Evaluation of Reinforcement Learning Algorithms in Drone Flight Control for 2D Space

Oliver Wiech
University of Nottingham

Sijan Rana
University of Nottingham

Andreas Nikolaidis
University of Nottingham

***Abstract*—The following study aims to achieve automated movement control for a drone through the application of three discrete Reinforcement Learning algorithms. Two Temporal Difference methods, Q-learning and Sarsa, and Monte Carlo are applied to train the agent (drone) to automate its movement. The environment considered is a 2D virtual space in python. The states in state-space are composed with information from the location and the discretized velocity of the agent. The actions are also discretized into movements in up, down, right and left directions. The actions are results of either ϵ-greedy or optimal policy for all the three methods aforementioned. In order to score the performance of the flight-controller a distance dependent reward function is utilized. The study has managed to show that discrete RL algorithms can automate the drone's flight. A second aim was to make the agent reach more than one targets. The agent manages through training to achieve the first target consistently and multiple times the second target, not consistently though. Improvements to get consistently the second target include the finding of a better exploitation to exploration ratio.**

I. Introduction

The flight controller is responsible for the adjustment of the its motors to coordinate drone movement. It takes in sensor data from other instruments such as gyroscopes and accelerometer, to leverage the input thrust required to perform certain actions such as stationary flight or controlled movement.

Our project aims to additionally integrate reinforcement learning algorithms to these flight controllers to automate drone movement to specified targets. In particular, we utilise model-free based techniques: Q-Learning, Monte-Carlo and Sarsa to train an agent drone to reach specific targets in a simplified simulated 2D environment.

II. Methods

A. Reinforcement Learning

Reinforcement Learning as described in [1] is a sub-field of Machine Learning composing of four main principles: agent, environment, action and reward. An agent interacts with its environment with actions. Each action observed by the environment returns a state and reward to the agent. Using this reward as a metric to guide the agent's selection of predefined actions, the agent learns to optimise its actions by maximizing this reward through many iterations in its interaction with the environment.

The core concepts which underpins the reinforcement algorithms evaluated in this project are the Bellman equations below.

$$V_\pi(s) = R(s, \pi(s)) + \gamma \sum_{s'} P(s'|s, \pi(s)) V_\pi(s') \quad (1)$$

The Eq. 1 Bellman equation is from optimisation methods in dynamic programming [2], which is utilised by reinforcement algorithms in assigning a value to a given state under a given policy π. The agent aims to construct optimal policy π^* by defining actions maximising its state and rewards. Mathematically this is defined as:

$$V_{\pi^*}(s) = \max_a \{R(s, \pi(s)) + \gamma \sum_{s'} P(s'|s, a) V_{\pi^*}(s')\} \quad (2)$$

B. Environment

The environment is simulated in 2D space with gravity in Python using the module pygame. The code [3] that creates and displays the simulation is written by Jamie Mair which our code expands upon.

C. Agent

Our agent is represented by a drone which receives thrust values from the flight controller. As we are working with 2D space, only pitch and motor thrust (n = 2 motors) is required to move the drone in the environment. The agent drone can move with action $a \in \{\text{Up, Down, Left, Right}\}$ which are more precisely defined by discrete thrust values and also affected by pitch and pitch velocity for lateral movement.

D. Learning Algorithms

1) Monte Carlo: In the Monte Carlo method, the agent drone learns from experience by directly interacting with the environment. It is a model-based method and the more simpler of the three algorithms evaluated. At each time step t of the simulation, the drone observes a reward r_t from action a_{t-1} in state s_t. However, it differs from other algorithms as the update to the value function V_π, state-value function Q_π and policy π are episodic.

In our implementation, each epoch represents an episode of learning where our agent drone creates a trajectory $\omega_t^T = \{(s_{t'}, a_{t'}, r_{t'})\}_{t'=t}^T$ where $a_t \in \{\text{left, right, up, down}\}$. The summation of each reward r_t in a given trajectory gives a return R. Each epoch runs the simulation for T time steps after which the value $V_\pi(s_t)$ and state-value $Q_\pi(s_t, a_t)$ are

updated using the generated trajectory. The value function is given by $V_\pi(s_t) = \mathbb{E}[R_t|s_t = s]$ where the R_t is the discounted return from a given ω_t. A discount factor γ is applied to R_t to control the agent drone's prioritisation of immediate over future rewards. The policy update is given by $\pi(s_t) = \arg\max_a(R_t + \gamma V_\pi(s_{t+1}))$.

2) Sarsa: State-Action-Reward-State-Action (Sarsa) is an on-policy temporal difference (TD) control method. Therefore, it interacts with the environment and updates the policy based on actions taken. The Sarsa method updates the Q-values based on the current policy used. For the evaluation of the action-value function the TD error, δ_t, update method is employed (equation 3). The equation that updates the state-action value function Q for the Sarsa method is as follows:

$$\delta_t = r_{t+1} + \gamma * Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \tag{3}$$

$$newQ(s_t, a_t) = Q(s_t, a_t) + a * \delta_t \tag{4}$$

where α represents the learning rate and γ the discount factor. The equation (4) describes how the new state-action value function $(newQ(s_t, a_t)))$ is updated based on the current state-action value function $(Q(s_t, a_t)))$, the reward for moving to the next state $(r_{t+1}$ and the state-action value function of the next state $Q(s_{t+1}, a_{t+1})$.

