# Autonomous Quadrotor Flight in Simulation using Reinforcement Learning

**Dhruv Mauria Saxena**
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
dsaxena@andrew.cmu.edu

**Ratnesh Madaan**
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
ratneshm@andrew.cmu.edu

**Rogério Bonatti**
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
rbonatti@andrew.cmu.edu

**Shohin Mukherjee**
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
shohinm@andrew.cmu.edu

## Abstract

Autonomous navigation of Unmanned Aerial Vehicles (UAVs) in real environments is still a big challenge in robotics, despite great advancements in sensor technologies and motion planning algorithms. In this work we use a deep learning model to allow a simulated UAV to avoid obstacles changing its yaw angle while flying forward. We only use using monocular images as input for Q-learning in 1.4M iterations with the environment. A video of learned policy can be found in this link.

## 1 Introduction

Real-time obstacle avoidance for UAVs is a challenging topic in robotics. In the case of static obstacles, most algorithms use some sort of 3D reconstruction of the environment, either with occupancy maps or SLAM to create feasible paths in the environment (1). These approaches, however, requires heavy computational power and sensor capabilities, and is challenging for small UAVs. In addition, traditional path planning does not work well in the case of reactive obstacle avoidance, when a very fast policy response is necessary.

Therefore, there is a need to provide UAVs the capability of mapping high-dimensional sensor input to complex decision-making in a fast response time. Learning is a possible solution for this need, with the vehicle using past experiences and interactions with the environment to improve future behavior, both in simulation and reality.

Past approaches to using learning for the control of UAVs include supervised learning such as the work of Ross et al. (2). Despite DAgeer being a powerful control technique, it requires human supervision and also many real-life tests, which are expensive and dangerous in the case of UAVs. Self-supervised approaches that can be scaled up using simulations are more attractive in this context.

Other recent work has looked at the problem of autonomous quadrotor navigation using reinforcement learning (3). They use a much simplified state representation for their learning algorithm. The state is a vector $s \in \mathbb{R}^3$, with $s = [\boldsymbol{x} \quad b]$, where $x$ is the position of the quadrotor on a 2D grid, and $b$ is its battery level. This approach fails to encode the complexity of sensors in real-worl problems into the motion planning decisions, and therefore is very unlikely to be transferred to reality. In our work, we want to the state to be much higher dimensional being the entire simulator image.

10703 Class Project (2017).

Reinforcement learning algorithms require an extremely large number of iteractions with the environment, and in order to avoid crashes with a real quadrotor, it is preferrable to learn policies in a realistic simulated environment. Following this principle, Sadeghi and Levine (4) present a learning-based approach for reactive obstacle avoidance for autonomous quadrotor flying it in a simulated environment using monocular images as input for the network. After learning a policy using simulations, they performed experiments in reality, avoiding obstacles in indoor settings with a UAV.

In this project our objective is to learn obstacle avoidance for an autonomous quadrotor in simulation using deep Q-learning. We use the DQN architecture popularised by Google DeepMind to play Atari video games (5) to train a network to output desired yaw commands by taking the simulator's monocular image as input.

Our contributions in this project are the following:

- Set up an open-source simulation environment with a quad-rotor to be used for our own project and by others in reinforcement learning research
- Train a deep Q-network capable of flying a drone autonomously in the Gazebo environment, avoiding obstacles receiving monocular images as input

## 2 Methods

In order to achieve our objectives of creating a simulation environment for drones we set up a simulation environment, a protocol for episodes, a deep Q-network for training and devised a learning algorithm based on rewards. These steps are explained in the following subsections.

### 2.1 Gym-Gazebo Environment

Figure 1 shows a close up of the quadrotor in Gazebo.

