POLITECNICO
MILANO 1863

Using Reinforcement Learning to Tune PID Controller of Quadcopters

Master Thesis
Computer Science and Engineering

Author

Yazeed Alrubyli

Advisor

Prof. Andrea Bonarini

2021-2022

Abstract

Unmanned Aerial Vehicles (UAVs), and more specifically, quadcopters, must be stable during their flights. Altitude stability is usually achieved using a PID controller built into the flight controller software. Furthermore, the PID controller has gains that need to be adjusted to achieve optimal altitude stabilization during the quadcopter's flight. Therefore, control system engineers need to adjust those gains using extensive modeling of the environment, which might change from one environment and condition to another. As quadcopters penetrate more sectors, from the military to the consumer sectors, they have been put into complex and challenging environments more than ever before. Hence, intelligent self-stabilizing quadcopters are needed to maneuver through those complex environments and situations. Here, we demonstrate that by using online reinforcement learning with minimal background knowledge, the altitude stability of the quadcopter can be achieved using a model-free approach. We found that altitude stabilization can be achieved faster with a small memory footprint using an activation function like Sigmoid. In addition, using this approach will accelerate development by avoiding extensive simulations before applying the PID gains to the quadcopter. Our results demonstrate the possibility of using the trial and error approach of reinforcement learning combined with activation function and background knowledge to achieve faster quadcopter altitude stabilization in different environments and conditions.

Keywords: reinforcement learning (RL), Q-learning, PID tuning, unmanned aerial vehicle (UAV), quadcopter

Abstract in lingua italiana

Unmanned Arial Vehicles (UAV), I veicoli aerei senza pilota, e più specificamente i quadricotteri, devono essere stabili durante i loro voli. La stabilità dell'altitudine si ottiene solitamente utilizzando un controllore PID integrato nel software del controller di volo. Inoltre, il controllore PID ha guadagni che devono essere regolati per raggiungere una stabilizzazione dell'altitudine ottimale durante il volo del quadrirotore. A tal fine, gli ingegneri dei sistemi di controllo devono ottimizzare questi vantaggi utilizzando un'ampia modellazione dell'ambiente, che potrebbe cambiare da un ambiente e condizione a un altro. Man mano che i quadricotteri penetrano in più settori, dall'esercito al settore dei consumatori, sono stati inseriti in ambienti complessi e impegnativi più che mai. Quindi, sono necessari quadricotteri intelligenti autostabilizzanti per manovrare attraverso quegli ambienti e situazioni complesse. Qui mostriamo che utilizzando l'apprendimento per rinforzo online con una conoscenza di base minima, la stabilizzazione dell'altitudine del quadrirotore può essere raggiunta utilizzando un approccio senza modello. Abbiamo scoperto che utilizzando una funzione di attivazione come una sigmoide, la stabilizzazione dell'altitudine può essere ottenuta più velocemente con un uso di memoria ridotto. Inoltre, l'utilizzo di questo approccio accelererà lo sviluppo evitando simulazioni estese prima di applicare i guadagni PID al quadrirotore. I nostri risultati dimostrano la possibilità di utilizzare l'approccio per tentativi ed errori dell'apprendimento per rinforzo combinato con la funzione di attivazione e la conoscenza di base per ottenere una stabilizzazione dell'altitudine del quadrirotore più rapida in diversi ambienti e condizioni.

Parole chiave: reinforcement learning (RL), Q-learning, PID tuning, unmanned aerial vehicle (UAV), quadrirotore

Acknowledgement

I would first like to thank my thesis supervisor Prof. Andria Bonarini of the Department of Computer Science and Engineering at Politecnico di Milano. The door to Prof. Bonarini's office was always open whenever I ran into trouble. He consistently allowed this thesis to be my own work, and without his trust, participation, and input, the research could not have been successfully conducted.

Finally, I must express my very profound gratitude to my parents Naif and Joza, and to my brother Nawaf for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without their unlimited continuous support. Thank you.

