ARL-TR-10286 ● FEB 2026

DEVCOM ARMY RESEARCH LABORATORY

Application of Reinforcement Learning to the Problem of Multiple UAVs Attacking a Counter-UAV Turret

by James Humann, Michael Dorothy, Stephen Nogar, and Christopher Kroninger

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TR-10286 ● FEB 2026

Application of Reinforcement Learning to the Problem of Multiple UAVs Attacking a Counter-UAV Turret

James Humann, Michael Dorothy, Stephen Nogar, and Christopher Kroninger
DEVCOM Army Research Laboratory

REPORT DOCUMENTATION PAGE

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
		START DATE	END DATE
February 2026	Technical Report	1 January 2022	31 December 2025

4. TITLE AND SUBTITLE
Application of Reinforcement Learning to the Problem of Multiple UAVs Attacking a Counter-UAV Turret

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
5d. PROJECT NUMBER	5e. TASK NUMBER	5f. WORK UNIT NUMBER

6. AUTHOR(S)
James Humann, Michael Dorothy, Stephen Nogar, and Christopher Kroninger

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
DEVCOM Army Research Laboratory ATTN: FCDD-RLA-WF Aberdeen Proving Ground, MD 21005	ARL-TR-10286

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited.

13. SUPPLEMENTARY NOTES
ORCID: James Humann, 0000-0003-3858-3873; Michael Dorothy, 0000-0002-0006-7462; Stephen Nogar, 0000-0002-5233-6767; Christopher Kroninger, 0000-0002-5489-8042

14. ABSTRACT
This report explores the use of reinforcement learning (RL) in the development of collaborative strategies for multiple unmanned aerial vehicles (UAVs) to attack a counter-UAV turret. We first validate the use of RL by recreating the optimal one- and two-UAV strategies and comparing the RL strategies to the theoretical best strategies and a selection of naïve strategies. We then discuss how the simplifications used in modeling and training could be applied to higher numbers of UAVs. In the case of the 1 vs. 1 game, tabular Q-learning was used to train an agent that could win 95.6% of the games that an optimal agent could win. This was 18.6% more than a naïve approach that guided an agent straight at the turret. In the more complex 2 vs. 1 case, a soft actor-critic algorithm was trained that could win 99.4% of the games that an optimal agent won. The success of the RL-trained agents serves as a proof of concept for applying RL to this domain. It is a promising direction for future work in deriving successful strategies for 3 vs. 1 and higher games. The RL algorithms, parameters, and modeling approach reported here should transfer to speed training in this problem space.

15. SUBJECT TERMS
multiagent systems, reinforcement learning, autonomy, game theory, simulation

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT	b. ABSTRACT	C. THIS PAGE		
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UU	35

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER
James Humann	(520) 691-5069

STANDARD FORM 298 (REV. 5/2020)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

List of Tables

Executive Summary

This report explores the use of reinforcement learning (RL) in the development of collaborative strategies for multiple unmanned aerial vehicles (UAVs) to attack a counter-UAV turret. Optimal strategies for one or two UAVs, derived from differential game theory, have been reported in the literature, but no results have been reported for three or more UAVs. We first validate the use of RL by recreating the optimal one- and two-UAV strategies and comparing the RL strategies to the theoretical best strategies and a selection of naïve strategies. We then discuss how the simplifications used in modeling and training could be applied to higher numbers of UAVs.

In the case of the 1 vs. 1 game, tabular Q-learning was used to train an agent that could win 95.6% of the games that an optimal agent could win. This was 18.6% more than a naïve approach that guided an agent straight at the turret. In the more complex 2 vs. 1 case, a soft actor-critic algorithm was trained that could win 99.4% of the games that an optimal agent won.

The success of the RL-trained agents serves as a proof of concept for applying RL to this domain. It is a promising direction for future work in deriving successful strategies for 3 vs. 1 and higher games. The RL algorithms, parameters, and modeling approach reported here should transfer to speed training in this problem space.

1. Introduction

Groups of unmanned aerial vehicles (UAVs) will be deployed to autonomously attack opposing forces in future battlefields. These groups will be more effective if they operate collaboratively, possibly sacrificing some group members so that others are successful. One concept to counter UAVs is to use an autonomous turret that can aim at them and shoot them down. This report explores the concept of using groups of UAVs to overwhelm a counter-UAV turret through strategic maneuvering.

Under simplified assumptions of constant-velocity linear motion of the UAVs and constant angular velocity of a turret, the optimal strategies for a one UAV versus turret engagement (i.e., "1 vs. 1 game") have been found in Von Moll et al.[1] The optimal 2 vs. 1 game strategies have also been found. Provably optimal strategies for more than two attackers have not been reported. While prior attempts have used differential game theory to produce analytical results, here we attempt to use reinforcement learning (RL). In this report, we use RL in an attempt to recreate the control strategies for the 1 vs. 1 and 2 vs. 1 cases and lay out a strategy for using RL to develop strategies for higher N vs. 1 games.

1.1 Reinforcement Learning

RL is a branch of artificial intelligence. It provides mathematical frameworks for learning from experience.[2] The basis of most RL models is a state–action–reward loop. An agent analyzes its state and takes an action. As a result, it receives a reward and is transported to a subsequent state. The agent's goal is to maximize the rewards received over the lifetime of the game. The consequences of actions must be learned by trial and error through interaction with the environment, which differentiates RL from classical control methods.[2] RL training approaches are focused on estimating the value of taking specific actions in specific states. They often rely on machine learning (ML) to estimate rewards or decide which actions to take.[2]

To formalize an RL problem, we define a policy, a reward signal, and a value function. (More complex approaches may also define a model of the environment for planning.) The policy defines how an agent behaves, mapping states to actions. The reward signal reflects the goal, and the value function predicts the long-term total rewards that an agent might receive for taking specific actions in specific states.[3] These fundamental RL elements are illustrated in Figure 1.

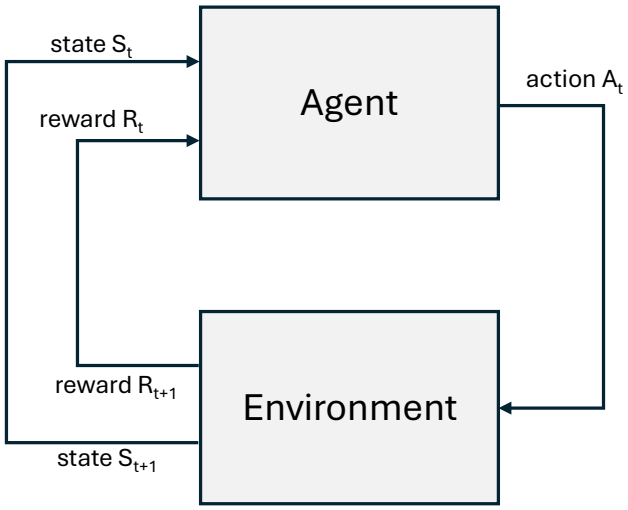

Figure 1. Depiction of the classical RL framework, with an agent that takes an action at each timestep, receives a reward, and observes a new state to inform its action at the next timestep. Adapted from Sutton and Barto.[3]

In mathematical notation, the agent observes the state of the environment $S_t \in \mathcal{S}$, where $\mathcal{S}$ is the set of all possible states. Given this observation, it performs an action $A_t \in \mathcal{A}$, where $\mathcal{A}$ is the set of possible actions. The state–action pair results in a reward R_{t+1} and transitions to a new state S_{t+1}.[3] The action is selected by a policy $\pi: \mathcal{S} \rightarrow \mathcal{A}$.

