The University of Sydney

Faculty of Engineering

School of Aerospace, Mechanical and Mechatronic Engineering

Application of Machine Learning in Unmanned Aerial Vehicle Control

A thesis submitted in partial fulfilment of requirements for the degree of Bachelor of Engineering (Honours)(Aeronautical)

by

Junfeng Guo
460058788

November 7, 2019

Statement of Contribution

This thesis project is completed between February 25, 2019 and November 7, 2019, entitled "Application of Machine Learning in Unmanned Aerial Vehicle Control" and under the supervision of professor K.C Wong, which has been accepted by the School of Aerospace, Mechanical and Mechatronic Engineering of the University of Sydney as partial fulfilment of the requirements for the degree of Bachelor of Engineering(Honours)(Aeronautical).

The following area of works were carried out for this thesis:

- Reviewing of existing literature of reinforcement learning algorithm and investigation of improvement about deep reinforcement learning components.
- Developing of unmanned aerial vehicle controller for both simulation used and real drone testing.
- Construction of simulation environment and real drone testing environment with detailed component used and detailed explanation of the function in each component.
- Training and Robustness Testing of the algorithm with two flight tasks in increment of complexity.
- Benchmarking of real drone control compare to result in simulation

Junfeng Guo

Associate Professor KC Wong

Abstract

Machine learning is being developed for this recent years and becoming a more and more popular study in different fields and industries. The developing of the reinforcement learning from basic Q learning to asynchronous algorithm allows more practical autonomous control can be achieved.

In unmanned aerial vehicle, autonomous control always be a heat and challenge topic. In general, autonomous control can be achieved by designing Proportional–Integral–Derivative controller. Traditional method has been proven a reliable control in different extent of autonomous, however, a few flaws exist which make autonomous control with machine learning comparable. In some of the control task, low level control of the UAV required. Multiple sensors needed to be calibrated by using PID controller. This could lead to a complex control system design. On the other hand, designing PID controller for autonomous task is usually applied in explicit environment, whereas autonomous control with machine learning can be achieved in implicit and uncertain environment. As a result, investigating of application of machine learning in unmanned aerial vehicle control is essential to explore the performance of machine learning.

A few authors have pointed out successful applications of machine learning in unmanned aerial vehicle control. Autonomous control with asynchronous algorithm such as deep deterministic policy gradient, or proximal policy optimization show a better performance in UAV control compare to traditional value-based method. These algorithms contain properties of solving large data and real time problem which allows to achieve continuous control. This is essential in UAV since the action taken by the drone cannot be discritized.

In this thesis, a literature review of designing a sufficient training agent is given. A few aspects are covered including, learning rate which can affect the training accuracy, batch size which influence the batch normalization of training data, activation function that to prevent gradient vanish and designing reward function that to generate better guidelines for training. A WIFI connection method is provided to ensure a stable testing. A framework to apply machine learning in continuous control is set up for both simulation environment and real drone testing environment. A drone controller is designed for the use of both simulation and real drone control.

An increasing level of objectives are trained and the results are demonstrates that using of DDPG provides autonomous control up to some extent. Two robustness tests are

done. The trained model shows robust in the position recovery tasks whereas the target chasing is not achieve in second test. A benchmarking of real drone control compare to the simulation has done. The results show the real drone has different level of side slipping at takeoff and hover. And some extents of noise are captured in the flight orientation which needed to be adjusted by design a PID controller for tuning the action.

Contents

List of Figures

List of Tables

Nomenclature

Symbols

g_t	gradient at current mini-batch
J	Policy value
m_t	Mean of gradient
$\hat{m}_t$	Bias-correct mean of gradient
n_t	Variance of gradient
$\hat{n}_t$	Bias-correct variance of gradient
q_0, q_1, q_2, q_3	Quaternion
v_x	linear velocity in x direction
v_y	linear velocity in y direction
v_z	linear velocity in z direction
x	x position
y	y position
z	z position

Greek Symbols

β^t	Unbias factor
η	Learning rate
γ	discounted factor
ϕ	Roll angle
$\dot{\phi}$	rate of change of roll angle
Φ	some functions over the state
ψ	Yaw angle
$\dot{\psi}$	rate of change of yaw angle
θ	Pitch angle
$\dot{\theta}$	rate of change of pitch angle
θ^π	Policy weight

Abbreviations

CPU	Central Processing Unit
DDPG	Deep Deterministic Policy Gradient
DNS	Domain Network System
DQN	Deep Q Network
GPU	Graphics Processing Unit
PC	Personal Computer
PID	Proportional–Integral–Derivative
PPO	proximal policy optimization
ReLU	Rectified Linear Units
RL	Reinforcement Learning
ROS	Robotic Operation System
SDK	Software Development Kit
SGD	Stochatsic Gradient Descent
UAV	Unmanned aerial Vehicle

Chapter 1

Introduction

Machine learning becomes more and more popular since 1990s [3] followed by the developing of variety of mathematical tools and logic theories. The more integrated algorithms make application of machine learning in different industries such as aeronautical engineering become possible. Machine learning is a broad concept of using theories from computer science with suitable mathematical tools to achieve some extent of artificial intelligence. The term "Learning" indicates the different tools used to obtaining the numerical knowledge from a selected goal. Different use of the algorithm logic lead to diversity of machine learning. The three most common categories are supervised learning, unsupervised learning and reinforcement learning. In supervised learning, input data and desired output are passed to the computer and classify the input data from the desired output. In unsupervised learning, only input data is given and the algorithm can categorizes the data based on the explored features. Therefore, these two type of learning are mainly used in data mapping or data analysis. Unlike supervised learning and unsupervised learning, reinforcement learning is focusing on finding optimized action that can achieve the given goal based on the corresponding observation in the environment. Therefore, reinforcement learning is built for the application based on taking action such as video games and robotic control.

In unmanned aerial vehicle autonomous control, a general method to achieve the goal of automation is to implement different type of controller. The well developing of techniques to the controller provides higher performance in autonomous control. However, the traditional autonomous control contains several disadvantages, and some tests [4] has shown better performance of using reinforcement learning. Firstly, increasing number of autonomous control objectives can lead to more complicated in design the system. For instance, multiple waypoints tracking with low level control may require large amount of controller for orientation control, position control and velocity control. Hence, the complexity of designing the entire controller increase follow by the increasing of difficulties for each task. Furthermore, tuning of control value in controller such as Proportional-Integral-Differentiate controller requires different extent of compromise. For instance, in some cases, large damping will occur if fast converge required. This indicate the control system is less robust compare to using reinforcement learning as controller since less tuning is required. Another advantages of using reinforcement learning is that, task with implicit information can also achieve success of autonomous control. If a case requires to

maintain less power consumption during the flight, an observation about battery power can be added to reach the objective of minimized power usage. However, in traditional method, power consumption calculation is required. More information is needed to achieve the goal. Therefore, reinforcement learning shows competitive to traditional autonomous control. On the other hand, the selection of RL algorithm is crucial according to the training task. There are two main types of RL in model-free category: value-based reinforcement learning and policy-based reinforcement learning [5]. In value-based algorithm, a popular algorithm is Q-learning which optimised the action based on Q-value, and in policy-based algorithm, a popular algorithm is called deterministic policy gradient the optimised action is picked based on a series of best strategies. This lead to two different characteristics that, valued-based RL usually suitable for discretised action but for policy-based RL continuous action is allowed. Therefore, policy-based RL will be mainly applied in this thesis.

The applications of reinforcement learning in UAV autonomous control are presented by several teams. ETH team has presented excellent performance on autonomous control by using proximal policy optimization [6]. However, the drone platform is built specialised for the training task. This lead to machine learning application can only suitable for a specific drone and the Vison system is used for location capture. This system is less cost performance compare to OptiTrack which makes the training less applicable [7]. At the same time, lower level control is presented in the report. The application therefore required higher accuracy of the action control. Unlike the ETH team, there are applications combine both PID controller and RL to achieve autonomous control [8],[9],[10]. Siyi and their teams [8] achieves a satisfactory result using PID controller with typical Q learning in autonomous target tracking whereas William's team [9] and Roman's team demonstrates good performance in traditional trajectory control such as altitude control and landing control. However, this combination of using PID controller with machine learning only show to achieve simple trajectory and are not purely RL autonomous control. Accordingly, there is a gap to implement more complex RL algorithm to achieve higher difficulties of trajectory autonomous control with higher cost performance equipment for position and orientation capture. The main objective the thesis is to focus on the application of reinforcement learning in UAV autonomous control, and explore how well the performance could be. Simulation-to-real approach will therefore be illustrated through graphical visualisation.

In this thesis, Deep Deterministic Policy Gradient is implemented. The performance for using DDPG for autonomous control is made and the results will be visualized and discussed in the following sections. Simulation setup and real drone test setup will be shown in the following. Some future works of this thesis will be discussed.

Chapter 2

Literature Review

In the previous chapter, the objective for the thesis is presented. Reinforcement learning is the main focus on this thesis. The goal of literature review is to demonstrate the deep deterministic policy gradient algorithm in details so that to provide enough knowledge to apply in autonomous control. The finding of the literature review is essential since each part of the algorithm plays important role that could affect the training result. The result can further influence the accuracy of model if apply to physical testing. Hence, the detail research for each component in DDPG are necessary. On the other hand, the implementation of the trained model from simulation to real world testing could help to provide a map of how well the algorithm can perform in real life. As a result, some researches for network connection is vital.

There are several challenges that could affect the precision and accuracy in the training result. A good constructed neural network allows to present a high standard of training. The two most important component: learning rate and activation function, influence the training in a great extent. The investigation of learning rate and activation function is therefore essential. On the other hand, memory replay in DDPG provides experience study for the agent. Batch size provides the scale of memory that the agent can study, hence the volume of each batch also vital. To correctly guide the agent, reward function plays an important role in the training. Hence, the formation of the reward and penalty mechanism is needed to be investigated. All these factors demonstrate the performance of training. Incorrect of using the component can lead to fail training and cannot be used on the real drone testing. A precise and accurate neural network model can therefore provide better performance on the actual control.

In this chapter, literature reviews is organised to give a brief introduction of deep deterministic policy gradient algorithm, followed by some details explanation of each core component inside the algorithm, fulfilled with functions, and the importance of these components will be discussed. These core elements will be discussed in details including the batch size in memory replay, the learning rate for the agent, the activation function for automatic tuning output values and the reward engineering for training guidance. Later in the thesis, the efficient functions and values for these components will be selected based on the theories mentioned in this chapter. Moreover, The network connections between each device is essential because the fluency of connection affects the stability of testing,

thus some methodology for the network connection will be mentioned.

2.1 Deep Deterministic Policy Gradient

One of the most popular reinforcement learning algorithm is called Deep Q learning which is firstly published by Google Deepmind at 2013 [11]. Deep Q learning combines the properties from both deep learning and Q learning. The upgrade of reinforcement learning allows the agent to learn from the estimate Q values as well as the Q-action through experience replay. By integrating the neural network, deep Q learning has much faster learning speed than traditional Q learning. Even though DQN has proved to have human-level performance in some video games, the characteristics of Q learning limit the performance in more practical autonomous control. In DQN, if the Q function such as reward function is too complicated, the agent cannot learn properly. Moreover, DQN has a large cost in discretizing the action space. As a result, DQN may fail to learn in more complicated environment such as continuous action robotic control.

In 2015, Google Deepmind published a new algorithm called deterministic policy gradient which based on asynchronous learning with learning architecture of actor-critic network and combined with deep neural network [12]. This algorithm allows to allocate multi-thread for training. This means the training algorithm can allocate more computer resources such as more CPU core or GPU core used for training. In DDPG, Q value is estimated but not be used as an action. Instead, the policy gradient is evaluated to obtain the desire action. DDPG provides estimation about possibilities distribution of action, therefore, DDPG allows to solve the control problem with continuous and stochastic action space. The figure 2.1 shows the general architecture of DDPG algorithm. Two groups of neural network are required to achieve the actor-critic learning. In the actor, action evaluation network provides real-time estimation for the next action. The generated action will actually proceeded. the other network called action target network which duplicate all the actions the robot has done. Both of the action networks then pass the values to the two corresponding critic network. The critic network therefore evaluate the action at real-time and from past experiences. The combination of learning method lead to higher efficiency since the real time action is evaluated at the mean time the random past experience also be studied.

DDPG algorithm has general learning procedure as shown (figure 2.2)in the following pseudo code provided by Google Deepmind [12]. As per the algorithm, the critic network and actor network is initially created with some starting weight values. Once the training begin, an initial state will be observed from the robot and be transferred to the actor network. Corresponding action is provided based on the initial state. Reward and new state can therefore be observed once the action has done, all these experiences will be stored in

the memory for experience replay study. The evaluation is calculated which denoted as y_i and the corresponding loss is measured. Finally, the actor policy is updated based on policy gradient provided by David Siler in 2014 [13]. Finally, the target networks that is used for memory replay are updated. This form the entire structure of DDPG.

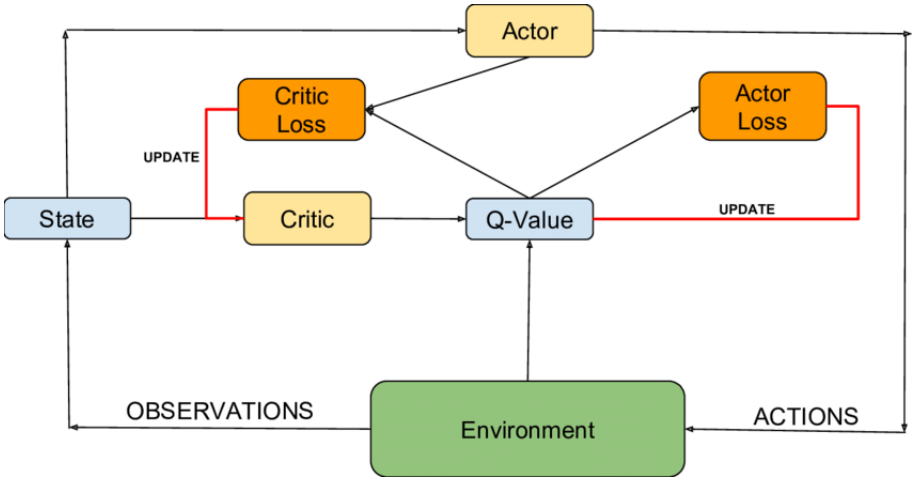

Figure 2.1: Learning Architecture of DDPG Algorithm

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Figure 2.2: **Algorithm** Deep Determininstic Policy Gradient

This state-of-art algorithm therefore shows the potential to achieve continuous control. According to the mechanism, the dual network allows evaluation at real-time and past experience. In robotic control, it is essential that each action the robot has taken will

lead to either good or bad consequence. Robot might reach desire goal in coincidence, but the action at that period is trainable. Dual network increases the intelligence of learning more complex task meanwhile keeping high efficiency. Oppositely, because of the episodic learning, DQN learn from experience with good and bad experience. This can lead to higher difficulty of learning complicated task especially for continuous control. As a result, using of DDPG to achieve the continuous control will be priority in this thesis, and some comparison of DQN will be done in later chapter.