Yazeed Alrubyli

Contents

Abstract **i**

Abstract in lingua italiana **iii**

Acknowledgement **v**

Contents **vii**

1 Introduction **1**
1.1 Reinforcement Learning 2
1.1.1 Q-learning 5
1.2 Activation Functions 6
1.2.1 Sigmoid Function 6
1.3 PID Controller 7
1.3.1 Trial and Error Tuning Method 8
1.4 Contributions 9
1.5 Thesis Outline 9

2 Literature Review **11**

3 Tuning Quadcopter PID Controller Using Q-learning and Sigmoid **15**
3.1 Problem Formulation 17
3.2 Experiment Setup 20

4 Results and Discussion **25**

5 Conclusions **31**
5.1 Future Developments 31

Bibliography **33**

A Appendix A: Quadcopter Assembly and Simulation **41**
A.1 Off-the-shelf Components 42
A.2 3D Printed Components 51

B Appendix B: Simulation Code **55**
B.1 Q-learning Algorithm with Sigmoid Function 55
B.2 PID Controller 58

List of Figures **65**

List of Tables **67**

List of Symbols **69**

1 | Introduction

In recent years, Unmanned Aerial Vehicles (UAVs) are being used in many applications, some of which are complex and challenging [7]. Areas that UAVs have been applied to belong to critical sectors like health [41, 44], and military [21, 23], but also such vehicles can be seen in the civilian sectors [5, 22, 57]. UAVs applications are inspection [45, 47], surveillance [28, 59], patrolling [42, 65], or rescuing [17, 27, 64], to name a few. Furthermore, these applications are becoming increasingly challenging. Therefore, autonomy is needed to assist in solving such complex problems [7]. Problems such as maneuvering through unknown spaces while avoiding obstacles automatically, intelligently planning and following a path, and finally completing the required mission successfully without the need for human intervention [55]. According to [21], there exist different UAVs constructions to be used for a given application, and they can generally be

- Fixed-wing
- Rotary-wing
- Flapping-wing
- Multirotor

Multirotor UAVs are attractive because of their compact size, simple construction, low-cost maintenance, maneuverability, and vertical motion [31, 33], and the most common member of multirotor UAVs is the quadcopter [14]. On the other hand, quadcopters suffer from increasingly challenging tasks to achieve. Furthermore, there are too many moving parts that need to be automated, such as correcting for disturbances, actuator degradation, or communication and processing delays are a few of the issues that need to be addressed by the flight controller [7]. To altitude stabilize the quadcopter, control algorithms such as Proportional–Integral–Derivative (PID) controller [53], fuzzy control [25], or linear–quadratic regulators (LQR) [71] is used [39]. PID controllers are widely applied in quadcopters, due to their simple implementation and tunning flexibility. When UAVs operate in complex unknown environments, intelligent adaptive controller is needed [33].

1.1. Reinforcement Learning

Machine learning is making machines learn complex generalized models from data to achieve a set of usually unforeseen tasks, and is traditionally divided into three main categories, supervised learning, unsupervised learning, and Reinforcement Learning (RL). RL is a branch of machine learning where an agent learns by trial and error, that is while acting in an environment, the agent will get rewarded if the desired action is achieved, and punished otherwise, the agent learns to do more of the desired actions to maximize its reward [35]. According to [63], there are three general categories for the RL algorithms. Each of these has its use cases.

- **Dynamic Programming Methods** require the model of the environment, but they are mathematically well developed.
- **Monte Carlo Methods** simple conceptually and do not require the model of the environment, but not suited for step-by-step computation.
- **Temporal-Difference Learning Methods** complex to analyze, but do not require the model of the environment, and are fully incremental.

RL problems can be formulated as Markov Decision Processes (MDPs). In MDPs, an agent is interacting with the environment over time. That is, by observing the environment state at each time step, the agent chooses a proper action to take which will transition the environment to some state, and therefore the agent will be given a reward as a consequence of the action taken. So, the agent's objective is to maximize the overall reward that can be received from taking a certain sequence of actions in a given environment, that is, the agent focuses not on the immediate reward but on the overall rewards that the agent will receive. Therefore, the components of MDPs are the following

- Agent
- Enviroment
- States
- Actions
- Rewards

Formally, in MDPs, S represent the set of states, A represent the set of actions, and R represent the set of rewards. At each time step $t = 0, 1, 2, ...$, the agent senses the enviroment to get its state $S_t \in S$. Based on the state observed by the agent at time t, the agent selectes an action $A_t \in A$ which can be represented as state-action pair (S_t, A_t).

