Using Reinforcement Learning to Teach Quadrotors How To Hover

Brian Neldon, Khaled Jabr
brian.d.neldon-1, khaled.jabr-1 @ou.edu

Abstract

Unmanned aerial vehicles (UAVs) controller design is an important building block in creating an autonomous UVA, and it has been the subject of much research in recent years. In this work, we present an approach to this challenge using reinforcement learning (RL) combined with a neural network to teach a simulated quadrotor how to hover. Using two techniques, Q-learning and SARSA, we managed to build an agent that can intelligently adjust its pose to hover in a certain spot. Our results showed that combining RL with a neural network and a suitable reward function can achieve our goal.

Introduction

Autonomous navigation for UAVs has gained much interest and research in recent years and has been utilized for a variety of applications. While quadrotor UAVs, commonly referred to as drones, are able to carry out many functions, the most essential function is the stabilization of its dynamic behavior, or hovering. The dynamic behaviour of a drone is nonlinear, which makes it more complex, but there has been done a lot of research on how to stabilize it.

Different works have used reinforcement learning to achieve various UAV tasks. Abbeel used reinforcement learning, namely differential dynamic programming, to create an autonomous autopilot for a helicopter to autonomously complete four main aerobatic maneuvers (Abbeel *et al.* 2006). Berger used RL to teach a helicopter how to hover upside down. His work developed a model by obtaining training data from a pilot tryouts, then once the model was ready, he used RL by formulating a Markov decision process with a quadratic reward(Berger *et al.* 2004). Omidshafiei used RL and hidden markov decision process to develop a quadrotor controller through discretizing state and action spaces, and using a gaussian reward function and an algorithm for finding the optimal policy (Omidshafiei 2015). In the work presented by Bou-Ammar ,the paper presents a dynamic model for a quadrotor UAV, then it proposes a stabilization solution using RL and fitted value iterations to approximate the value function and design a controller(Bou-Ammar *et al.* 2010) .

Our work is focused on using reinforcement learning to teach a quadrotor drone how to hover. We define hovering as the drone starting at a specific position (0, 0, -5) and staying within distance of 0.5 meters of the starting point.

While using RL and seeking to teach our UAV a specific task, which is similar to the works mentioned, especially Bou-Ammar, our work differed from other works in two main ways. First, we did not use a predetermined dynamic model for the drone. For instance, Bou-Ammar uses a heavily physics-based controller ((Bou-Ammar *et al.* 2010). In our project, we did not assume any knowledge of physics and instead used a model-free agent that relies on feedback from the simulated environment, Microsoft AirSim. Second, we used different RL techniques, namely Q-learning and SARSA, and incorporated neural networks with them.

For simulation, we used AirSim, an open-source drone environment simulator released by Microsoft in 2017. Their stated objective was to allow robotics and artificial intelligence programmers to more easily and efficiently acquire data from flight tests. Within the simulator, any number of conditions can be fabricated to collect relevant data. The simulator is built on top of Unreal Engine for realistic physics calculations and graphics processing.

Methodology

Learning Model

For our state representation, we decided to use the position and orientation of the drone since those were readily available from the simulators API: S = (x, y, z, Φ, θ, ψ) where x, y, and z are the position coordinates in the NED coordinate system used by AirSim and Φ, θ, and ψ are roll, pitch, and yaw representing the orientation angles rotating around the x, y, and z planes respectively.

For our action space, A = (dΦ, dθ) where dΦ is roll delta and dθ is pitch delta. These values are deltas because they are not absolute; they change the previous roll and pitch by that amount. Changing our action space from absolute to relative angles drastically decreased our state space, from millions to dozens. We decided to lock ψ to a constant value in order to further constrain our action space from cubic to quadratic complexity. In the final iteration, the agent could choose to change its roll and pitch by a degree angle in [-5, 5] with a step size of 2.5. This lead to 25 distinct actions.

Approach

In order to handle continuous values, we implemented a three-layer feed-forward neural network with TensorFlow. The neural network served as a function approximator for our Q-values. The network was updated with stochastic gradient descent using a sum of squares loss function with a learning rate of 0.00001. There were six inputs to represent the state, 10 neurons in the hidden layer, and 25 in the output layer, one for each action. The outputs generated Q-values for a particular state input.

