OXFORD JOURNAL
OF STUDENT
SCHOLARSHIP

Model-Free Reinforcement Learning for Projectile Targeting with Inherited Platform Velocity from a Moving Omnidirectional Robot

Aditya Rastogi
arastogi0129@gmail.com

ABSTRACT

Projectile targeting from a moving robot is more difficult than targeting from a stationary platform because a launched projectile inherits the translational velocity of the robot. This study investigated whether a model-free reinforcement-learning controller could learn to compensate for this effect while controlling a simulated turret-mounted launcher. A Proximal Policy Optimization (PPO) model was developed in Unity using ML-Agents. The controller used eight observations comprising target geometry, robot chassis velocity, turret state, and aiming progress to generate continuous yaw, pitch, and flywheel-speed commands. The training was done with multiple independent seeds at chassis speed in range 0-1.5 m/s and bearing randomized across 360^0 for 6 million training steps achieving high training performance (>99% final hit rate). Frozen-policy evaluation was subsequently conducted over 1,000 episodes at each of stipulated chassis speeds. The controller achieved interception accuracies of >99% for stationary chassis as well as for speed up to 0.5 m/s moving towards and 1.0m/s moving away from target. The accuracy degraded to 85.9% for speed of 1.5 m/s moving towards target, reflecting difficulty in acquiring target from closer distances when platform is also moving at higher speed. The evaluation data was benchmarked against inverse ballistics baseline. These results demonstrate high interception accuracy across the evaluated moving-platform conditions.

Keywords: *reinforcement learning, Proximal Policy Optimization, projectile motion, robotic targeting, omnidirectional robot, continuous control*

INTRODUCTION

Mobile robots are increasingly expected to perform tasks while remaining in motion rather than stopping before each action. Omnidirectional platforms, such as Mecanum-wheel systems, are particularly suited to this operation because they can translate in different directions while independently controlling their orientation (Gfrerrer, 2008). While this improves maneuverability, it creates an additional challenge when the robot carries a projectile launcher: the motion of the robot affects the motion of the projectile. In the 2025–26 FIRST Tech Challenge (FTC) season, robots score by launching 5-inch balls into a goal situated at a height on a corner of a rectangular field (FIRST, 2026). Time spent stopping to aim is time not spent scoring, and during the autonomous period the robot has to aim entirely by itself.

A projectile fired from a stationary launcher begins its flight with a velocity determined mainly by the launch speed and direction. When the same projectile is fired from a moving robot, it also inherits the translational velocity of the robot at the instant of launch. A targeting controller must therefore account for both launcher settings and platform motion. One approach to this problem is to calculate the required compensation directly using projectile kinematics. If the position of the target, velocity of the robot and launch velocity are known, a model-based controller can estimate the projectile trajectory and determine suitable launcher settings. Reinforcement learning provides an alternative approach in which a controller learns a relationship between its observed state and the actions that produce successful outcomes.

Reinforcement learning (RL) is a machine-learning approach in which an agent repeatedly interacts with an environment, selects actions, and receives rewards based on their outcomes (Sutton & Barto, 2018). Over many interactions, the agent attempts to develop a policy that selects actions associated with higher cumulative reward. RL has been investigated extensively in robotics for sequential decision-making and continuous-control applications (Kober et al., 2013; Tang et al., 2025).

Applying reinforcement learning to robotics also involves challenges. Its performance can depend on the observations provided to the agent, the reward function, the learning algorithm, and the conditions encountered during training. Simulation is therefore useful for initial investigations because a controller can experience a large number of interactions without requiring an equivalent number of physical trials.

Deep reinforcement learning has been investigated in several areas of robotics, including robot manipulation and mobile-robot navigation (Han et al., 2023; Le et al., 2024). Continuous-control problems are particularly relevant because many physical actuator commands are not naturally represented by a small number of discrete choices (Recht, 2019). The targeting problem considered in this study is also a continuous-control problem. A turret can vary its yaw, pitch, and projectile launch speed over continuous ranges. A controller must therefore determine not simply whether to move left or right, but how much to adjust several actuator commands simultaneously.

Proximal Policy Optimization (PPO) is a policy-gradient reinforcement-learning algorithm developed for stable policy optimization (Schulman et al., 2017). PPO updates a learned policy using experience collected through interaction with an environment while restricting excessively large changes during individual policy updates. The algorithm is well suited to continuous-control problems in which multiple actuator commands must be adjusted simultaneously. The moving-platform targeting problem provides a compact way to investigate whether PPO can acquire physically meaningful compensation behaviour, combining projectile kinematics with continuous robotic control.

In this study, the targeting problem was investigated using a Unity-based PPO controller with three-dimensional simulation of an omnidirectional robot equipped with a turret-mounted flywheel launcher. The PPO controller received information describing relative target position, chassis velocity,

turret orientation, and progress through the aiming period. It produced three continuous outputs controlling turret yaw adjustment, turret pitch adjustment, and flywheel rotational speed. Projectile motion was evaluated analytically while incorporating the planar chassis velocity inherited by the projectile at the instant of launch.

The closest earlier work is in robotic throwing. TossingBot (Zeng et al., 2020) learned to throw arbitrary objects into boxes by combining a physics-based ballistic estimate with a learned correction, or "residual", that accounted for effects the physics left out, such as differences in mass distribution and aerodynamics between objects. That design keeps an analytical model at the centre of the controller. The present study asks the opposite question: how much of the targeting problem can a policy learn when it has no ballistic model at all when choosing its actions?

RL has also been applied to aiming at targets. Li et al. (2021) trained a deep RL policy to control a two-axis gimbal shooting at moving targets, reporting greater accuracy than classical controllers. In missile guidance, Hu et al. (2023) trained a policy using the TD3 algorithm that maps line-of-sight rate directly to acceleration commands for maneuvering targets. In both cases, however, it is the target that moves. In the problem studied here, the target is fixed and the launcher moves, so the projectile carries the platform's velocity with it. As far as the author is aware, learned control for this inherited-velocity case has not been studied before.

The purpose of this study was to investigate **whether a model-free PPO controller could learn accurate projectile targeting from a translating robotic platform without being directly supplied with an inverse ballistic targeting solution.** The objectives were to develop a continuous-control PPO policy for turret yaw, pitch, and flywheel speed; train the controller at all field positions and intended range of chassis velocity; and evaluate the final frozen policy under regulated chassis-motion conditions. The observations are benchmarked against an analytical model based on kinematics.

METHODOLOGY

Simulation Environment

To ensure real-world applicability, the geometric parameters of the field and the physical specifications of the robotic platform were adopted from the 2025–26 FIRST Tech Challenge competition format. The simulated field measured 3.66 m square (12 feet square). The target position is fixed and placed at one of the corners of the field. For the sake of simplicity, the launcher is assumed at the centre of chassis and same field constraints are applicable on both launcher and the robot chassis. The Robot can take aim from anywhere in the field except from a minimum radial distance of 1.0 m from the target pole. The projectile is a smooth spherical ball with a 5-inch diameter and the target a circular goal basket with a 13.5-inch diameter. For simplicity the projectile was considered a point object and after accounting for dimensions of both, effective radius of target was taken as 0.108 m.