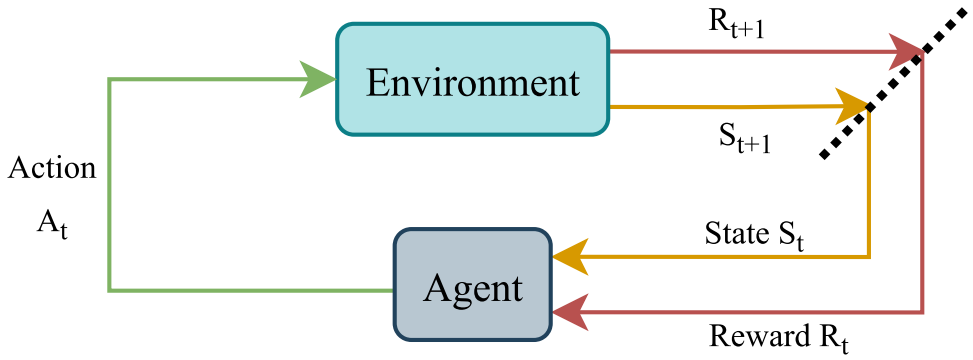


Figure 1.1: Agent-environment relation in a Markov decision process [3].

Based on the action taken at time t, the envormatoint will transtion to time $t+1$ with an enviroment state $S_{t+1} \in S$. Reward $R_{t+1} \in R$ will be given to the agent for the action A_t in state S_t whcih can be represented as a function of state-action pair as showen in equation (1.1). This process is illstrated in Figure 1.1.

$$f(S_t, A_t) = R_{t+1} \tag{1.1}$$

The expected return is what drives the agent to take a specific action in some state at a certain point in time. Mathematically, the equation (1.2) represents the cumulative return G at time t, where the T is the final time.

$$G_t = R_{t+1} + R_{t+2} + ... + R_T \tag{1.2}$$

The previous definition of expected return makes sense if the time T is defined. In another word, if that task at hand is continuous, a discounted return is represented instead as shown in equation (1.3) where γ is the discount rate that will be used to determine how important future rewards and its number of between 0 and 1. If γ is low it means the agent cares more about the immediate rewards and less about the future ones. On the other hand, if $\gamma = 1$, then the agent cares about immediate and future rewards equally to drive its actions.

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+2} + ... \\ &= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \end{aligned} \tag{1.3}$$

The agent's objective therefore is to maximize G_t from equation (1.3). Policies represent

the probability that an agent will select an action given a state. Value functions, on the other hand, are responsible of measuring how good it is to pursue an action or a state. A policy π followed by an agent represent the probability distribution over actions $a \in A(s)$ in state $s \in S$, that is, $\pi(a|s)$. State-value function v_π shows how good a state s is under the policy π. Mathematically, equation (1.4) gives the expected return for starting with state s at time t and following policy π.

$$\begin{aligned} v_\pi(s) &= E\left[G_t|S_t = s\right] \\ &= E\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s\right] \end{aligned} \tag{1.4}$$

Action-value function q_π shows how good for the agent to take action a in state s under policy π. Mathematically, equation (1.5) gives the expected return for taking action a starting with state s at time t and following policy π.

$$\begin{aligned} q_\pi(s,a) &= E\left[G_t|S_t = s, A_t = a\right] \\ &= E\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a\right] \end{aligned} \tag{1.5}$$

RL algorithm is therefore trying to find the optimal policy. TO that matter, in order to compare policies, we say that policy π is better than $\pi^{'}$ if the expected return of π is higher or equal to the expected return of $\pi^{'}$. Mathematically,

$$\pi \geq \pi^{'} \text{ if and only if } v_\pi(s) \geq v_{\pi'}(s) \text{ for all } s \in S \tag{1.6}$$

Optimal state-value function $v_*(s)$ as showen in equation (1.7) is the maximum expected return achievable in each state by any policy π.