2.2 Learning Rate

Learning rate is one of the important hyperparameter in reinforcement learning especially in algorithm such as back-propagation and DDPG. The selection of learning rate influence how fast the agent can response to observed error [14]. In general, the different values of learning rate can affect the accuracy and speed of training. Equation 2.1 shows the policy descent used in DDPG which η denoted as learning rate [15], and the figure 2.3 demonstrates an idea of how the learning can influence the final training result.

$$\theta^{\pi} \leftarrow \theta^{\pi} + \eta \bigtriangledown_{\theta^{\pi}} J(\theta^{\pi}) \tag{2.1}$$

For the algorithm using policy decent, the policy value J is required to converge to zero. The minimum location of policy indicates a successful training with high accuracy. As per the equation and graph, the if the learning rate is set to a very low value, the changing of the policy gradient becomes less sensitive. As a result, the variation is too small, therefore the network requires more update. In this case, the training episode will increase and lead to fairly slow in training speed. On the other hand, if the training rate is too high, the variation of the policy gradient is too sensitive. the large learning rate, therefore, causes drastic updates which lead to divergent behaviours. In this case, the robot will show behaviour of repeating action that is not desired to the objective. Thus, an optimised value of learning rate can increase the training speed. Moreover, an optimised learning rate could reduce the training loss. As per the algorithm showed in figure 2.2, the loss equation is estimated based on the difference between the training reward y_i and Q value, which y_i is estimated according to policy descent. As a result, a bad learning rate can lead to high loss in the training.

However, the policy gradient curve is not exactly quadratic shape can could be varied. If constant learning rate is used, slow converge or divergent will occur at some local points. As a result, to avoid this issue, some mathematical tools are used to vary the learning rate through out entire training. Schedule learning rate and adaptive learning rate are the two main type of categories. IN schedule learning rate, single math equation is used for planning the learning rate. One of the famous schedule learning rate optimizer is

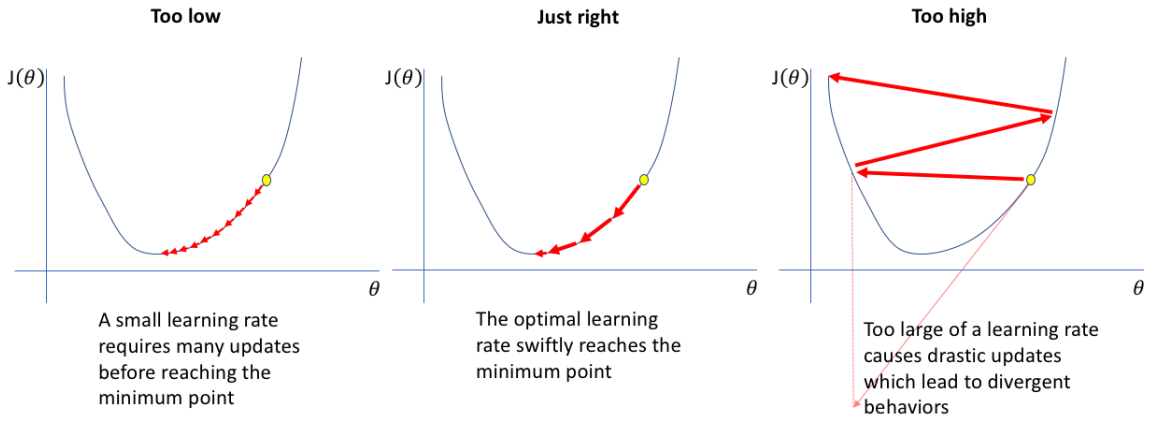

Figure 2.3: Effect of Learning Rate [1]

Stochastic Gradient Descent (SGD). This optimizer allows to update the policy gradient at each sample batch. For the large amount of data, the update policy gradient can converge before all the sample is run due to the episodic update [16]. However, this also lead to a problem of cost fluctuation even though the updating speed is fast. The accuracy for this optimizer therefore reduce and cannot reach the global optimization even though local minima is achieved.

Adaptive Moment Estimation (Adam) is another popular optimizer that can make the learning rate self-adapted according to the training data. This optimizer has similar characteristics to moment optimizer [17] which developed based on Newton Momentum Law. The policy descent keeps an exponentially decaying average of past gradient m_t [16]. In the equations, m_t and v_t estimates the mean and variance of the gradient respectively.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{2.2}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{2.3}$$

The athors of Adam observe that the bias will become zero when mean and variance are initilized at initial time step [18]. Therefore, they elilimiate the bias by computing bias-corrected the mean and variance:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2.4}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{2.5}$$

Hence, the gradient descent can be updated by using Adam update rule (equation 2.6). Which β_1 is purpose to have value of 0.9 and β_2 is purposed to be 0.999 and ϵ to be 10^{-8}. According to author of Adams, they have demonstrated a satisfactory performance

of the optimizer for multi neural network. The training cost drop much faster than other optimizer such as SGD and AdaGrad [19]. This is essential for implementing DDPG algorithm since two group of neural network: actor and critic, are used for the training. As a result, Adam optimizer can be used in this thesis in order to accelerate the training and increase the accuracy.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t \tag{2.6}$$

2.3 Batch Size

Batch size in deep reinforcement learning denotes the the number of samples that will be propagated through the network. Batch, therefore, can be treated as a package of data. Hence, the batch size is crucial in neural network training since the number of experiences back propagated to network can influence the training speed. Before optimising the batch size, another important concept should be introduced. Batch normalization is an important tool to accelerate the deep neural network training [20]. The job of batch normalization is to normalize the input data from a connection layer in neural network. The batch normalization can be treated as a hidden layer in between input layer and activation layer. The data that be normalized can be effectively activated by activation function. As per figure 2.4, if a *tanh* activation function is used, data passed to activation layer without normalization will become saturated in value either 1 or -1. This makes the data become inefficient when distributed to next layer for the calculation. The data with normalization will keep in the range of 1 to -1, thus the data is more valuable for training in next neural network layer.

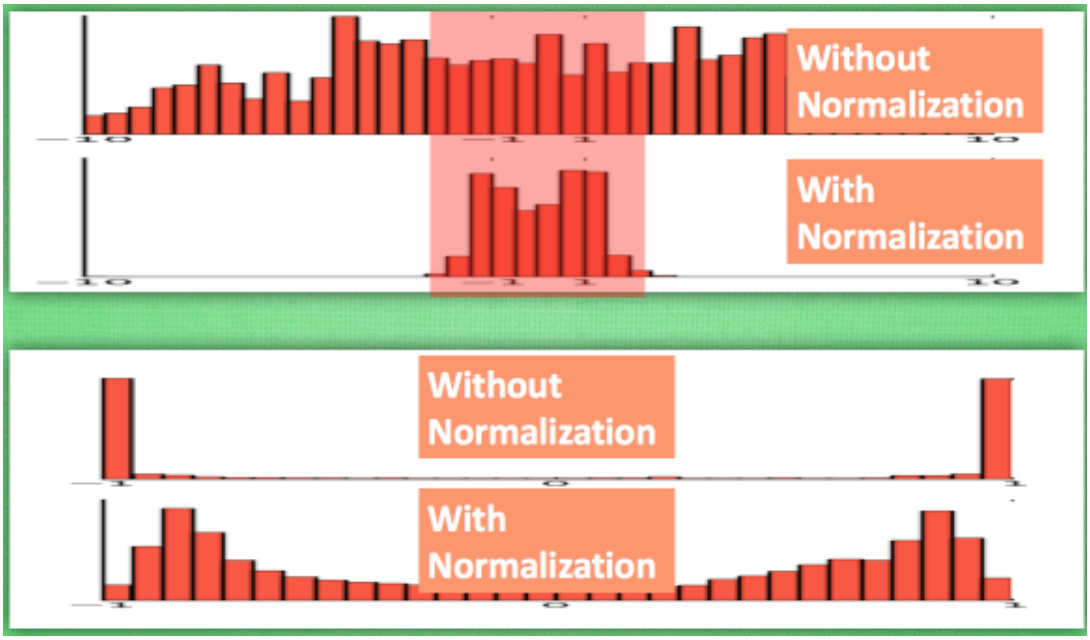

Figure 2.4: Data With and Without Normalization [2]

In this case, the batch size play an important role in batch normalization. In DDPG, the actor and critic network are updated by sampling a mini-batch uniformly from the replay buffer and this replay buffer is allowed to be big [21]. As a result, a large batch size can be used. In general, a big batch size allows to perform better normalization. However, several author points out some issues when using large batch size and demonstrate a better performance when using smaller batch size. Carlo in his report compares the accuracy of using batch size vary from 2 to 1024 with batch normalization [22]. The result demonstrates that by using SGD optimizer, the larger batch size lead to dramatic drop of training accuracy from 128 to 2048. Nitish and his team observe the similar result in different aspect [23]. Their team using Adam optimizer in the training and large batch size is applied in the experiment. The result illustrates a convergence of minimum sharp causes the poor generalization for deep learning. As a result, the generalization gap [24] occurs followed by a drop of training accuracy. These two papers indicates that, using of large batch size can lead to a drop of accuracy and poor generalization. On the other hand, the increasing of complexity in neural network also lead to unnecessary of using large batch size. GPU can process faster if small batch size is used [25].

As a result, the selecting of batch size can affect the performance of training in greater extent. These paper indicate a using of large batch size that over 128 can lead to poor generalization and drop of accuracy. Small batch size is, therefore, recommended. However, a set of batch size needed to be tested to obtain the optimized model. In this thesis, batch size of 32 and 64 will be tested. The result will be shown in the final use of batch size.

2.4 Activation Function

Activation function is an essential component inside neural network. The need for activation function is to convert linear input data and model to non-linear ouput. This helps to learn for higher order polynomial beyond one degree for deeper network [26]. The figure 2.5 demonstrates a typical neural network with single hidden layer. The input value such as action or state is passed into the neural network and some weighted are added. Then the sum of weighted input will be passed to an activation function and generate a non-linear output for the model. This output can be the predicted action or predicted Q value.

There are a few popular activation functions can be used for DDPG:

- **Linear** is a linear activation function that provides output proportionally. This allows to gives a range of activations value rather than limit the output to a limit. Linear activation function is used when connecting of different hidden layer is required. Due to the linear curve, the derivative of this function is constant. As a

result, the constant gradient can lead to divergence of the training.

$$f(x) = x * m \tag{2.7}$$

- **Sigmoid activation function** using an exponential function to obtain a smooth and positive derivative. This function can keep the output in domain of [0,1], however, due to the derivative, this function always suffer from vanish gradient problem since the derivative is too small when deeper hidden layers are added. The multiple of all the weighted gradient from each hidden layer become too small and lead to less sensitive of model learning.

$$f(x) = \frac{1}{e^{-x}} \tag{2.8}$$

- **Rectified Linear Unit (ReLU)** is firstly proposed by Nair and Hinton in 2010 [27]. This activation function has properties similar to linear activation function. In this function, negative input will be tuned to zero and positive input will be tuned to maximum. Hence, the function rectifies the values of the inputs less than zero and prevent from vanish gradient problem. The linear output in positive domain allows to accelerate the speed of training.

$$f(x) = \begin{cases} x_i, if\, x_i \geq 0 \\ 0, if\, x_i \leq 0 \end{cases} \tag{2.9}$$

- **Hyperbolic Tagent Function (Tanh)** has a smooth zero-centred function with output range between -1 and 1. Compare to Sigmoid, Tanh activation function can obtain a gradient of 1 when input is zero thereby aiding the backpropagation process. However, similar to Sigmoid, the vanish gradient problem still exist due to the non-zero output at negative input.

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.10}$$

As a result, selecting appropriate activation function in the neural network is crucial. Activation function distribute high proportion of performance in different algorithm. Activation function such as Sigmoid and Tanh should be used properly to avoid the vanish gradient problem especially in algorithm with backpropogation such as DDPG. In next chapter, the setup for the neural network will be demonstrated. Different activation functions are used at different layer of both actor and critic neural network.