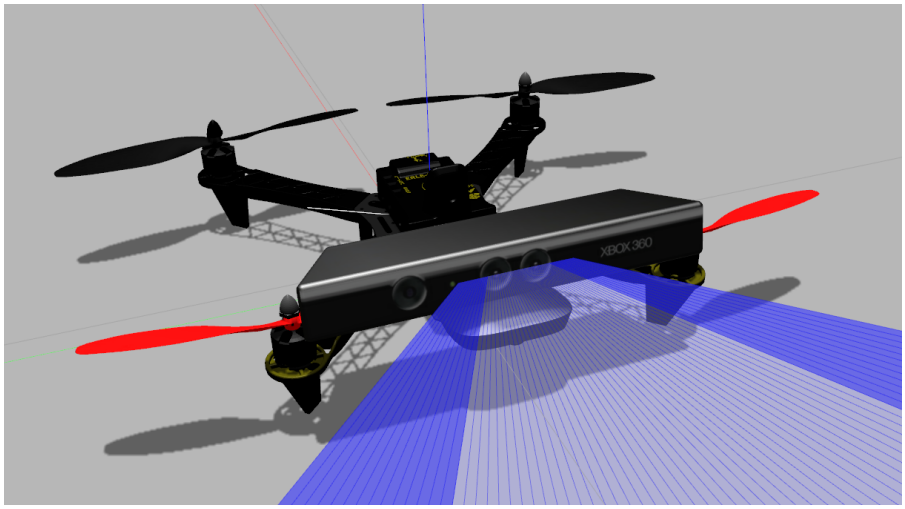


Figure 1: Closeup of our quadrotor in Gazebo.

Erle Robotics [1] released an open-source simulation environment named *Gym-Gazebo* (6), which interfaces the OpenAI Gym [2] with ROS [3] and Gazebo [4]. This allows us to simulate a quadrotor in a Gazebo, communicate with it using ROS, and execute reinforcement learning algorithms using the OpenAI Gym setup, restarting the drone after each episode and giving steps after the selection of

[1] http://erlerobotics.com/blog/
[2] https://gym.openai.com/
[3] http://www.ros.org/
[4] http://gazebosim.org/

each action. More details about the simulation environment and associated protocols are given in Section 2.2.

There are several pieces of software that interact with each other to make our simulation possible. First and foremost, the quad-rotor and obstacles are rendered and simulated in Gazebo. This simulation is wrapped inside an OpenAI Gym environment which our reinforcement learning agent interacts with to learn an obstacle avoidance policy. The quad-rotor is equipped with a laser rangefinder and front-facing camera. The choice of actions, current state of the quad-rotor, and sensor readings are communicated back and forth between the Gym environment and Gazebo using ROS messages. A screen-shot of Gym-Gazebo is shown in Figure 2. Currently, our obstacle field is a randomly generated forest of cylinders with 4m of height and 1m in diameter.

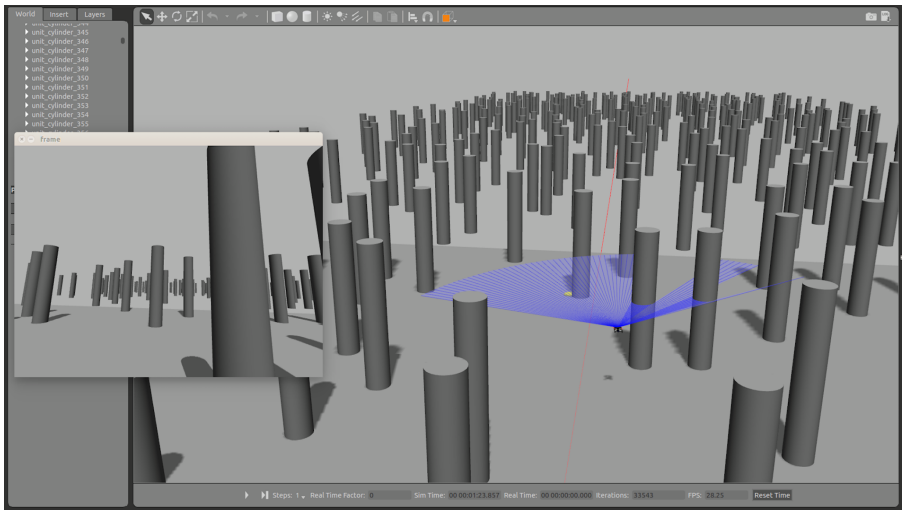

Figure 2: Screenshot of Gym-Gazebo environment setup with cylinders as obstacles.