$$v_*(s) = \max_\pi v_\pi(s) \tag{1.7}$$

Optimal action-value function, Q-function, as showen in equation (1.8) is the maximum expected return achievable in each state-action pair by any policy π.

$$q_*(s,a) = \max_\pi q_\pi(s,a) \tag{1.8}$$

Bellman optimality equation (1.9) considers the expected return given the state-action pair. The return is composed of the reward for taking an action a in-state s at time t, and the maximum expected return for the state-action pair (Q-value) that will be taken in time $t+1$ multiplied with the discount rate γ.

$$q_*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(s',a')\right] \tag{1.9}$$

1.1.1. Q-learning

The goal of Q-Learning algorithm is to find the optimal policy by learning the optimal Q-values for every state-action pair. Specifically, the algorithm iteratively updates q_* for each state-action pair using the Bellman equation 1.9. The algorithm will iterate updating the Q-value for each state-action pair using until q converges to q_*, that is, the optimal Q-function. Q-table is used to store Q-values for each state-action pair. Columns represent the actions, and rows represent the states. Therefore, the dimension of the Q-table is the number of actions by the number of states. Since the agent knows nothing about the environment at the beginning the table may be initialized to zero. In each episode, the agent chooses an action based on the state and highest Q-value given the state in the Q-table. But, to prevent the agent from finding a local minimum as the highest value in the table at the point in time, a factor ϵ is introduced to make the agent explore at the beginning, and as it learns and gets more experiences, it increases exploiting what the agent learns to maximize the return. $\epsilon - greedy$ strategy is give a balance between exploration and exploitation to avoid to prevent the agent to exploit a local maximum. The idea is to make the agent explore the most at the beginning, and by using a decay factor, this number will be reduced by the decay factor in every episode till reaches zero, which means the agent is greedy and tries to exploit what has been learned 100% of the time to maximize the total return. To know if the agent should choose exploration or exploitation at each episode, a random number between zero and one is drawn and compared with ϵ, then exploitation takes place, that is, choosing the highest Q-value in the Q-table given the state. Otherwise, the agent will explore, and action is chosen randomly given the state. α is the learning rate that is a number between zero and one, which can be seen as how quickly the agent abandons the previous Q-value in the Q-table for the new Q-value. In different words, $\alpha = 1$ means the newly calculated Q-value will replace the old one in the corresponding state action cell in the Q-table [63].

$$q(s,a) = (1-\alpha)q(s,a) + \alpha(R_{t+1} + \gamma \max_{a'} q(s',a')) \tag{1.10}$$

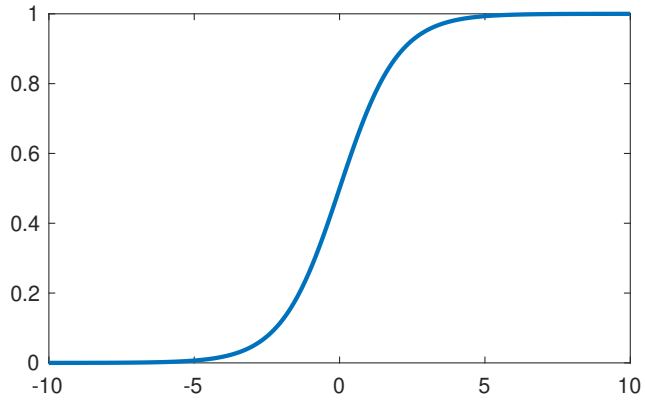


Figure 1.2: Sigmoid function [60].

1.2. Activation Functions

In neural networks, activation functions are used to introduce a nonlinearity, that is, a nontrivial mapping from the input to the output passing through cascaded layers of neurons with corresponding activation functions [60]. Furthermore, activation functions come in different flavors, to name a few

- Sigmoid
- Tanh
- ReLU
- Leaky ReLU
- SoftMax
- Linear

1.2.1. Sigmoid Function

Sigmoid function as defined in (1.11) is a great tool to squash any input to be in the range between 0 and 1 (see Figure 1.2). It is not symmetrical around zero, which means the values will be strictly nonnegative [60].

