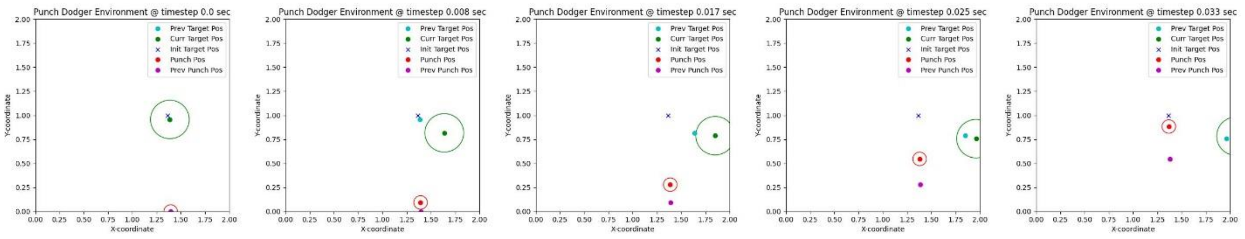


Figure 8: TD3 OU

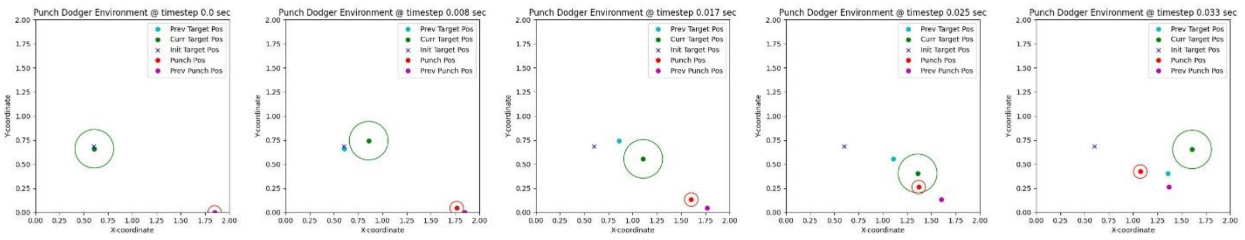


Figure 9: TD3 OU

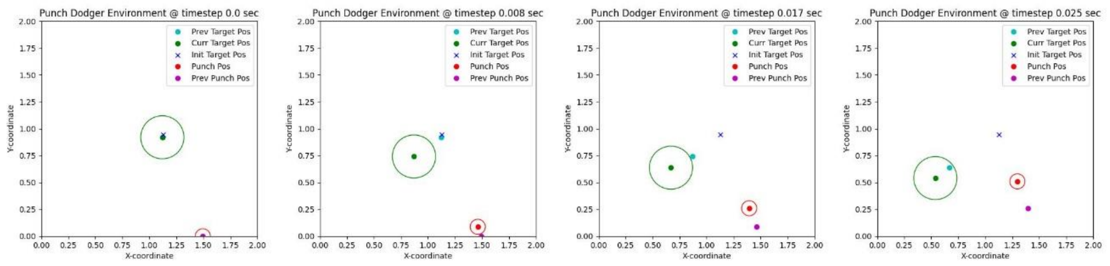


Figure 10: DDPG Normal

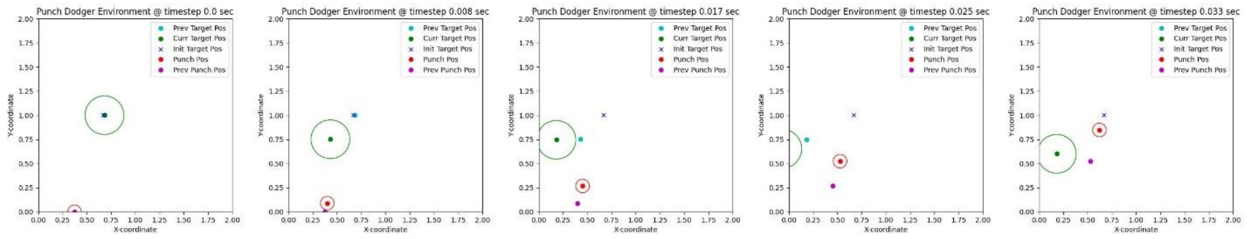


Figure 11: DDPG Normal

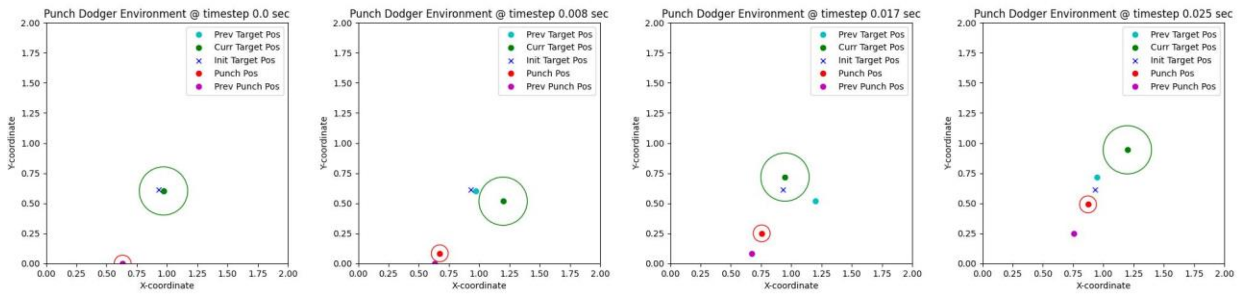


Figure 12: DDPG OU

4. Conclusion

In this study, my primary objective was to develop a reinforcement learning model capable of optimally dodging punches in real-time. I aimed to bridge the gap between theoretical environments explored in existing research and the complexities of real-life scenarios. Throughout the research, I strived to create a simulated environment and algorithm that closely emulated the challenges of a computer vision system, addressing factors such as tracking area, target and punch location updates, and the real-time nature of punch trajectories.

From my findings, I found no significant difference between the models in their general performance. This was a surprise that the DDPG models performed so well, given their disposition to overfit models. The models for most cases effectively follow the reward structure of maintaining an ideal distance away from the punch and staying in bounds as evidenced by their demonstration samples.

Having physical hardware limitations increases the complexity of learning optimal punch dodging trajectories, because you are not given a stable stream of data to learn from. The average episode length was around 5 steps, meaning that you have 5 frames to determine the trajectory of a punch and where is the proper place to move.

5. Future Work

The trajectory of punch movements is a critical aspect that warrants further refinement in future work. Enhancing the accuracy of punch trajectories could involve a comprehensive analysis of the mechanics underlying different types of punches. Investigating the intricacies of punch dynamics, including variations in speed, angle, and force, would contribute to a more realistic simulation environment. Additionally, a notable avenue for exploration would be the simulation of consecutive punches with a delay between them. Simulating sequential punches introduces a layer of complexity, requiring the model to anticipate and respond to evolving threats over time. This extension would not only broaden the scope of the research but also align with scenarios encountered in real-world situations where adversaries may employ varied attack strategies. The sequential punch trajectories would also warrant a new reward structure because the algorithm would have to account for punches that may have varying ideal distances.