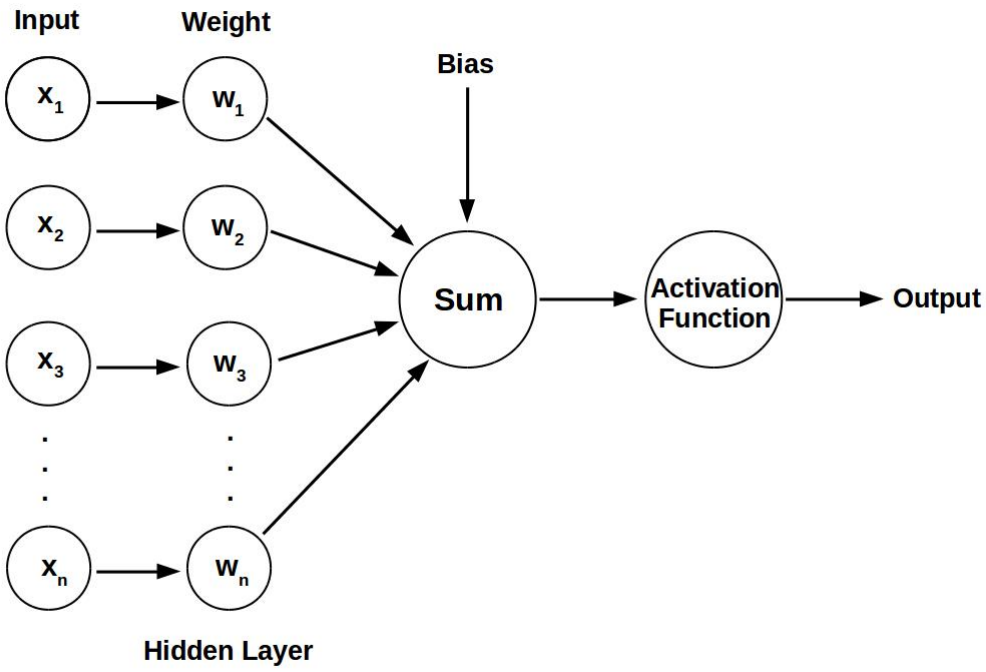

Figure 2.5: Neural Network with Activation Function

2.5 Reward Engineering

Reward engineering is one of the biggest problem to correctly guide the agent to learn correctly from the given goal. In machine learning, reward can be treated as sweat candy at the goal that attract the agent to capture it in any methods. Usually, in Q value based algorithm, reward guides the agent to provide maximum Q value with corresponding action and in policy-based algorithm, reward indicates the agent to obtain a best policy to achieve the goal. Accordingly, setting up training reward logically can improve the agent learn the given objective precisely and accurately.

Daniel Dewey points out the reward function allows the training agent "to do right thing" [28]. He demonstrates two main concepts motivate the proper design of training reward. good training agent contains two feature: generality and autonomy.

- **Generality:** High level of generality shows more ability of agent to succeed at multiple tasks in different contexts. The trained agent is not limited to a small domain-limited bundle of policies. A trained agent with more generality can demonstrates consideration of accurate policy under different environments. A more general agent can therefore discover a dominant policy with more diverse actions. This allows the agent to explore reliable source of reward. Therefore, there is a challenge of reward engineering that, the more general agent can lead to higher diversity and less predictable of action. This make the design of engineering more difficult.

- **Autonomy:** Autonomy of the agent indicates the ability to function successfully without human interaction. High level of autonomy ensure the agent to take a task without human corrective action. An autonomous agent usually design for environment that exceed the human intervention. This makes reward engineer hard to adjust the undesirable behaviour fast enough. On the other hand, autonomous agent tries to exclude the human approval reward. This forces the reward function mechanisms to become more automatic. Together, these two effects of autonomy cause the design of reward function contain less scheduled or fixed reward but more environmental reward.

These two features of training agent cause to a big challenge in designing appropriate reward function especially for UAV autonomous control. However, Stuart Russell and their team introduce a methodology of designing reward function based on the the principle of Markov decision process [29]. A reward function that without discount over the observe state can lead to unlearning of original task. An example of training UAV fly in four waypoint circle can be used to demonstrate the problem. With non-discounted reward function, there is no punishment if the drone is flying away from the trajectory. Because the rewards are acquired once the drone reach the waypoint, only positive reward will obtain while training. This lead to a problem that the drone may only fly to reach the waypoint rather than fly along the trajectory. Thus, the drone is "distracted" from the reward locate at the desire waypoint and the drone is learnt the decision or policy independent from time. Stuart and their team point out a design theorem by using potential-based shape function (equation 2.11).

$$F(s, a, s\prime) = \gamma\Phi(s') - \Phi(s) \tag{2.11}$$

Where γ is the discount factor and Φ is some functions over the state. A reward function that is potential-based satisfy the necessary and sufficiency condition. This indicate that, if the reward function does not achieve the criteria of necessity and sufficiency, then only local optimal policy is achieved for either current decision process or future decision process, the optimal policy is therefore independent from each decision process. Oppositely, if the reward function satisfies the two criteria, the function is therefore potential-based and every optimal policy for current decision process is also the optimal policy for next decision process.

Accordingly, due to the task of UAV autonomous control, some extents of autonomy and generality should be achieved. As a result, by using the reward engineering design method proposed in previous. A semi-autonomic reward function can be found based on the observation state. Some implement rewards can be introduced to supplement the learning guidance. The detailed design will be demonstrated in the following chapter.

2.6 Network Connection

For most of the private router, the local DNS is set to 192.168.x.x. This lead to an issue that both drone and host computer occurs DNS conflict because of the same range of default DNS starts with 192.168.x.x. Even though the last two numbers can be varied, the conflict still happen and make the connection fail. Unlike other commercial drone such as DJI Tello, the default router IP is configured not in the range of 192.168.x.x, in Parrot AR Drone, the default router cannot not be changed permanently.

According to the Parrot AR Drone developer guide [30], the Parrot AR Drone is accessible by using appropriate SDK tool. Unfortunately, the tool kit is not provided, therefore, the direct reconfigure of the WIFI is not workable in this case. Petrovicz Benedek giving an idea of forcing Parrot AR Drone to match a network IP address in an unencrypted network [31]. The Parrot AR Drone system is insecure due to the open source. This provide a back door to hack the drone system in wireless. Other device can shut down the drone easily by simply connect to the drone WIFI then ping a command to shut down the drone. Accordingly, this allows the drone WIFI to be reconfigured to any WIFI signal. However, the drone may shows unstable in the connection because the drone WIFI is forced to connect to any IP address offered by the router. If the drone WIFI is distributed to a host IP address, the connection will fail once the host device dominate the IP address.

Therefore, Cristinel Ababei suggests another solution using wireless bridge connection [32]. According to the scenario, a Wifi router is used as a internet connection center. Computer and a WII controller are connected to the router in Ethernet. The router then connect to the drone by using bridge connection. The IP address for the router is reconfigured to match the same IP address to the UAV. However, the connection method leads to a problem that the router no longer receive other internet signal once the router is used for bridge connection. Therefore, this method may not be suitable for the use of multiple network connection, but still provide some inspirations of changing the DNS of router to achieve the connection.

Chapter 3

Experiment

In this chapter, the methodology of implementing machine learning algorithm in UAV autonomous control is established. In the first section, the experiment setup is demonstrated. Both simulation and real drone test setup are established in order to provide a stable platform for training and testing. In the simulation setup, the overall platform will be visualized. The visualisation of the simulation is based on Gazebo, and the communication platform ROS is used to receive and send any data from all the ports that relate to the simulation. The real drone testing environment is also demonstrated. The structure of the real drone test is similar to the simulation. The only variation is the component that to receive drone location and orientation. OptiTrack is used to provide satisfy location and orientation capture and recording when testing for the real drone. The training task is introduced from an easy trajectory to a more complicated trajectory. The simple trajectory can be used to demonstrate the functionality of the algorithm. The increasing of trajectory complexity allows to show the robustness of the algorithm. In the second section, the neural network policy for DDPG algorithm will be demonstrated. The selection of mini-batch size, activation function are discussed according to the method reviewed in literature review chapter. The reward functions are designed based on the training objective. Several considerations to designing the reward function are listed. Parrot AR Drone 2.0 will be used in both simulation and real drone testing. Same model applies to minimize the uncertainty of the drone dynamic variation. If different UAV models apply in the separate environment, the implementation of machine learning becomes difficult.

3.1 Environment

3.1.1 Simulation Setup

The simulation allows to provide a training environment before apply to the real test. Therefore, setting up a simulation is essential. After the drone is trained, some tests can be done on the simulation environment. This can largely avoid the potential risk from directly apply the machine learning to the real drone. These potential risks including control failure such as flight failure, drone crashing can cause serious damage to the real drone. Hence, the use of simulation environment allows to provide robustness of the testing. Moreover, drone durance is needed to be considered when doing the testing. In this case, simulation environment allows to train the drone without using the real drone.

The trained model can therefore be applied to real drone testing after training from the environment.

Gazebo is a physics engine that provide simple but powerful capability of graphical rendering [33]. In this physics engine, 3D environment can be generated. This is essential for robotic control especially for UAV control since three dimensional space are required to demonstrate fully the interaction between drone and the environment. This physics engine provides flexible and stable simulation for the training. In the Gazebo world, Parrot AR Drone model is imported and default world with no landscape is used to simulate the indoor environment with flat floor and limited space.

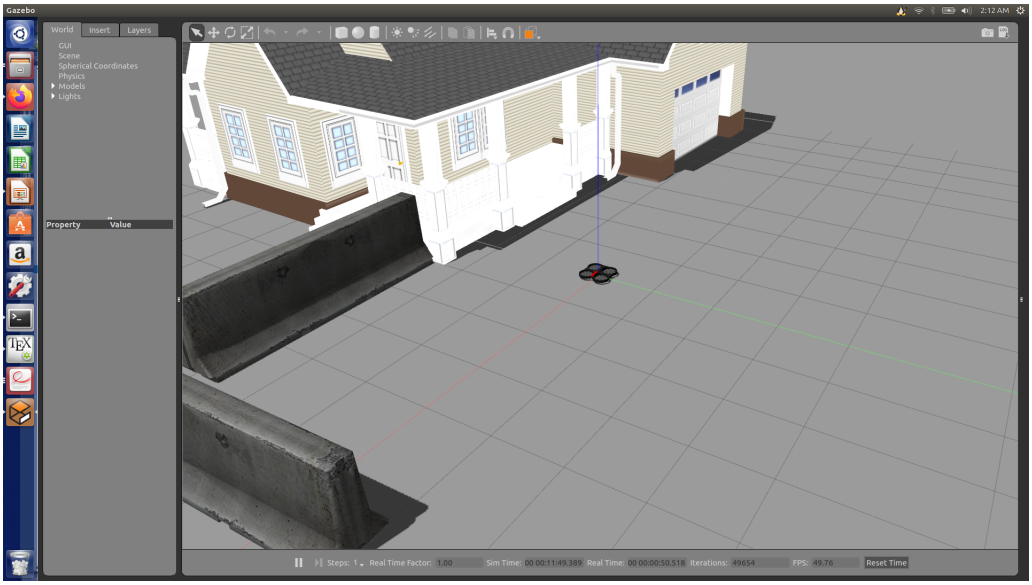

Figure 3.1: Gazebo World and Parrot AR Drone 2.0

Robotic Operating System (ROS) is an open source structure that design for robotic control. Unlike other robotics control software, ROS is a distributed framework of process that allows to design each executable file individually and couple at runtime [34]. This can largely increase the flexibility of setting up the simulation. The combination application of machine learning, simulation and real drone control can lead to a fairly complicated system. As a result, ROS provides a robust platform that allows different packages to collaborate fluently by building accessible node for each executable files. Moreover, ROS is supported for multiple language including Python, C++. Therefore, it is easy to implement different programming language in a single testing.

Figure 3.2 shows the simulation structure and demonstrate how the components cooperate together. In the simulation, ROS is the communication platform that publish and subscribe the information from each node. Drone controller allows to access to ROS by registering a node to the platform. The drone controller then can send command and receive position and orientation information from ROS. The agent is the machine learning agent that proceeding the training of the simulation by passing the state, reward and

the criteria of breaking training loop. Then the agent will estimate the action based on the values. $tum_simulator$ [35] is an open source package that generate the simulation model of Parrot AR Drone and the simulation world. The AR drone in the simulation world therefore doing action based on the command sent by drone controller, and Gazebo visualise the entire movement. 500 episodes training is set in the purpose of shortening the simulation time meanwhile giving a good result of showing a successful training.

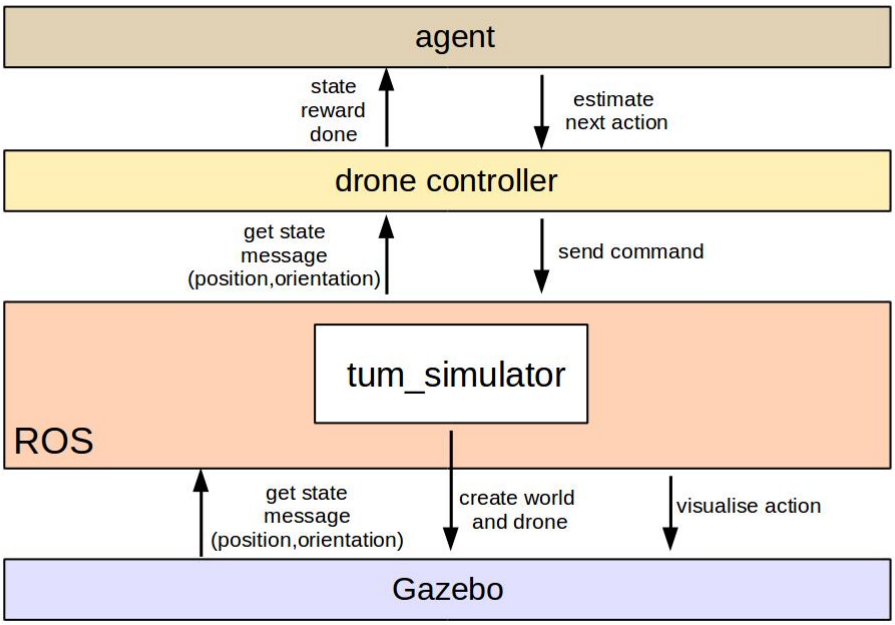

Figure 3.2: Simulation Setup Structure

3.1.2 Real Test Setup

After the training for the tasks are accomplished, the trained models are saved and can immediately replay by simply restore the model file. Note that same drone (Parrot AR Drone 2.0) is used for both simulation and real drone testing. The table 3.1 in the following shows the basic physical parameters for the drone. As per the table, an inboard WIFI router allows wireless connection between the drone and other devices. Accordingly, offboard control can be accomplished. However, the low endurance limits the drone be trained at real time. As a result, pre-train of the model should be achieved and transferred the trained model as well as trained policy to real drone testing.

