BirdsEye: Dynamic Vision-Based Autonomous Rover Path Planning and Navigation in Real Time

"All team members have reviewed and approved this report and acknowledge the contributions and responsibilities of each member as outlined herein." - 5/6/2026

Aaditya Sharma (Team Lead)
Department of Computer Engineering
Virginia Tech
Blacksburg, VA
aaditya07@vt.edu

Algorithm Developer: Implemented PyGame simulation, Double DQN, and model integration in physical system.

Bryce Pollak
Department of Computer Engineering
Virginia Tech
Blacksburg, VA
brycep05@vt.edu

Algorithm Developer: Implemented Double DQN as well as target detection mechanism in the physical system.

Tamirlan Zharmagambetov
Department of Computer Engineering
Virginia Tech
Blacksburg, VA
tamirz@vt.edu

State Estimation Developer: Handled state estimation within simulation and car detection in the physical system.

Adam Eshaq
Department of Computer Engineering
Virginia Tech
Blacksburg, VA
eshaqam23@vt.edu

Hardware/Software Developer: Designed, assembled RC car body and circuitry; implemented remote control functionality.

***Abstract*—This project addresses the problem of autonomous rover navigation using a top-down camera to detect obstacles and reach a specified goal in real time. The problem is compelling because it integrates computer vision, reinforcement learning, robotics, and real-time processing—core areas in computer engineering and intelligent systems. To train the reinforcement learning model, a custom environment with a rover and a target was implemented to model the problem. It is important to note that a calibration script was used to ensure simulation closely mimicked real world conditions. A vision-based perception pipeline was developed to evaluate the current state. The agent was able to learn a navigation policy through simulation which was subsequently transferred to a physical rover. A top-down view of the field was received from a webcam propped up by a tripod. Computer Vision techniques were used to gather state information. Color detection using HSV values being the primary approach. State information was extracted from webcam frames, fed to the model, which sent the ideal action to the physical rover. The system was largely successful in reaching the green cylindrical target.**

I. Introduction

Autonomous navigation in dynamic environment requires reliable perception and adaptive planning. Modern-day autonomous systems rely on sensor data to map their positions and leverage agents to make decisions based on map data. This can be costly when scaled to dozens of robots, the cost increasing linearly with each additional robot.

This project seeks to address the problem of autonomous rover navigation using only a top-down view of the environment to detect obstacles and reach a specified goal. This is a proof of concept that demonstrates an inexpensive approach to autonomous navigation and planning that can be expanded to warehouse settings, for example.

In pursuit of this goal, we developed a simulation environment that closely resembled real-world conditions to serve as a training ground for our agent. As a real-world system does not have access to ground truth state values, a vision-based perception mechanism was used to extract state information from the simulator environment. This state information was then used by a Double Deep Q-Network to determine the best action to take to get to the target as fast as possible.

We proceeded to build a physical system, propped up a webcam using a tripod, and developed an entirely new perception pipeline to account for all the imperfections of a real-world system. State was estimated, passed into the DDQN, which sent the ideal action to the physical rover to further the goal of reaching the target. This is BirdsEye.

Fig. 1. BirdsEye Rover and Target

II. Methodology

The high-level approach to this problem starts with designing and assembling the physical rover. This was not the first step in our process, but there is a reason why rover assembly and remote control should be implemented first. The next step is designing a simulation environment where the model will be trained. When designing the simulation environment, it is important to calibrate the simulation parameters such that it matches the expected behavior of the physical rover in space. Initially, the simulation was developed with its own parameters for speed of rover

movement (pixels/sec) and rotation (rads/sec). However, it doesn't take long to recognize that these values don't typically match up with physical systems. The model should adjust to reality, not the other way around. With the environment established, a state estimation mechanism needs to be implemented. The state representation of the environment along with a rewards ruleset for selecting actions that approach the target will then be used for training the Double DQN agent that will control the rover. After the model has been trained, the physical system needs a way to create the state representation needed by the model to make an action decision. Thus, a new perception mechanism needs to be established. The state representation will be passed to the model, which will send remote control commands to the microcontroller on the rover.

A. Rover + Calibration

For simulation purposes, calibrate simulation parameters according to motion patterns of the rover.

1) Rover: The rover chassis and motor mounts were designed using 3D CAD software. They were 3D printed. The circuitry is comprised of an HC-05 bluetooth module for receiving commands from the agent, an Arduino Uno microcontroller, and an L289n motor driver. PySerial is the python library used for bluetooth communication between the rover and our local computer. Actions are mapped to messages with format: "[l_wheel_vel],[l_wheel_dir],[r_wheel_vel],[r_wheel_dir]".