Basic approaches take discretized values as input and output discretized values. More advanced approaches, such as those for controlling robots, have continuous inputs and outputs. One such approach, deep deterministic policy gradient, can take continuous inputs with no knowledge of the underlying model, and give continuous outputs to solve many simulated physics tasks.[4] Other algorithms can similarly act as black boxes, mapping inputs to outputs with no internal model of the environment or even the agent.

The two main approaches to RL are *value function*-based methods and *policy search*-based methods.[2] A value function estimates the expected cumulative return of applying a policy from a given starting state (which includes estimates of the future states that will be visited and their values). Sampling methods are used to estimate the value function for any state–action pair, and an RL agent can take actions that maximize the expected value. Policy search methods do not build up a value function, but instead directly train an optimal policy.[2] These often rely on parameterization of the action space. If a gradient in reward is found with respect to the parameters, this gradient can be used for optimization of the policy. Estimating the gradient using strategic sampling or advanced techniques such as ML is the focus of most of these approaches. *Actor-critic* methods combine the

value function and policy search approaches. The critic learns a value function that in turn is used to calculate gradients for the actor's policy search.[2]

Model-free RL methods can be quite expensive because of the number of samples required for training. Even simple models may require millions of samples, and this number grows rapidly as dimensions are added to the input.[5] The samples obtained are also highly dependent on the agent's actions in its environment, and there are long-term dependencies that blur the relationship between action and reward. Nonetheless, it has been widely applied to problems from board games to classification to purely visual robotic control with great success and remains one of the most successful model-free learning methods in the literature.[2]

RL can also operate on visual inputs, not relying on access to underlying physical parameters.[2] For example, models have been trained to play video games simply by processing the pixels of the games' graphics and outputting control commands,[3,6] just as a human would do when processing visual input.

The main RL types that we use in this paper are Q-learning (and its offshoots), and soft actor-critic (SAC). Q-learning relies on a lookup table that contains the expected value of taking a given action in a given state. Advanced techniques focus on estimating the values in this Q-table, so as not to rely on a discretized action and/or state space that grows exponentially. The SAC algorithm operates in continuous state–action spaces and was first described in Haarnoja et al.[5] Its goal is to simultaneously maximize expected reward and entropy (random behavior). This incentivizes exploration of the state–action spaces during training.

1.2 Related Work

The SAC algorithm, as described in Haarnoja et al.,[5] was used to achieve state-of-the-art performance in diverse continuous control benchmarks. These benchmarks came from the OpenAI gym benchmark suite and the 21-dimensional humanoid control task.[7,8] These benchmarks test the ability to control multijointed robots, with the SAC algorithm performing the best in four out of six benchmarks, and nearly the best in the other two. It was compared against deep deterministic policy gradient, proximal policy optimization, twin delayed deep deterministic policy gradient, and soft Q-learning.

The Deep Q Network (DQN) is an RL approach that replaces a tabular Q-value table with a deep neural network that approximates the state–action–value mapping.[2,9] It successfully demonstrated visual control of diverse Atari video games. Its use of a deep neural network to replace the value function in a classical RL model is illustrative of a common tactic in combining ML and RL.

We build on the work in differential game theory by Von Moll et al.[1] In that work, the authors showed that optimal strategies could be derived for a turret and attacking UAV in a confrontation where the UAV wins if it enters a prescribed radius around the turret, and the turret wins by aligning its boresight with the UAV before this happens. In the same article, the game is extended to the two UAVs versus one turret scenario, and game-theory optimal strategies are similarly derived. This analysis was expanded to games with partial information, where the turret does not know the attackers' maximum speed.[10] In Shiskika et al.,[10] it was shown that the attackers could force the turret into a dilemma by masking their true maximum speed. A slightly different formulation was used in Von Moll et al.[11] to derive optimal strategies when the turret and attacker are simply attempting to sense one another in a finite time period.

2. RL Applied to the 1 vs. 1 Game

The one drone versus one turret game has a provably optimal solution from differential game theory, as given in Von Moll et al.[1] The UAV can be proved to win when it enters a circle, called the ν-circle. An attacking agent's optimal move is to fly at full speed toward the tangent of the ν-circle. There are always two tangents, and the agent should choose the tangent that is away from the boresight of the turret.

We attempted to recreate this finding using a naïve RL approach as a learning exercise to discover good hyperparameters, scaling, normalization, and bounds.

2.1 Introduction to 1 vs. 1 Game

The setup for the game is given in Figure 2. An attacking UAV at point A is attempting to impact a turret at the origin (point O). The turret is pointed at the angle β. The angle from the x-axis to line OA is the angle of the attacker. The angle θ represents the difference between this angle and the turret's look angle.

The turret can rotate with rotational velocity ω rad/s. The UAV can move with velocity v m/s. The quotient v/ω, which has units of length, is an important quantity, which we name ν (Greek letter nu). When the UAV is a ν-length radius away from O, its maximum angular velocity about O (obtained by traveling perpendicular to the segment OA) will be equal to ω. Thus, outside a circle of radius ν, the turret can always reduce θ by turning toward the UAV, and inside the circle, the UAV can always outrun the turret's look angle.

The UAV and turret have competing goals as follows:

- Turret's win condition:

- The turret attempts to reduce the angle θ to 0. It is assumed that a turret would then fire a bullet or laser at an oncoming UAV, destroying it.

- UAV's win condition:
  - Nominally, the UAV must reach point O to destroy the turret. In a real battle, this would conclude with the detonation of an explosive or simply crashing into the turret to disable it.
  - We can identify a winning region, rather than a point. The winning region is anywhere within the ν circle. Once inside this circle, the UAV has a guaranteed victory, because it can maintain a rotational velocity about O that is higher than ω. Thus, in this region, it can circle around the turret indefinitely without being caught and attack at will.
  - So, for the purposes of our analysis, we will consider the UAV to be victorious, and the turret to have lost, at the moment the UAV reaches the ν circle.

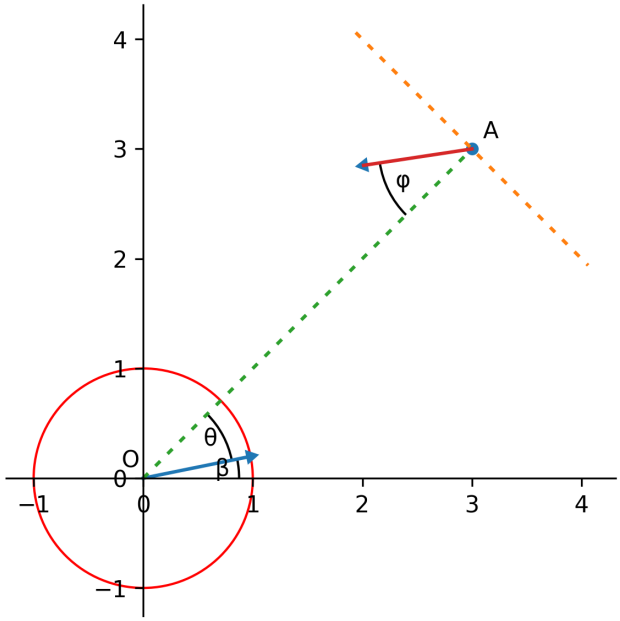

Figure 2. Setup of the 1 vs. 1 game. The turret's look angle is β. The angle from the look angle to segment OA, which includes the attacker, is θ. The agent's action space is its choice of movement angle φ. All coordinates are normalized so that the ν-circle has radius 1. All angles are measured positively counterclockwise.