Table 3.1: Parrot AR Drone 2.0 Basic Configurations

Parameter	**Value**
mass	0.42 kg
battery	Lithium-polymer 3-cell (11,1 CV), 1500 mAh
Interface	USB and WIFI 802.11n
Dimension	52cm x 52cm
Endurance	approx 12 minutes with speed of 5 m/s

To setup a stable real drone testing environment, reliable devices should be selected to ensure the high efficiency of the training since offboard control without PID controller is used. If high latency environment is used, the reliability for the testing will drop and the transmission of control action by the reinforcement learning agent becomes inaccurate. The following table 3.2 shows the core hardware used for the real drone testing. The test environment is setup similarly according to a OptiTrack indoor control implementation paper[36]. Latency exists when using WIFI connection and LAN connection. In general, LAN connection will generate 1 ms of latency and OptiTrack camera capture has latency of 5 ms. The transmission latency of WIFI signal from WIFI router to host computer is around 1 ms. By using ICMP ping, the latency between client computer and the UAV can be tested. The result of average latency in 20 times is around 4.23 ms. These delay of signal transmission are satisfactory at this level of control since only high level command is sent to control the drone.

Table 3.2: Testing Environment Setup Devices

Component	**Part**	**Detail**
Camera	Optitrack 41W	4.1MP Horizontal FOV 51°
	Optitrack 17W	1.7MP Horizontal FOV 70°
Host	PC Windows OS	CPU: Intel CoreI7-6700K NIC: Intel E1G42ETBLK Dual Port Adapter
Client	Alienware m17x R4 Linux OS	CPU: Intel coreI7-3740QM GPU: Geforce GTX 1060m
Network	WI-FI Router	TY-LINK ARCHER AR2600
Drone	Parrot AR Drone 2.0	Specification at table 3.1

Parrot AR Drone contains an inboard WIFI router that allows mobile phone, computer or other devices to control the UAV. However, most of the laptop can only acceptable for single WIFI connection, thus combined connection of LAN with WIFI is required for the testing. As per the method suggest in section 2.6, changing router LAN IP address to a new range (192.168.0.0 to 192.168.0.255 or 172.16.0.0 to 172.31.255.255, or 10.0.0.0 to 10.255.255.255) and plunging in Ethernet port with laptop then connecting the drone

WIFI with laptop. To be mentioned, using of any LAN IP address start with 192.168.1.x cannot be connected to the Parrot AR Drone due to the similar LAN route. Finally, the communication between physical position capture system, drone and ground station (laptop or client PC) are set up. The figure 3.3 shows the overview of the communication environment.

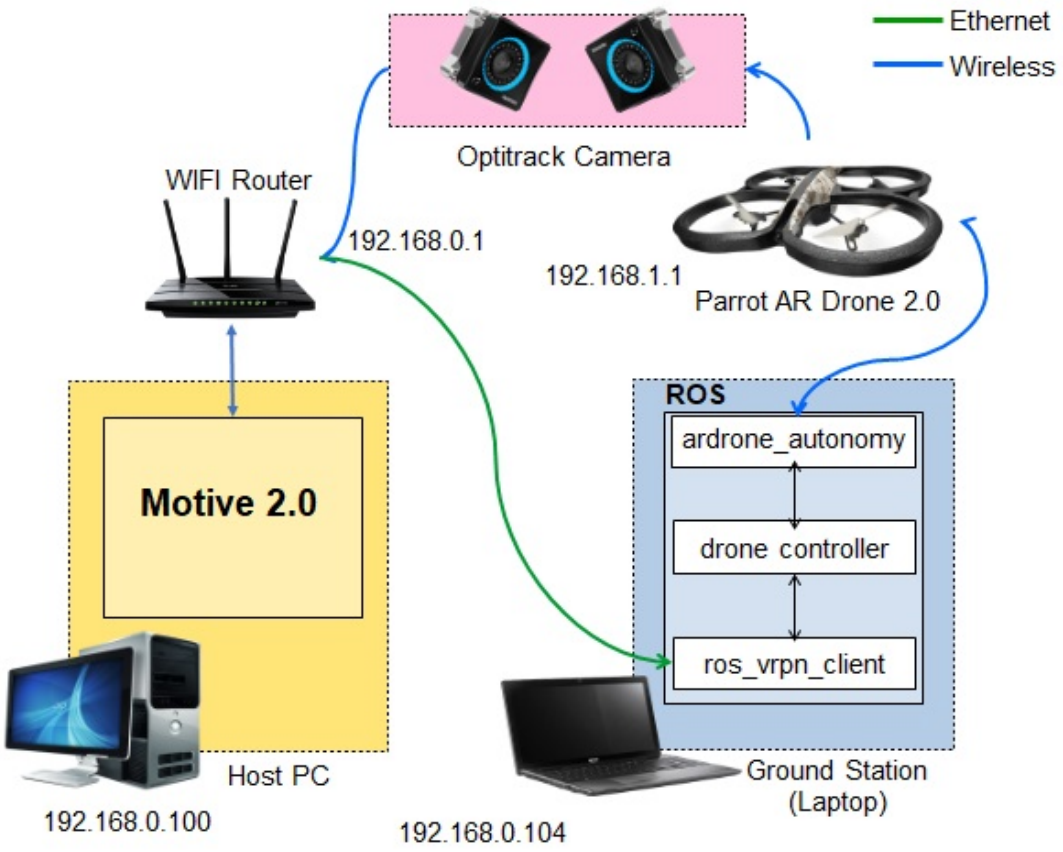

Figure 3.3: Communication Setup

As per the figure, the ground station as well as the client computer connects between the host computer. $ardrone_autonomy$ is an opened source [37] package for Parrot AR Drone 1.0 and 2.0. This package acts like a drone driver that helps to fetch the IP address from the UAV and subscribe to ROS as a node. This allows command sending from the self-built controller to the UAV. Another opened source package ros_vrpn_client helps to match the IP address with host PC. This package ensures the data transmission from host PC to client PC. The position and orientation information can therefore transmit to client PC from host PC.

In real drone testing, the coordination and orientation of Earth frame and body frame should be aligned. In simulation, figure 3.1 shows the coordination in Earth frame is aligned with the drone body frame, which the red line is x-axis, green line is y-axis and blue line is z-axis. The motion in simulation does not require a transformation. However, the coordination and orientation in real test are different based on the setup. The figure

3.4 shows the x-y plane in body frame is 90 degree anti-clockwise to Earth frame. The reason of allocate the drone to face in y direction is that, y-axis represents the length of OptiTrack lab room whereas x-axis represent the width of the lab room. As a result, when coordination and orientation are recorded, some small translation required to align the attitude system.

3.1.3 Drone Controller

In offboard control, UAV does not require any RC transmitter. UAV flight control can be operated by setting up corresponding command line. In this case, stability is essential for the controller to make sure each command sent to the drone is valid and satisfied. Control error comes to be vital once human interaction with the drone disappear. It is essential that to design a controller specify for Parrot AR Drone because the dynamic for each commercial UAV is different.

According to the developer guide[30], the AR drone is a quadrotor equipped with four brush-less current motor with gear box and propeller. As per figure 3.4 Two pairs of motor (1,3) and (2,4) rotate in opposite direction. motor pair (1,3) spin in counter-clockwise direction and motor pair (2,4) spin in clockwise direction. The forward motion is achieved by pitching in X direction in body axis. Front motor reduces the velocity and rear motors increase the velocity. Similarly, going left or right motion is accomplished by rolling. Corresponding motors velocity are changed based on the required motion.

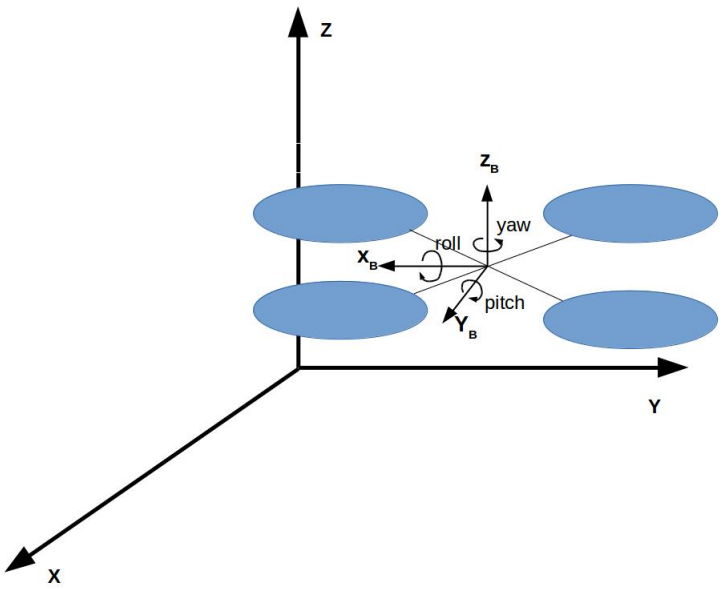

Figure 3.4: Parrot AR Drone Motor Motion in Real Frame

The dynamics of the drone can be described in twelve degree of freedom including: three absolute position $[x, y, z]$ of center of mass of the quadrotor , three euler angles $[\phi, \theta, \psi]$ to describe the orientation of quadrotor , three linear velocity $[v_x, v_y, v_z]$ to describe the

rate of change of position, three angular velocity $[\dot{\phi}, \dot{\theta}, \dot{\psi}]$ to describe the rate of change of orientation. Due to the encapsulation of Parrot AR Drone, even though the motor speeds and other low level information are established by the AR drone SDK [30], the control for these values are invalid. Therefore, according to SDK, only high level control for the drone can be accomplished and the motion is controlled in the way of command lines. Six degree of freedom is used to achieve the control including: three linear velocity corresponding to linear motion and three angular velocity corresponding to angular motion.

The drone controller is consisted of three main components: data receiving, data processing and command sending. The figure 3.5 visualizes the structure of entire controller which is developed based on an opened source code [38]. Data is subscribed from either OptiTrack or Gazebo environment through ROS. These data contains the position and orientation if in the real world scenario. In Gazebo, drone velocity is passed to the controller for checking the control accuracy. According to the flow chart, when the controller is used, all navigation data will be subscribed by subscriber from Gazebo or OptiTrack through ROS. At the same time, a publisher is generated for sending command to ROS for further control. The reset part of the controller allows to initialize the state for the drone. Initialize position is generated based on the training task. The drone is sent to takeoff and the states are observed as initial observation.

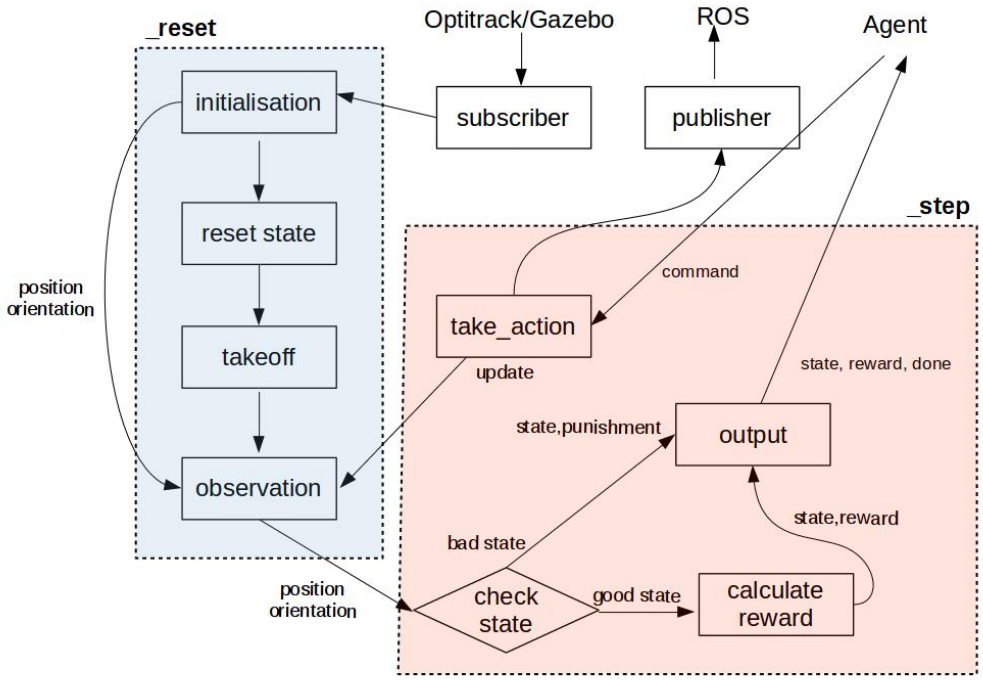

Figure 3.5: Drone Controller Structure

When the drone requires control or training, drone action such as linear motion (moving forward, backward, left, right,etc.) is sent through ROS. In training or result testing,

state of the drone is checked after every steps the drone made. If the altitude or location is out of bound or the orientation of drone is too bad, punishment will be given and training will stop. If the state is within the limit, reward will be calculated based on the distance to target position. States, reward and whether the simulation is finished are returned to machine learning agent for further learning. Manual control is acceptable in this controller. Basic commands such as takeoff, land and set motions can be achieved by simply call the corresponding function.

3.1.4 Training Tasks

The selection of training objective should be considered based on the potential of algorithm. The complexity of training objective should be evaluated before the training start. Even though DDPG can be used for continuous control, the performance for DDPG is also limited if complex trajectory, such as four way point circling, is trained. To test the performance of DDPG in autonomous control, lower level of trajectory can be used for the early training. In this thesis, a simple trajectory will be trained followed by a training of a more advanced flight task.

Firstly, a single takeoff and hover task is set to test the usability of each algorithm. In this task, the drone will be sent to takeoff from origin point (0,0,0) then hover at altitude of 1.5 m above the ground (coordination of (0,0,1.5 m)). In this scenario, the drone can "touch" the target position easily which allows the training agent receive positive reward easily. After the simple trajectory test for the algorithms, a more advanced trajectory can be introduced to check the performance of the algorithms. Single way point tracking with random initial takeoff position can be used for the advanced test. This ensure some level of difficulties meanwhile ensure the possibility of success in the training By just varying the location of the drone.

Based on the real drone testing, indoor flight control will be processed. In this case, to make sure the training scenario is as real as possible, the drone is placed randomly inside a $3m \times 3m$ region on the ground. Full trajectory will be simulated including initialize drone position, takeoff, approach to target position and hover. The target position is located at 1.5 meter above the origin. This origin can be defined at any position. To simplify the training, the origin in the simulation is located at (0,0,0) with respect to the Gazebo environment. Thus the target position is located at (0,0,1.5) as shown in figure 3.6.

