Automate Takeoff and Stabilize Hovering of a Micro Quadcopter

A B.Tech Project Report

Submitted for the fulfilment of BTP I

In the department of

Computer Science and Engineering

by

Amijeet Thakur
[Roll No. 17MF10004]

Under the supervision of

Dr. Sudip Misra

Abstract

Unmanned Aerial Vehicles (UAVs) have extensive use cases from defense and military purposes to new unmet demands in the industry like autonomous delivery of packages. Drones have a better advantage over other fixed wing UAVs, mainly because of their higher degree of mobility, stable maneuverability and flexible deployment. Among all kinds of UAVs, drones have a higher advantage over their other winged cousins in areas which are remote or places where the flying space is less relative to the size of the vehicle. This is mostly because drones are much smaller than most winged airplanes. Quadrotors can take off and land vertically, which makes autonomous hovering easy and quick. This is almost an impossible maneuver for other winged aerial vehicles. In this paper we propose a reinforcement learning model for a quad-copter to learn to take off from a fixed position on the ground and hover directly above it, maintaining its height and position in space. We have used an actor-critic based algorithm, multilayer perceptron (MLP) policy along with real time physics simulation environment pybullet physics, to simulate real world physics to train our drone. Our simulation results show that such a learning approach achieves successful and quick hover.

I Introduction

In recent times the interest of people in autonomous Unmanned Aerial Vehicles(UAVs) has grown thanks to recent technological developments in Machine Learning, AI and extensive research in quad-copters. With increased popularity, demand and potential applications in the future of autonomous systems, there is an increased need to take full advantage of their capabilities. With increasing share of cellular data among other wireless data traffic, developments of new technologies like 5G and need to connect remote places to a stable internet connection, keeping in mind the cost of such systems, UAVs are being seen as the solution to such a problem. UAVs could hover over a mobile phone, laptop or a vehicle and take the role of a Service Provider or a Base Station for a variety of use cases. In this paper we will discuss the application of reinforcement learning (RL) to take off a quad-copter and hover over a point in space. We have used Bitcraze's Crazyflie 2.0 nano-quadcopter as our drone for conducting this experiment along with other technologies like OpenAIGym to train our RL model and Pybullet physics to simulate real world physics for simulation and object collision in our experiment.

This paper is organised as follows. Section 2 includes research papers and description of the work related to this study. Section 3 has two parts, the first subsection contains the

basics of a quadrotor like the kinematics, dynamics and propulsion models of a quadrotor. The second subsection contains a brief introduction to reinforcement learning and its broad types. Section 4 describes the framework and technologies we have used in brief. Section 5 contains 2 subsections. We show the experimental setup in the first subsection and the results in another. This section contains the environmental details, workflow of the controller and the reward function. Conclusions are added to section 6 and finally references of work related to our work have been added to section 7.

Ⅱ Literature review

In this section, we describe the previous studies concerning UAVs using the learning controller. Takuya Sugimoto *et al.* applied Reinforcement Learning to the AR.Drone 2.0 (Parrot) quadrotor helicopter [1]. They applied Q-learning technique to hover the UAV over a point on the ground having a diameter of 100 mm. They used a real drone for learning purposes rather than using a simulation. Their work was compared to various other simulation based studies using reinforcement learning to control a quadcopter.

Anush Manukyan *et al.* have used a deep reinforcement learning model on a multicopter for learning a stable hovering task [2]. They present a framework based on OpenAI GYM, Gazebo, Robotic Operating System and RotorS MAV simulator. For training they have used a model-free, on-policy, actor-critic based algorithm called Trust Region Policy Optimization (TRPO). Two neural networks have been used as nonlinear function approximators. Their experiments show that such a learning approach achieves successful results, and facilitates the process of controller design.

Jichiang Tsai *et al.* present a method using ArUco markers as a reference to improve the accuracy of a drone on straight take-off, flying forward, and landing based on Reinforcement Learning (RL) [3]. Even they have used a Q-Learning algorithm to train their reinforcement learning model. Experimental results show the learning process can achieve more accurate results and converge more quickly with more states and actions adopted by the Q-Learning method.

Bou-Ammar et al. [7] propose a nonlinear autopilot for quadrotor UAVs based on feedback linearization for precise and fast stabilization. Their approach is based on reinforcement learning, where after each generated control action the agent receives feedback by a simple reward function. Using a value iteration method, they achieve faster convergence of the learning process.