The final targeting environment was implemented using Unity, a real-time 3D development platform, together with the ML-Agents framework (Juliani et al., 2020). The simulation represented an omnidirectional robot equipped with a turret-mounted flywheel launcher and a fixed elevated target. The environment used a physics timestep of 0.05 s, corresponding to a 20 Hz control rate. Each episode contained 30 applied aiming actions, giving a 1.5 s aiming period before one projectile trajectory was scored. The maximum speed investigated during training was 1.5 m/s. A successful interception was defined analytically rather than by visual collision in Unity. A projectile was classified as a hit when its calculated parabolic trajectory under the influence of negative "g" crossed the target-height plane in descending direction (i.e. achieving a peak height and then descending into the target plane) with a horizontal error (hit tolerance) of radius 0.108 m or less. The criterion therefore represented target-plane interception rather than physical entry into a finite three-dimensional goal.

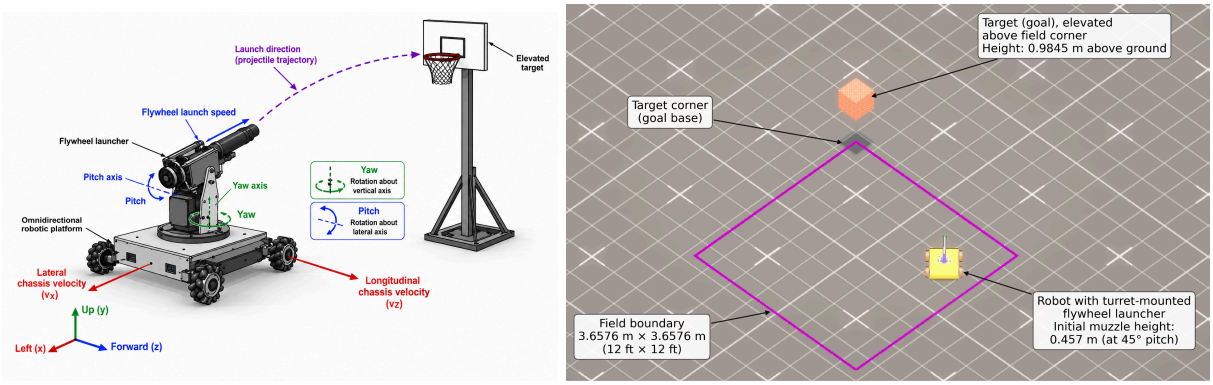


Figure 1. System schematic showing the moving omnidirectional robotic platform, chassis velocity components, turret yaw and pitch, flywheel launch speed, projectile launch direction, and elevated target. Image is the screenshot of the Unity simulation environment.

Parameter	Value
Target location	Fixed field corner
Target center height above ground	0.9845 m
Initial muzzle height above ground	0.457 m, at 45° pitch
Initial target-muzzle height difference	0.5275 m
Hit tolerance	0.108 m horizontal miss distance
Crossing used for scoring	Descending only
Field size	3.6576 × 3.6576 m (12 x 12 ft)
Chassis speed	0-1.5 m/s
Movement direction	Randomized across 360°
Launch pitch limits	0°-90°
Initial pitch	45°, fixed
Yaw limits	-90° to +90° relative to robot
Initial yaw	0°, fixed
Physics timestep	0.05 s
Control frequency	20 Hz
Aiming actions per episode	30
Aiming duration	1.5 s
Flywheel radius / diameter	0.048 m / 0.096 m
Velocity-transfer factor	0.5
Maximum exit speed	10.05 m/s

Table 1. Parameters for Simulation Environment

Projectile Kinematics

Projectile trajectories were evaluated analytically in world coordinates. Flywheel rotational speed was first converted into projectile exit speed. If N represents flywheel speed in revolutions per minute, angular velocity was calculated as:

$$\omega = \frac{2\pi N}{60}$$

The corresponding projectile exit speed was calculated as: $\nu = k\omega r$

Here, the tangential surface speed of the flywheel was defined as ωr, where ω is the angular velocity and r is the flywheel radius. For single-wheel flywheel launchers equipped with a hood, the projectile exit

speed is typically well below the wheel's surface speed. To account for this, a velocity-transfer factor $k = 0.5$ was introduced to convert the flywheel surface speed to the projectile's center-of-mass launch speed. Using a flywheel radius r of 0.048 m and a maximum rotational speed N of 4000 RPM, the maximum projectile exit speed evaluated to 10.05 m/s. The exit velocity was directed according to the turret yaw and pitch. The planar velocity of the chassis at the instant of firing was then added to the turret-directed projectile velocity to obtain effective projectile velocity. The initial projectile velocity was therefore represented as:

$$\mathbf{v}_0 = \mathbf{v}_{launcher} + \mathbf{v}_{chassis}$$

This velocity addition represents the central physical effect investigated in the study. A projectile launched from a translating robot inherits the planar velocity of the chassis at the instant of firing in addition to the velocity produced by the launcher. After launch, projectile motion was calculated under constant gravitational acceleration using:

$$\mathbf{p}(t) = \mathbf{p}_0 + \mathbf{v}_0 t + \frac{1}{2}\mathbf{g}t^2$$

where p_0 is the projectile position at launch, v_0 is its initial world-frame velocity, and the gravitational acceleration vector was defined as (0, -9.81, 0) m/s². Using the robot position coordinates relative to target, the effective projectile launch velocity, pitch and yaw angles, the simulator analytically determined the positive times at which the projectile crossed the height of the target. When descending positive crossings existed, the horizontal error was calculated at the crossing and used for scoring. A shot was classified as a successful interception when this error was 0.108 m or less. If error was greater than 0.108 m or no positive target-height crossing existed, the trajectory was classified as unsuccessful.

Observation Space

The PPO controller received eight continuous observations describing target geometry, chassis motion, turret state, and progress through the aiming period. All geometric and velocity observations were expressed relative to the robot before being supplied to the policy. The eight observations supplied to the PPO controller are summarized in Table 2.

Observation	Description
Chassis lateral position	Robot-frame lateral target position
Chassis forward position	Robot-frame forward target position
Relative target height	Robot-frame vertical target position
Chassis lateral velocity	Robot-frame lateral chassis velocity
Chassis forward velocity	Robot-frame forward chassis velocity
Turret yaw	Current logical turret yaw
Turret pitch	Current logical turret pitch
Elapsed aim progress	Applied aiming actions divided by 30

Table 2. PPO controller Observation Space

The two chassis-velocity observations were scaled by 2.0 m/s, and turret yaw and pitch were scaled by 90°. Elapsed aiming progress was represented as the number of applied aiming actions divided by the 30-action aiming horizon. ML-Agents additionally applied learned running normalization to the observations. The 2 m/s value was used as an observation-scaling reference and does not represent the maximum training speed; PPO controller was trained up to 1.5 m/s, while 2.0 m/s was included only as an additional evaluation condition.