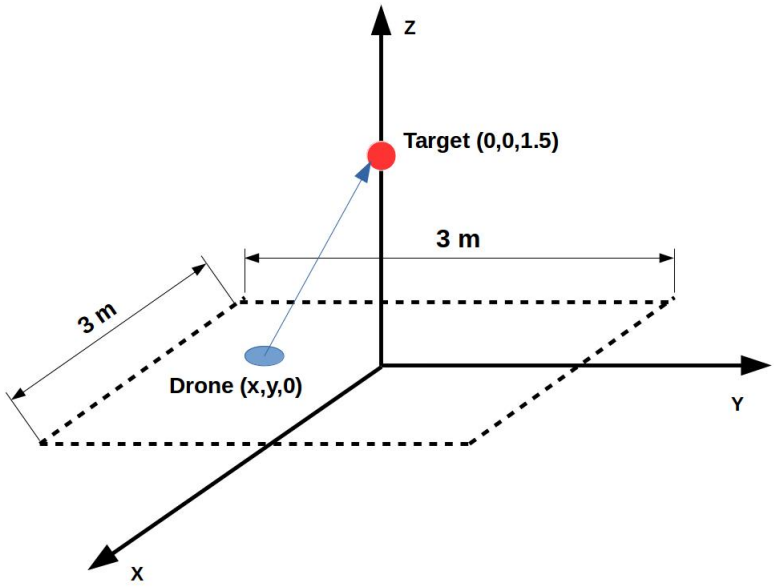

Figure 3.6: Random Initial Position to Target Position

After the drone is trained, two robustness tests will be executed. At first robustness test, once the drone arrive the target position, a random action will generate to simulate the push effect by human in real life. This allows to check whether the drone will return to target position. Another robustness has scenario that, after the drone is trained, vary the target position and test whether the drone can follow to the new desire location. These two robustness tests allow to show the general the drone is trained to be.

3.2 Learning Agent Setup

3.2.1 Observation/State

In order to provide a good guide line for the agent to train the UAV properly, correct setting for the observation is essential. In the training, observation allows to provide information about the state of drone in the environment after do some actions. According to the training task, the drone should approach the desire location and stabilize that location. The distance between the drone and the desire location becomes important in the training, hence, the linear distance from drone to goal position can be used as one of the observation. This also lead to a more diverse observation. the position (x,y,z) of drone can be set as another observation. To train the drone hover at the goal location, a criteria of whether the drone arrive the location can be used as one of the observation. These five states allow to give satisfactory training since the training task does not contain any requirement of time consumption or battery consumption and high level control is used.

3.2.2 Reward Function

The reward setup of the training task can be divided into three sections: A limitation for the drone not fly beyond the physical boundary or dynamics, a punishment about the distance from drone to target location within the limited boundary and a reward for staying at the desire location.

Boundary and dynamics limitation: The idea of the limitation is to restrict the action taken by the drone to stay at a reasonable range. For instance, in indoor environment, there is physical limit such as the room size that the drone cannot fly across. In real test, this may lead to a crash on wall or drop to the floor. As a result, a punishment for the drone to cross the boundary can limit the drone to fly within a physical area in real-life. Furthermore, unreasonable dynamics also needs to be restricted. In the action space, only linear actions are used for the flight, therefore, the roll and pitch for the drone should be limited to 0.7 rad/s. Accordingly, these two criteria can be formulate as a fixed punishment as shown in the following if the drone cross the limit.

$$r = -5 \begin{cases} if\, x, y > \pm 3m \\ if\, z < 0.3m \ or \ z > 3m \\ if \phi, \theta > 0.7 rad/s \end{cases} \tag{3.1}$$

Punishment for away from goal position: According to previous reward, the drone can be kept inside a space while training or testing. Another punishment can therefore be introduced to guide the drone fly close to the target position. The reward is designed based on the method review in section 3.2.2. To main the criteria of necessity and sufficiency, the linear distance from the drone to desire location can be used as the reward. The basic principle is to obtain more punishment if the drone fly away from the target point. The equation **??** shows the designed reward function as a punishment since a negative reward help to reduce the distraction from the reward. A discounted factor of 0.03 is used to prevent the issue of unable to learn due to the accumulated negative reward at each episode.

$$r = -0.03 * \sqrt{(x_i - x_g)^2 + (y_i - y_g)^2 + (z_i - z_g)^2} \tag{3.2}$$

Which x_i, y_i, z_i is the current location and x_g, y_g, z_g is the desire location.

Reward for staying at the desire location: Finally, once the drone reach the desire location, it is important to tell the drone stay at the location. This can be easily set up by using a fixed reward. In this case, for each step the drone stay at the location, a reward is received. A reward of 1 can be used since the reward cannot be too large so that the drone only focus on touching the location and also cannot be too low so that the

drone unable to learn properly. On the other hand, this reward can be obtained within a range of 0.3 meters. This setting allows to loose the training objective so that make the drone easier to be trained.

$$r = 1 \tag{3.3}$$

The design of the reward functions for the training task contain both schedule reward and automatic reward. The combination of designing the reward allow to provide a better guidance for training.

3.2.3 Neural Network Setup

The neural network for the DDPG training can be setup based on the methodology suggest in chapter 2. The figure 3.7 demonstrates the neural network structure for actor and critic network. In actor network, three hidden layers are used to improve the performance of training and ReLU activation function is used for each of this layer to ensure a fast training without gradient vanish. A tanh activation function is used at the output layer for constraint the estimate action. In the critic layer, two separate layers are used to train the model in observation space and action space. The observations in first input layer are passed through two hidden layer with a ReLU and linear activation function. Actions in second layer are passed through a hidden layer with linear activation function. Linear are used for combining the two separate neural network so that to avoid the unexpected value loss. The combine layer therefore goes through a hidden layer with ReLU and eventually output by a linear activation function.

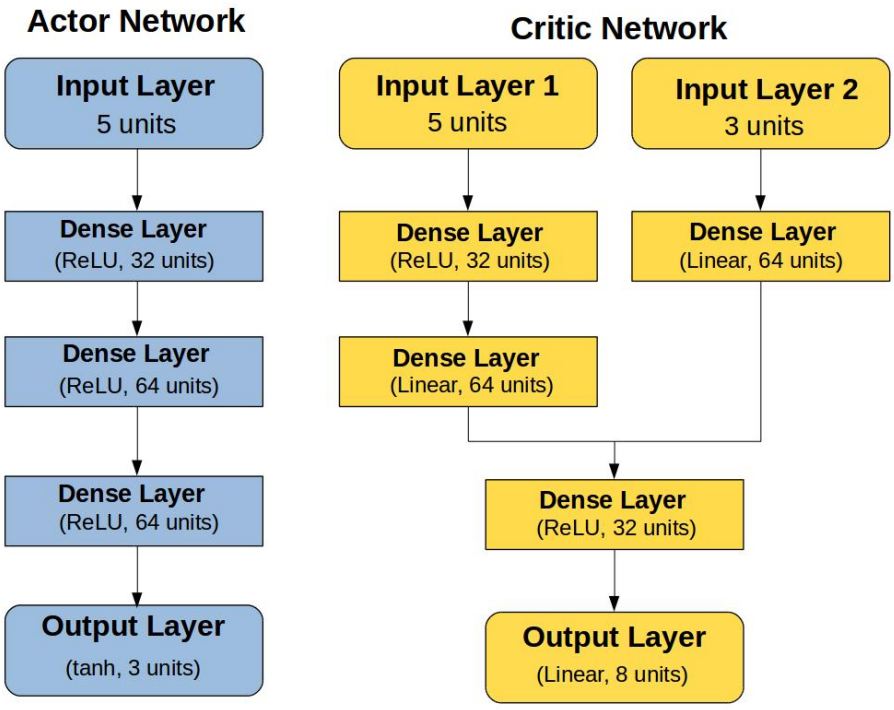

Figure 3.7: Nerual Network Strcuture

Chapter 4

Result

In this chapter, the simulation results are discussed. In particular, a simulation comparison of simple takeoff-hover task between DDPG and DQN algorithm are presented with reward graph. The performance of DDPG compare to DQN is discussed. Subsequently, simulation result of higher complexity trajectory will be presented. Two robustness tests result for the algorithm are illustrated to show the level of autonomous of trained drone. In addition, a comparison of the fly control without implementing machine learning between simulation and real drone testis is demonstrated. The aim of the comparison is to investigate the difference between simulation and the real drone so that to acquire better understanding of control error. This error could eventually affect a failing of implementing the trained model in real approach. Therefore, the investigation show high necessity in this thesis.

4.1 Simulation Result

4.1.1 Fixed Takeoff and Hover Task

According to the training schedule, the simulation is firstly trained for a simple trajectory of takeoff from origin and hover at altitude 1.5 meter above origin. The following figure 4.1 shows the reward of the simulation with comparison of DDPG and DQN. As per the figure, DDPG has high reward over the training episode (red line). At first 200 episode, the drone is experiencing a learning phase. The drone will randomly choose an action to explore the environment meanwhile obtaining the experience for future study. After around 200 episode, the drone stop exploring the environment, instead, the drone exploits the memory and apply to the action. Therefore, the reward graph start to converge even though is not stable after 275 episodes. However, for the DQN, the reward does not converge to expected value. A flat negative reward obtain indicates that the drone does not learn to reach the target position. Even though there are some episodes show a high value of positive reward, drone cannot exploit those experiences and apply to the policy. In this case, the DDPG has shown a better performance even in the simple training task. The discretization of action space may be the reason causes DQN unable to train the model correctly.

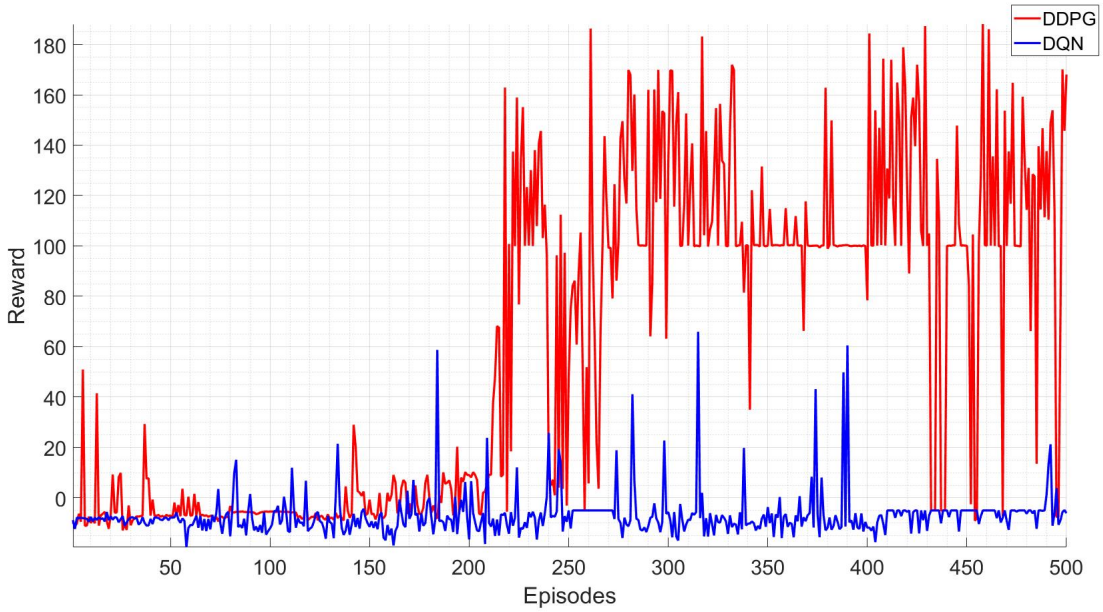

Figure 4.1: Reward for DDPG and DQN in Fixed Hover Task

4.1.2 Random Location Takeoff and Approach Target Location Task

The previous section demonstrates DDPG has a better performance when dealing with continuous control. A further training is therefore introduced to investigate the maximum performance of this algorithm in more complex drone control. As mentioned in section 3.6, an advanced trajectory is used for the training. The figure 4.2 in the following shows the reward the drone obtain in 500 episodes. In this scenario, the drone start to converge at around 275 episode. The drone show a surprisingly fast converge time compare to simple training task. As mentioned previous, the reward start to converge at 200 episode but a fluctuation occurs at around 250 episode. This indicate that the drone does not learn stable enough at that scenario. However, the training of more complex scenario reveal a more stable result. The fluctuation of the reward is mainly occur beyond 100 rewards. Although there are a few episodes show different extents of low reward, the training can still be determined as successful. Additionally, the damping of reward graph is caused by the waving of the drone during flying. The UAV flies in and out of the target position, thus the counting of step staying at target position range is refreshing. As a result, more total steps are used to stay at the target region and lead to higher reward obtain. This can be treated as a reward design flaw that the drone may try to maximize the reward. In this case, the drone may learn a policy that to touch and leave the target position so that to obtain higher reward.