2) Calibration: As previously stated, the simulation parameters need to be adjusted depending on the rover's motion patterns. As such, we implemented a calibration script that modifies simulation actions such that the linear and angular velocities between simulation and rover match.

B. State Estimation on Simulation

Refer to Appendix A for an image of the simulation environment. Refer to Appendix B for target representation.

For simulation purposes, this could have been done using the ground truth values. However, we decided to use this as practice for our physical perception system, which we will discuss in detail in the coming sections. Thus, to avoid redundancy, refer to *Physical Perception* section for our approach.

C. Algorithm Development

Given the state estimation result using our simulation perception pipeline, the model should be able to predict the best possible action.

1) Network Architecture: The agent uses a Double Deep Q-Network (DQN) implemented as a fully connected multilayer perceptron that maps the 22-dimensional state representation to Q-values over 5 discrete actions. The architecture consists of an input layer accepting the 22 normalized state features, three hidden layers of 256, 256, and 128 neurons respectively with ReLU activations, and an output layer producing one Q-value per action.

2) Training Procedure: The agent is trained using a standard Deep Q-Learning framework with experience replay and a target network for stability. The agent initially explores randomly and gradually shifts toward exploitation as training progresses. Epsilon decays over episodes to reduce exploration in later stages. An experience replay buffer stores transitions and uses random minibatch sampling to break temporal correlation in training data. Huber loss is used to reduce sensitivity to outliers in Q-value estimation with gradient clipping to prevent instability. Training was done across ~4000 total episodes with each episode limited to about 80 steps.

3) Objective Function: The agent is trained by minimizing the Huber loss between the predicted Q-value of the taken action and a bootstrapped target. The Double DQN formulation decouples action selection from action evaluation to reduce overestimation bias.

D. Physical Perception

The physical perception pipeline is responsible for reconstructing the 22-dimensional state vector from a raw camera frame. Each frame captured at 1920×1080 is first down sampled to the 900×600 logical map resolution to match the coordinate space used during training.

1) Car Detection Stage 1: The rover's red body is isolated using a dual-range HSV mask covering both hue wrap-around regions (0–5° and 170–180°) with a saturation floor. The largest surviving contour is taken as the car body, and a minimum-area rotated rectangle is fit to it to recover the car center and a raw estimate of the four corners. A filled convex hull of the rotated rectangle is then constructed as a tight footprint mask for the second stage.

2) Car Detection Stage 2: Heading is determined from a small white tape marker affixed to the front of the rover. A white HSV mask is computed at full-frame resolution and masked against the eroded red body region intersected with the convex hull. The best candidate contour is selected by a score combining region area and proximity to the body centroid. The forward direction vector is then computed as the normalized offset from the car body centroid to the tape marker centroid. Corner labels (FR, FL, BR, BL) are assigned by projecting the four rotated-rectangle corners onto the forward and right-hand axes derived from this vector.

3) Target detection: Operates independently of the car pipeline and uses HSV thresholding tuned to the specific hue, saturation, and value range of the colored goal marker placed in the arena. A binary mask is constructed from all pixels matching the threshold, and connected component analysis identifies the largest surviving region as the target. A minimum-enclosing circle is fit to that contour, with the circle centroid providing the target's (x, y) position and the circle radius serving as the reach threshold used by BirdsEyeRun.py to determine when the rover has arrived.

4) State Assembly: Once both detections succeed, state_generator.py assembles the 22-dimensional vector using the same normalization scheme as the simulator. Refer to Appendix B for captured state features.

E. *Final Pipeline*

The webcam captures real-time individual frames of the environment. Each frame is converted into a 22-feature state representation that the model uses to issue an action (0-4) to the physical system. The action is converted into a message compatible with Arduino Uno microcontroller that is onboard the rover. This message format is displayed in Figure 2. The microcontroller receives the message via an HC-05 Bluetooth module. The microcontroller breaks the message into individual motor commands, which are issued to the L289n motor driver to move the robot towards the target. Refer to Figure 2 for a diagram of the pipeline.