## 2.2 Episode Protocol

### 2.2.1 Simulation with MAVROS

Initially, we used ROS for the communication between various modules, and MAVROS [5] for communication to/from the quadrotor. This meant that the speed of our simulations was limited by the Gazebo $\leftrightarrow$ MAVROS communication protocol, giving us approximately $24,000$ environment steps in a day at best.

We had to heavily optimize the `init` and `reset` protocols for our simulation episodes in order to even get these speeds. In short, at the start of each episode, the quadrotor is sent a sequence of commands that arm the motors, make it takeoff to a specified altitude, and hold a constant half throttle command. During the episode, the quadrotor receives one of the 9 possible velocity commands (which are converted to motor PWM commands by Ardupilot automatically). While the quadrotor is in flight, our learning agent uses the images from the front-facing camera as an input for its learning algorithm.

The laser rangefinder on-board the quadrotor is only used to detect the end of an episode. Once the minimum laser range reading falls below a specified threshold, the end of an episode is triggered. After the end of an episode is triggered, the quadrotor enters the Return-To-Land mode. It flies above the obstacle field to come back to its initial launch position. The motors are then killed, Gazebo world reset, and start of the next episode triggered.

After optimizing the episode protocol as much as we could, the training speed was still prohibitively slow for any deep learning algorithm. For this reason, we decided to use a custom velocity controller to fly the quadrotor in simulation. This is much faster than MAVROS since we can directly update a Gazebo model's state (position in this case) via messages published to ROS topics. Most importantly,

[5] http://wiki.ros.org/mavros

the overhead from communicating over MAVROS, flight mode changes, and pre-flight checks is avoided, which tremendously increases the speed of the simulation.

#### 2.2.2 Custom Velocity Controller

Each episode in this simulation environment consists of the quadrotor spawning at $(0, 0, 2)$. It then starts flying forward with a small constant velocity, and at every timestep picks one of the 9 actions (uniformally discretized yaw angles between $-45°$ and $45°$). Same as the MAVROS simulation, once the minimum laser range reading falls below a specified threshold, the end of an episode is triggered. The end is triggered pre-emptively in the sense that the quadrotor never actually collides with a cylinder. The quadrotor is then reset back to the starting position, the forest of cylinders is randomized again, and a new episode starts.

This simulation is a lot faster than before. One of the major reasons is that because we use a fast velocity controller to calculate quadrotor position updates, we are not restricted to run the simulation in real time. Obviously this is at the expense of realistic dynamics, but that is not a concern for the purpose of this project. We are able to run the simulation at faster than real-time speeds and execute upwards of $1,000,000$ environment steps per day (about half as fast as the OpenAI Gym environment for Atari games on the same machine).

### 2.3 Network Architecture

The network architecture is similar to the one in (5). The network takes in preprocessed $84 \times 84 \times 4$ images as input. The first three layers of the network are convolution layers. In order, they contain 32 $8 \times 8$ filters of stride 4, 64 $4 \times 4$ filters of stride 2, and 64 $3 \times 3$ filters of stride 1. The last hidden layer is a fully-connected layer with 512 units. The three convolution layers and the penultimate fully-connected layer all have ReLU activations. The output layer is a fully connected layer with 9 outputs, one for each action in our case.

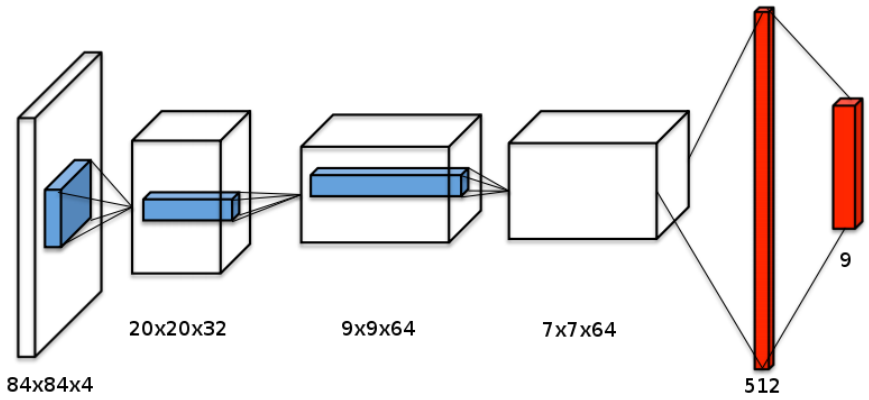


Figure 3: Network architecture used in this project. Dimensions of each layer are written below them in the diagram in $H \times W \times C$ format.