Reward

To measure the performance of our drone, we implemented two reward structures, labeled R1 and R2. R1 was a static reward system, in which the drone was awarded +10 if it stayed within 0.5 meters of the origin, and -50 elsewhere. R2 was a linear reward system. It awarded +10 within 0.5 meters, and elsewhere was a negative linear function of the distance from the origin. The farthest point away was a reward of -50.

Experiments and Results

Experiments

We hypothesized that we can use RL to create and implement a control system to take suitable actions that will stabilize the drones as it flies. We implemented two learning methods with three variations in actions selections and rewards, for a total of six experiments. The learning methods were Q-learning and SARSA. For both of those, we selected actions with ϵ-greedy or softmax and had a third variation with ϵ-greedy and a linear reward structure. Both Q-learning and SARSA require a discount factor, which we set at 0.1 to make the agent appreciate immediate rewards over future rewards.

Results and Analysis

Our performance metric is the average reward obtained per episode. We averaged the reward of all the actions taken per each episode. We realized that there was too much noise changing by the episode, so the next measure was averaging the reward over 100 episodes to smooth out the graph. These results we achieved after running each experiment for 20,000 episodes. Here are the results and an analysis on each of them:

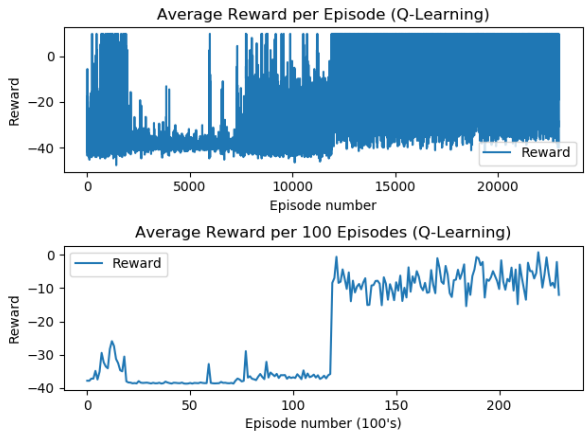

Figure 1: Q-learning R1 ϵ-greedy

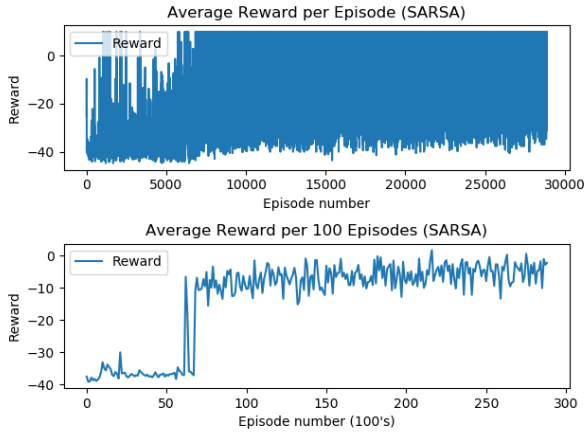

Figure 2: SARSA R1 ϵ-greedy

Looking at Figures 1 and 2, we can see that the reward for both approaches improved as number of episodes increased, which indicates that the drone improved at hovering. We can also see that SARSA learned approximately twice as fast as Q-learning, which we attribute to SARSA being an on-policy method. We believe it learned the stochasticity of the environment better than off-policy Q-learning which always takes the max value action.