Including chassis velocity provided the controller with information about the motion that would be inherited by the projectile at launch. However, the policy was not directly supplied with an inverse ballistic correction. The elapsed-progress observation allowed the PPO controller to distinguish between earlier and later stages of the 1.5 s aiming period.

Action Space

The PPO policy produced three continuous actions controlling turret yaw adjustment, turret pitch adjustment, and flywheel rotational speed. Unity converted these normalized outputs into physical launcher commands, respecting the specified mechanical limits.

The yaw action produced an angular adjustment of up to ±5° per 0.05 s control timestep, while the pitch action produced an adjustment of up to ±2° per timestep. These increments corresponded to commanded limits of 100°/s for yaw and 40°/s for pitch. Logical turret yaw was constrained between -90° and +90°, while pitch was constrained between 0° and 90°. These ranges represent hard angular limits rather than the angular distance traversed during a single episode. Over the 30-action aiming period, the incremental commands allow up to 150° of cumulative yaw adjustment and 60° of cumulative pitch adjustment, subject to these limits. At the start of each episode, turret yaw was initialized to 0° and turret pitch to 45°; these initial angles were fixed rather than randomized. Initial flywheel speed is 2800 RPM.

The third action controlled flywheel speed continuously from 0 to 4000 RPM. Unlike yaw and pitch, which were incremental commands, flywheel RPM was treated as an absolute command. RPM changes were applied immediately in the current simulation without modeling motor acceleration or an actuator command queue. The three continuous actions and their corresponding physical constraints are summarized in Table 3.

Action	Physical output	Constraint
Yaw adjustment	±5° per control timestep	−90° to +90°
Pitch adjustment	±2° per control timestep	0° to 90°
Flywheel speed	Continuous absolute RPM command	0–4000 RPM

Table 3. PPO Action Space

The continuous action space allowed the policy to coordinate projectile direction and launch speed throughout the 30-action aiming period. The robot chassis velocity is kept constant (magnitude and

direction) during the aiming episode. After these actions were completed, the final yaw, pitch, and flywheel speed were used to evaluate the projectile trajectory.

For each episode:

1. Reset the turret: yaw 0°, pitch 45°
2. Randomize motion: choose chassis speed from 0-1.5 m/s and a direction.
3. Choose a location across the field: reject paths that leave the field, pass within 1.0 m of the pole, or lack a feasible shot.
4. Aim while moving: for 30 decisions (1.5 simulated seconds) the network observes the target, chassis velocity, turret angles, and timer. It adjusts yaw, pitch, and RPM.
5. Fire and score: calculate the projectile's descending target-height crossing. A horizontal error ≤0.108 m counts as a hit.
6. Learn: PPO collects episodes and updates the network using hit rewards, miss penalties and aiming rewards.
   This repeats with new locations and velocities throughout training.

Reward Function

The reward function combined feedback applied during the 30-action aiming period with a final hit-or-miss reward at the end of the episode. At each aiming action, the controller received a small control-step cost and a bounded dense shaping reward, while the terminal reward was applied only after evaluating the final projectile trajectory.

At each of the 30 aiming actions, a control-step cost of -0.001 was applied. The controller also received a proximity-based shaping reward calculated from the projectile error predicted for its current launcher settings:

$$R_{shape} = \frac{0.25}{30} e^{-d/0.108}$$

where d represents the predicted scoring error. This term provided progressively greater positive feedback as the predicted projectile trajectory approached the target. When no valid descending target-height crossing exists, the shaping error is assigned the fixed fallback value, d = 10 m, the positive reward gets calculated approximately to 0. After the 30-action aiming period, one projectile trajectory was evaluated. A successful interception, defined as a horizontal target-plane error of 0.108 m or less, received a final reward of +1. An unsuccessful shot received a bounded penalty:

$$R_{miss} = -\min\left(1, \frac{d}{3.5}\right)$$

The complete reward structure encouraged the controller to improve its predicted targeting solution while applying a small cost to each control action and penalizing unsuccessful final trajectories. The maximum possible cumulative episode reward was 1.22, calculated as

$$R_{\max} = 30\left(\frac{0.25}{30}\right) + 1 - 30(0.001) = 1.22$$

Cumulative reward was used as a learning signal and was not interpreted as an interception-accuracy percentage.

PPO Training

The PPO controller was trained using eight observations and three continuous actions. It used separate actor and critic neural networks. The actor received eight observations and contained three hidden layers of 256 neurons with Swish activation. Its output represented the three continuous launcher actions. The critic used a separate 8-256-256-256-1 network with the same hidden activation. The actor contained 134,662 trainable parameters and the critic contained 134,145, giving a combined total of 268,807 trainable parameters. Running observation normalization was enabled, while recurrent memory, convolutional layers, and observation stacking were not used.

The PPO optimizer used Adam with an initial learning rate of 3×10^{-4}, a batch size of 256, rollout buffer of 8192 samples, five optimization epochs, a time horizon of 32, discount factor γ of 0.99, and GAE parameter λ of 0.95. The initial PPO clipping parameter was 0.2 and the initial entropy coefficient was 0.0005. Linear schedules were applied to learning rate, entropy, and clipping. The complete PPO configuration is summarized in Table 4.

Parameter	Value
Algorithm	Proximal Policy Optimization (PPO)
Framework	Unity ML-Agents
Backend	PyTorch
Observations	8
Continuous actions	3
Actor hidden layers	256 × 256 × 256
Critic hidden layers	256 × 256 × 256
Activation	Swish
Trainable parameters	268,807
Initial learning rate	3×10^{-4}
Batch size	256
Rollout buffer	8192
Optimization epochs	5
Time horizon	32
Discount factor, γ	0.99
GAE parameter, λ	0.95
Initial clipping parameter	0.2
Initial entropy coefficient	0.0005
Training seed	44,78,94
Configured training budget	6,000,000 steps

Final saved step	1,700,610
Software	mlagents 1.1.0, PyTorch 2.8.0, Python 3.10.11
Hardware	Intel(R) Core(TM) i5-13450HX (2.40 GHz)

Table 4. PPO configuration

Training Curriculum

The PPO controller was trained across all permissible chassis velocities and field positions located at least 1.0 m from the target pole. During each training episode, the robot's initial position and velocity were selected randomly, as detailed in the "Action Space" sub-section. To prevent training bias and ensure uniform learning, no distinct stationary-chassis training phase was executed. Instead, the 0 m/s condition was included within the randomized velocity distribution. This approach enabled the controller to learn the complex relationships between target geometry, turret orientation, flywheel speed, and projectile interception, while also adapting to the translational momentum of the moving platform.