The variables used are given in Table 1.

Table 1. Variables in 1 vs. 1 game.

Symbol	Meaning	Units	Note
β	Turret's look angle	rad	$(0 – 2\pi)$
θ	Angle from turret's look to UAV	rad	$(-\pi – \pi)$
v	Velocity of UAV	m/s	. . .
ω	Angular velocity of turret	rad/s	. . .
ν	Radius where turret and UAV have equal angular velocity	m	Greek letter nu, not English v. Calculated as v/ω
r	Radius from origin to the UAV	m	Length of segment OA

2.2 Scaling the 1 vs. 1 Game

The "curse of dimensionality"[3] can be mitigated by normalization and scaling of the game's parameters such that large equivalence classes can be found whose solutions are the same. We build on the normalization in Von Moll et al.[1] by normalizing all distances to the radius of the ν-circle. We also normalize time such that the turret needs one time unit to turn by 1 radian. This normalization also results in the UAV's speed being 1 in the normalized units. This greatly reduces the space of equivalent games.

2.3 RL Setup

We apply one of the most straightforward classical RL techniques to the 1 vs. 1 game here to serve as a proof of concept for the model, simulation, and application of RL in general. This exercise will also help us learn proper hyperparameters and levels of discretization that may be helpful in future analyses of more complex problems.

Q-Learning is a tabular RL approach that can be applied to discrete statespaces, where the agent has a set of discrete possible actions. The policy maps the state of the environment to an action that the agent should take at any given timestep. The action affects environment, changing the state, and thus possibly changing the optimal action for the next state. This is a closed-loop learning process, where the optimal actions are not known a priori; they are only discovered through repeated interaction with the environment. The agent must explore (take new actions to discover new states) and exploit (search deeply within a promising set of states) to determine optimal policies[3] and build the Q-table.

Our approach in this section is completely discretized. The timesteps in Figure 1 are discrete. The environment can only assume discrete states, and there is a discrete list of actions that the agent can perform. Because our domain is actually continuous, we must discretize the states and actions to apply this framework. Note

that the discretization is internal to the agent and only used to define the policy π. The policy will still be trained and tested in a continuous simulated world.

Through informal testing, we determined the following discretization parameters were a suitable tradeoff between training time and performance as shown in Table 2.

Table 2. Discretization parameters.

Parameter	Value	Note
Maximum radius	8	Rounded up from $1 + 2\pi$, allowing the turret to sweep the entire field before an agent would reach the ν-circle
No. radius levels	102	Nominally 100 brackets, with extra levels for $r < 1$ and $r > 8$
No. θ levels	59	Nominally 60 bracket limits, with 59 brackets between them
φ range	$[0 - \pi/2]$	See Section 2.4 for justification
No. φ levels	20	. . .

Upon experimentation, we saw that finer discretization for r was much more important at low values (near 1) than high values. This makes sense, because at low r, the state can change quickly, the ν-circle subtends a much greater angle from the agent's point of view, and the agent has more ability to outrun the turret by choosing large values for φ, making the choices in this region more varied and impactful. So we chose to do an exponential discretization, where the buckets were narrower at low r and wider at high r. We discretized θ into buckets of uniform size. See Table 3 for full discretization details.

Table 3. Discretization of angle and radius.

Angle, θ		Radius	
Range	**Discretization**	**Range**	**Discretization**
(0–0.0532)	1	< 1	0
(0.0532–0.106)	2	(1–1.021)	1
(0.106–0.160)	3	(1.021–1.042)	2
. . .	. . .	. . .	. . .
(3.035–3.088)	58	(7.516–7.674)	98
(3.088–3.142)	59	(7.674–7.835)	99
. . .	. . .	(7.835–8)	100
. . .	. . .	8 +	101

The central process of Q-learning is to learn the Q-table, which contains the value of any state–action pair (i.e., the value of taking a given action in a given state). This Q-value incorporates not just the immediate reward, but information about possible future rewards that can be gained in reachable states.

Again, there is a Q-value for every state–action pair in the Q-table. So, we can define a function over discretized values of θ and *r* that gives a Q-value: $Q(S_t, A_t)$. Before training, we do not know the entries in this Q-table. We can initialize them all to 0 and begin estimating them as we simulate the turret game and collect rewards and penalties. The total reward expected over a game is $G_t = R_{t+1} + R_{t+2} + R_{t+3} \ldots R_N$, where the game ends after N-1 steps. Since the rewards are state-and-action dependent, we cannot calculate any of the terms past R_{t+1} with certainty. We can at least assume that we are following an optimal policy, so that we choose the action with the highest expected reward in each state. We can also discount future rewards, due to their uncertainty. With discounting, our value function then becomes: $G_t = R_{t+1} + \gamma^1 R_{t+2} + \gamma^2 R_{t+3} \ldots \gamma^{N-1} R_N$, where $\gamma < 1$. This leads to an interesting iterative relationship where $G_t = R_{t+1} + \gamma G_{t+1}$. This iterative formulation can be exploited by using estimates of the value of future states to inform the value of the current state.[3]

Using a state as an index to the Q-table will access a list of Q-values. These Q-values are indexed by the actions the agent can take. These Q-values are equivalent to the total discounted reward an agent can expect from taking the indexed action in the indexed state and following an optimal policy in subsequent states. Because an agent will observe a state transition after taking action, it can use the Q-value of the resulting state to estimate the Q-value of the original state[12]:

$$Q(s_t, a_t) \leftarrow (1 - \alpha) Q(s_t, a_t) \; + \; \alpha (r_{t+1} + \gamma \max_a Q(s_{t+1}, a)).$$

The Q-values can be updated iteratively using this equation every time the agent performs an action. By learning the proper Q-values for every state–action pair, the agent can formulate an optimal policy that always chooses the available action with the highest Q-value for its current state. The variable α is called the learning rate, and it weights the new reward and experience versus the original estimate that already exists in the Q-table. A high learning rate will emphasize quickly learning from experience, at risk of overfitting and oscillations. A low learning rate is more stable, but may take longer to converge to optimal policies. In the extremes, a learning rate of 0.0 would never learn from experience, and 1.0 would forget all prior experience and treat every new experience as the ground truth. A balanced α value is an important learning hyperparameter.

2.4 State-Space Reduction and Discretization

By normalizing the distances and turret's angular velocity (and consequently the UAV's velocity), we are left with only the angles θ and β, and the radius *r* (in normalized units) to describe the state of the system. We also recognize that there is a rotational indifference in the game. The absolute angles of the players do not

matter, only the relative angle between them. We can thus perform our analysis by ignoring β and treating *r* and θ as the only relevant variables.

Finally, by inspection we can see that the only logical angles for φ are in the range 0°–90°. Less than 0°, the UAV will be traveling toward the turret's look angle. More than 90°, the UAV will be traveling away from the goal. This also leads to the simplification of symmetry for negative θ values. The optimal φ value for a given θ value should have symmetry, so the negative of the optimal φ is optimal for the negative of θ. This allows us to reduce our range for θ and φ to give the same resolution with fewer buckets, thus speeding training.