$$\sigma(z) = \frac{1}{1 - e^{-z}} \tag{1.11}$$

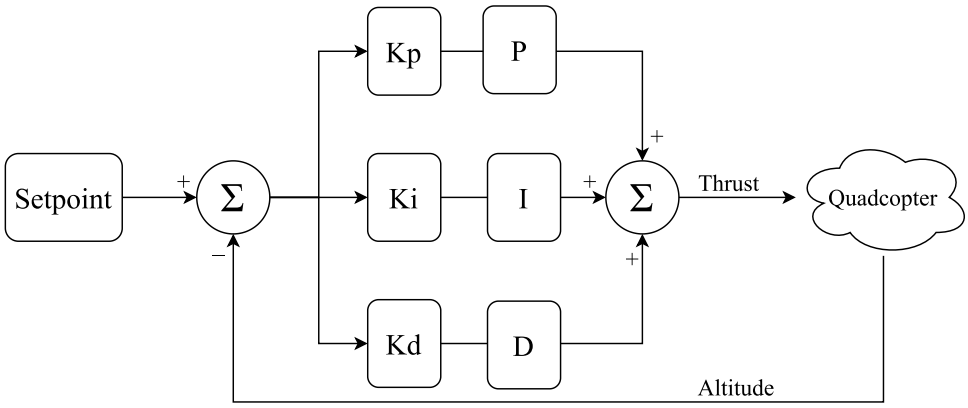


Figure 1.3: Quadcopter's PID controller example [3].

1.3. PID Controller

A control system consists of a plant (ex. quadcopter's controller), that when given an input, obtains the target output with a target performance. Control systems have two general configurations, open-loop, and closed-loop as shown in Figure 1.3. Unlike open-loop where feedback is not returned to the system to correct for errors, closed-loop allows for the system to recover from errors by adjusting the plant using a feedback loop [48].

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{1.12}$$

Due to the simplicity and practicality of the PID controller, many industrial processes are utilizing it [9]. The PID controller parameters depend on the uncertainty of the process and operational conditions [16]. When it comes to tuning the PID controller parameters, there are two general approaches, classical and modern [9]. Classical PID controller tuning methods are simple and computationally efficient. It gives a starting point to a necessary further tuning which is a shortcoming of these methods as the use they make assumptions about the plant and desired output. The following are a few of the well-known classical tuning methods for the PID controller

- Trial and Error Method
- Ziegler-Nichols Step Response Method
- Ziegler-Nichols Frequency Response Method
- Relay Tuning Method
- Cohen-Coon Method