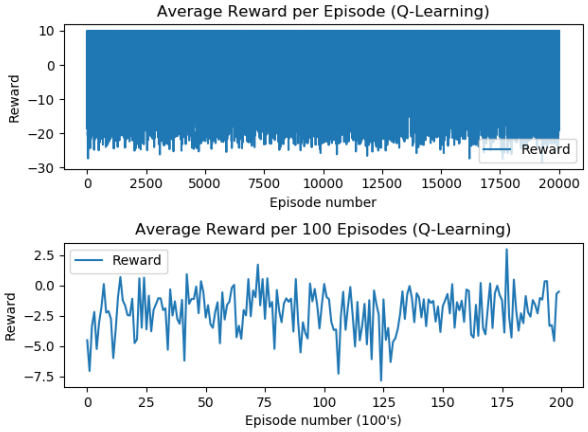

Figure 3: Q-learning R1 Softmax

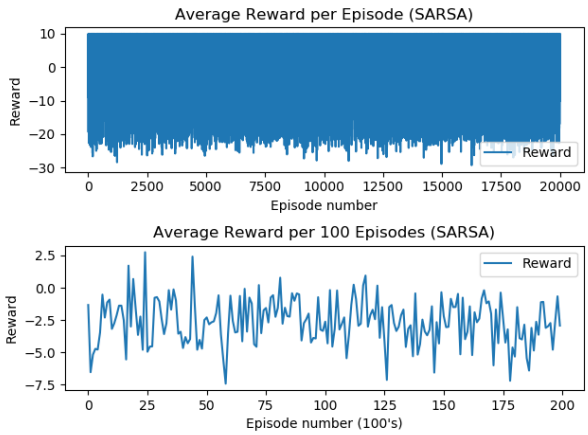

Figure 4: SARSA R1 Softmax

Looking at Figures 3 and 4, we can see that the average reward for both approaches oscillates between 2.5 and -7, which is unexpected, and it shows a flat or negligible learning curve for both uses of softmax. Our hypothesis for this lack of learning was that randomly selected actions made the drone imbalanced and might have started an irrecoverable flight path. This could explain the dips and oscillation in the graph.

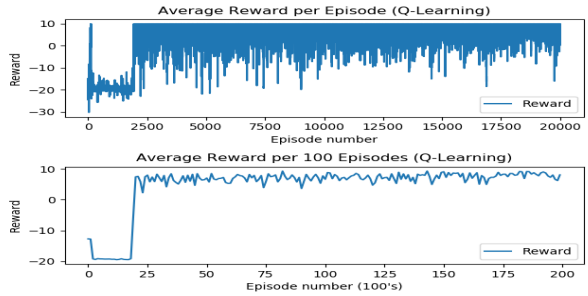

Figure 5: Q-learning R2 ϵ-greedy

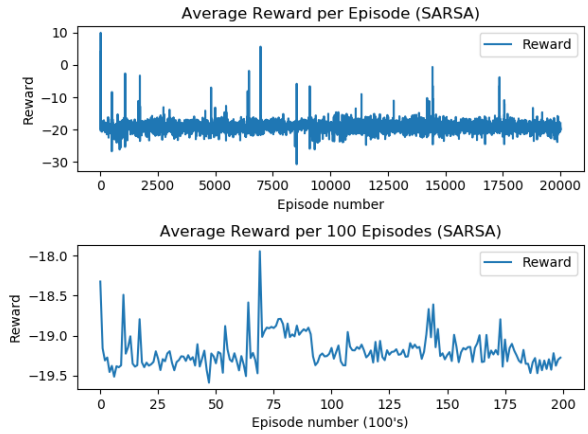

Figure 6: SARSA R2 ϵ-greedy

Looking at Figures 5 and 6 in the next page, we can notice a difference between Q-learning and SARSA. In figure 5, there is a big jump after 2500 episodes, in which the system after was receiving an average of 9 +/- 1, which initially indicates that the drone learned how to hover. We suspect that the way we structured our reward function R2 led to the spike in the graph. There was linearity in the reward as long as the drone was outside the target radius, but once inside the reward jumped to +10. This may have led to the large spike observed.

Conclusion and Future work

In this work, we showed that using ϵ-greedy RL techniques such as Q-learning and SARSA, we could develop and implement a control system that is capable of teaching a simulated drone how to hover. We conclude that both SARSA and Q-learning using R1 reward structure and ϵ-greedy action selection showed tendency towards learning. For future work, we hope to be able to use developments on AirSim, including its computer vision modes and lower level controls, to implement a deep Q-network (DQN).

References

Abbeel, P., Coates, A., Ng, A.Y., and Quigley, M. 2006. *An Application of Reinforcement Learning to Aerobatic Helicopter Flight*. NIPS.

Berger, E., Coates, A., Diel, M., Ganapathi, V., Liang, E., Ng, A.Y., Schulte, J., and Tse, B. 2004. *Autonomous Inverted Helicopter Flight via Reinforcement Learning*. ISER.

Omidshafiei, S. 2015.*Reinforcement Learning-based Quadcopter Control* .

Bou-Ammar, H., Ertel, W.,and Voos, H. 2010. *Controller design for quadrotor UAVs using reinforcement learning*. 2010 IEEE International Conference on Control Applications, 2130-2135.