Reinforcement Learning in Pursuit-Evasion Games

Jonathan Chang, Andrew Sack, Francis Straceski

Abstract

Reinforcement Learning in Pursuit-Evasion games is an important problem useful in training adversarial networks with multiple agents. Our paper tackles the single pursuer and evader problem in a simulation environment using a Q-learning algorithm to simultaneously train both agents independently over the course of thousands of iterations. Our simulation harnesses the power of the OpenAI gym API and creates a discretized space and action space of a virtual environment. A sensor model is used to provide useful and powerful information while training to help provide rewards and guidance to both the pursuer and evader.

1 Introduction

Reinforcement Learning in Pursuit-Evasion Games tackles the classical problem of the imperfect Pursuit-Evasion Game. One of the agents acts as the seeker, planning its movements and actions based on its observations of the environment. The evader attempts to find the optimal path away from the seeker using its own observations and knowledge base.

We use a Q-learning reinforcement learning algorithm to train adversarial models to compete against each other in the same game. The interface of our algorithm is based off the OpenAI gym API, where we designed a custom environment for our pursuit evasion game.

We discretize our space into a grid structure with 4 potential orientations. We define our action space into 6 possible movements and our observation spaces into 2 possible observations. We used various sensor models to see which sensing techniques provide the most useful and powerful information while training.

This is an important problem, as motion tracking and pursuit has many real-world applications. Some common Pursuit-Evasion applications involve search and rescue missions where the target could be unknowingly moving away from the seeker, corralling of uncooperative livestock, tracking vehicles through road networks, and human-follower assistive robots. Creating a generalized approach to the Pursuit-Evasion existing technology can be used to perform many tasks previously only able to completed by humans.

2 Background Related Work

In our Probabilistic Robotics course, we have focused on different probabilistic estimation algorithms. We have studied Kalman Filters and Particle Filters in depth and created simple versions of these filters.

Since there are many applications for pursuit-evasion tasks, there has been a large amount of prior work done. Research has explored search behaviors and algorithms for a pursuer to locate and follow the evader. Kim et al used two different pursuer policies, Greedy and Global-Max, and determined that Global-Max was quicker at locating an evader [2]. They also applied the same policies to the evader, demonstrating the similarity of the pursuit and evasion tasks. This paper also outlined a system architecture that can be used to translate high level planning into low-level movements.

Trafton et al looked at humans to inform how a robot can play hide-and-seek [4]. They observed a human child play rounds of hide and seek and looked at how they learned, then used the human learning process to train a robot.

D. Tolić and R. Fierro studied the use of markov chains and adaptive kalman filters to produce smooth, effective pathing for pursuers. Their approach predicts probabilities of different markov chains to calculate the optimal movement toward an escaping target. They focus on using low-level sensors and planning to make the target never leave the FOV of the pursuer. This method allows for low amounts of processing power and robot energy to be used while pursuing a target [3]. Their work proves that Markov chains can be used effectively for pathing in pursuit-evasion games. Our proposal will explain how Markov-Decision Process with Q-learning will be used to train the evader to smoothly and quickly make effective decisions.

3 Technical Approach / Methodology / Theoretical Framework

Like most reinforcement learning algorithms, training a pursuit evasion game follows the general formula of generating an initial state for the agents, picking a random or greedy action, calculating a reward, and updating the state-action pairing in the corresponding Q table.

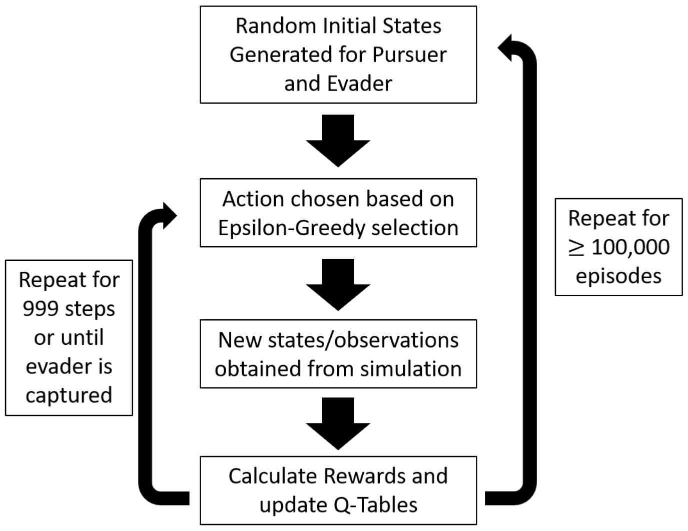
Random Initial States Generated for Pursuer and Evader
Action chosen based on Epsilon-Greedy selection
New states/observations obtained from simulation
Calculate Rewards and update Q-Tables
Repeat for 999 steps or until evader is captured
Repeat for ≥ 100,000 episodes