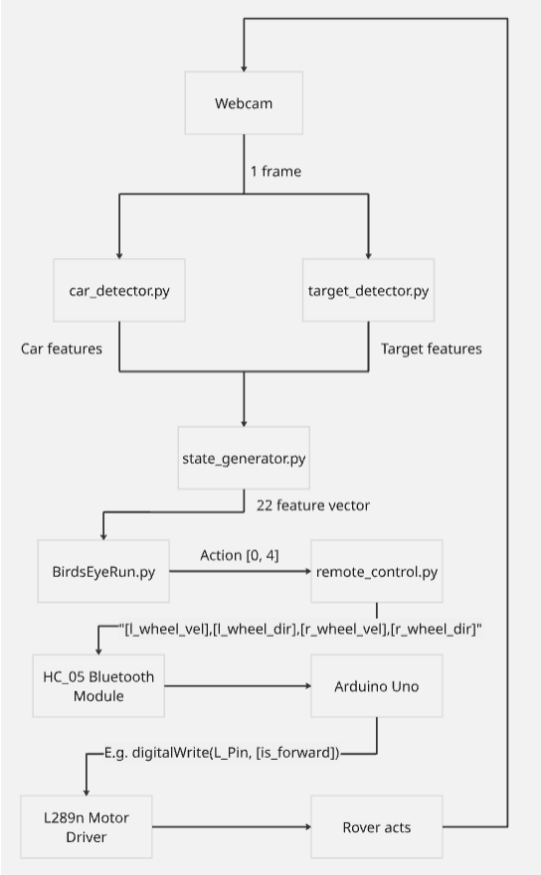

Fig. 2. Real-time system pipeline for autonomous rover

III. Results and Evaluation

The BirdsEye system was evaluated over 50 real-world episodes using the complete perception–control pipeline described previously. Performance was assessed in terms of task success, trajectory quality, spatial behavior, safety (wall avoidance), and efficiency in reaching the target.

A. *Overall Performance*

The system achieved a 100% success rate (50/50 episodes), indicating successful transfer of the learned policy from simulation to the physical environment. The reward statistics remained tightly clustered, with an average reward of 108.97 and a range of 103.48 to 116.20. Episode lengths averaged 26.6 steps, with a minimum of 13 and a maximum of 37 steps.

Figure 3 shows the reward per episode and corresponding episode lengths across all evaluation runs. The relatively low variance in both metrics suggests that the policy behaves consistently across different initial conditions rather than relying on favorable starting states.

Fig. 3. Agent Performance Over Evaluation Episodes

B. *Trajectory Behavior*

Representative agent trajectories are shown in Figure 4. Across multiple episodes, the rover exhibits smooth, goal-directed motion with minimal oscillation. In shorter episodes, the agent follows near-linear paths toward the target, while longer episodes show slight curvature due to initial misalignment or corrective rotations.

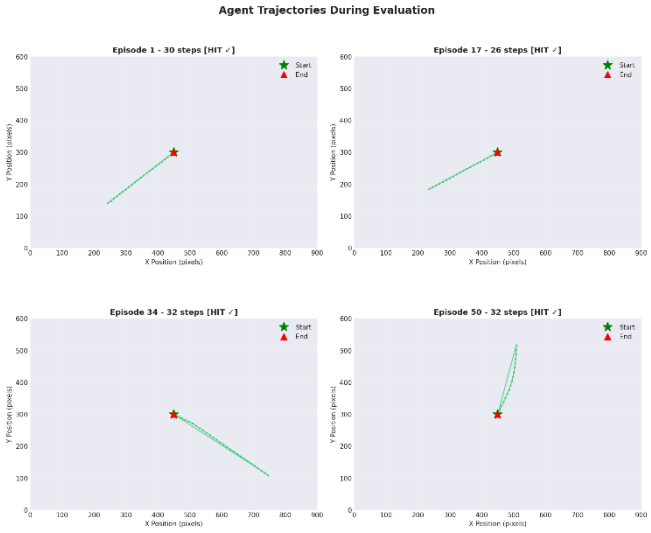

Fig. 4. Agent Trajectories During Evaluation

Despite these variations, all trajectories converge reliably to the target without divergence or erratic motion.

C. *Spatial Coverage and Visitation Patterns*

Figure 5 presents a visitation heatmap aggregated across all evaluation episodes. The heatmap reveals a high-density region near the target location, confirming consistent convergence behavior. Intermediate regions show moderate

visitation, corresponding to the agent's progression from initial positions to the goal.