3) Q-Learning: Q-Learning is a model-free method which learns what action to take given the state s_t. It is very similar to Sarsa algorithm, however it is off policy meaning that no specific to the task policy is needed to control the agent, the Q-values are updated as following:

$$newQ(s_t, a_t) = Q(s_t, a_t) + \alpha(r_t + \gamma maxQ(s_{t+1}, a^g) - Q(s_t, a_t)) \tag{5}$$

What is different to Sarsa in the equation is that Q-Learning uses greedy policy to pick a greedy action a^g in the new state s_{t+1} that maximises its Q value: $maxQ(s_{t+1}, a^g)$.

III. Current Methodology

A. State and Action Space

Since the simulation is done in continuous space, the states and actions had to be discretised. The state space was decided to consist of: positions x and y of the drone, each multiplied by 100 and then rounded to an integer, velocity x and y rounded to the first decimal place, and positions x and y of the target. The state vector s can be represented then as a tuple of numeric parameters: $s =$ (drone.x, drone.y, velocity.x, velocity.y, target.x, target.y).

The actions decided by the agent are a result of two heuristics, one for the horizontal motion and one for the lateral motion. These heuristics depend only on two hyper-parameters which were tuned based on a trial and error method to avoid overshooting the target. Additionally, the actions were split into thrusting up, thrusting up but slower and similar discretisation was performed for the other directions (left, right, down), hence action-vector $a =$ (Up, Down, Left, Right).

B. Reward Function, Scoring, Learning and Discount Rate

Multiple reward functions were tested, some of them penalised for the time t taken to reach the target, others rewarded the agent for getting into specific places on the simulation. The final reward function that was decided to be the best is defined as:

$$r(s, a, s') = \begin{cases} 1 - Distance & \text{-0.75<x<0.75, -0.5<y<0.5} \\ -10 & \text{otherwise} \end{cases} \tag{6}$$

-0.75 and 0.75 are the left and right boundaries of the simulation, while -0.5 and 0.5 are down and up boundaries, hence the agent was heavily penalised for leaving the rendered game space. To compare the models performance, we set a scoring that takes into consideration the average time taken to reach the first target per episode.

The two parameters, learning rate α and discount rate γ take values of [0, 1]. The α dictates the weights of the new updates of Q-values i.e. setting it to 0 means that the values are never updated while high value allows quick learning. The γ factor penalises very distant rewards, telling the agent that immediate rewards are worth more that getting the same reward in more time-steps. Both α and γ were set to 0.9 in our implementation.

C. Greedy and Optimal Policy

The agent behaves by either exploiting the information available stored in the Q-Table or by exploring, that is taking actions randomly so it discovers new states and sees the rewards of being in the new state. During the training exploration-exploitation can be balanced by using epsilon greedy policy. Epsilon ϵ is set which sets the probability of making a random action instead of going with the optimal one. The rewards for making action a in state s are stored in the Q-Table and updated after each time-step t. After the training is completed, an optimal policy can be used which is equal to taking $maxQ(s_t, a_t)$ from the Q-Table.

IV. Results

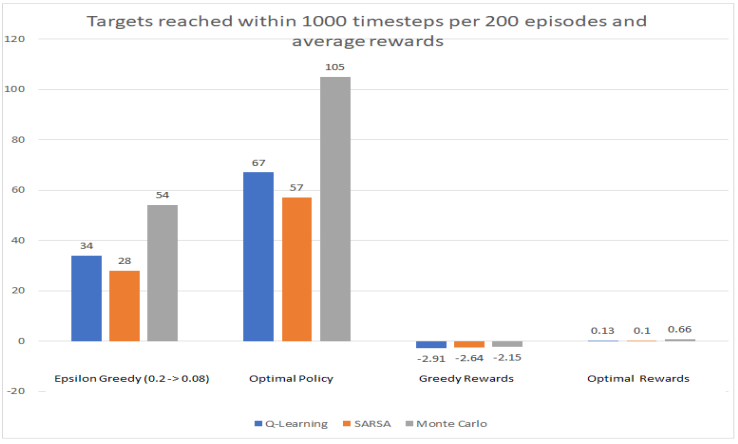
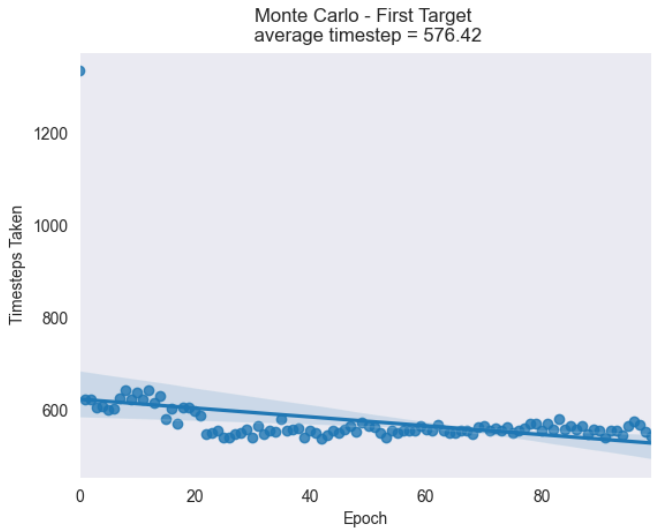
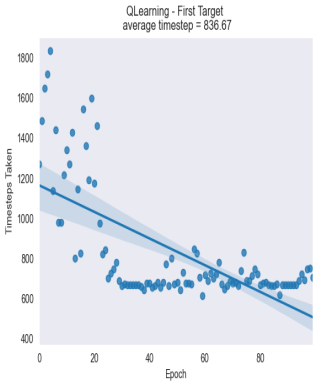