Figure 1: Specific Q-Learning Algorithm applied to the optimization of the Pursuit Evasion Game

3.1 Notation and Problem Formulation

For this implementation of a Pursuit-Evasion Games, the goal was to train both the pursuer and the evader in an adversarial nature. Both agents are trained at the end of each step, holding separate Q-tables. We define a successful adversarial policy as a policy which produces the same average reward for each agent. This policy is effectively a discretized solution to the Hamilton-Jacobi-Isaacs equation, a control equation used to find optimal policies. For the case of a pursuer and an evader this equation takes the form:

$$\min_{u \in U(x)} \max_{v \in V(x)} \left\{ l(x,u,v,t) + \frac{\partial G^*}{\partial t} + \sum_{i=1}^{n} \frac{\partial G^*}{\partial x_i} f_i(x,u,v,t) \right\} = 0$$

Where u and v are the evader and the pursuer, l is the reward function, f is the state transition function, x is the state space, t is time and G^* is the optimal policy we want to solve for. [1]

Q-learning algorithms use the Monte-Carlo approach to obtain an optimal policy, therefore we mush be sure to both set up an appropriate reward function that will return the desired behavior for a pursuit-evasion game, and create a simulation that is able to mathematically optimize our reward function.

3.1.1 Pursuit Evasion Reward Formulation

$$R_{pursuer} = \begin{cases} No\ Capture & R_{pursuer} = -1 \\ Evader\ Observed & R_{pursuer} = +2 \\ Evader\ Captured & R_{pursuer} = +100 \end{cases}$$

$$R_{evader} = \begin{cases} \textit{No Capture} & R_{evader} = +1 \\ \textit{Pursuer Observed} & R_{evader} = -2 \\ \textit{Evader Captured} & R_{evader} = -100 \end{cases}$$

This implementation of a pursuit-evasion game has imperfect information between the pursuer and the evader. This means that the state of one agent is not known by the other agent. The only way the agents gain information about each other is from its own front-facing sensor. Because gaining knowledge of the other robots location is intuitively important for pursuing, we reward the pursuer for sensing the evader, and penalize the evader for sensing the pursuer to encourage it to move away.

Simulation Setup

3.1.2 Environment and Action Space

The environment space for each robot is represented as: $x = \{x, y, \theta, z\}$. Each agent can occupy an x, y coordinate that does not contain a wall, as well as have an orientation θ, and an observation of its fron sensor z. These states can be discretized to any number of possible values, but for this study we stuck to a 20 by 20 grid, 4 orthogonal orientations, and 2 possible sensor values (seen or not seen).

In this implementation action space consists of six possible actions for each robot as shown in table 2. These actions are the permutations between a discrete set of linear and angular components

Action	***Linear Component***	***Angular Component***
Stop	0	0
Forward	1	0
Turn Left	0	1
Turn Right	0	-1
Move and Turn Left	1	1
Move and Turn Right	1	-1

Table 1: Robot Actions and their linear and angular components

3.1.3 Sensor and Observation Space

Observation	***Returned***
Sensed	0
Not Sensed	1

Table 2: Observation space for each robot

The observation space for each robot was represented $x = \{z\}$. This is a simple sensor where the robot has a 180 degree field-of-view from it's orientation and a range of 5 units to create a semicircle. If the other robot falls in the sensor range, it returns that the other robot was sensed. Originally, the sensor sensed 5 different distances, but this greatly increased the state space, slowing down training, and did not improve the robot's performance, so the observation space was changed to binary values. The field-of-view was set to 180 degrees, so that the robots could sense the opposing robot for more than one time step as they passed by. This is illustrated in Figure 2.

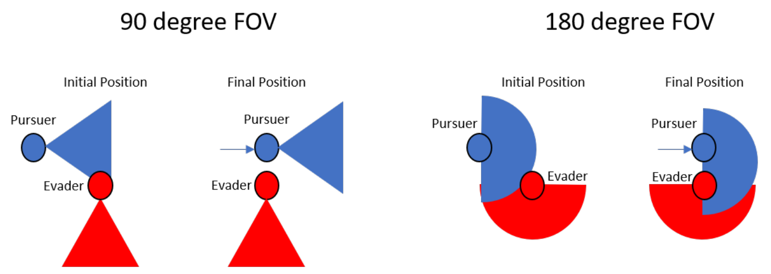

Figure 2: Effect of Field-of-View on detecting passing robot

3.1.4 Hyperparameter Selection

Pursuit evasion games are quite complex and nuanced games, even with only 2 agents in a highly discretized environment. This problem further complicates when the agents do not have full information about each other. Intuitively, one could even break the problem down into 2 problems: Scanning the map to determine the general area of the evader, then chasing, trapping, and capturing. As a result, it is important that the learning process is able to capture all the nuanced behavior needed to determine an optimal policy.