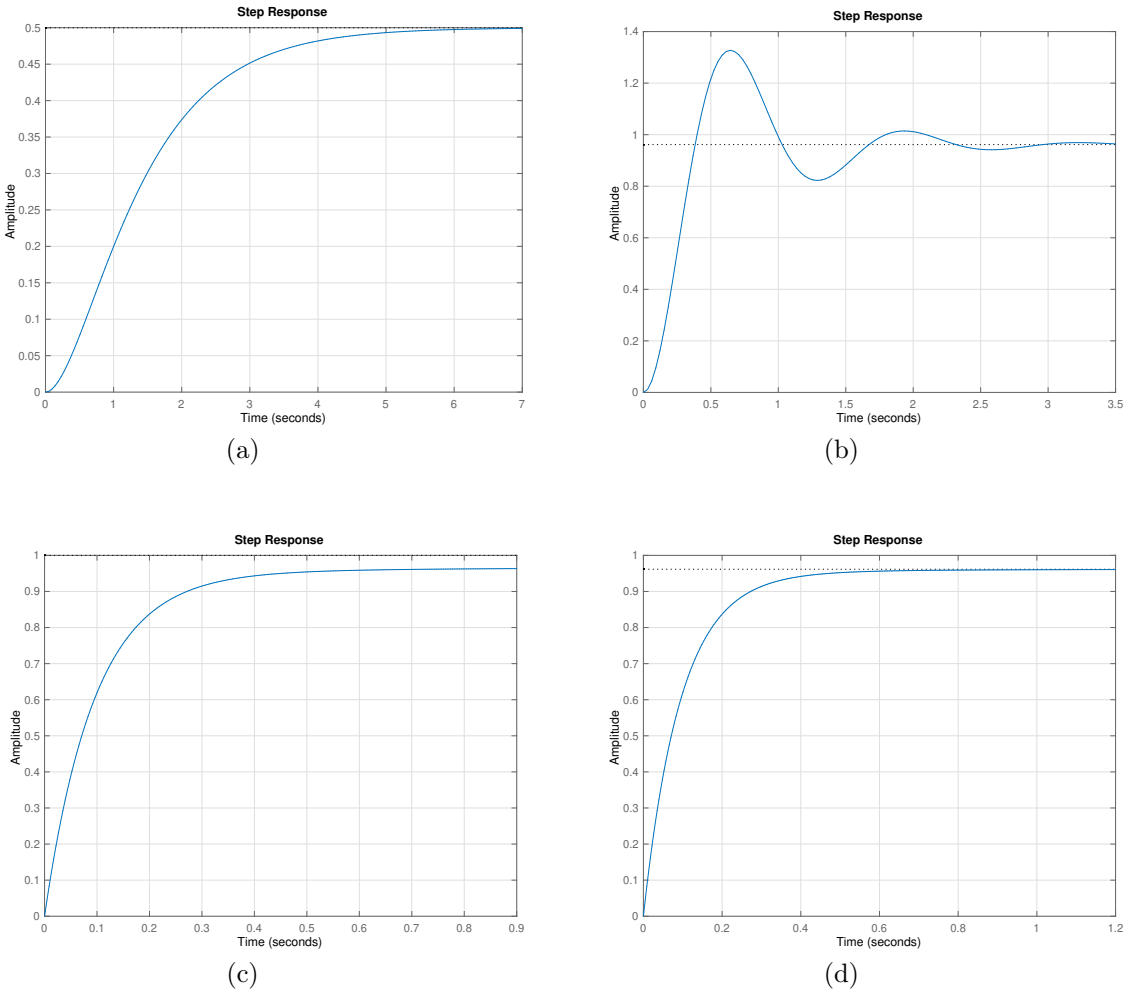


Figure 1.4: (a) Without PID (b) P (c) PD (d) PID [34]

1.3.1. Trial and Error Tuning Method

If the system must remain online, one tuning method is to first set K_i and K_d values to zero. Increase the K_p until the output of the loop oscillates; then set K_p to approximately half that value for a "quarter amplitude decay"-type response. Then increase K_i until any offset is corrected in sufficient time for the process, but not until too great a value causes instability. Finally, increase K_d, if required, until the loop is acceptably quick to reach its reference after a load disturbance. Too much K_d causes excessive response and overshoot. A fast PID loop tuning usually overshoots slightly to reach the setpoint more quickly; however, some systems cannot accept overshoot, in which case an overdamped closed-loop system is required, which in turn requires a K_p setting significantly less than half that of the K_p setting that was causing oscillation [13].

1.4. Contributions

A solution has been presented to tackle two main problems raised when using tuning the quadcopter's PID controller within the context of reinforcement learning. First, is the full online tuning problem. Second, the memory footprint might expand rapidly due to the nature of Q-learning algorithms. The presented solution solves the first problem by engineering the required function to effectively guide the plant to reach the setpoint. Furthermore, the second problem has been solved using the sigmoid activation function to limit the dimension of the Q-table.

*Contribution accepted and nominated for **Best-Paper Award** at the IEEE International Conference on Mechatronics and Automation (ICMA), 2022.*

1.5. Thesis Outline

The thesis is organized into six chapters. Chapter 2 presented an important literature review concerning the state of the art in tuning the PID controller using RL. Also, tuning quadcopters PID controllers are discussed in chapter 2. Chapter 3 present a novel approach of tuning quadcopter PID controller using PID. Furthermore, the simulation setup has been presented in chapter 4. Chapter 5 shows the results with a discussion of the proposed method. The conclusion is presented in chapter 5 with an overall summary of the thesis, and possible future developments.