The PPO controller was trained for 6 million steps for three separate independent seeds to ensure the reproducibility of the result. Hit rate and miss distance showed incremental learning progress as captured in Figure 2. At the end of training budget the hit rate and miss distance on all the three seeds plateaued towards max value. To monitor intermediate progress, model checkpoints were saved every 1 million steps during the latter half of the training process (beginning at 3 million steps). Rather than employing an early termination condition, the model was permitted to exhaust the predefined training budget. The value represents a training hit-rate summary rather than independent evaluation accuracy. The frozen final policy was therefore evaluated separately after training.

Evaluation Procedure

Training performance and final controller evaluation were treated separately. After training, the final policy was frozen so that no further policy updates occurred during evaluation. The training hit rate is the proportion of successful interceptions recorded during training while the policy was still being updated whereas interception accuracy is the percentage of successful interceptions obtained after training, with the frozen policy, calculated from 1,000 evaluation episodes at each prescribed chassis speed.

$$\text{Interception Accuracy} = \frac{N_{\text{hits}}}{N_{\text{shots}}} \times 100\%$$

where N_{hits} represents the number of successful target interceptions and N_{shots} represents the total number of evaluated shots. A successful interception used the same analytical criterion as the simulator: the projectile trajectory had to cross the target-height plane from above with descending velocity with a horizontal error of 0.108 m or less.

The final evaluation protocol used 1,000 shots at nine chassis speeds: -2.0, -1.5, -1.0, -0.5, 0, 0.5, 1.0, 1.5, 2.0 m/s. The direction is kept as radially moving towards the target and radially moving away from the target (negative speeds). The same evaluation seed (44) and hit radius of 0.108 m were used across the reported conditions. Hits, misses, and interception accuracy were recorded separately for each speed using

the frozen policy. The 2.0 m/s condition was included as an additional evaluation beyond the maximum 1.5 m/s chassis speed used during training.

To capture the magnitude of compensation required across varying chassis speeds and target distances, additional metrics—specifically launch pitch, flywheel RPM, flight time, and the projectile's angle of inclination at the target—were gathered and plotted. For comparison, corresponding data were generated using a physics-based baseline, hereafter referred to as the Analytical controller. This controller utilizes inverse ballistic equations to calculate the exact theoretical pitch and RPM required to achieve a valid interception at each given chassis velocity and spatial distance.

To isolate the impact of accurate chassis-velocity observations on targeting performance, two ablation experiments were conducted using the frozen policy. In the first case, the controller was intentionally supplied with a constant chassis speed observation of 0 m/s, despite the platform actually moving at various non-zero velocities. In the second case, the controller received shuffled, incorrect chassis velocity inputs that did not correlate with the platform's actual motion. For both scenarios, interception accuracy and mean miss distance were evaluated across different actual chassis speeds. These conditions were also applied to the physics-based Analytical controller to establish a comparative baseline for performance degradation.

RESULTS

Training Performance

During training the policy was simultaneously exposed to all chassis velocities and movement vectors—including moving towards, moving away from, and translating laterally left and right relative to the target. All the three seeds were trained for exactly 6M budgeted training steps. Figure 2 shows the training progression across two parameters: hit rate and miss distance and their final value for each seed. These values indicate that the final policy achieved a high rate of successful target-plane interceptions within the training environment. However, because the statistics were collected during training, the hit rate was not used as the final independent accuracy of the controller.

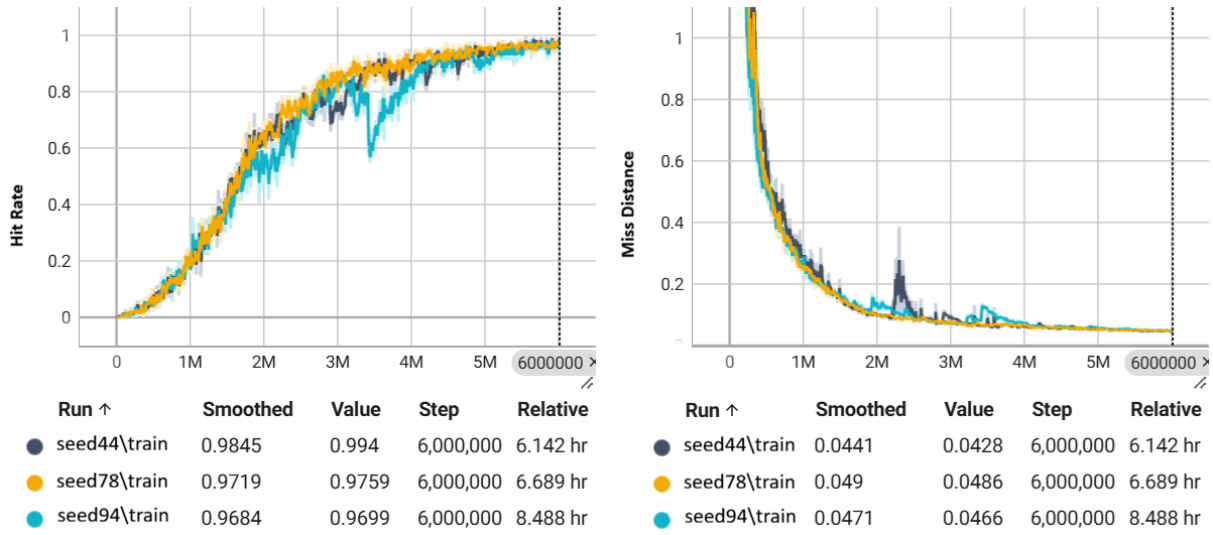


Figure 2. Training performance of PPO controller showing interception hit rate and miss distance as functions of training step

Final Frozen-Policy Evaluation

Signed Radial Chassis Speed (-ve away, +ve towards pole)	**No. of Episodes**	**N_{Hits}**	**N_{Misses}**	**Interception Accuracy**
-2 m/s	1000	752	248	75.2%
-1.5 m/s	1000	993	7	99.3%
-1.0 m/s	1000	994	6	99.4%
-0.5 m/s	1000	995	5	99.5%
0 m/s	1000	998	2	99.8%
0.5 m/s	1000	992	8	99.2%
1.0 m/s	1000	991	9	99.1%
1.5 m/s	1000	859	141	85.9%
2.0 m/s	1000	523	477	52.3%