2.5 Limits

The maximum radius of 8 for training was very conservative, meant to capture any configuration where a naïve look at the problem would indicate the possibility of a UAV win. A more restrictive analysis is based on the geometry of Figure 3. As the turret travels the angle γ, the UAV also travels a linear distance γ due to the normalization. If γ is chosen such that it sweeps the entire range of the UAV, the UAV cannot win. Through numerical computation, we determined that the maximum value for γ (and radius of the UAV's range) is 0.9322, corresponding to a starting radius for the UAV of 5.074 to ensure that the UAV range does not overlap the winning region. (Even this is slightly optimistic for the UAV, as we will see in testing optimal strategies).

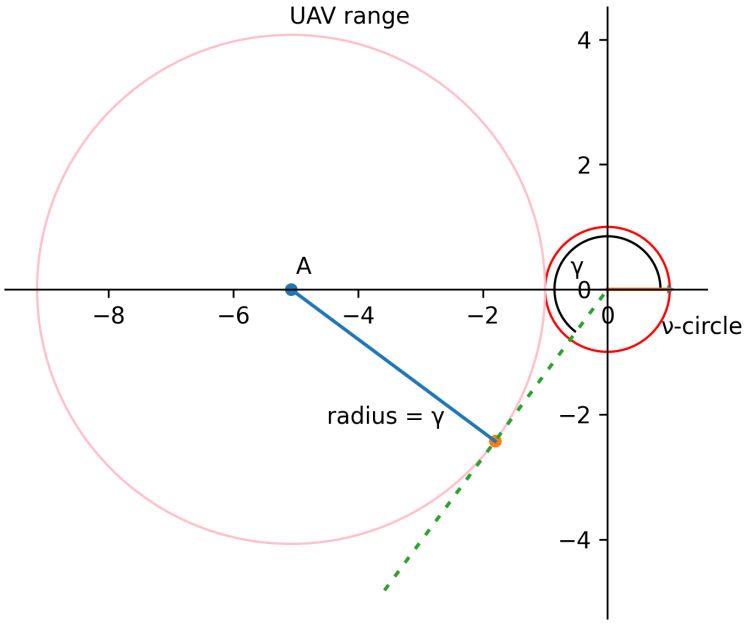

Figure 3. Determination of maximum starting radius of the UAV for a possible win. The radius of the UAV range is equal to the angle traveled by the turret in normalized coordinates.

2.6 Training Hyperparameters and Simulation

We used ε-greedy exploration, with linear ε decay. We used a learning rate of α to update the Q-table values. We also decayed α. We trained for 4,800,000 simulated games with a γ discount rate of 0.96.

We did some reward shaping to prevent uncontrolled growth of the Q-values. To shape the rewards, we gave a reward at each step for the amount that the distance to the origin was reduced. We clipped the target Q-value (estimated Q-value of the best action in the next state) to be no less than -2π. This was a response to runaway Q-values in very disadvantageous states, where the UAV might transition from a state to the same state (due to discretization), while collecting a penalty for getting hit, and adding it to the target value that already anticipates getting hit in that state. We also gave a penalty of -2π for getting hit by the turret (and ended the game). We gave a reward of $\pi + \mathrm{abs}(\theta)$, rewarding the agent for not only reaching the ν-circle, but doing so with the maximum angular distance between it and the turret.

To simulate, the UAV made a control decision, and all the agents' positions were updated every 0.02 normalized time units.

2.7 Training Results

After training, we ran the trained agent through a test battery against three other control strategies:

- The game-theory optimal behavior from Von Moll et al.[1]
- An agent that always moves straight into the origin
- An agent that moves randomly

We used initial conditions of *r* linearly spaced at 40 points from 1.1 to 5.0, and θ linearly spaced at 200 points from 0 to 6.252, for a total of 8,000 tests. The angle β always starts at 0°. The UAV was placed at the coordinate to begin the game, and then moved in timesteps of 0.02 in normalized time. Reaching the η-circle counted as a win for the UAV. Getting hit by the turret counted as a loss. Ties were awarded to the turret.

We show the results for these initial conditions given the various control strategies in Figure 4 and the learned Q-values in Figure 5. The best-performing agent is the game-theory optimal agent, as expected. The RL agent is second best. It won every game that the straight and random agents won, and then some. The total number of wins using RL increased by 18.6% over the greedy straight in algorithm. The widest winning region in Figure 4, in orange, is captured by the game theory agent, and is

a cardioid shape, as derived in Von Moll et al.[1] The total number of wins for each strategy are given in Table 4.

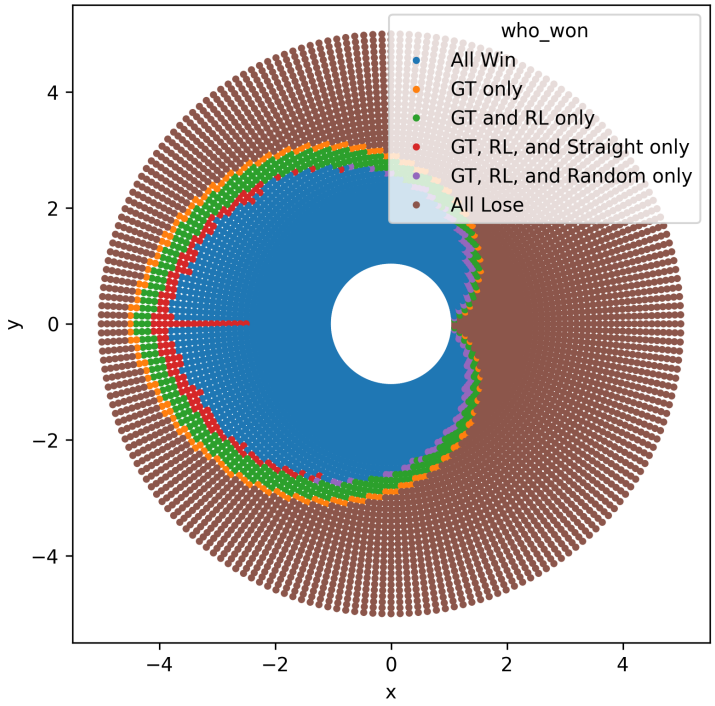

Figure 4. Results of various control strategies from given initial conditions. The color of the point gives the result of the game when a UAV starts at that point. The turret always starts at an angle of 0°.

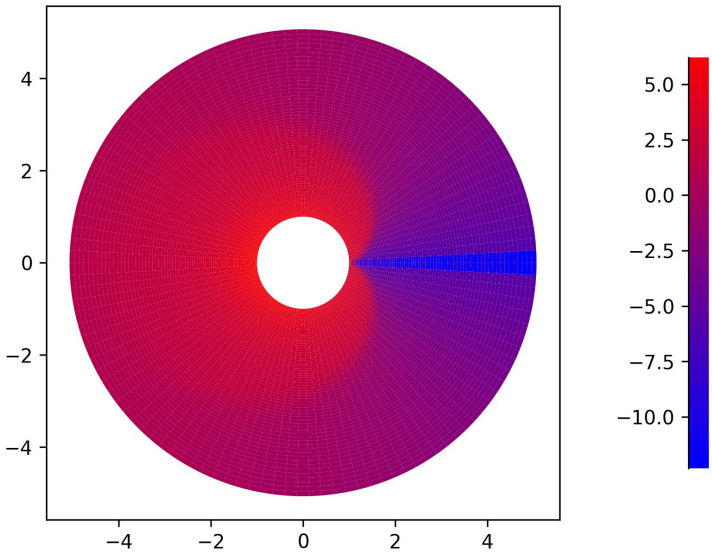

Figure 5. Q-values from the trained Q-table for a given radius and θ.

Table 4. Results of RL in the 1 vs. 1 game.