In order to determine appropriate hyperparameters, the Agent Rewards vs Training Episode graphs were consulted in Figure 4 to ensure that both reward functions converged to the same value. Each run also had to pass the eye-test to make sure the agents weren't leaning unnatural behavior.

- Learning Rate $= 0.01$
  Low learning rate needed to avoid overtraining. Adversarial networks often have flukey behavior (randomly running into another robot) that we don't want to influence our Q tables.

- Iteration Number $= 100,000+$
  High number of iterations needed to converge on a policy for low learning rates.

- Future Discount Factor = 0.9
  We don't discount future states too much because they include important information about evading / capturing.

3.2 Code

All of the code for this project was implemented in python and is available along with some of the results on GitHub[1].

4 Experimental Results / Technical Demonstration

4.1 Evaluation Metrics

Three metrics were used to evaluate the training of our Q-learning model:

1. Final Reward of Pursuer
2. Final Reward of Evader
3. Number of Time Steps until Capture

All of these were recorded for each episode and logged along with the Q-tables generated from the training. A median filter was applied to the data and the results were plotted, visualizing the change in these metrics as the model was trained.

Additionally, the program had the ability to save each step of the simulation as an image, such as in Figure 3. A tool was created to stitch the images into a video, allowing for visual analysis of agent behaviors.

4.2 Results

The results discussed in this section were generated with the following parameters:

- **Learning Rate:** 0.01
- **Future State Discount:** 0.9
- **Number of Episodes:** 100,000
- **Steps per Episode:** 999
- **Sensor FOV:** 180 degrees
- **Sensor Range:** 5 units

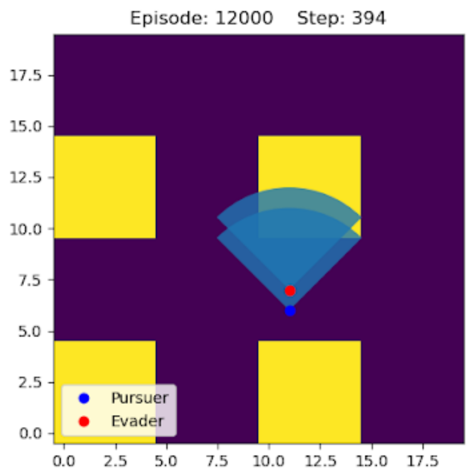

Figure 3: Still image from simulation of Pursuer capturing Evader

While the raw data shows a wide range of results at all stages of training, when the data is filtered, trends become visible. As shown in Figure 4, for the first 40,000 episodes, the evader has more positive final reward values and the pursuer has more negative values. This means that the pursuer is winning more frequently, by lasting longer without capture. This is supported up in Figure 5, where the average number of steps is approximately 300 for the first 40,000 episodes.

After 40,000 steps, a notable change occurs. The median Pursuer and Evader rewards both become much closer to 0 and the number of steps becomes approximately 100. This indicates that the Pursuer has improved to match the ability of the Evader and capture it more quickly.

4.3 Demonstration Videos

Some interesting behaviors are visible in the videos of the simulation[2].

In Trial 9[3], the pursuer senses the evader and approaches it. It temporarily loses the evader and the evader moves away, but the pursuer is able to sense it again and captures the evader in 75 steps.

In Trial 7[4], the pursuer methodically traces along the edges of the walls

[1]All developed source code for this project is available at `https://github.com/asack20/RL-in-Pursuit-Evasion-Game`

[2]videos are available at `https://github.com/asack20/RL-in-Pursuit-Evasion-Game/tree/master/results/04_May_2020_17_59_56/video`

[3]Trial 9 video available at `https://github.com/asack20/RL-in-Pursuit-Evasion-Game/blob/master/results/04_May_2020_17_59_56/video/epis00009_anim.mp4`

[4]Trial 7 video available at `https://github.com/asack20/RL-in-Pursuit-Evasion-Game/blob/master/results/04_May_2020_17_59_56/video/epis00007_anim.mp4`