Pham et al. [12, 13] propose a RL based framework for a UAV to reach a point in 3D space where the exact mathematical model of the environment may not be available. Their framework is a combination of a PID controller and a Q-Learning technique. For conducting experiments they have used the Parrot AR.Drone 2.0. Their experimental results show that the agent is able to reach the desired point in the least number of steps.

Ⅲ Background

In this section we present a brief introduction of the kinematics and dynamics of a quadrotor and an introduction to reinforcement learning and actor-critic methods.

Quadrotor Platform

Quad-rotors belong to the class of Vertical Take-off and Landing (VTOL) aerial vehicles. As other VTOL vehicles, they are inherently unstable and need well-designed controllers for successful flight. A Quadrotor has six degrees of freedom (6 DoF) and includes nonlinear aerodynamics as well as actuator dynamics and saturation. All states are controlled by controlling the thrust of the four rotors. More about kinematics and dynamics of a quadrotor can be found in the following works [4], [5]

A picture of our aerial platform, Crazyflie 2.0, is depicted in Figure 1 and its main specifications are listed in Table I. Our drone of choice in this experiment is Bitcraze's Crazyflie 2.0 nano-quadcopter. Our Crazyflie includes a 10 DOF IMU sensor, which includes a 3 axis accelerometer sensor (MPU-9250), a 3 axis gyro sensor (MPU-9250), 3 axis magnetometer sensor (MPU-9250) and a high precision pressure sensor (LPS25H).

Specification	Value
Size	9 cm^2
Weight	27 g
Battery	240mAh LiPo battery, 3.7V
Flight time	7 mins
Motors	4.2V, 810mA, 12000 rpm/V
Propellers	45mm

Table 1 : Some important specifications of crazyflie 2.0

A. Kinematics

To proceed with deriving the kinematic model of our drone we would need two coordinate systems, one which will be fixed to the body of the drone, ie. the center of mass (CM) of the drone, B = {X^B, Y^B, Z^B} and the second, fixed to the ground G = {X^G, Y^G, Z^G}. The Z^B axis of the body fixed frame is assumed against the thrust direction and X^B is pointing towards the first motor. The Z-X-Y quadrotor orientation of the drone in the ground frame is given by the rotation matrix R

$$\mathbf{R} = \begin{bmatrix} c_\beta c_\alpha & -c_\beta s_\gamma & s_\beta \\ c_\alpha s_\gamma - c_\gamma s_\alpha s_\beta & c_\alpha s_\gamma + s_\alpha s_\beta s_\gamma & c_\beta s_\alpha \\ s_\alpha s_\gamma + c_\alpha c_\gamma s_\beta & c_\gamma s_\alpha - c_\alpha c_\gamma s_\beta & -c_\alpha c_\gamma \end{bmatrix} (1)$$

Here α, β and γ are the roll pitch and yaw angles of the drone respectively and c_θ and s_θ being sine(θ) and cosine(θ) respectively.

As mentioned before, our crazyflie also has a 3 axis IMU gyroscope sensor which would give us roll rate, pitch rate and yaw rate at every timestep. Let $\boldsymbol{\omega}_\beta = [\, \omega_{,x}\, \omega_y\, \omega_z\,]^T$ be the angular velocity of the the drone in the body fixed frame. Let $\boldsymbol{\Theta} = [\, \alpha\, \beta\, \gamma\,]^T$ be the Euler angles vector. Where ω_x , $\omega_{,x}$, $\omega_{,z}$ represents the angular velocity about the x axis, y axis and the z axis respectively. The relation between the $\boldsymbol{\omega}_\beta$ and the three euler angles is given by

$$\omega = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} c\theta & 0 & -c\phi s\theta \\ 0 & 1 & s\phi \\ 0 & s_\theta & c\phi c\theta \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \mathbf{H}\dot{\theta} (2)$$

After solving for time derivative of Euler angles, we get

$$\dot{\theta} = H^{-1}\omega = \begin{bmatrix} 0 & s_\alpha t_\beta & -c_\alpha t_\beta \\ 0 & c_\alpha & s_\alpha \\ 1 & s_\alpha / c_\beta & c_\alpha / c_\beta \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} (3)$$

Since the above equation 3 is non linear, the vector $\boldsymbol{\Theta}$ can only be obtained by numerical integration.