### 2.4 Learning parameters

The parameters used for our deep Q-network experiment were the following:

- Target network update frequency = 10,000 steps
- Number of burn-in steps before training = 50,000 steps
- Network training frequency = 10,000 steps
- Batch size = 32 steps
- Number of steps with environment = 1,500,000
- Max episode length = 2500 steps
- Evaluate model every 50,000 steps

### 2.5 Learning Algorithm and Rewards

The task we are working on is autonomous outdoor flight. As a result, the primary metrics to measure our performance are the duration of uninterrupted flight, and the average distance flown without intervention. We will evaluate these metrics after every policy improvement iteration of the algorithm.

The Deep Q-network updates the parameters according the following equation.

$$w \leftarrow w + \alpha \left( r + \gamma \max_{a' \in A} Q_{w^-}(s', a') - Q_w(s, a) \right) \nabla_w Q_w(s, a) \tag{1}$$

Here $Q_{w'}$ is the target network and $Q_w$ is the online network.

### 2.6 Reward Shaping

We tailor our reward function to explicitly give emphasis to four inter-related things:

- Forward flight.
- Distance from obstacles.
- Crashing into obstacles, or exiting the obstacle field.
- Reaching the goal point.

The quadrotor gets a small positive reward for every timestep that it does not crash. Each obstacle has a radially emitted potential field between $0.5m - 1.5m$ distance from it. The quadrotor receives a linearly increasing negative reward inside this potential field. At $\leq 0.5m$, there is a large negative reward to signify a crash and the end of the episode is triggered. The same occurs if the quadrotor exits the obstacle in any direction. The goal point is defined to be at $(x_{max}, 0, 2)$ for some $x_{max}$. There is a positive reward that is inversely proportional to the distance from the goal point. There is a large positive reward for being within some threshold of the goal point. At every timestep, the quadrotor receives the sum of all these rewards.

Written down, the reward at time $r_t$ is,

$$r_t = 0.25 + \sum_{o \in \mathcal{E}} \left( \frac{1}{d(q, o)} \mathbb{1}\left[0.5 \leq d(q, o) \leq 1.5\right] - 10 \times \mathbb{1}\left[d(q, o) < 0.5\right] \right) + 10 \times \mathbb{1}\left[d(q, g) < 0.5\right] - 10 \times \mathbb{1}\left[q \notin \mathcal{E}\right]$$

where $\mathcal{E}$ is the environment, $o$ denotes obstacles in the environment, $q$ denotes the quadrotor, $d(\cdot, \cdot)$ is the Euclidean distance function, $\mathbb{1}$ is the indicator function which evaluates to 1 when its argument is true, and $g$ denotes the goal point.

## 3 Results

In this section we present the results obtained in learning obstacle avoidance using monocular images in the Gym-Gazebo simulation environment.

First, we successfully achieved the goal of creating an open-source Gym-Gazebo simulator for quadrotors for deep reinforcement learning research. The code can be found in this Github link. It is important to notice that we used elements from Erlecopter's quadrotor simulation environment, which as not entirely prepared for resetting simulations, and we built more code upon it, being one of the greatest contributions the custom velocity controller.

In the training phase we ran a total of 1.4 million steps in the environment, which accounts for roughly 150 thousand episodes of the drone flying until ultimately crashing into a pillar.

As seen in Figures 4-6, only after roughly 800,000 interations with the environment we could see a significant increase in the reward per episode and also in episode length in both the training and test configurations.

It it not only our numerical results that indicate learning over time. A qualitative analysis of the quadcopter flying in the environment also shows the learned policy to be better than a random policy, and the video can be found in the following link.