Strategy	Wins/8000	Note
Game theory optimal	3746	. . .
RL	3580	. . .
Straight in	3019	The straight and random behaviors had similar total wins but won in different scenarios.
Random	2987	This was helped by using the same mirroring and restriction of φ as in the RL training.

Approximately the same cardioid shape can be seen in Figure 5. This bright red region is where the Q-table reflects expectations of a large reward for winning. The slim bright blue region, centered around $\theta = 0$, indicates a strong expectation of a penalty for losing. This region even overestimates the penalty. The penalty for losing is -2π, but the Q-table has values almost double that. This is because it is nearly always a loss choosing any action from that state. In some cases, the UAV does not lose immediately, but ends up in the same state (due to discretization). And after training for some time, it may even lose while transferring to the same state, thus adding the value of the penalty for losing to the already penalized value for being in that state.

Figure 6 shows the control angles chosen by the RL for a given radius, versus the optimal angle, which is actually independent of angle. The discernible trends are the clustering of RL-chosen points at very small angles (because those maximize the immediate reward of moving closer to the origin) and the lack of large angles at large radius (because that only wastes time while the turret is in a highly advantageous position. With the lack of large angles at a large radius, it is able to mimic the downward slope of the game theory optimal curve. Although the RL approach does not match well with game theory in Figure 6, it is still able to perform comparably overall, because the average course taken over several timesteps approximates what the optimal course should have been. Many of the values with the most error also exist in cases where the UAV has little-to-no chance of winning anyway; therefore, the optimal choice does not change the expected value much and is not discovered by the RL training.

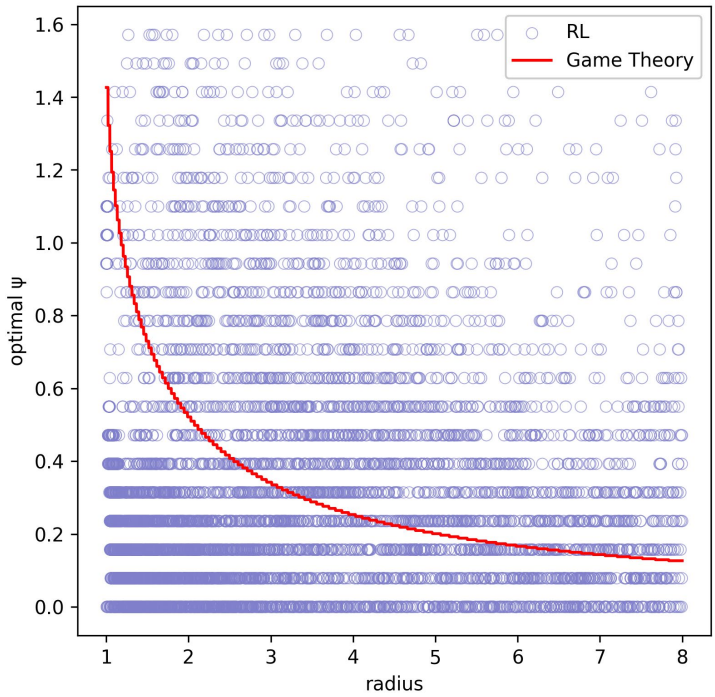

Figure 6. Optimal angle for the UAV at a given radius. The blue markers represent RL-trained values at various θ levels and have slight jitter on the x-axis (since they were calculated for a radius range due to discretization). The line shows the optimal value from game theory, which is actually independent of θ.[1]

After this tabular approach, we trained a new RL using a more sophisticated approach called Deep Q Network that trains a neural network to estimate the Q-value of any state–action pair. We used the same discretization of actions and the same mirroring. We did not have to discretize the state space, as DQNs can make predictions on continuous state spaces. The results that we got were very similar. In 100 cases, the discretized tabular approach outperformed the DQN. In two cases, the DQN outperformed the tabular approach. In all other cases, the outcome (win/lose) was the same between the two RL approaches. The training values we used are in Table 5.

Table 5. Training values

Parameter	Value
Deep neural network model	2 hidden layers of 64 neurons each
Training timesteps	2.5 E7
Exploration fraction	0.7
Replay buffer size	2.5 E6
Batch size	32
Learning rate	0.0001
Target network update interval	10,000
All other values	Stable baselines 3 DQN defaults

There were two cases discovered in an early DQN approach that showed very interesting results; the DQN outperformed the game theory approach in these cases. This is theoretically not possible. Upon investigation, we saw that it was an artifact of the simulation and tiebreaking rules, where in simulation, the game theoretic agent reached the winning circle and the turret reached the agent's angle in the same 0.02-s timestep, with the tie being awarded to the turret. In the DQN approach, the agent took a slightly more direct angle, winning in an earlier timestep, at the expense of less angular margin (the game theory approach attempts to maximize the angular margin at victory). The special cases are at radius 1.2, and angle 0.0942477796076938, and its reflection across the x-axis. Many digits after the decimal are required to produce this anomaly.

We finally attempted to train the RL using SAC. SAC, like DQN, uses Q-networks to train the reward estimate and the policy. The defining feature of SAC is the "temperature," which balances exploration with expected reward, and is adjusted throughout training to maintain this balance. We used Stable Baselines 3 default values for SAC and it did not compare favorably to our prior attempts. As seen in Figure 7, SAC never outperforms DQN or the tabular approach.

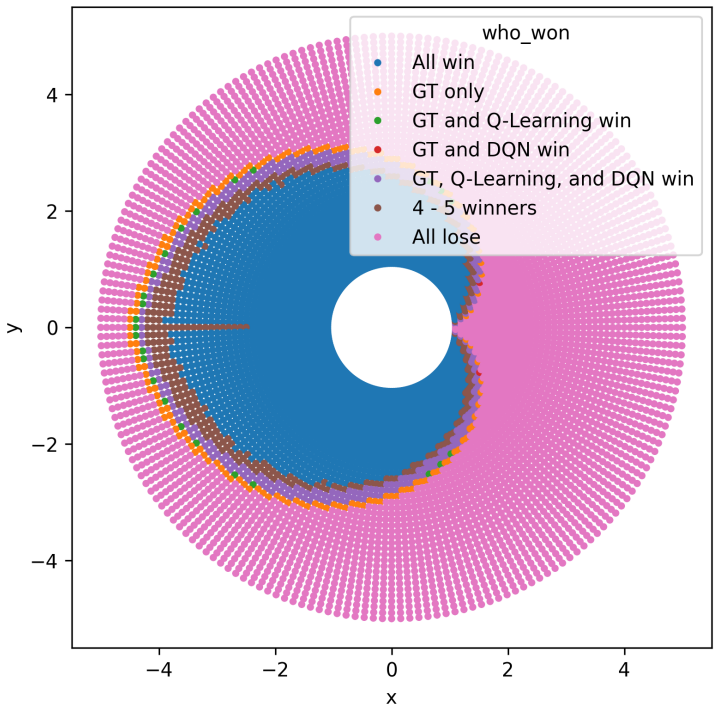

Figure 7. RL results with SAC. SAC never outperforms the tabular or DQN approaches. When it outperforms naive approaches, it is in the "4–5 winners" group.

Overall, the RL training was successful, albeit imperfect, as seen in Figures 4 and 6. We will use this simulation, evaluation framework, normalization, and mirroring in future applications with more sophisticated missions and RL approaches.

3. RL Applied to the 2 vs. 1 Game

In this section, we attempt to use RL to train agents in the 2 vs. 1 game. This game has been shown to have optimal strategies based on differential game theory, in the same work that analyzed the 1 vs. 1 game.[1] Our research question is whether RL and simulation can recreate these optimal multiagent behaviors.