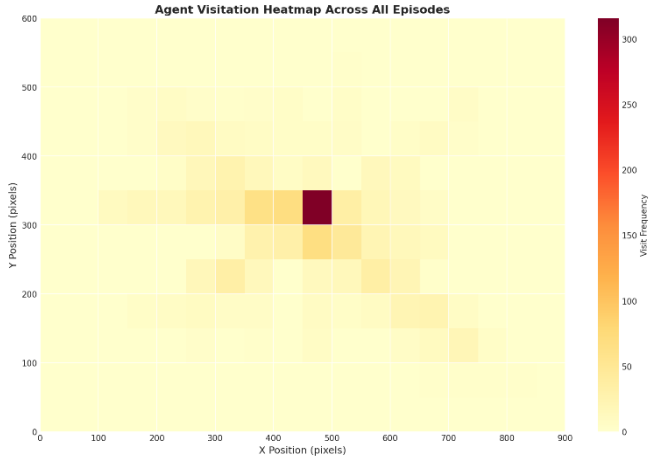

Fig. 5. Agent Visitation Heatmap Across All Episodes

Importantly, there is no excessive clustering near the boundaries of the environment, suggesting that the learned policy prioritizes direct goal-seeking behavior rather than exploiting environmental edges.

D. *Wall Avoidance Behavior*

Wall avoidance performance is illustrated in Figure 6, which shows both the minimum distance to walls over time and the distribution of wall proximity across all steps. The agent consistently maintains a safe margin from boundaries, with most distances falling between 0.25 and 0.50 (normalized).

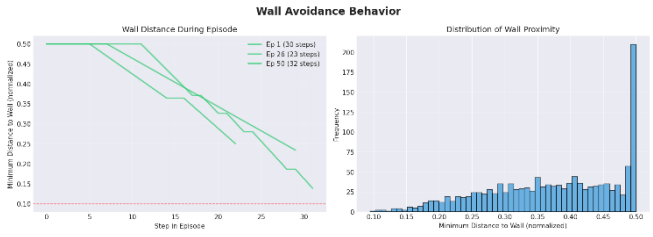

Fig. 6. Agent Wall Avoidance Performance

The safety threshold of approximately 0.10 is never approached during evaluation, indicating that the rover does not exhibit collision-prone behavior. Overall, these results demonstrate that the agent has learned an implicit understanding of boundary constraints through the state representation.

E. *Target Finding Efficiency*

Figure 7 shows the distance-to-target progression during representative episodes. In all cases, the distance decreases smoothly, indicating stable convergence. Initial distances vary depending on starting position, but final distances consistently fall within a narrow range corresponding to the success threshold.

The comparison of initial versus final distances across all episodes (Figure 7) further highlights the robustness of the policy. Regardless of the starting configuration, the agent reliably reduces the distance to the target and terminates within the goal region.

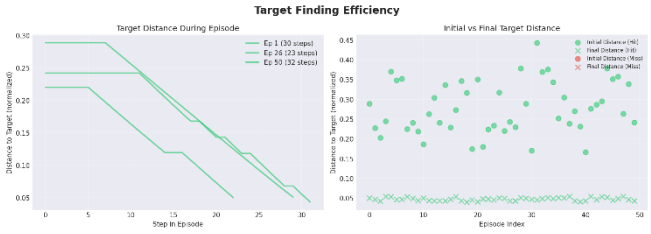

Fig. 7. Agent Target Finding Efficiency

This behavior confirms that the learned policy generalizes well across different spatial configurations and does not depend on specific initial conditions.

F. *Action Selection Analysis*

The distribution of selected actions is shown in Figure 8. The agent predominantly chooses forward (568) and backward (516) actions, while rotational actions are used less frequently (159 left rotations and 85 right rotations).

Fig. 8. Distribution of Agent Selected Actions

This distribution indicates that the agent prefers linear motion once it is approximately aligned with the target, using rotations primarily for initial orientation correction. Overall, action distribution reflects an efficient control strategy in which rotational adjustments are minimized and forward motion is prioritized for rapid convergence.

G. *Temporal Stability*

Figure 9 illustrates performance trends over all evaluation episodes. Both reward and episode length remain stable across time, with no evidence of degradation, instability, or policy drift.

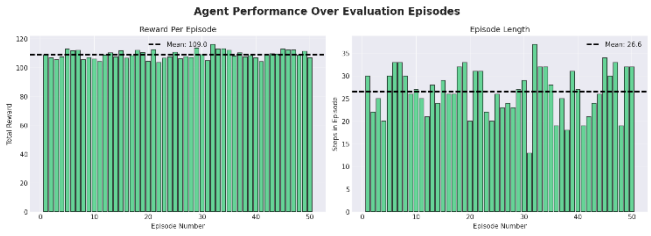

Fig. 9. Agent Performance Trends Over All Evaluation Episodes

This consistency indicates that the system is robust to real-world noise in perception and actuation. The integration between the vision-based state estimation and the Double DQN controller remains reliable throughout extended operation.