Table 5. Evaluation data (seed 44): Interception Accuracy across chassis speeds

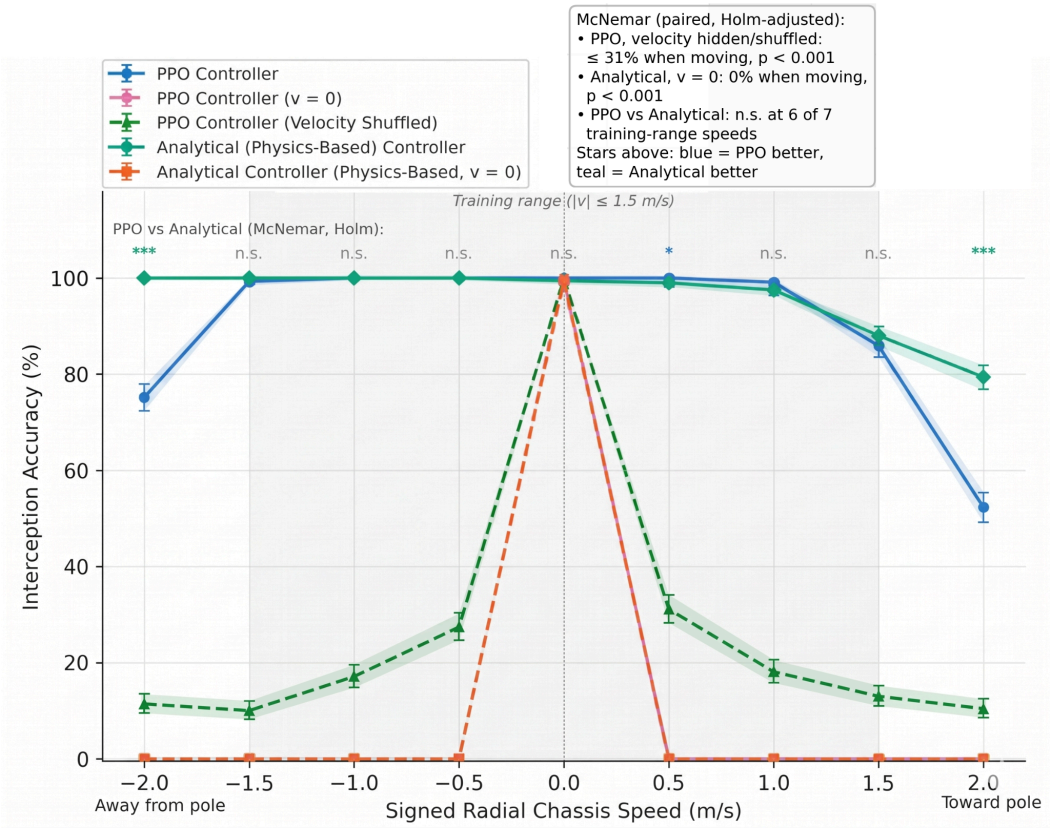


Figure 3. Evaluation data (seed 44) for Frozen-policy interception accuracy across chassis speeds. PPO Controller (v=0) & Analytical Controller (v=0) are overlapping, so only one line is visible. Pair, Holm adjusted McNemer test comparison between PPO Controller & Analytical Controller performance also indicated.

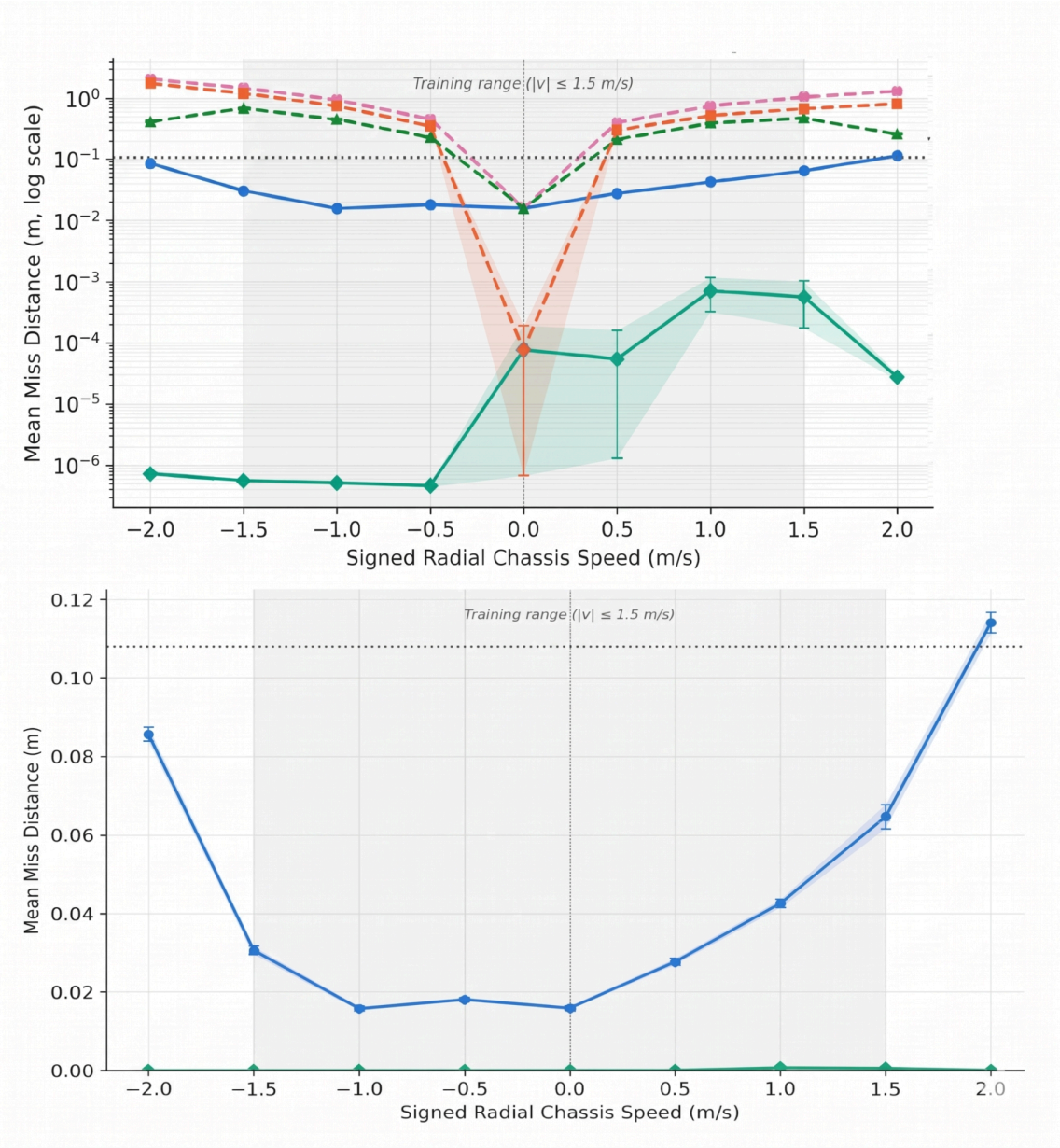


Figure 4. Evaluation data (seed 44) for Frozen-policy Mean Miss Distance across chassis speeds (bands/bars bootstrap 95% CI)

The PPO controller achieved high interception accuracy across the evaluated moving-platform conditions. In a 1000 episode evaluation run on each speed the hits and misses statistics are tabulated in table 5. The interception accuracy is >99% for stationary and slower chassis speeds. The most challenging cases are where radial distance to target are shorter and chassis speed is moving towards target with higher speed (1.5 m/s), there is dip in accuracy as well as increase in mean miss distance but still within tolerance level.