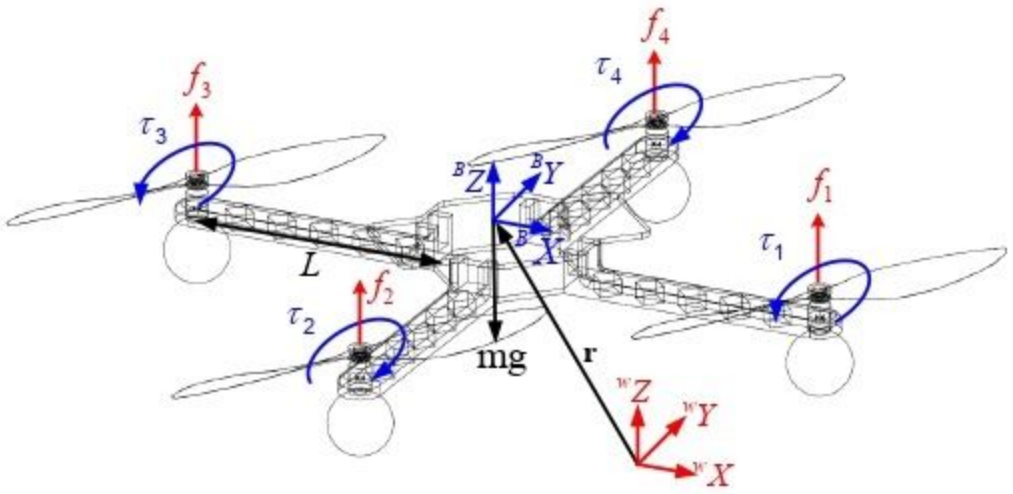

Figure 1 : Figure showing the ground frame and the body fixed frame.

B. Dynamics

During hovering our drone would be subjected to two types of forces, the force of the gravity (mg) and force generated by the thrust of motors, $T(f_1+f_2+f_3+f_4)$. Representing the above system using Newton-Euler formulation:

$$m\ddot{r} = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} + R \begin{bmatrix} 0 \\ 0 \\ f_1 + f_2 + f_3 + f_4 \end{bmatrix} (4)$$

Also the $\boldsymbol{\tau}$ or the resultant torque applied to the drone system, having a moment of inertia I, and angular acceleration ω is:

$$\tau = I\dot{\omega} + \omega \times I\omega (5)$$

C. Propulsion Model

Here we would find the lift force f_i and the τ_i of each motor. The steady state thrust force generated by each motor in air can be modeled as

$$f_i = k_f \omega_i^2 (6)$$

Besides lift force f_i(i = 1, 2, 3, 4) the rotating motors would also produce torque τ_i(i = i, 2, 3, 4) And the torque vector, τ is also found to be proportional to the squared of the rpm of each motor, therefore

$$\tau_i = k_\tau \omega_i^2 (7)$$

Now in the very special case of hovering, the forces in the z direction should sum to zero. Also the torque applied on the frame of the quadrotor should be zero. Hence:

$$f_1 + f_2 + f_3 + f_4 = mg \qquad (8)$$

Since the direction of rotations of the rotors opposite to each other are opposite, we find the torques balance themselves always, in the case of hovering.

Reinforcement Learning

In this subsection, we explain behavior learning using reinforcement learning [6]. In the reinforcement learning framework, a robot learns a suitable state-action mapping without prior knowledge of the dynamics of the robot and its environment [7]. The optimal policy is learnt by trial and error interactions with its environment. The agent receives a positive or negative reward for each action taken. Having a goal of maximizing the cumulative reward, the agent learns to take the correct sequence of actions.

Reinforcement Learning can be carried out through two main methods namely :

Value based: They try to find or approximate the optimal value function, which is a mapping between an action and a value. The higher the value, the better the action. The most famous value based RL method is Q learning and all its enhancements like Deep Q Networks, Double Dueling Q Networks, etc.

Policy-Based: Policy-Based algorithms like Policy Gradients and REINFORCE try to find the optimal policy directly without the Q -value as a middleman.