3.1 Introduction to the 2 vs. 1 Game

When two UAVs are attacking a single turret, they have the opportunity to work together, possibly sacrificing one UAV, to win games that neither UAV could win solo. Again, from Von Moll et al.,[1] we know that there is a game-theory optimal strategy for two UAVs to follow to maximize the range of starting conditions that result in a UAV win.

The win conditions and motion models for the 2 vs. 1 game are similar to the 1 vs. 1 game from Section 2, with the following additions.

- The turret must destroy *both* UAVs to win.
- The first UAV to be destroyed stops in place, while the other continues.
- *Either* UAV can reach the winning region to count as a UAV win.

The game-theory optimal behavior for the two UAVs, if they cannot win solo, is for one to become a "runner," to draw the turret's boresight toward it for as long as possible, and for the other to become a "penetrator," moving in a 1 vs. 1 optimal fashion.[1] The turret's optimal behavior is to move at maximum speed, first toward one UAV to kill it, and then toward the other UAV. It chooses the order of attack by simulating the optimal moves the attackers could make (choosing to become runners or penetrators) and strategically forcing its first target to become a runner.[1] Initial conditions that call for these strategies are shown in Figure 8. Here, Attacker 1 is at (0, 4) and not shown. The turret's angle is 0°. Attacker 2 may start at any position in this graphic. The black positions are already inside the winning region, η. Attacker 2 can win from the dark gray positions regardless of Attacker 1's behavior, simply by playing its 1 vs. 1 optimal strategy. In the medium gray positions, the team can win, and Attacker 2 should be the penetrator. In the light gray positions, the team can win, and Attacker 2 should be the runner. The colored positions show the closest approach that the team (either attacker) can achieve if playing optimally, but they cannot beat an optimal turret. Similar regions can be delineated with different initial conditions for Attacker 1.

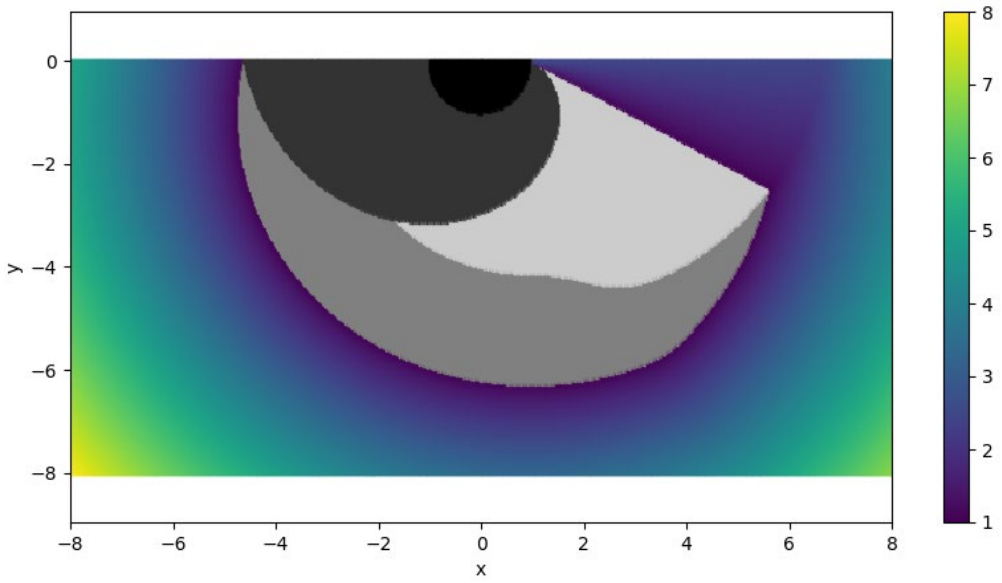

Figure 8. Typical phase space of 2 vs. 1 game.

3.2 Initial Conditions

All scenarios will begin with a UAV on either side of the turret's boresight. This makes a true 2 vs. 1 game, where the turret has a dilemma in which UAV to target first. If both are on the same side, the turret simply turns toward that side and does not have to make a choice. A "side" is defined as ±180° from boresight. The game may evolve such that both UAVs are on the same side. This will be allowed, but their behavior will then revert to the game-theory optimal single-UAV behavior. It may be advantageous for the RL to try to force the scenario into such states, so we will allow the simulations to continue. The state and action spaces for the RL are defined in Table 6.

Table 6. State and action spaces in the 2 vs. 1 game.

Symbol	Meaning	Units	Note
θ_1	Angle from turret's look to UAV on positive side	rad	$[0 - \pi]$
θ_2	Angle from turret's look to UAV on negative side	rad	$[0 - \pi)$
φ_1	UAV 1's heading, measured from radial line to turret	. . .	$[\frac{-\pi}{2} - 0]$
φ_2	UAV 2's heading, measured from radial line to turret	. . .	$[0 - \frac{\pi}{2}]$
v	Velocity of UAVs	m/s	. . .
ω	Angular velocity of turret	rad/s	. . .
ν	Radius where turret and UAVs' angular velocity are equal	m	. . .
r_1	Radius from origin to the UAV on positive side	m	. . .
r_2	Radius from origin to the UAV on negative side	m	. . .

The variables are shown graphically in Figure 9.

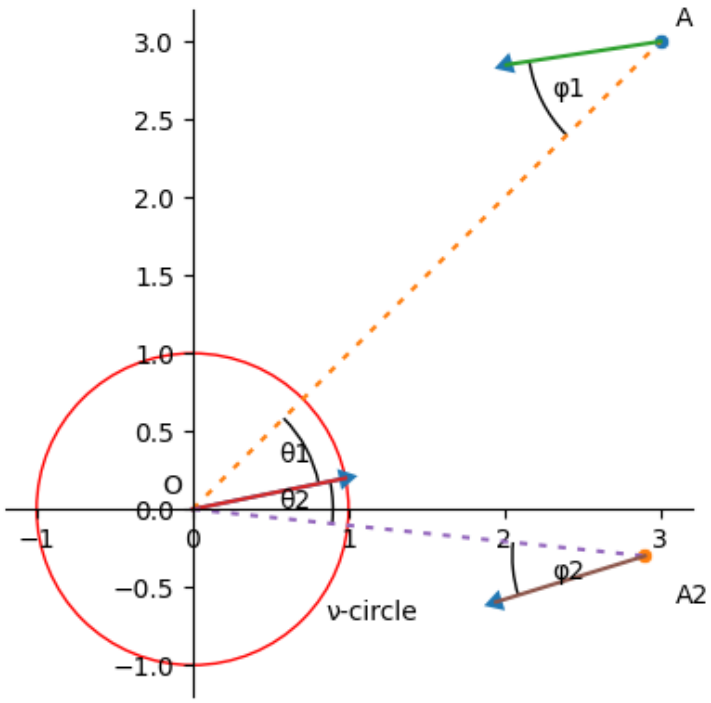

Figure 9. Variable setup in the 2 vs. 1 game. θ angles are measured from the turret's boresight, and φ angles are measured from the respective radii from the agents to the origin. Due to the limits imposed, no angles will change sign. Counterclockwise angles are positive.