Same observation is for the 2.0 m/s condition, which exceeded the maximum training speed, the accuracy dipped to 52.3%, while performing better in moving away case of -2 m/s (75.2%). This indicates that the learned policy remained moderately effective over speeds slightly beyond training range, although the result does not establish performance at arbitrary speeds beyond the training range.

The two ablation cases for v=0 and velocity shuffled gave comparable accuracy at stationary speeds and almost zero accuracy for non-zero chassis velocity. This shows that without appropriate kinematic compensation (pitch and RPM), the projectile will mostly miss the target. This is confirmed by the corresponding plot for Analytical controller for v = 0. As illustrated in Figure 4 both the PPO controller and the Analytical controller maintained miss distances well below the 0.108 m error tolerance at all trained speeds. Conversely, while the PPO controller (v=0), PPO controller (velocity shuffled) and the Analytical controller (v=0) achieved acceptable targeting accuracy under stationary conditions, both exhibited substantial deviation from the target at all non-zero chassis velocities.

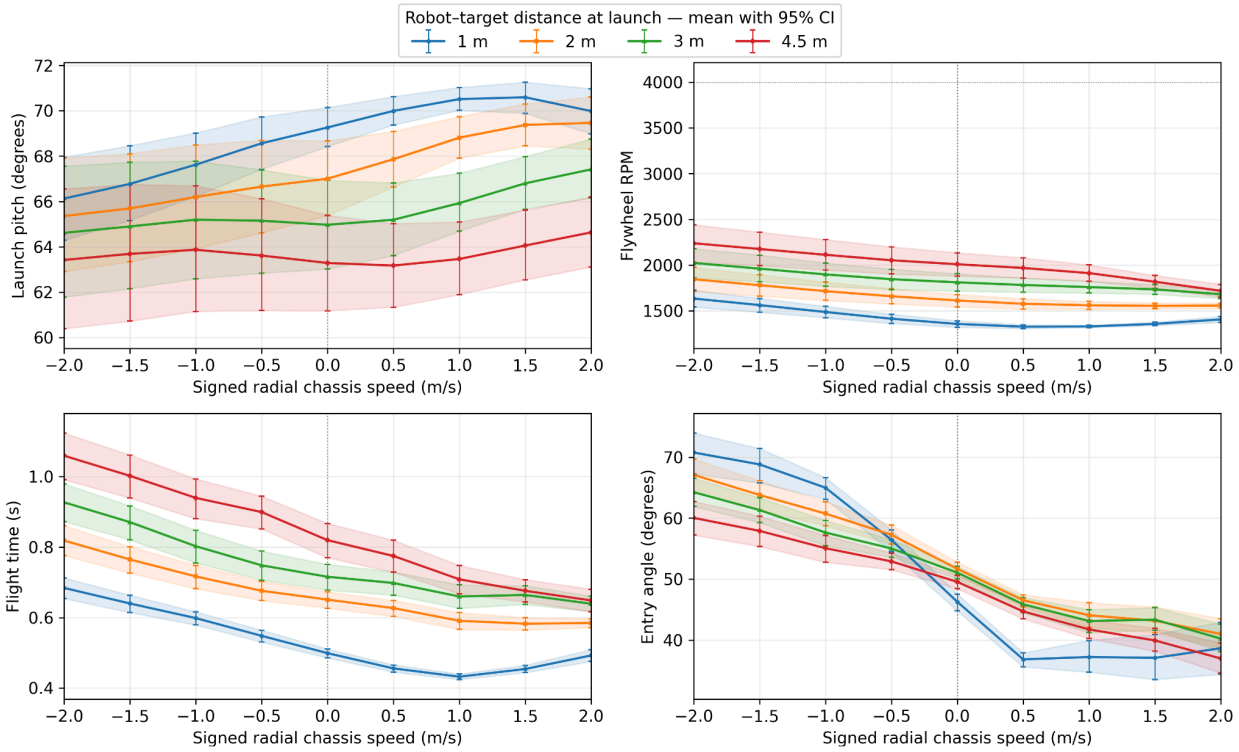


Figure 5. Evaluation data for Frozen-policy PPO controller kinematics parameters - Launch Pitch, Flywheel RPM, Flight Time and Entry Angle at target across evaluated chassis speeds

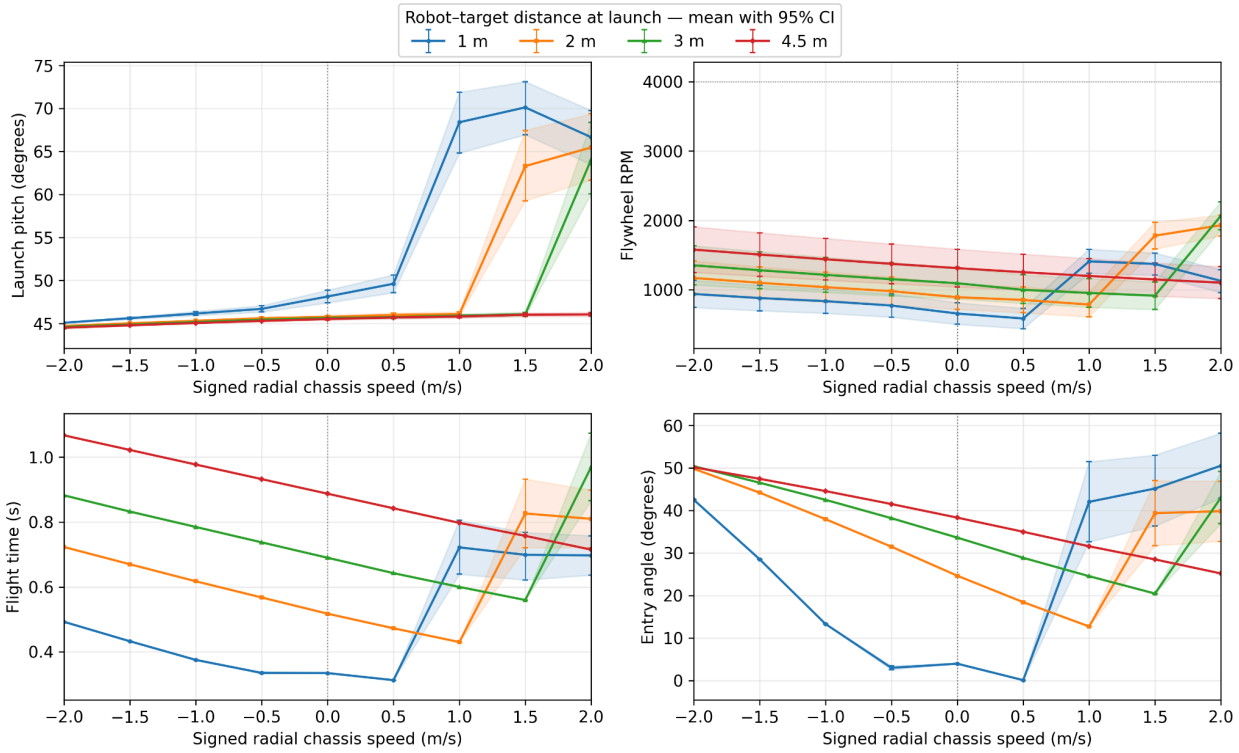


Figure 6. Analytical controller kinematics parameters across chassis speed - Launch Pitch, Flywheel RPM, Flight Time and Entry Angle at target