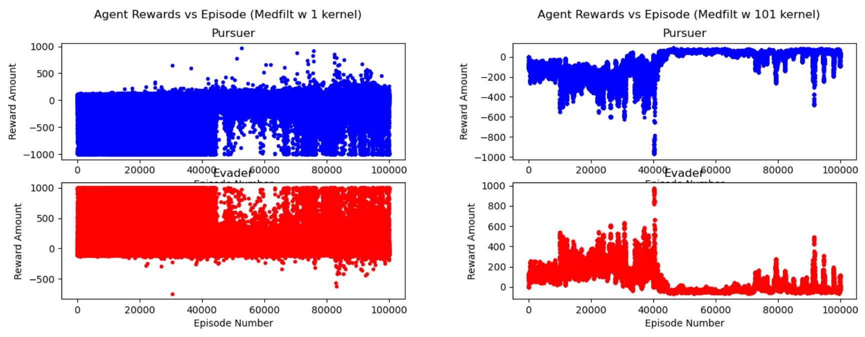

Figure 4: Agent Rewards vs Training Episode Unfiltered and with a Median Filter with a 101-wide window

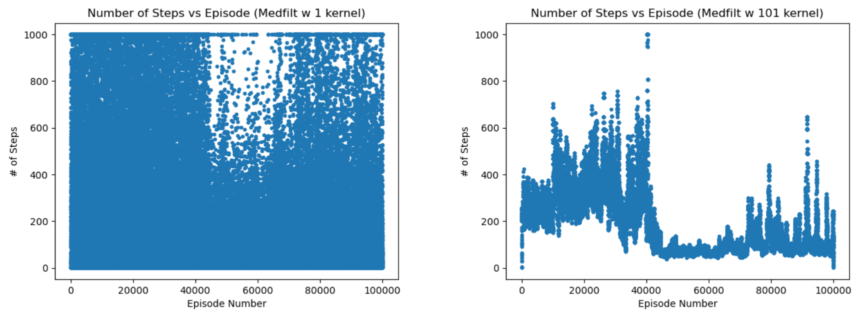

Figure 5: Number of Steps vs Training Episode Unfiltered and with a Median Filter with a 101-wide window

searching for the evader. However, it does not check in the bottom right corner. The evader learned this and hides in that corner, so the pursuer is unable to capture the evader

In Trial 5[5], the pursuer again traces the walls. However, in this trial the evader attempts to hide in the top right corner. The pursuer does search in this corner and captures the evader in 96 steps.

5 Conclusion and Future Work

Pursuit-Evasion games have been studied and solved using many different algorithms and optimizations. In this paper, reinforcement learning was the algorithm used to train the adversarial agents. Using a single pursuer and evader, we could create a simplified environment that fell within reasonable computational power to provide enough training on the robots.

[5]Trial 5 video available at `https://github.com/asack20/RL-in-Pursuit-Evasion-Game/blob/master/results/04_May_2020_17_59_56/video/epis00005_anim.mp4`

Under the Q-learning framework using the OpenAI gym API, we presented a successful model for training the pursuer and evader to approach a perfectly adversarial convergence where the rewards for both the pursuer and evader went to 0. We were able to create an effective Q-table with a reward distribution and sensor observations that properly trained the behavior of the robots.

We learned that overtraining is a classic problem to approach in reinforcement learning. Producing the raw Q-learning algorithm was relatively trivial, but refining the hyperparameters and optimizing the ideal sensor were the keys to a successful algorithm. The key was to provide simple enough metrics for the robots to understand and learn instead of having too many parameters to optimize.

For future work, we would look to modularize our simulation and look at result from multi-agent games and giving optional sensor readings. The observation space could be thoroughly more complex by implementing multiple sensors that can be used and using a Kalman filter to make observations and predictions. This would thoroughly increase the complexity of the robots. A more complex and less-discretized world environment would also be interesting to lead to more interesting applications into real-world scenarios.

References

[1] Volkan Isler, Dengfeng Sun, and Shankar Sastry. Roadmap based pursuit-evasion and collision avoidance. In *Robotics: Science and Systems 2005*, pages 257–264, June 2005.

[2] H. J. Kim, R. Vidal, D. H. Shim, O. Shakernia, and S. Sastry. A hierarchical approach to probabilistic pursuit-evasion games with unmanned ground and aerial vehicles. In *Proceedings of the 40th IEEE Conference on Decision and Control (Cat. No.01CH37228)*, volume 1, pages 634–639 vol.1, Dec 2001.

[3] D. Tolić and R. Fierro. Adaptive sampling for tracking in pursuit-evasion games. In *2011 IEEE International Symposium on Intelligent Control*, pages 179–184, Sep. 2011.

[4] J. Gregory Trafton, Alan C. Schultz, Dennis Perznowski, Magdalena D. Bugajska, William Adams, Nicholas L. Cassimatis, and Derek P. Brock. Children and robots learning to play hide and seek. In *Proceedings of the 1st ACM SIGCHI/SIGART Conference on Human-Robot Interaction*, HRI '06, page 242–249, New York, NY, USA, 2006. Association for Computing Machinery.