H. Physical Car Detection Results

The physical perception pipeline was also evaluated qualitatively using top-down camera frames from the real environment. As shown in Figure 10 & 11, the rover body is successfully detected using the red HSV mask, and the fitted rotated bounding box accurately tracks the car footprint. The labeled corners (FL, FR, BL, BR) remain consistent across different positions and headings, which confirms that the corner-ordering logic is working correctly.

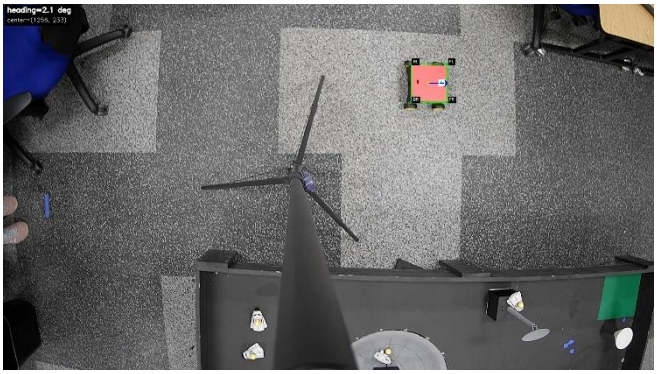

Fig. 10. Detection of a 2.1° Rotated Car

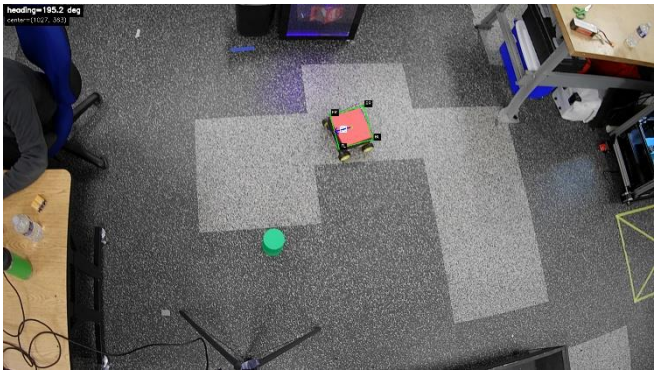

Fig. 11. Detection of a 195.2° Rotated Car

The heading estimate is also shown directly on the camera frames. The detector correctly identifies the front-facing direction of the rover using the white tape marker, producing stable heading estimates such as 2.1° and 195.2° across different orientations. This is important because the reinforcement learning model depends on heading, forward direction, and corner coordinates as part of the 22-feature state vector.

I. Summary

The evaluation results demonstrate that the BirdsEye system achieves reliable and consistent autonomous navigation in a real-world environment. The agent successfully reaches the target in all trials, maintains safe distances from boundaries, and follows efficient trajectories with minimal corrective behavior.

These findings confirm that the combination of a calibrated simulation environment, a structured state representation, and a Double DQN policy is sufficient to bridge the gap between simulation and physical deployment for this class of navigation tasks.

IV. Discussion and Key Insights

Several design decisions were critical to achieving stable and efficient performance in the BirdsEye system. First, a fully connected MLP was selected over a convolutional architecture because the input to the model is a structured 22-dimensional feature vector rather than raw image data. Since no spatial correlations need to be learned, convolutional layers would introduce unnecessary computational overhead without improving performance. In addition, Double DQN was chosen over vanilla DQN to mitigate Q-value overestimation. By decoupling action selection and evaluation through separate policy and target networks, Double DQN provided more stable training and improved convergence behavior in both simulation and real-world deployment.

A key practical challenge observed during testing was the rover overshooting the target and pushing it after contact. This issue stemmed from the mismatch between pixel-level detection and the physical footprint of the rover. To address this, the perceived target radius was scaled by a factor of 4.25, ensuring that the stop condition is triggered before physical contact.

Another major source of instability was noise in the perception pipeline, particularly in heading estimation. Small variations in the red-mask contour caused inconsistent bounding box fits, which occasionally resulted in incorrect front/back assignments and large heading flips. These errors propagated directly into the control loop, degrading performance. To resolve this, temporal smoothing was introduced by blending successive heading estimates, along with a deadband that suppresses minor fluctuations below 2.5°. Additionally, restricting the white marker search to the rover's rotated footprint eliminated false detections caused by background artifacts. Together, these changes produced stable and consistent state estimates across all rover orientations.