Fig. 1: Number of first targets reached under greedy and optimal policy, and corresponding average rewards achieved.

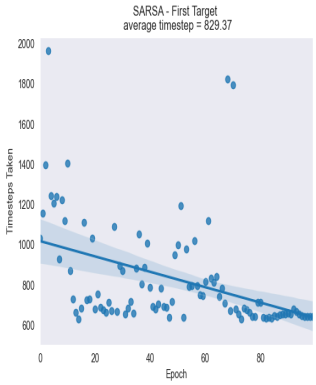
The agent was trained for 8000 episodes (about 5 hours) by applying one of the three RL algorithms, Monte Carlo, Q-learning and Sarsa, at a time. Before executing the three algorithms for the ϵ-greedy policy, ϵ was set to 0.2 and reduced to 0.08 over time. Within the 8000 episodes the agent manages to reach the first target consistently and also multiple of times the second target for all three of the algorithms applied.



(a) Monte Carlo



(b) Q-Learning



(c) Sarsa

Fig. 2: Time to reach the first target during training using epsilon-greedy policy.

The Monte Carlo method, as also shown in figures 1 and 2, outperforms the Q-learning and Sarsa algorithms to reach the first target. This superior performance of the Monte Carlo method, at least for the first 200 episodes, compared to the two TD methods can may be due to some bias. The TD methods inherently carry a bias depending on the position that the algorithm will initiate. This bias is asymptotically eliminated with the number of iterations. Since figure 1 compares for the first 200 episodes, position bias might still be significant causing the Monte Carlo method to appear superior. Continuing, from figure 1 it can be seen that with an optimal policy all of the algorithms perform better for the first 200 episodes than using the ϵ-greedy policy. This can be explained by the luck of exploration and by considering that the reward function is distant dependent and thus the reward is "driving" the drone to the first target. Figure 2 demonstrates the average time steps required for the drone to reach the target. Figure 2 also shows that the agent is properly trained, for all three of the methods used, to reach the first target since the average time-steps required to reach the target is reduced with experience (episodes).

V. Discussion

A. Improvements

Improvement with respect to the agent (drone) would be to manage to at least find two targets consistently. One way to do that is to keep the exploration higher for a longer period of time but allow for a larger number of episodes. However, this approach is computationally expensive without guarantee that the drone will consistently find the second target. In order to maintain the same amount of episodes (8000) but increase the performance, a better exploration to exploitation ratio can be found. The highest value of exploration is during the time the agent is trying to find the first target. This may not allow the drone to explore the action-value space of the second target enough, and therefore does not converge to finding two targets. Another way to improve the current model is with the usage of parallel programming. One can have the agent learning simultaneously to reach all of the target positions and then merge the information to get the agent to reach all four targets.

B. Conclusion

The study found that a drone moving into a continues space can be controlled by the discrete RL algorithms Q-learning, Sarsa and Monte Carlo. Both Sarsa and Q-Learning produced almost identical results after convergence, noteworthy is that Sarsa took longer time during the training to converge. Monte Carlo had the best results from the three algorithms when finding only the first target. The study is at the phase where the drone manages to consistently reach the first target and multiple times the second target. In order to achieve the discretization of space, discrete states that consist information about the location and the velocity of the agent were generated. Additionally, the action-space was discretized to up, down, left, right movements. The reward function utilized is a simple distance dependent function. Lastly, to improve the model to at least reach two targets consistently, either more steps with highest exploration can be utilized or a better exploration to exploitation ratio can be used.

References

[1] Richard S. Sutton, Andrew G. Barto. *Reinforcement Learning: An Introduction, second edition*. MIT Press, 2018

[2] R. Bellman, The Theory of Dynamic Programming, 1954, https://projecteuclid.org/download/pdf_1/euclid.bams/1183519147

[3] Jamie Mair, Drone Flight Controller Github Repo, https://github.com/JamieMair/DroneControllerMLiS.