3.3 Mirroring

As in 1 vs. 1 game, there is symmetry in the problem, such that the state space for the RL to train over can be cut in half by mirroring. For the RL, "Agent 1" will always be on the positive side of the turret's boresight. To choose which agent is "Agent 1," we pick the agent with the largest absolute value of θ. If tied, we choose the agent with the largest *r*. If still tied, we do not mirror the game, as it is symmetrical anyway. In the game of Figure 9, the agent labeled A_1 would be "Agent 1" for the RL since $\text{abs}(\theta_1) > \text{abs}(\theta_2)$, and the game does not have to be mirrored, since it is already on the positive side.

Figure 10 shows a game and its mirrored equivalent. The starting positions are mirrored around the x-axis. The system controls cause the final state to be mirrored across the x-axis as well, demonstrating the equivalence of the games.

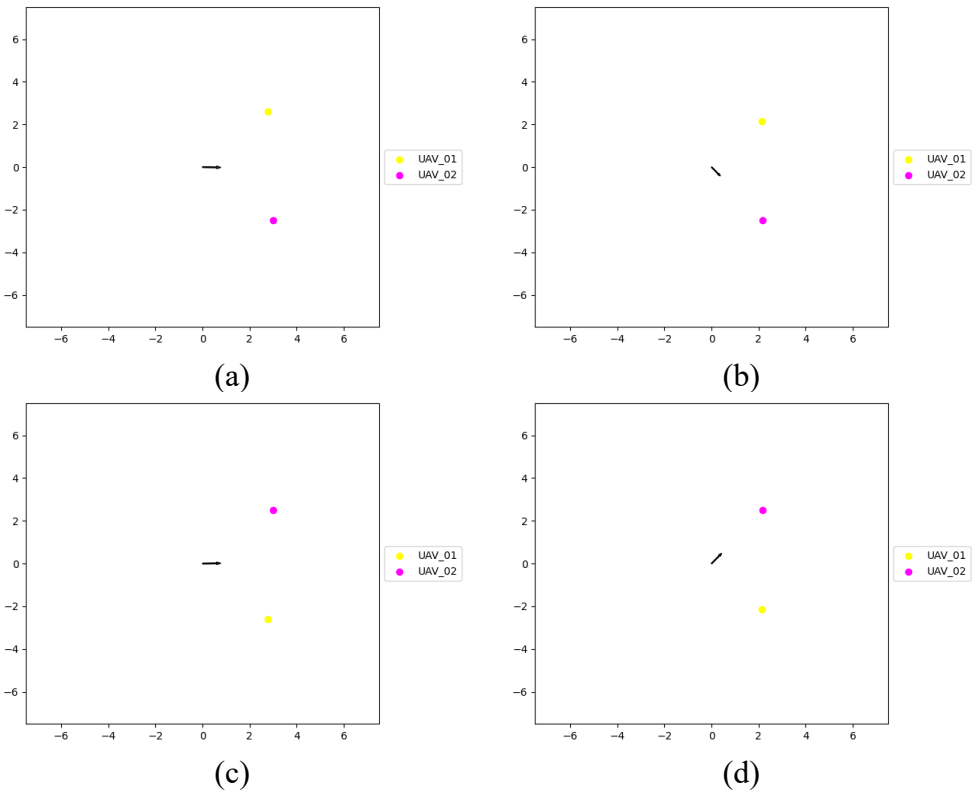

Figure 10. Unmirrored game beginning (a) and end (b), and mirrored game beginning (c) and end (d). Note that the games end when one attacker reaches the 1 vs. 1 winning region.

3.4 Control Values

To control the agents' behavior, we need to choose their respective φ values. This leads us to a four input (θ_1, r_1, θ_2, r_2) and two output (φ_1, φ_2) control problem, where the ranges of φ_1 and φ_2 are $[\frac{-\pi}{2} - 0]$ and $[0 - \frac{\pi}{2}]$, respectively.

3.5 RL Approach

We once again used SAC as the first attempt at RL for this problem. We chose SAC because it can be used with continuous inputs and multiple continuous outputs. This black-box approach essentially treats the two agents as a single system to be controlled. The simulation and controller run at 0.02 normalized time unit intervals. At each interval, the controller chooses φ_1 and φ_2. The RL agent was trained using the Stable Baselines 3 defaults for SAC. After 1,500,000 simulated timesteps, a version of the trained agent was saved. This version was then reloaded and trained for another 1,500,000 timesteps, with the SAC entropy reset. This process was intended to be repeated for as long as necessary to satisfactorily train the agent, and we saw good results after the first iteration. This report analyzes the second iteration (trained for 3,000,000 timesteps).

3.6 RL Results

We tested the results against three other baseline control strategies and the game-theory optimal strategy. In summary, these were the controllers tested:

- Both attackers choosing randomly
- Both attackers moving straight in toward (0, 0)
- Both attackers following the optimal 1 vs. 1 strategy
- RL-chosen direction for each attacker
- Game-theory optimal direction for each attacker (not simulated, simply calculated as in Figure 8)

The simulation and controller updated 50 times in every normalized time unit. The turret always followed the 2 vs. 1 game-theory optimal control strategy, so the RL was training against the best possible target.

We simulated and trained the 2 vs. 1 strategy assuming that the 1 vs. 1 game had already been solved. The simulation ended with an attacker win if either attacker reached the 1 vs. 1 winning region (anything within the orange cardioid shape in Figure 7, rotated with the turret's boresight). The simulation ended with a turret win if the turret shot down both attackers.

To assess performance, we ran 10,000 test cases with one attacker starting at (0, 4). The other attacker was started at radii in the range 1.1 to 8 and angles $-\pi$ to 0. From those starting positions, the attackers ran one of the five controllers mentioned previously. Figure 11 shows the result of these tests.

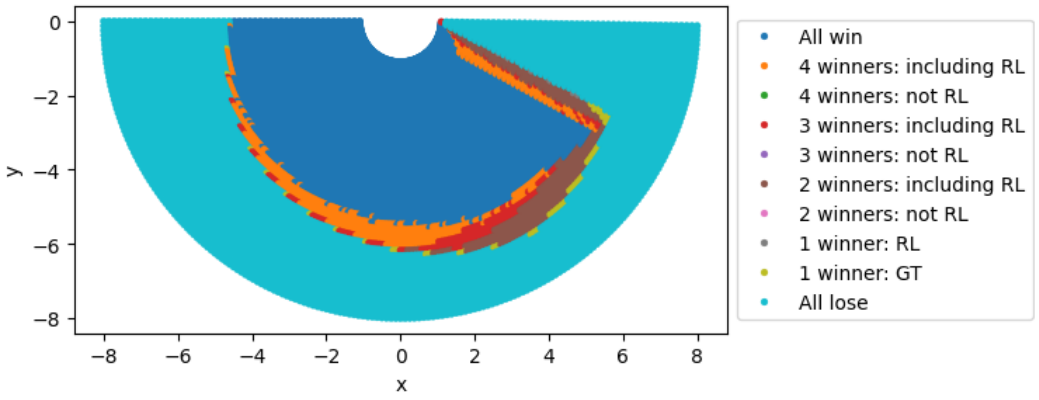

Figure 11. Results of various controllers for the attackers, with one attacker's initial condition at (0, 4) not shown. The brown region shows where the RL-trained agents performed better than all except the game-theory optimal agent. The light green region shows where only the game-theory optimal agents could win.

In Figure 11, we see that the RL agent performed as well or better than the more naïve strategies in most cases. The brown region especially shows its outperformance of all except the game-theory optimal strategy. Table 7 shows the

number of wins for each strategy. The RL agent was able to win in more than 99% of the initial conditions that the game-theory optimal agent could win. It lost a raw total of 40 more games and won 396 more than the next-best strategy (applying the 1 vs. 1 optimal strategy to both agents).