Usually policy-based techniques are better for continuous and stochastic environments, have a faster convergence, while Value based are more sample efficient and steady. In our case we have used an Actor-Critic Method to take advantage of all the good stuff from both value-based and policy-based while eliminating all their drawbacks. These algorithms work by splitting the model in two, one for computing an action based on a state and another one to produce the Q values of the action. The actor takes as input the state and outputs the best action. It essentially controls how the agent behaves by learning the optimal policy (policy-based). The critic, on the other hand, evaluates the action by computing the value function (value based). Those two models participate in a game where they both get better in their own role as the time passes. The result is that the overall architecture will learn more efficiently than the two methods separately.

After each action selection, the critic evaluates the new state to determine whether things have gone better or worse than expected. That evaluation is the error:

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t),$$

if *p(s, a)* are the values at time of the modifiable policy parameters of the actor, indicating the tendency to select (preference for) each action when in each state . Then the strengthening or weakening described above can be implemented by increasing or decreasing *p(s, a)* by:

$$p(s_t, a_t) \leftarrow p(s_t, a_t) + \beta\delta_t,$$

where β is positive step-size parameter.

This is just one example of an actor-critic method. Another common dimension of variation, as in reinforcement comparison methods, is to include additional factors varying the amount of credit assigned to the action taken, a_t. For example, one of the most common such factors is inversely related to the probability of selecting a_t, resulting in the update rule:

$$p(s_t, a_t) \leftarrow p(s_t, a_t) + \beta\delta_t\Big[1 - \pi_t(s_t, a_t)\Big].$$

IV Framework Structure

Training is one of the most critical steps in the learning and development of robot systems through reinforcement learning. Since aerial robots are a high priced system, field training could become one of the main problems in reinforcement learning and development. When using real robots, whenever there is a crash or a failure of a system, the robot might get damaged by our trail and error learning methodology. Whereas in the case of simulation, our robot to be trained has no potential harm during training. Therefore the best alternative is to use a simulator that mimics the dynamic behavior of a real-word quadcopter and its environment. The training of the robot can be performed safely and the optimal policy can be learned efficiently, which can later be run on the real robot, without any changes. Ideally, the agent should not be able to differentiate between a simulated environment and the real world.

In this work we designed such a framework by combining a few open source toolkits, such as: OpenAI Gym [8] and Pybullet Physics [9].

OpenAI Gym is a toolkit that provides several robotic environments for developing and comparing reinforcement learning algorithms[10]. pybullet is an easy to use Python module for physics simulation, robotics and deep reinforcement learning based on the Bullet Physics SDK. The model of our drone, crazyflie 2.0 can be loaded as a URDF file and an environment can be created with real world physics parameters like gravity, density of air, etc of our choice where our quadcopter would train.

By combining all these two powerful tools together, we obtained a new gym-pybullet framework which provides a real-world like environment for training our agents. The learning process in pybullet-gym is being executed in an episodic setting, where the agent learns to perform the given task through a series of episodes. In each episode, the agent interacts with its environment until the environment reaches a terminal state. There are a

few ways to define the terminal states either by defining constraints or by defining a timestep. In this work the agent can interact with its environment until it breaks down the defined constraints or if it overpasses 500 steps, meaning that it performs 500 actions during one episode.

V Experimental Validation

In this section we present the experimental setup, reward function, the neural network structure and results in detail.

A. Experimental Setup:

This subsection deals with how we train our RL model for continuous robotic control problems.

1. **Environmental Details:** As mentioned before our pybullet-gym framework, used for training our agent to perform a takeoff and hovering task is in a continuous state and action space domain. Like any other training, the core of the learning procedure is based on two very important functions, the *reset* and the *step* function. The reset function is called when we finish training one episode. This function resets the environment to its initial conditions. The step function is used when an episode is being trained. We have here used a very widely implemented step function. At each step, the step function returns the following very important outputs:

- Observation: Observation is an object which contains 20 values related to the environment and the agent. It contains the x, y and z position of the drone in the world frame(in meters, 3 values); Quaternion orientation in the world frame(4 values); Roll, pitch and yaw angles in the world frame(in radians, 3 values); the velocity vector in the world frame(in m/s, 3 values); the angular velocities in the world frame(in rad/s, 3 values); Motors' speeds (in RPMs, 4 values).
- Reward: Reward is a floating point number that denotes the reward the agent gets after performing each action.
- Done: Done is a boolean value which governs whether execution of a step should be halted and the environment should be allowed to reset. Returning a *True* value of done resets the environment when the above mentioned conditions pass, ie. when an episode finishes or one or more constraints fail.