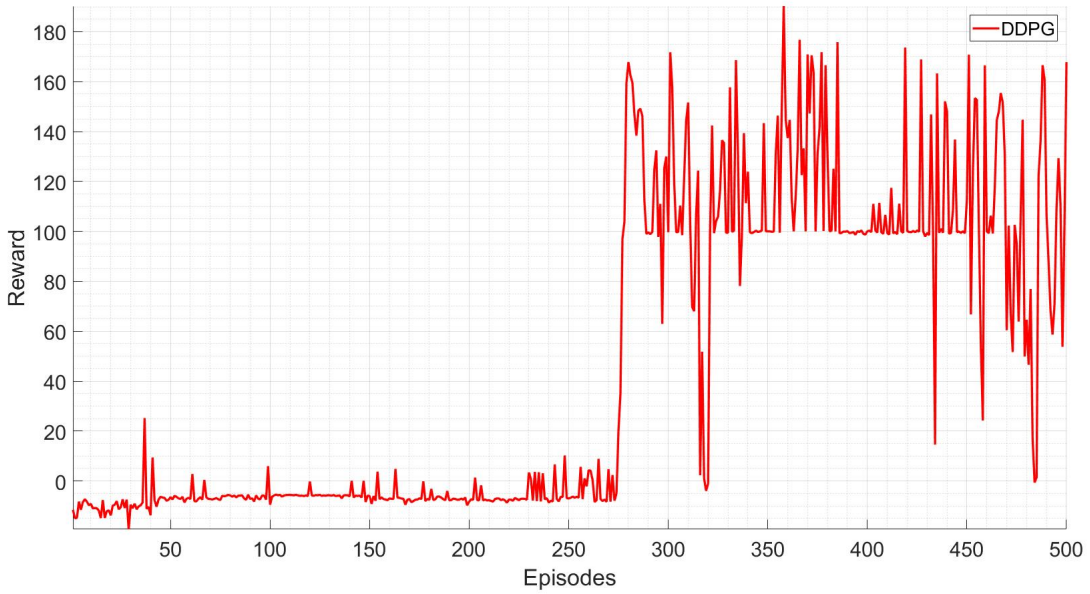

Figure 4.2: Reward for more advanced trajectory

The figure 4.3 visualizes how the drone learns to reach the given goal location. Since random location is generated while training, the start position for each episode is varied. At first 50 episode, the drone still exploring the environment, therefore, the drone quickly crash to ground due to random action. Later at 250 episode, the training effect start to reveal. Even though the drone is still unable to reach the desire location, the trajectory demonstrates a trend of approaching the desire location. After 350 episodes, the drone is well trained. In this case, the trajectory shows a fast arrive of desire location. Due to the setting space around the desire location as mentioned in section 3.2.2, there is a gap between the drone converge point to the desire location. This accuracy problem can be adjusted by redesign a more precise and automatic reward function in the case when the drone reach nearby the target position.

The training for random takeeoff location trajectory illustrate a sufficient result. The reward graphs demonstrate a convergence at around 275 episode which also imply less training episode needed (500 episodes training requires around four and half hours). The reward and trajectory graph both show a stable converge in great extent after 275 episode. However, the exist of location error means there are potential to improve the reward function so that to obtain more accurate result. In the next section, a robustness test for the trained drone will be demonstrates to show the extent of generality of the agent.

The results for these two training tasks can be compared to a similar trajectory approach experiment by Rohit Murthy [39]. The author simulating a simple trajectory control with battery constraint by using DDPG and Parrot AR Drone. In his result, even though a video of the trained drone is shown, the reward function does not converge according to the reward setting whereas this thesis demonstrates a better result of convergence which

indicates a better performance compare to the author's result. However, some limitations after convergence is shown in this thesis. The problems are discussed in ETH paper [6], the unstable effect exist in using of DDPG whereas the problems are less identical in using PPO and TRPO.

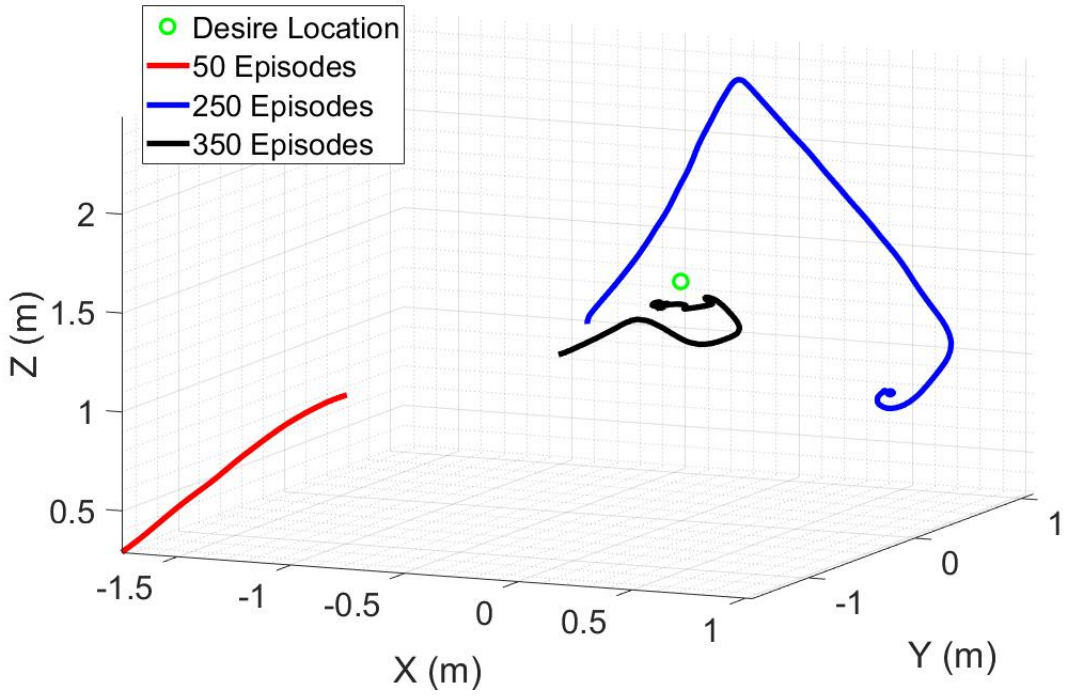

Figure 4.3: Trajectory for Random Takeoff Task at episode 50, 250 and 350

4.1.3 Robustness Test for Trained Drone

As mentioned in section 3.6, the first robustness test has task that, once the drone reaches the desire location, push away the drone and observe will the drone return to the desire location. In real world, this can be easily achieve by push away the drone manually. In the simulation, this action is simulated by adding an action to the drone at a period of time after the drone arrive the desire location. The figure 4.4 shows the trajectory of the entire robustness test. The trained drone shows an ability to return to desire point once be pushed away from the target location. Figure 4.5 and 4.6 show a closer look at the position change of drone and the corresponding flying command that the training agent estimate for the drone.

As per figure 4.4, the drone quickly reaches to desire location at around 30 steps. To avoid a high estimated velocity, the command is clipped to 0.25 m/s as shown in figure 4.6. Some damping effect occur on the position graph show the drone is "swinging" around a fixed point. This damping is caused by a step velocity publish to the drone. Since no observation and reward about velocity is added, the drone seems to be unable to keep zero velocity at desire location. Instead, the drone needs to move around to maintain the goal position. Nevertheless, the drone velocity start to reduce when approach the desire

location. This implies that the drone realizes a reduce of v_z allows it to stay at the desire region. Subsequently, the push away action added at the 90 step and release at 130 step. A drop of altitude followed by a position adjustment as shown in the figures. Since no turning action is added, the drone tries to add negative velocity to recover the position and finally return to desire location. The recover time takes about 52 steps which is around 5.2 seconds (each action step is 0.1 s).

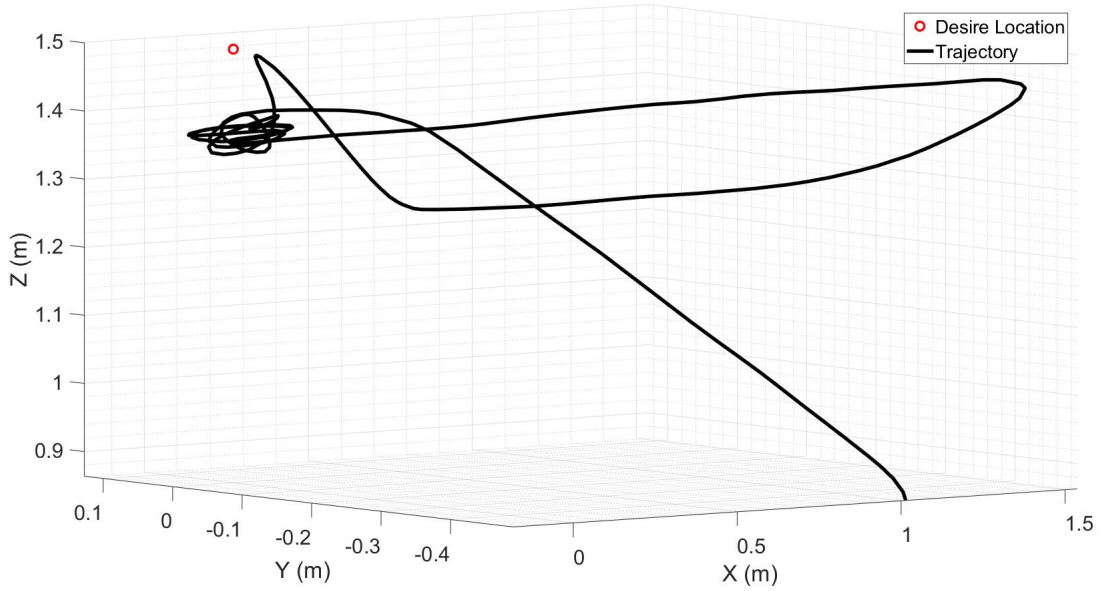

Figure 4.4: Trajectory of 'Push Away' Robustness Test

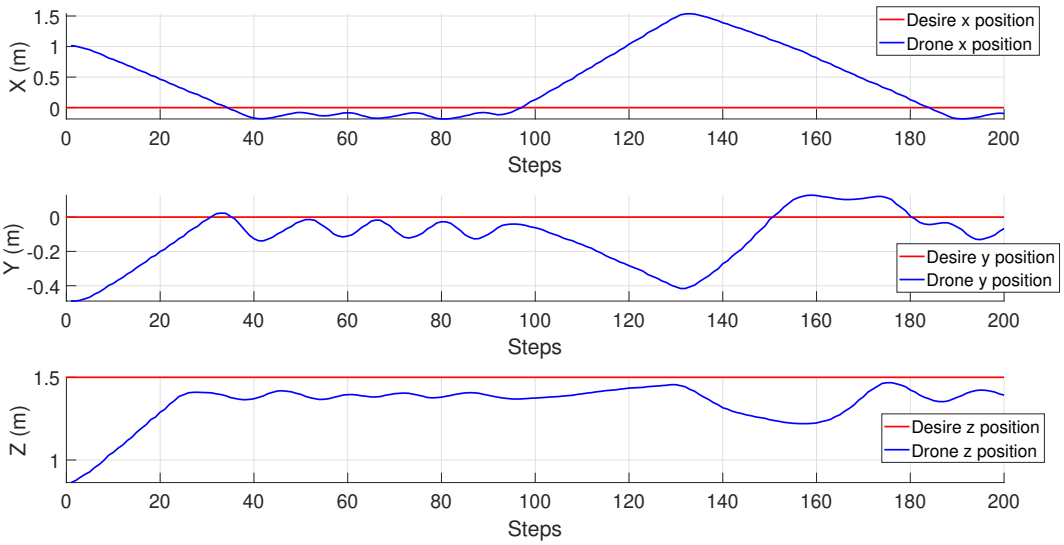

Figure 4.5: Position Variation of Drone at Robustness Test

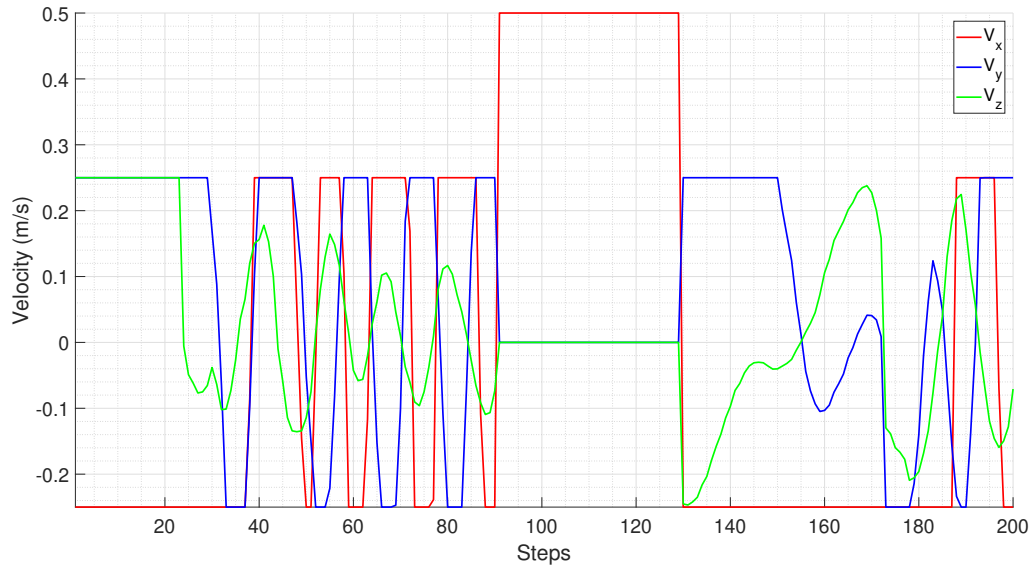

Figure 4.6: Actions Estimated at Robustness Test

The second robustness test has scenario of varying the desire location. By setting a desire location of (1,1,1.5), the testing result is shown in the following. As per figure 4.7, a vary of desire location does not affect the drone's decision. An approach of original destination is shown and this indicate that the drone can achieve some extent of autonomous. However, for high standard generality, the drone may require better architecture of algorithm or better reward function to achieve higher level of autonomous based on a less relevant training task.