Figure 5 illustrates the kinematic parameters in action, plotting the policy output values (pitch, RPM and corresponding flight time and angle of inclination at target) demonstrating that the learned policy actively compensates for chassis translation across different radial distances from the target pole. Distance from target is an important parameter to study as for nearer distances policy needs to more actively compensate for inherited chassis velocity, specially the horizontal component. When the platform moves away from the target, the controller increases flywheel RPM to generate a higher exit velocity, effectively offsetting the inherited backward horizontal velocity. Conversely, as the platform moves toward the target, the controller decreases flywheel RPM to compensate for the added forward horizontal velocity. Simultaneously, the launch pitch becomes steeper as forward chassis speed increases, deliberately reducing the horizontal component of the exit velocity (proportional to $\cos\theta$). The trends are more eccentric towards speed of 1.0 and 1.5, where the policy chooses higher RPM as well as pitch and the trajectory height and flight time increases. With pitch the vertical velocity increases and horizontal decreases to compensate for increased chassis speed. These behavioral trends closely align with the baseline established by the Analytical controller (Figure 6). While the exact magnitudes of pitch and RPM differ between the two models, this variation is expected, as multiple valid combinations of pitch and RPM can successfully intercept the target under the same chassis speed and distance conditions. The distance 4.5m from pole is outside the training range, a held-out case, however the chart shows similar trends as for other distances from pole, showing the policy is able to handle near boundary cases too.

DISCUSSION

In an idealised physics-based simulation, if the target position, the robot's velocity, the projectile's exit speed, and gravity are known, one can work the projectile equations backwards to find the yaw, pitch, and flywheel speed that will hit the target. On a real robot, however, the equations are only approximately right: balls slip on the flywheel, the flywheel slows after each shot, battery voltage drops during a match, and position and velocity estimates arrive with noise and delay. An analytical controller cannot account for effects missing from its model unless each one is measured and added by hand, whereas a learned controller can, in principle, absorb them through training. We ask the question: can a learned controller discover the compensation for platform motion at all for real noise-laden situations? The ideal simulation is the right place to ask this, because the correct answer is known exactly, so any failure can be traced to the learning rather than to the physics. A controller that cannot reach high accuracy here would have little chance in a noisier environment. One that can is a credible starting point for that harder test, which this study prepares for. The experiments in this study are numerical: every shot is simulated rather than fired. This is standard practice for training learned controllers, because RL needs a very large number of trials and simulation provides them quickly, cheaply and without risk to hardware (Choi et al., 2021).

The main objective of this study was to determine whether a model-free reinforcement-learning controller could acquire projectile-targeting behaviour while accounting for platform motion. Evaluation results showed strong baseline performance. The PPO controller was trained using an observation space comprising the robot's position relative to the target (lateral and forward displacement), its relative velocity (lateral and forward chassis velocity), the target's relative height (kept fixed), and the current turret yaw and pitch. Additionally, elapsed aiming progress was included as an input parameter, enabling the controller to distinguish between the early and late stages of the 30-action aiming period. Explicitly providing chassis velocity ensured the policy had the information needed to compensate for the translational momentum the projectile inherited at launch. Ultimately, the model's robust performance demonstrates the sufficiency of this selected parameter space.

The model established the feasibility of continuous learned control of turret yaw, turret pitch, and flywheel speed. A subsequent examination of the implementation identified several environmental aspects that needed improvement, including episode handling, chassis-motion implementation, sampling validation, observation timing, and reward design. The tuned reward structure provided bounded feedback throughout the aiming period rather than relying on the much larger terminal reward. The dense shaping term rewarded reductions in predicted scoring error, while the final hit-or-miss component preserved successful interception as the main objective.

The frozen-policy evaluation showed consistently high interception accuracy across the evaluated moving-platform conditions, including at the maximum chassis speed used during training. The controller also maintained high interception accuracy at the additional 2.0 m/s evaluation condition. Because the PPO controller was trained only up to a nominal chassis speed of 1.5 m/s, this condition provides an

initial test outside the training-speed range. However, a single additional speed does not establish unrestricted generalization. A decline in targeting accuracy is observed with increase in chassis speed, and further testing would be required before attributing it to curriculum effects, policy specialization, target sampling, or another aspect of the simulation.

RL results can change substantially between training runs that differ only in their random seed keeping all the other parameters the same, which makes single-run results hard to trust (Henderson et al., 2018). For this reason, the controller in this study was trained with three independent seeds. Evaluation results and interception accuracy across various chassis speeds remained consistent across all three runs, demonstrating the maturity and reproducibility of the learned model.

The current study examines whether a model-free policy can learn the required mapping from target geometry and robot state to continuous launcher actions. While the PPO controller received the kinematic information necessary to respond to chassis motion, it was never explicitly provided with the inverse ballistic solution required to calculate the correct action. Although PPO is fundamentally model-free—meaning the policy itself does not rely on a ballistic model to select actions—the training process did utilize model-derived information via the dense shaping reward. At each aiming step, this reward was calculated using the predicted projectile error for the current launcher settings. For future deployment on physical hardware, this simulator-derived feedback would naturally be replaced by real-world sensor measurements to guide the training process.

Limitations

The study was conducted entirely in simulation. The targeting results therefore describe performance within the implemented Unity environment and do not represent measured accuracy on a physical FTC robot. Projectile flight was modeled using analytical point-particle kinematics with constant gravity and inherited planar chassis velocity. The model did not include aerodynamic drag, projectile spin, Magnus forces, flywheel slip, mechanical vibration, or projectile or launcher deformation. The current simulation assumes immediate execution of flywheel RPM commands and does not include motor acceleration or an actuator command queue. The reported scope did not include sensor noise, actuator latency, or battery-related variation affecting flywheel speed.

The scoring criterion was simplified. A successful interception was defined as a projectile trajectory crossing the target-height plane from above and intersecting within 0.108 m horizontally of the target centre. This criterion does not model physical entry into a finite three-dimensional target or collision with a real scoring structure.