Bibliography

1. Kasiri, S., Fookes, C., Sridharan, S., & Morgan, S. (2017). Fine-grained action recognition of boxing punches from depth imagery. Computer Vision and Image Understanding, 159, 143–153. https://doi.org/10.1016/j.cviu.2017.04.007
2. Zordan, V., & Hodgins, J. K. (2002). Motion capture-driven simulations that hit and react. SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation. https://doi.org/10.1145/545261.545276
3. Gamage, N. M., Ishtaweera, D., Weigel, M., & Withana, A. (2021). So Predictable! Continuous 3D Hand Trajectory Prediction in Virtual Reality. UIST '21: The 34th Annual ACM Symposium on User Interface Software and Technology. https://doi.org/10.1145/3472749.3474753
4. Shum, H. P. H., Komura, T., & Yamazaki, S. (2007). Simulating competitive interactions using singly captured motions. VRST '07: Proceedings of the 2007 ACM Symposium on Virtual Reality Software and Technology. https://doi.org/10.1145/1315184.1315194
5. Shum, H. P. H. (2013, May 1). Preparation Behaviour Synthesis with Reinforcement Learning. University of Edinburgh Research Explorer. https://www.research.ed.ac.uk/en/publications/preparation-behaviour-synthesis-with-reinforcement-learning
6. Won, J., Gopinath, D., & Hodgins, J. K. (2021). Control strategies for physically simulated characters performing two-player competitive sports. ACM Transactions on Graphics, 40(4), 1–11. https://doi.org/10.1145/3450626.3459761
7. Qingxu Zhu, He Zhang, Mengting Lan, & Lei Han. (2023). Neural Categorical Priors for Physics-Based Character Control.
8. Stanley, E, R. (2020). Maximal punching performance in amateur boxing: An examination of biomechanical and physical performance-related characteristics (Doctoral dissertation). University of Chester, UK.
9. Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., ... & Wierstra, D. (2015). Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971.
10. Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D. &amp; Riedmiller, M.. (2014). Deterministic Policy Gradient Algorithms. Proceedings of the 31st International Conference on Machine Learning, in Proceedings of Machine Learning Research 2(1):387-395 Available from https://proceedings.mlr.press/v32/silver14.html.
11. Leoyon. (n.d.). Deep Deterministic Policy Gradient (DDPG) – LIAO YONG Technology Space. http://www.liaoyong.net/deep-determinstic-policy-gradient/
12. Fujimoto, S., Hoof, H., & Meger, D. (2018, July). Addressing function approximation error in actor-critic methods. International conference on machine learning (pp. 1587-1596). PMLR.