2. **Workflow of the controller:** The workflow of our controller is fairly simple. Our agent takes the position and orientation as the input and returns 4 values, that are the RPM of the 4 motors as the next action. Since the drone

starts training from a position on the ground and our task is to take off and hover. We apply particular bounds to our continuous action space. Our action space is continuous and starts from ω_{min} and ends at ω_{max}. ω_{min} being the minimum RPM required by each motor to hover the crazyflie and ω_{max} being the max RPM that a single motor of crazyflie can touch.

3. **Reward function:** The most critical part of reinforcement learning is the reward function. Only the reward function is capable of showing how our agent is performing while training. A good reward function would give us accurate results in less time, whereas a bad reward function would give us just the opposite. There could be two types of reward functions, one which only give a positive reward when the task is complete and the other, which increments reward as our agent reaches closer to the goal. Such a reward function is called a shaped reward function and helps our agent to converge faster. The former reward function doesn't give enough information to the agent about its behavior[11]. Such a function gets feedback only when it succeeds or fails the task. Meaning that, while the agent is exploring its environment, it does not receive any information if it is performing good or worse. And only when it stumbles to the success condition then it can learn the sequence of actions that brought it there. In our case, we have designed a smooth, shaped reward function which provides continuous information about the quadcopter's behaviour. Our function is the negative of the distance between the drone's current position and the desired position to hover. Our results show that our quadcopter successfully trains to takeoff and hover over a point in a reasonable amount of time. We successfully train our agent in 500000 timesteps and our agent completely trains in about 41.5 minutes on average which boils down to a 200 Hz running model on a typical 4 gigabyte machine.

B. Results

Both the learning and the testing procedures have been performed in the previously mentioned gym-pybullet framework, using crazyflie 2.0 nano-quadcopter as our learning agent. The learning procedure starts with our drone at position $x = 0$, $y = 0$ and $z = 0$. From there our drone learns to take-off and hover over the origin. We successfully train our agent in 500000 timesteps and our agent completely trains in about 41.5 minutes on average which boils down to a 200 Hz running model on a typical 4 gigabyte machine.

All experiments and training have been executed on a machine, equipped with 4GB of RAM and a 2.2Ghz AMD A8 CPU running on Ubuntu 18.04 operating system.

VI Conclusion

In this work we present a Reinforcement Learning approach that trains a quadcopter, in our case, Crazyflie 2.0 nano-quadcopter to take off and hover at a point in 3D space. We used an Actor-Critic learning algorithm for this purpose, using 2 neural networks, having 64 nodes each. The action space of our problem was continuous, the RPM of the 4 motors of our quadcopter. Our learning procedure is performed in a gym-pybullet framework where we have tried to minimise the difference between our simulation and a real world physics environment.

The whole learning procedure is performed in about 1050 policy iterations which is about half a million simulation steps. Our results showed that using such learning technique it is possible to successfully train an agent to perform a stable hovering task in a continuous action space domain.

As future work we would try to implement such a controller on a real Crazyflie. We would try to replicate this work using only the onboard sensors, improving the performance of our learning algorithm. We would also like to integrate takeoff as well as landing optimally on a single quadcopter controller.