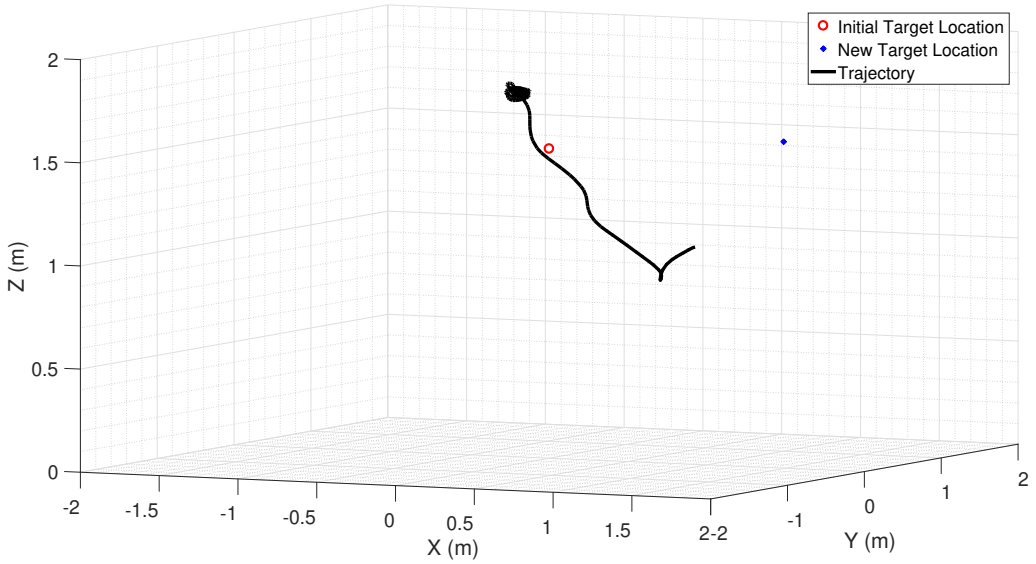

Figure 4.7: Trajectory of Vary Target Location Robustness Test

The position error still affecting the result at both robustness test. According to the robustness tests, the drone is able to perform a recover procedure after arrive the desire location. However, the drone is unable to perform a vary target chasing. As a result, the

trained drone shows some standard of autonomy and generality, yet the trained drone is not suitable for higher level mission or more complex trajectory. This implies that, to design for higher level of autonomous drone, more complex algorithm or better reward function is needed.

4.2 Comparison of Simulation to Real Drone Flight

In general, the trained model can be restored and applied to any platform. Machine learning package such as Keras or Tensorflow allows the saving and recalling function. However, transferring trained model from simulation to real drone testing requires some benchmarking. In particular, the drone control between simulation and physical fly is different. Even though the dynamic of the simulated Parrot AR Drone is built based on the real drone SDK, there are still some errors exist that need to be adjusted before applying the reinforcement learning model.

In this section, two flight control are demonstrated to investigate the control error between simulation and real drone control. The first flight control is simple takeoff and hover control. The drone will takeoff and hover for 10 seconds. The real drone orientation and position are recorded. Due to the different, setup of coordinate system, the simulation coordinate is varied from the testing environment. By comparing the recorded data to simulation data, the x-axis in Gazebo corresponding to y-axis in real world and y-axis in Gazebo is corresponding to x-axis in real world. The orientation therefore needed to be trained according to the axis variation. The coordinates in the result below will be shown in Gazebo reference frame.

The figure 4.8 shows the trajectory when the drone takeoff and hovering. As per the figure, both simulation and real flight control show different level of side slipping when hovering for 10 seconds. However, in simulation, hovering is more stable compare to the real flight which showing some damping in the altitude. Figure 4.9 and 4.10 demonstrate more detailed comparison between the simulation and the real drone control. In figure 4.9, the x position and y position show different extents of movement when hovering for both simulation and real drone control. The altitude graph shows a different height after takeoff. In simulation, the drone has a takeoff altitude of 1 m with no variation whereas the real drone has lower altitude of around 0.8 m with variation. Theoretically, there should less difference in the takeoff altitude since the command to send takeoff is default for Parrot AR Drone 2.0. A possible reason that lead to this problem is because of the battery weight. Since in the simulation, the battery weight is ignored whereas the real drone has a 11.1V battery with 107g. Since the lift generate to takeoff is default for the drone, the adding weight may lead to less counter force due to the weight and therefore the altitude drop.

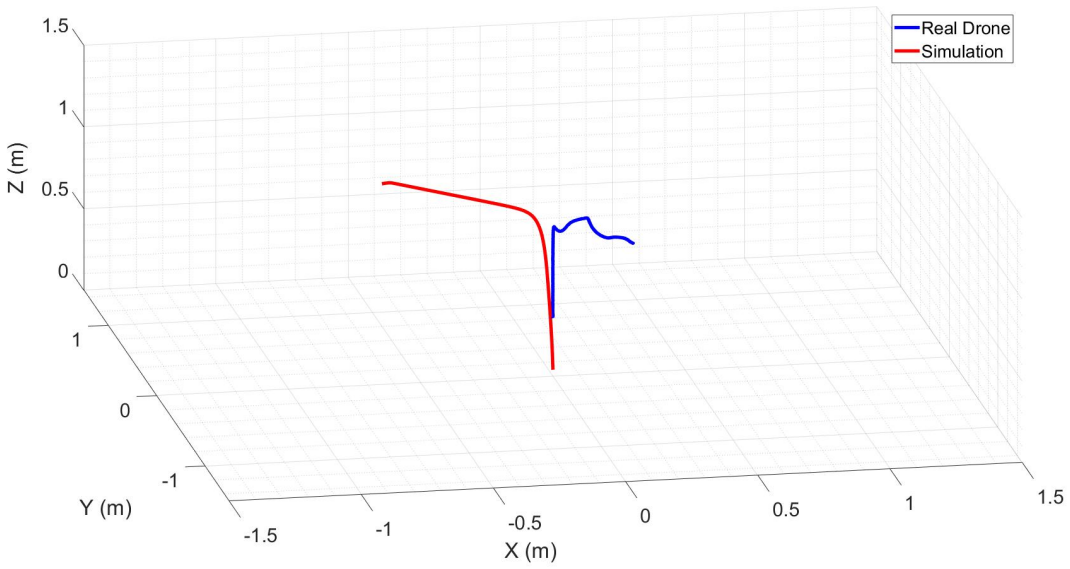

Figure 4.8: Trajectory of Hover Control in Simulation and Real Drone Testing

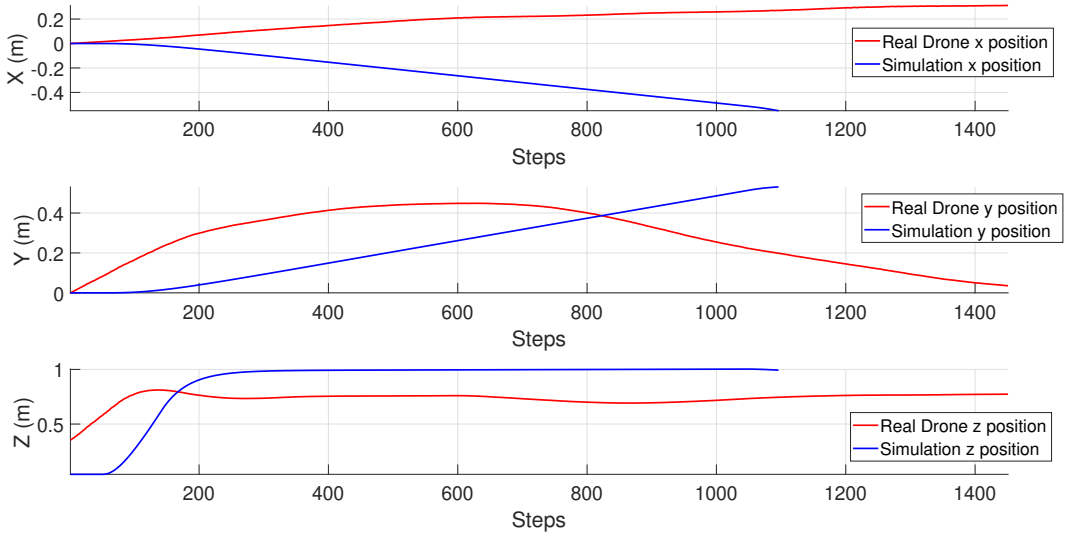

Figure 4.9: Position of Hover Control in Simulation and Real Drone Testing

Figure 4.10 shows the orientation for the simulation and real drone. When the done is hovering, the simulated drone has approximately zero roll and pitch angle, whereas real drone has an initial roll angle when takeoff. This initial roll angle causes an initial movement in y-direction as shown in the trajectory graph. Although the orientation for simulation is approaching to zero, a side slipping still can be observed as mentioned in the previous. This is caused by the different initial motor speed when takeoff which leads to a drifting in position. On the other hand, both roll and pitch angle for the real drone control show different extent of noise. Even though the overall trend is converged to zero, but the noise is serious when implementing the trained model to real life. Additionally, a turning in the value of yaw angle exist for simulation. This can be explained that, since

the quaternion is captured, the conversion from quaternion to Euler angle is necessary. However, due to the transformation formula, the arctan of the equation lead to switch up for the angle in some cases.

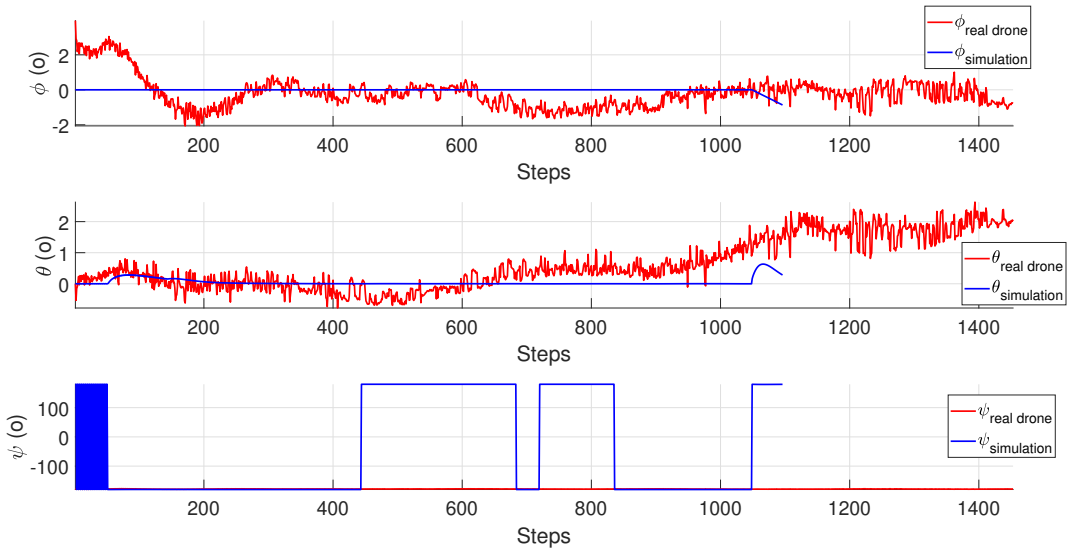

Figure 4.10: Orientation of Hover Control in Simulation and Real Drone Testing

Another flight control is tested to reveal the stability of the real drone. A roll command is sent to both simulation and real drone in this scenario. The results are shown in the following figures. As per figure 4.11, simulation shows a more stable flight compare to the real drone. Even though there is a little bit variation in the x direction as shown in figure 4.12, the overall flight path is reasonable. But for the real drone, a huge side slip occur during takeoff. This can finally lead to unstable in machine learning control. Moreover, in the orientation graph, the real drone occurs to have negative pitch angle. The drone shows a head-down phenomena. If further adding pitch angle, the forward motion for the drone may not be as accurate as the simulation. As a result, the control errors for the real drone needed to be adjusted.

Therefore, a basic benchmarking for the real drone control has done with comparison to simulation. The real drone shows different extents of unstable and control noise. The effect of these error will lead to inaccurate of testing the model. At the same time, simulation also shows some unstable when hovering. Therefore, some designs of controller (such as PID controller) are required to tuning the control error for both real drone and simulation model.