Table 7. Number of wins for each strategy.

Name	No. of wins	Fraction of game theory
Game theory optimal	6188	1.00
RL	6148	0.994
1 vs. 1 optimal	5752	0.930
Straight in	5503	0.889
Random	5359	0.866

Table 8 shows a cross-comparison of the top three strategies. All three had the same result in the vast majority of cases. In 390 cases, the game theory optimal and RL strategies both "outperformed" the 1 vs. 1 strategy. In a small number of cases, RL and/or 1 vs. 1 outperformed the game-theory optimal strategy. This is theoretically impossible. The game-theory optimal behavior was outperformed due to the discretized timesteps and tiebreaker behavior of the simulation. The RL was taking advantage of the simulation that it was trained in, and the game theory results were directly calculated, not simulated, so did not have this opportunity.

Table 8. Comparison of top three strategies. A + indicates a win for that strategy.

1 vs. 1	RL	Game theory	Count
+	+	+	5751
. . .	. . .	. . .	3805
. . .	+	+	390
. . .	+	. . .	6
+	+	. . .	1

4. Possible Extensions to N vs. 1

This report analyzed the cases of 1 vs. 1 and 2 vs. 1 attacker versus turret games. The game-theory optimal behaviors in these games have been described in Von Moll et al.,[1] and here we attempted to recreate the optimal strategies by training agents using RL in simulation. To date, we know of no game-theory optimal results in the 3 vs. 1 or higher N vs. 1 games.

To extend the RL approach to N vs. 1 games, lessons learned in this effort could be applied. All games should be normalized. Mirroring should be considered to reduce the state space. For example, in the 3 vs. 1 game, there would be one agent on one side of the turret, and two on the other. The side with one agent could be always considered the positive side, with the inverse situation mirrored. (If all three are on

the same side, behavior reduces to the 1 vs. 1 game). Also, in training, the simulation could stop and award a win to the agents as soon as any two are in a configuration that would guarantee a win in the 2 vs. 1 game. A similar approach can be applied as N grows: find a suitable way to mirror the games and reduce the state space; then, check for winning configurations among a subset of agents that have been found for (N – 1) and below to award an immediate win during training. In this way, the training will be applied to the most compact state space while relying on lessons learned from simpler cases.

5. Conclusion

5.1 Summary

In this report, we used RL to recreate optimal strategies for UAVs attacking a turret. In the case of one UAV attacking a turret, our most successful RL training used tabular Q learning, and was able to win in 18.5% more cases than a naïve straight-in strategy, and 4.43% fewer cases than a provably optimal analytical strategy. In the case of two UAVs versus one turret, RL trained agents were able to win 99.4% of the games that the theoretically optimal agent could win, outperforming the next best strategy by 6.4%.

We also discussed how the lessons learned in this work may be applied to higher N vs. 1 games, which so far do not have optimal strategies reported in the literature. The techniques for successful training include normalization of the game, mirroring of the states, and halting the simulation when a winning condition is found in a subset of the UAVs. The success of training these agents in RL is an encouragement to move to the higher N vs. 1 games, using RL to ascertain strategies that have not yet been uncovered through analytical application of game theory.

5.2 Limitations

This work did not do extensive parameter tuning in the RL training process. The training algorithms, such as the learning rate, can have a very large effect on the success of the agent's behavior. We relied on informal exploration and using the defaults of Stable Baselines 3 in this work, but future work should do more systematic and thorough parameter exploration.

We also limited ourselves to the 1 vs. 1 and 2 vs. 1 games, which already have a provably optimal strategy. This was so that we could build a simulation and pipeline for this class of games with a ground truth to compare them to. In the 2 vs. 1 analysis, we only postprocessed and analyzed the case where one attacker's initial

condition is (0, 4), and this analysis should be extended throughout the half plane to make sure that the RL's good performance is not localized to these specific initial conditions (the RL agents were trained for the entire domain).

Finally, we considered only first-order physics and deterministic simulations to be consistent with the differential game theory in prior work. Real applications will be stochastic, and real robots will have force and acceleration constraints on their dynamics.

5.3 Future Work

The most straightforward extension of this work will be to apply RL to 3 vs. 1 and higher N vs. 1 games. The ultimate goal is to create controllers that are computationally lightweight enough to be run on inexpensive UAV companion computers. RL can also be used to train attackers and turrets in stochastic environments, where the first-order physics and perfect information assumptions necessary for the analytical approach do not hold.

6. References

1. Von Moll A, Shishika D, Fuchs Z, Dorothy M. Turret–runner–penetrator differential game with role selection. IEEE Transactions on Aerospace and Electronic Systems. 2022;58(6):5687–5702. https://doi.org/10.1109/TAES.2022.3176599

2. Arulkumaran K, Deisenroth MP, Brundage M, Bharath AA. Deep reinforcement learning: a brief survey. IEEE Signal Processing Magazine. 2017;34(6):26–38. https://doi.org/10.1109/MSP.2017.2743240

3. Sutton RS, Barto AG. Reinforcement learning: an introduction. Vol. 1, No. 1, Cambridge: MIT press; 1998. p. 9–11.

4. Lillicrap TP et al. Continuous control with deep reinforcement learning. arXiv preprint; 2015. arXiv:1509.02971. https://doi.org/10.48550/arXiv.1509.02971

5. Haarnoja T, Zhou A, Abbeel P, Levine S. Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor. Proceedings of the International Conference on Machine Learning; 2018 July 10–15; Stockholm, Sweden. PMLR, vol. 80. 2018;1861–1870. https://doi.org/10.48550/arXiv.1801.01290

6. Mnih V et al. Human-level control through deep reinforcement learning. Nature. 2015;518(7540):529–533. https://doi.org/10.1038/nature14236

7. Brockman G et al. OpenAI gym. arXiv preprint; 2016. arXiv:1606.01540. https://doi.org/10.48550/arXiv.1606.01540

8. Duan Y, Chen X, Houthooft R, Schulman J, Abbeel P. Benchmarking deep reinforcement learning for continuous control. Proceedings of the International Conference on Machine Learning; 2016 June 19–24; New York, New York. PMLR, vol. 48. 2016;1329–1338. https://doi.org/10.48550/arXiv.1604.06778

9. Bellemare MG, Naddaf Y, Veness J, Bowling M. The arcade learning environment: an evaluation platform for general agents. Journal of Artificial Intelligence Research. 2013;47(1):253–279. https://doi.org/10.1613/jair.3912

10. Shishika D, Von Moll A, Maity D, Dorothy M. Deception in turret defense game: information limiting strategy to induce dilemma. IEEE Open Journal of Control Systems. 2025;4:385–399. https://doi.org/10.1109/OJCSYS.2025.3611726

11. Von Moll A, Gerlach AR, Bakker C, Rupe A, Pachter M. Constrained turret defense with fixed final time. IFAC-PapersOnLine. 2024;58(28):965–970. https://doi.org/10.1016/j.ifacol.2025.01.121

12. Wiering M, Van Otterlo M, editors. Reinforcement learning: state-of-the-art. Springer; 2012. https://doi.org/10.1007/978-3-642-27645-3

List of Symbols, Abbreviations, and Acronyms

DQN	Deep Q Network
ML	Machine Learning
RL	Reinforcement Learning
SAC	Soft Actor-Critic
UAV	Unmanned Aerial Vehicle

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
1 (PDF)	DEVCOM ARL FCDD RLB CI TECH LIB