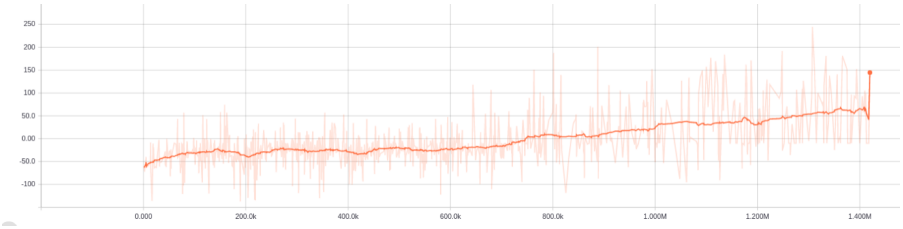


Figure 4: Train reward with respect to number of steps

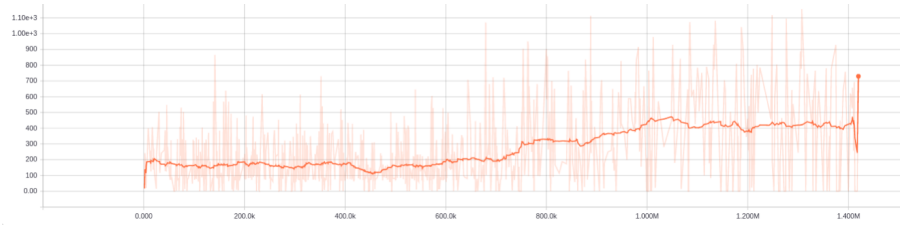


Figure 5: Train episode with respect to number of steps

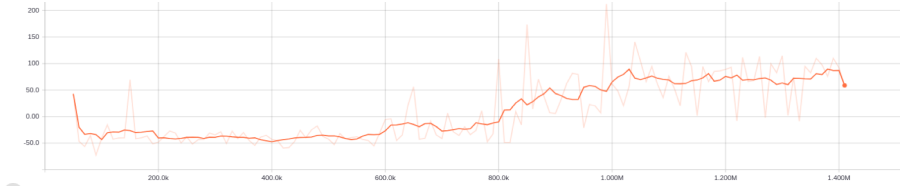


Figure 6: Test reward with respect to number of steps

## 4 Discussion

In this work we built an open-source Gym-Gazebo environment for reinforcement learning and learned a policy for obstacle avoidance in a simulated Gazebo environment where the quadcopter chooses its next yaw angle based on monocular images as input. With this learned policy we can successfully navigate around randomly positioned vertical pillars.

In our work we show that deep Q-learning have the potential to be used for end-to-end training of policies for quadcopters. Even though our results are promising, there are very important steps that still need to be accomplished before using learned policies in a real physical system:

- **High variance in rewards:** even though the general trend for rewards and episode length is to grow as the number of training samples increases, we still observe extremely high variance, which is an issue in physical systems. Future work in this area involves developing performance guarantees for learned policies
- **Action space in multiple dimensions:** our work allowed only change in yaw for the quadcopter. Future work can address learning in higher-dimensional action spaces, changing the altitude, pitch roll and yaw of the vehicle
- **Discrete versus continuous actions:** instead of discretizing commands, we can learnin policies in continuous space, allowing for more smooth movements
- **Transfer learning:** policies learned with monocular images in simulation must be transferable to real images. Two possible solutions for this issue are using photo-realistic simulators in the learning process or learning obstacle avoidance in disparity space

### Acknowledgments

The authors acknowledge Weikun Zhen for the Gazebo model of the simulated quad-copter.

## References


[1] S. Hrabar, "3d path planning and stereo-based obstacle avoidance for rotorcraft uavs," in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pp. 807–814, IEEE, 2008.

[2] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. D. Bagnell, and M. Hebert , "Learning monocular reactive uav control in cluttered natural environments," in *IEEE International Conference on Robotics and Automation*, IEEE, March 2013.

[3] N. Imanberdiyev, C. Fu, E. Kayacan, and I. M. Chen, "Autonomous navigation of uav by using real-time model-based reinforcement learning," in *2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pp. 1–6, Nov 2016.

[4] F. Sadeghi and S. Levine, "(cad)2 rl: Real single-image flight without a single real image," *arXiv preprint arXiv:1611.04201*, 2016.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[6] I. Zamora, N. G. Lopez, V. M. Vilches, and A. H. Cordero, "Extending the openai gym for robotics: a toolkit for reinforcement learning using ros and gazebo," 2016.