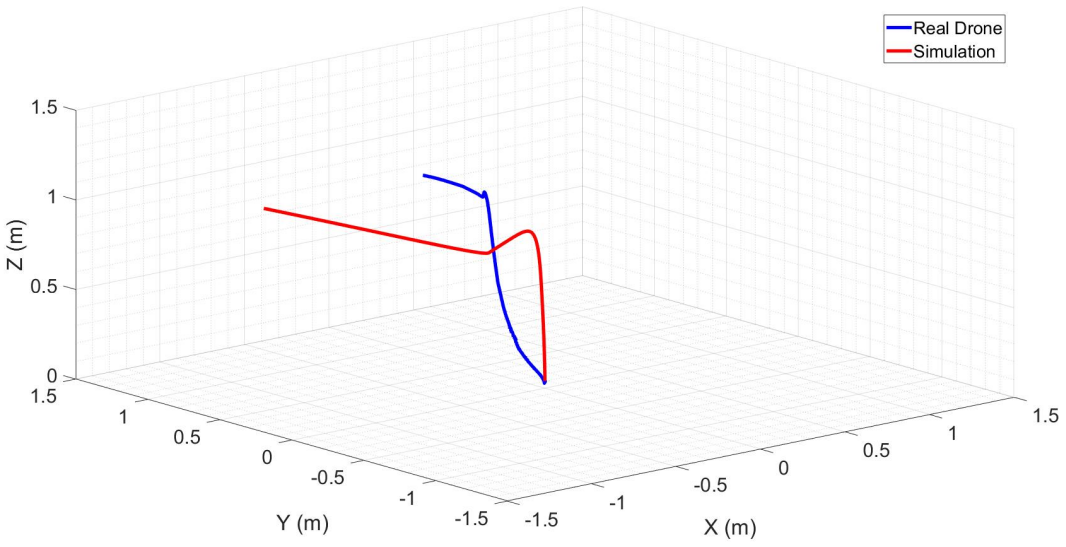

Figure 4.11: Trajectory of Roll Control in Simulation and Real Drone Testing

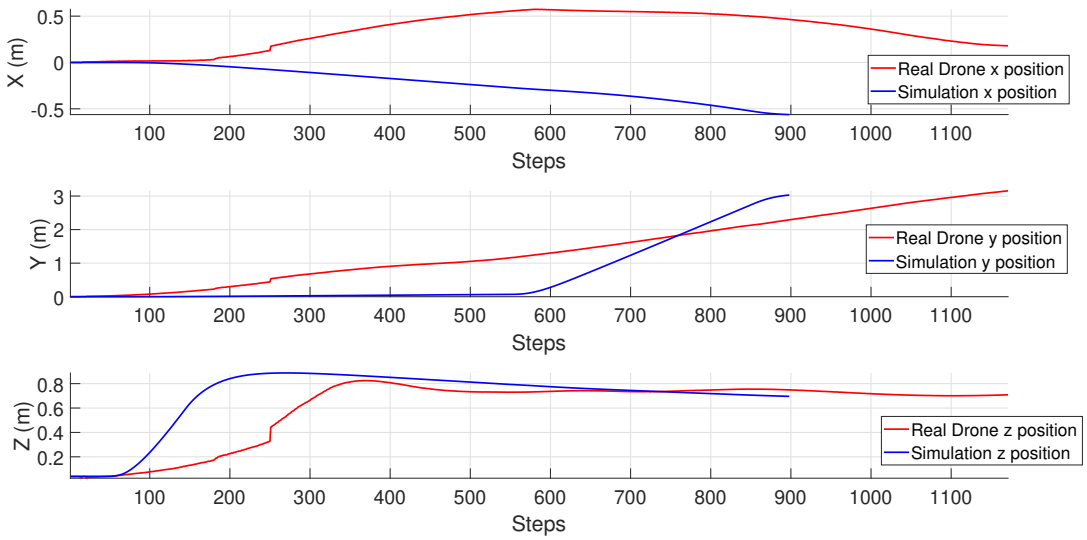

Figure 4.12: Position of Roll Control in Simulation and Real Drone Testing

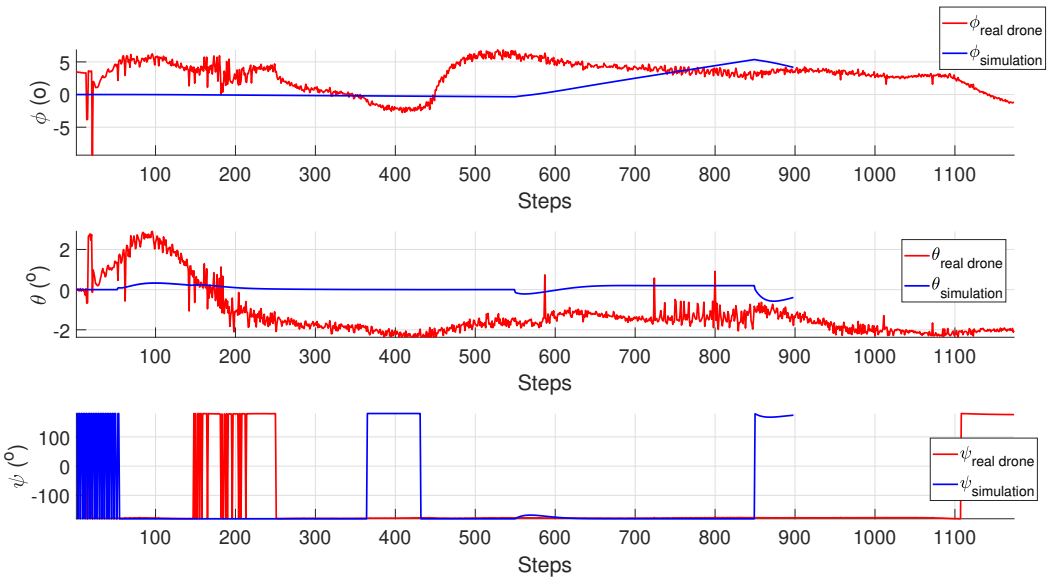

Figure 4.13: Orientation of Roll Control in Simulation and Real Drone Testing

Chapter 5

Conclusion

The objectives discussed in section 1 are partially achieved in this thesis work. A detailed framework of applying machine learning to real drone control is constructed for both simulation and real drone testing. A brief demonstration of the functionality about DDPG is presented. Subsequently, Limited number of reinforcement learning theories are discussed to provide improvement of algorithm performance in different extents. Designing concept of reward engineering is illustrated under a main theorem. Methodology of network connection are presented, inspired by a few Parrot AR Drone connection setting.

The machine learning application framework is built to couple with both hardware and software. Simulation environment is constructed to provide reliable and sufficient training circumstance. Real drone testing environment is built to ensure stable testing and implementation of reinforcement learning. A drone controller is developed to ensure collaboration between software and hardware, and simulation to real drone testing.

The primary goals of the experiment is to identify the performance of machine learning in unmanned aerial vehicle autonomous control. Studies of the applied algorithm in simulation provides a good understanding of machine learning performance based on the construction. These can further be implemented into real drone testing.

5.1 Result Summary

The main outcomes of this thesis can be summarized by the following pints:

- The machine learning implementation framework is developed to ensure high stability and reliability in simulation and real drone testing. Minimum calibrations are required if diverse flight platform adoption is needed.
- The simulation result presents some extents of UAV autonomous control. A simple train task is presented followed by a comparison of two popular algorithms. Better performance is shown in deep deterministic policy gradient algorithm. A more realistic training task is then presented. The result reveals DDPG still be able to learn sufficiently.
- Result from robustness tests show some extent of autonomy and generality of the

trained model. Although higher level of flight control is not achieved, the trained model is sufficient in the simulation and can be applied to real drone testing.

- Benchmarking of simulation and real drone flight is presented in this thesis work. The physical testing demonstrate a medium level of error which can affect the testing result if implementing the trained model to real life.

5.2 Future Work

Certain improvements are suggested through out the development of the thesis regarding to model construction and learning tasks formation. In this section, the advices for future work is divided into two parts. First refers to improvement of training task and the second refers to model improvement.

Training Task Improvement
The following suggestions are indicate as future work to improve the training task to achieve higher level of autonomy and generality:

- Training for more complex trajectories such as four way points circling or upgrading level of random takeoff tracking with random generated way points. This allow the drone achieve higher level of autonomy.

- Better design of reward function. This can be achieved by replacing schedule reward or fixed reward. For instance, a reward function based on velocity can be used to keep the drone stay at desire location. More automatic reward and environmental reward allows to increase to autonomy of the trained drone.

Model Improvement
The following suggestions are indicate as future work for model improvement to promote the accuracy of control and higher efficiency and robustness on training:

- Using higher level of reinforcement learning or combination of different reinforcement learning techniques. Higher level of reinforcement learning such as PPO or TRPO provide better allocation of computer resources for training. This can largely increase the training speed and can be used to achieved more complicated task. Learning technique such as hindsight experience replay can supplement the algorithm to be more sufficient while training.

- Design PID controller for tuning the drone action. This allows to provide more accurate control while applying the machine learning to autonomous control. This helps to build up a more robust control system for both simulation and real drone testing.

The code for this thesis can be accessed at https://github.com/ChiRanTou/DDPG-ardrone.

Reference

[1] Jeremy Jordan. Setting the learning rate of your neural network. https://www.jeremyjordan.me/nn-learning-rate/, 1 March 2018.

[2] Mofan Zhou. Batch normalization. https://morvanzhou.github.io/tutorials/machine-learning/ML-intro/3-08-batch-normalization/, 06 December 2016.

[3] Pat Langley. The changing science of machine learning. *Machine Learning*, 82(3):275–279, 2011.

[4] Brett L Moore, Todd M Quasny, and Anthony G Doufas. Reinforcement learning versus proportional–integral–derivative control of hypnosis in a simulated intraoperative patient. *Anesthesia & Analgesia*, 112(2):350–359, 2011.

[5] Jonathan Hui. Rl - value learning. https://medium.com/@jonathan_hui/rl-value-learning-24f52b49c36d, Oct 2018.

[6] Jemin Hwangbo, Inkyu Sa, Roland Siegwart, and Marco Hutter. Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, 2(4):2096–2103, 2017.

[7] Dominic Thewlis, Chris Bishop, Nathan Daniell, and Gunther Paul. *A comparison of two commercially available motion capture systems for gait analysis: High end vs low-cost.* 2011.

[8] Siyi Li, Tianbo Liu, Chi Zhang, Dit-Yan Yeung, and Shaojie Shen. Learning unmanned aerial vehicle control for autonomous target following. *arXiv preprint arXiv:1709.08233*, 2017.

[9] William Koch, Renato Mancuso, Richard West, and Azer Bestavros. Reinforcement learning for uav attitude control. *ACM Transactions on Cyber-Physical Systems*, 3(2):22, 2019.

[10] Roman Barták, Andrej Hraško, and David Obdržálek. A controller for autonomous landing of ar. drone. In *The 26th Chinese Control and Decision Conference (2014 CCDC)*, pages 329–334. IEEE, 2014.

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[12] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[13] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. 2014.

[14] D Randall Wilson and Tony R Martinez. The need for small learning rates on large problems. In *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No. 01CH37222)*, volume 1, pages 115–119. IEEE, 2001.

[15] Tianbing Xu, Qiang Liu, Liang Zhao, and Jian Peng. Learning to explore with meta-policy gradient. *arXiv preprint arXiv:1803.05044*, 2018.

[16] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[17] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[18] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[19] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[20] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[21] Riashat Islam, Peter Henderson, Maziar Gomrokchi, and Doina Precup. Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. *arXiv preprint arXiv:1708.04133*, 2017.

[22] Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks. *arXiv preprint arXiv:1804.07612*, 2018.

[23] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[24] Yiding Jiang, Dilip Krishnan, Hossein Mobahi, and Samy Bengio. Predicting the generalization gap in deep networks with margin distributions. *arXiv preprint arXiv:1810.00113*, 2018.

[25] Adam Stooke and Pieter Abbeel. Accelerated methods for deep reinforcement learning. *arXiv preprint arXiv:1803.02811*, 2018.

[26] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.

[27] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[28] Daniel Dewey. Reinforcement learning and the reward engineering principle. In *2014 AAAI Spring Symposium Series*, 2014.

[29] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping.

[30] P Stephane, B Nicolas, E Pierre, and D Frederic. Ar. drone developer guide sdk 2.0, 2012.

[31] Petrovicz Benedek. Kamera alapú beltéri navigáció az ar drone eszközön, 2017.

[32] Cristinel Ababei. Lab: Controlling an ar.drone.

[33] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2149–2154. IEEE, 2004.

[34] AmandaDattalo). Ros/introduction. http://wiki.ros.org/ROS/Introduction, Aug 2018.

[35] TUM Computer Vision Group). tum simulator. https://github.com/tum-vision/tum_simulator, March 2014.

[36] Pengfei Yu, Derrick Ho, and KC Wong. An adaptable indoor flight test implementation for small uavs.

[37] Mani Monajjemi. ardrone tutorials. https://github.com/AutonomyLab/ardrone_autonomy.

[38] Mike Hammer. ardrone tutorials. https://github.com/mikehamer/ardrone_tutorials, Oct 2013.

[39] Rohit Murthy, Harikrishnan Suresh, and Chris Song. Learning control policies for quadcopter navigation with battery constraints.

Appendix A

Methodology of WIFI Connection

One way to accomplish the WIFI connection is proposed to use an extra router connect to the root router that link with OptiTrack. This method can avoid the DNS address conflict issue. However, the methodology is not able to achieve the using of both network simultaneously. As per the figure in the following, although the secondary router has a different DNS compare to the Parrot AR Drone, since the network connection is linked to root router, the LAN is route the root router which still cause a DNS address conflict (IP Address of 192.168.1.1 for both root router and Parrot AR Drone). Therefore, this method is not achievable. The only way to ensure the network works simultaneously is described in the main thesis which can be achieved by changing the router DNS to any IP available address except 192.168.1.x.

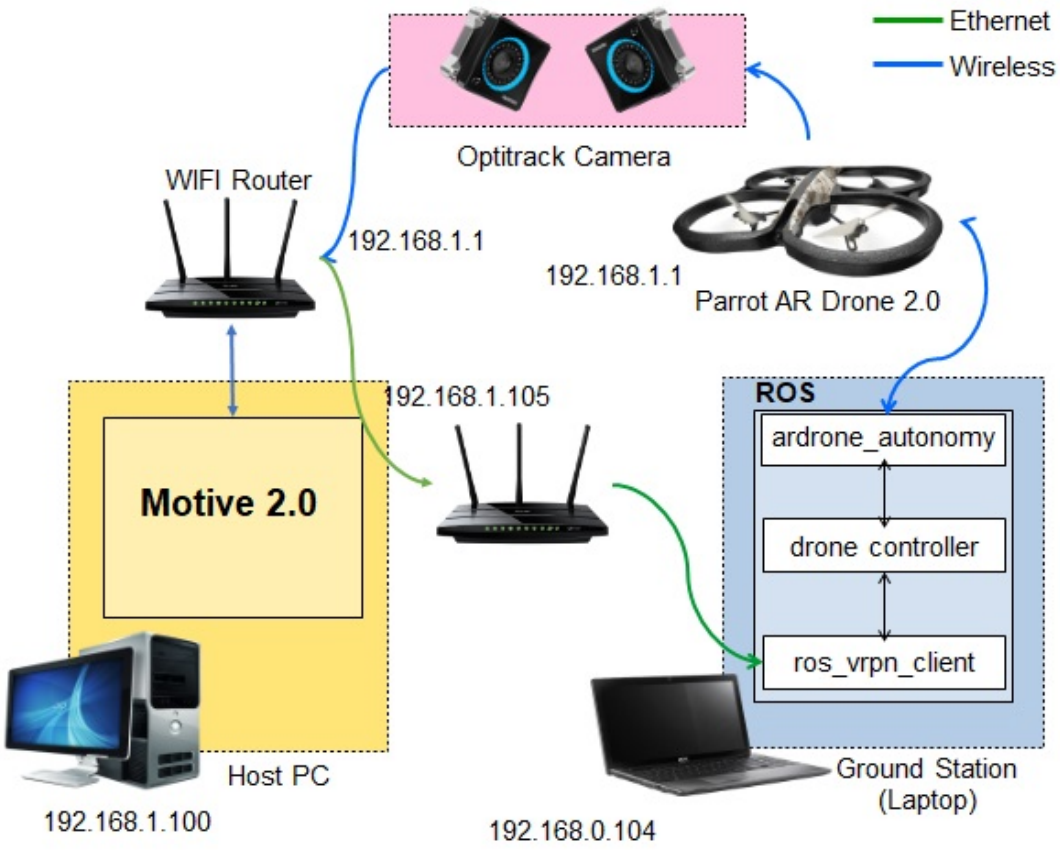

Figure A.1: Network Connection Through an Extra Router