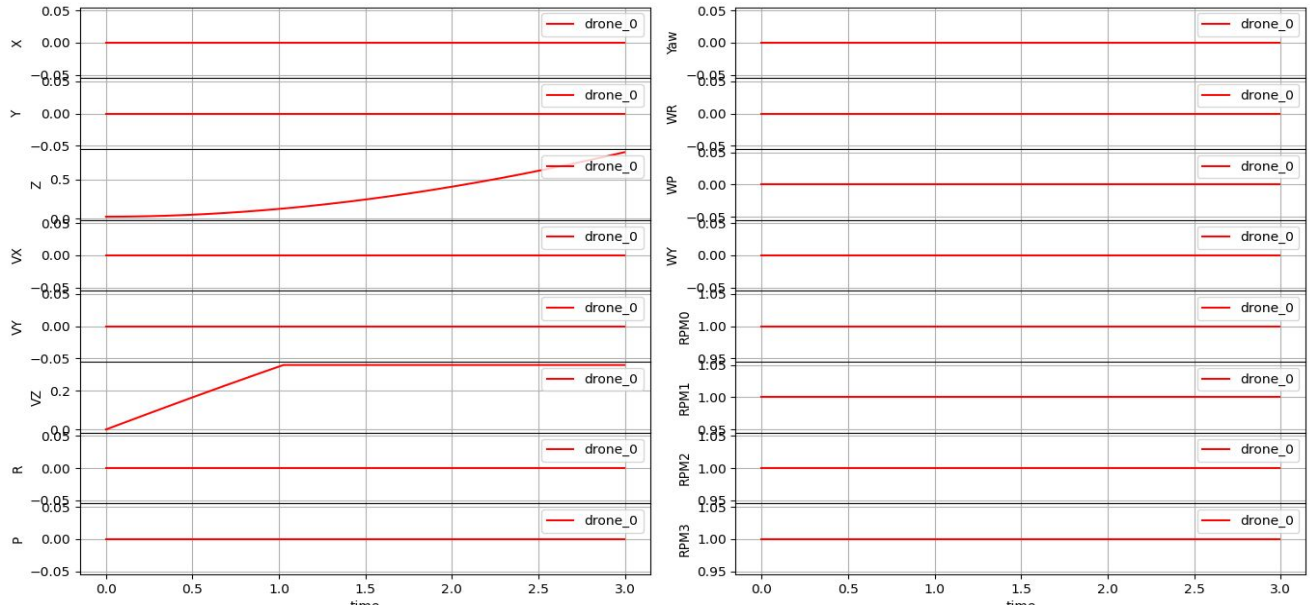

Figure 2: A graph showing test results of our trained model.

Ⅶ References

1. Takuya Sugimoto and Manabu Gouko. “Acquisition of hovering by actual UAV using reinforcement learning.” Information Science and Control Engineering (ICISCE), 2016 3rd International Conference on. IEEE, 2016.
2. Anush Manukyan, Miguel Olivares-Mendez, Matthieu Geist and Holger Voos. “Deep Reinforcement Learning-based Continuous Control for Multicopter Systems.” International Conference on Control, Decision and Information Technologies (CoDIT), 2019 6th International Conference on. IEEE, 2019.
3. Jichiang Tsai, Peng-Chen Lu, and Ming-Hong Tsai. “Accuracy Improvement of Straight Take-off, Flying Forward and Landing of a Drone with Reinforcement Learning.” International Conference on Consumer Electronics (ICCE-TW), 2019 International Conference on. IEEE, 2019.
4. Gabriel Hoffmann, Haomiao Huang, Steven Waslander, and Clarie Tomlin.“Quadrotor helicopter flight dynamics and control: Theory and experiment,” in AIAA Guidance, Navigation, and Control Conference, South Carolina, 2007.
5. Luis Rodolfo García Carrillo, Alejandro Enrique Dzul López, Rogelio Lozano, Claude Pégard. “Modeling the quad-rotor mini-rotorcraft,” in Quad Rotorcraft Control. Springer, 2013, pp. 23–34.
6. Jens Kober, Andrew Bagnell, and Jan Peters. “Reinforcement learning in robotics: A survey.” The International Journal of Robotics Research 32.11 (2013): 1238-1274.
7. Haitham Bou-Ammar, Holger Voos, and Wolfgang Ertel. “Controller design for quadrotor UAVs using reinforcement learning.” Control Applications (CCA), 2010 IEEE International Conference on. IEEE, 2010.
8. OpenAI Gym toolkit https://gym.openai.com
9. Pybullet physics engine https://pybullet.org/wordpress/
10. Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, Wojciech Zaremba. “Openai gym.” arXiv preprint arXiv:1606.01540 (2016)
11. Aditya Gudimella, Ross Story, Matineh Shaker, Ruofan Kong, Matthew Brown, Victor Shnayder, Marcos Campos. ”Deep reinforcement learning for dexterous manipulation with concept networks.” arXiv preprint arXiv:1709.06977 (2017).
12. Huy Xuan Pham, Hung Manh La, David Feil-Seifer, Luan Van Nguyen. “Reinforcement Learning for Autonomous UAV Navigation Using Function Approximation.” 2018 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR). IEEE, 2018.

13. Huy Xuan Pham, Hung Manh La, David Feil-Seifer, Luan Van Nguyen. “Autonomous uav navigation using reinforcement learning.” arXiv preprint arXiv:1801.05086 (2018).