Future Work

The benefits of developing a Reinforced Learning based approach are manifold. Real-world environments are heavily influenced by non-ideal interferences such as sensor inaccuracy, wheel slip, battery drain, and aerodynamic drag. Formulating explicit mathematical models for each of these effects is notoriously

difficult, and specific parameters are often left entirely unscoped. This complexity increases even more when these environmental interferences interact. Implementing such math-heavy analytical solutions in a real-time, low-latency environment requires millions of time-critical calculations. This significantly increases the system's compute, storage, and overall costs. In contrast, a pre-trained reinforcement-learning model in the real world environment can act as a universal approximator, which naturally internalizes these compounding variables to provide a workable, low-cost solution for low-duty robotic applications.

Building upon the foundation established in this study, subsequent work is currently ongoing to incorporate hardware-inspired imperfections into the simulation. Specifically, this includes sensor noise perturbing target and chassis-state observations, actuator latency delaying the execution of turret commands, and variations in available flywheel speed representing battery or launcher-performance changes. Because evaluating these variables falls outside the scope of the present study, they will be introduced as separately defined experiments rather than assumed to be part of the current model. The findings from this expanded scope will be published subsequently once the methodology and results are proven.

CONCLUSION

This study investigated model-free reinforcement learning for projectile targeting from a translating omnidirectional robotic platform. A Unity-based simulation combined analytical projectile motion with a PPO controller that selected continuous turret yaw, pitch, and flywheel-speed commands.

The PPO controller is provided with a continuous set of observation parameters representing the robot's spatial position, kinematic motion, and elapsed aiming progress within an episode. A shaped reward function provides the necessary feedback to guide and improve the learning process. The controller was trained across the complete range of chassis velocities and target positions, strictly bounded by predefined field and motion constraints. Subsequent frozen-policy evaluation demonstrated high interception accuracy across all tested moving-platform conditions.

The study demonstrates that a model-free PPO controller can learn to coordinate turret orientation and projectile launch speed while accounting for chassis motion in a simulated continuous-control task. The results provide a foundation for eventual evaluation on physical robotic hardware. Furthermore, it paves the way for further work in real-world scenarios, where a reinforcement learning policy can implicitly model complex hardware imperfections to provide a low-cost, effective control solution.

ABOUT THE AUTHOR

Aditya Rastogi is a class 12 student at The Heritage Xperiential Learning School, Gurugram, India. He takes interest in robotics and machine intelligence, with a specific focus on the evolution and application

of pervasive machine assistance. He intends to pursue engineering, with focus in information systems and mechatronics.

REFERENCES

Bengio, Y., Louradour, J., Collobert, R., & Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning* (pp. 41–48). Association for Computing Machinery. https://doi.org/10.1145/1553374.1553380

Choi, H., Crump, C., Duriez, C., Elmquist, A., Hager, G., Han, D., Hearl, F., Hodgins, J., Jain, A., Leve, F., Li, C., Meier, F., Negrut, D., Righetti, L., Rodriguez, A., Tan, J., & Trinkle, J. (2021). On the use of simulation in robotics: Opportunities, challenges, and suggestions for moving forward. *Proceedings of the National Academy of Sciences, 118*(1), Article e1907856118. https://doi.org/10.1073/pnas.1907856118

For Inspiration and Recognition of Science and Technology. (2026). *2025–2026 FIRST® Tech Challenge competition manual* (Team Update 32). ttps://ftc-resources.firstinspires.org/ftc/archive/2026/game/manual

Gfrerrer, A. (2008). Geometry and kinematics of the Mecanum wheel. *Computer Aided Geometric Design, 25*(9), 784–791. https://doi.org/10.1016/j.cagd.2008.07.008

Han, D., Mulyana, B., Stankovic, V., & Cheng, S. (2023). A survey on deep reinforcement learning algorithms for robotic manipulation. *Sensors, 23*(7), Article 3762. https://doi.org/10.3390/s23073762

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., & Meger, D. (2018). Deep reinforcement learning that matters. *Proceedings of the AAAI Conference on Artificial Intelligence, 32*(1). https://doi.org/10.1609/aaai.v32i1.11694

Hu, Z., Xiao, L., Guan, J., Yi, W., & Yin, H. (2023). Intercept guidance of maneuvering targets with deep reinforcement learning. *International Journal of Aerospace Engineering, 2023*, Article 7924190. https://doi.org/10.1155/2023/7924190

Hwangbo, J., Lee, J., Dosovitskiy, A., Bellicoso, D., Tsounis, V., Koltun, V., & Hutter, M. (2019). Learning agile and dynamic motor skills for legged robots. *Science Robotics, 4*(26), Article eaau5872. https://doi.org/10.1126/scirobotics.aau5872

Juliani, A., Berges, V.-P., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., & Lange, D. (2020). *Unity: A general platform for intelligent agents* (arXiv:1809.02627v2). arXiv. https://doi.org/10.48550/arXiv.1809.02627

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., & Hadsell, R. (2017).

Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences, 114*(13), 3521–3526. https://doi.org/10.1073/pnas.1611835114

Kober, J., Bagnell, J. A., & Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research, 32*(11), 1238–1274. https://doi.org/10.1177/0278364913495721

Le, H., Saeedvand, S., & Hsu, C.-C. (2024). A comprehensive review of mobile robot navigation using deep reinforcement learning algorithms in crowded environments. *Journal of Intelligent & Robotic Systems, 110*, Article 158. https://doi.org/10.1007/s10846-024-02198-w

Li, B., Jin, T., Zhu, Y., Li, H., Wu, Y., & Zhao, D. (2021). Moving target shooting control policy based on deep reinforcement learning. In *2021 8th International Conference on Information, Cybernetics, and Computational Social Systems (ICCSS)* (pp. 183–188). IEEE. https://doi.org/10.1109/ICCSS53909.2021.9722012

Ng, A. Y., Harada, D., & Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning* (pp. 278–287). Morgan Kaufmann.

Peng, X. B., Andrychowicz, M., Zaremba, W., & Abbeel, P. (2018). Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 3803–3810). IEEE. https://doi.org/10.1109/ICRA.2018.8460528

Recht, B. (2019). A tour of reinforcement learning: The view from continuous control. *Annual Review of Control, Robotics, and Autonomous Systems, 2*, 253–279. https://doi.org/10.1146/annurev-control-053018-023825

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). *Proximal policy optimization algorithms* (arXiv:1707.06347). arXiv. https://doi.org/10.48550/arXiv.1707.06347

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (2nd ed.). MIT Press.

Tang, C., Abbatematteo, B., Hu, J., Chandra, R., Martín-Martín, R., & Stone, P. (2025). Deep reinforcement learning for robotics: A survey of real-world successes. *Annual Review of Control, Robotics, and Autonomous Systems, 8*, 153–188. https://doi.org/10.1146/annurev-control-030323-022510

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., & Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 23–30). IEEE. https://doi.org/10.1109/IROS.2017.8202133

Zeng, A., Song, S., Lee, J., Rodriguez, A., & Funkhouser, T. (2020). TossingBot: Learning to throw arbitrary objects with residual physics. *IEEE Transactions on Robotics, 36*(4), 1307–1319. https://doi.org/10.1109/TRO.2020.2988642