From these experiences, several key insights emerged. First, structured state representations combined with classical vision techniques can outperform more complex end-to-end approaches in constrained environments, enabling efficient learning with simpler models. Second, sim-to-real alignment is essential; accurate calibration of motion dynamics and coordinate systems is a prerequisite for successful policy transfer. Third, temporal stability in perception is more critical than raw detection accuracy, as small frame-to-frame inconsistencies can lead to large control errors in closed-loop systems. Finally, the results demonstrate that robust autonomous navigation can be achieved using low-cost hardware and minimal sensing, highlighting the practicality and scalability of vision-based approaches for real-world applications.

V. Conclusion and Next Steps

This project has been a very exciting undertaking for the BirdsEye team. Our model performs very well when utilized in the physical environment. We successfully implemented an autonomous rover that relies solely on an overhead camera for perception, demonstrating that a low-cost, vision-based approach can achieve reliable real-time navigation. The system consistently reaches the target with stable trajectories and minimal corrective behavior, validating both the simulation-to-real transfer and the effectiveness of the perception–control pipeline.

There are, however, several limitations to our current implementation. Due to time constraints, obstacle detection and avoidance were not incorporated into the system. Extending the state representation to include obstacle features, along with appropriate reward shaping, would allow the agent to operate in more complex and dynamic environments. Additionally, the current HSV-based color detection approach is sensitive to lighting conditions and specific color choices, which limits scalability. While effective for this proof-of-concept, it would not generalize well to more realistic scenarios with varying textures, colors, or multiple agents.

Future work will focus on addressing these limitations and expanding the system's capabilities. A more robust perception pipeline could be developed using learning-based detection methods or adaptive color segmentation to improve generalization under varying environmental conditions. Incorporating obstacle avoidance through additional sensors or enhanced vision-based features would enable navigation in cluttered environments. Furthermore, extending the system to support multiple rovers would introduce challenges in coordination and collision avoidance, opening the door to multi-agent reinforcement learning.

Overall, this project demonstrates that autonomous navigation can be achieved with minimal sensing and computation when paired with a well-designed state representation and reinforcement learning framework. BirdsEye serves as a strong foundation for future work in scalable, low-cost autonomous systems.

APPENDIX A

SIMULATION ENVIRONMENT

APPENDIX B

STATE REPRESENTATION

State Representation		
Index	***Feature***	***Normalization***
0-1	Front-right corner (x, y)	x / 900, y / 600
2-3	Front-left corner (x, y)	x / 900, y / 600
4-5	Back-right corner (x, y)	x / 900, y / 600
6-7	Back-left corner (x, y)	x / 900, y / 600
8	sin(heading angle)	Already in [-1, 1]
9	cos(heading angle)	Already in [-1, 1]
10-11	Forward direction vector (x, y)	Already unit-length
12-13	Target position (x, y)	x / 900, y / 600
14	Target radius	radius / 900
15	Distance to target	dist / diagonal
16-17	Delta to target (dx, dy)	dx / diagonal, dy / diag
18	Distance to left wall	car_x / 900
19	Distance to right wall	(900 - car_x) / 900
20	Distance to top wall	car_y / 600
21	Distance to bottom wall	(600 - car_y) / 600

APPENDIX C

REWARD FUNCTION

Reward Function		
Condition	***Reward Condition***	***Typical Magnitude***
Every step	-0.1	-0.1
Hit target	+100.0	+99.99
Hit boundary	-100.0	-100.1
Moving closer	+50 * (delta_d / 1082)	+0.01 to +0.17
Moving farther	-50 * (delta_d / 1082)	-0.01 to -0.17

APPENDIX D

HYPERPARAMETERS

Hyperparameters		
Parameter	***Value***	***Description***
Learning rate	5e-4	Adam optimizer step size
Discount (gamma)	0.99	Future reward discount factor
Batch size	128	Transitions sampled per learning step
Buffer capacity	200,000	Maximum replay buffer size
Epsilon start	1.0	Initial exploration rate (100% random)
Epsilon end	0.01	Minimum exploration rate
Epsilon decay	0.997	Multiplicative decay per episode
Tau	0.005	Soft target update rate (Polyak averaging)
Learn every	4 steps	NN update frequency (skips reduce compute)
Gradient clip	10.0	Max gradient norm to prevent exploding updates
Loss function	Huber	SmoothL1Loss, robust to outlier rewards