2 | Literature Review

Using machine learning (ML) to tune the PID controller has been accomplished in the literature. Methods that utilized supervised learning algorithms such as Gradient descent have been explored [52, 74]. Those approaches suffer when non-linearity systems are the ones being investigated in which the PID controller is not stable. Furthermore, fuzzy logic (FL), genetic algorithms (GA), and artificial neural networks are approaches used in the artificial intelligence (AI) literature to tune PID controllers [10, 11, 37, 46, 56, 58, 61, 70]. Deep neural network (DNN) and RL are two of the solutions that have been suggested by the literature to model the PID controller and tune its gains for non-linear systems [4, 10, 74]. By the same token, RL algorithms such as Q-learning have been used extensively with different flavors to address such a problem of non-lieanrity [10, 11]. Further, a combination of RL and deep learning called (Deep RL) which is a subfield of ML shows potential by the work of [62] and [12], the former faced an issue with the reward function of the RL algorithm, and the latter faced an issue with the controller stability. In addition, a work proposed by [16] that uses an offline tuning technique based on Deep RL, but using it for online tuning is computational expensive. RL off-policy algorithms such as Deep Deterministic Policy Gradient (DDPG) [43], Soft Actor-Critic (SAC) [29] and Twin Delayed DDPG (TD3) [24] have been used for control applications, and more specifically in robotics [8].

When it comes to quadcopters, they are highly non-linear systems [1]. Classical PID tunning methods are not effective in reaching optimal gains with such systems in which an adaptive controller should be used [8]. Furthermore, a neural network and FL might still be used to build the required adaptive controller [18, 39]. Also, GA has also been applied to overcome the complexity of building an adaptive PID controller but the time to settle is still needed to be minimized for real-world systems [2]. Complexities such as delayed response, and strong and high-frequency disturbances can complicate the control tasks and prevent the classical PID controller to generalize to unforeseen situations. On the other hand, if the goal is to maximize the system adaptability in complex environments, RL might be used [33].

Table 2.1: The Q-table of the work in [36].

	u1300	...	**d1700**
1	$v(s_1, a_1)$	...	$v(s_1, a_4 0)$
⋮	⋮	⋮	⋮
40	$v(s_4 0, a_1)$	...	$v(s_2 0, a_4 0)$

A review has been presented in [7] using deep reinforcement learning on drones for various purposes like navigation and PID tuning. One of the reviewed papers proposed a design to implement a standard PID controller to ensure stable maneuvering. By taking various learning parameters as an input to the controller, along with the PID controller gains, and based on the current state of the UAV, and the reward expected, the next state is updated. The tuning process aims to reach the optimum values for the three gains of the PID controller. The simulation was done to reach optimum values for the gains by increasing the reward and decreasing the penalty over time. Applying the algorithm to a physical quadcopter, using the proposed PID algorithm made the learning process computationally expensive, and it took more time than the batteries can stand. The RL algorithm used is Q-learning which aims to maximize the overall reward signal. Q-learning helps the drone to reach the optimal values for the PID controller gains based on taking the proper action to get as much reward as possible. The Q-table of the Q-learning algorithm is used to store the Q-values related to each state which represents the UAV position. The problem that has been faced is with regards to the Q-table as claimed that could lead the size of the table to grow in a way that will make it hard to implement the algorithm in reality due to hardware memory size and battery limitation. This issue has been fixed by using an approximation technique Fixed Sparse Representation (FSR). Because the used technique is an approximation, that caused sometimes an overshooting of the quadcopter.

The work done by [36] in 2020 is similar to the one investigated in this thesis, that is, using Q-learning for PID controller of quadcopters to hold a given altitude. The work has a Q-table of 800 cells and a complex reward function. The structure of the Q-table represents 40 actions as columns, and 20 states as rows. The actions consist of the value of the Pulse Width Modulation (PWM) to activate certain motor speeds either up or down for the 4 motors of the quadcopter, for example, *u1500* means up force with 1500 PWM (see Table 2.1). The episodes exceed 2000, the first 1000 episodes are for exploration without decay, from 1000 to 2000 episodes the decay rate λ will reach around half. Thereafter, exploitation will increase takeover.

In cases like tuning the PID controller by using RL algorithms, and more specifically, using Q-learning, the Q-table is the bottleneck as shown in examples from the literature. The state of the environment is continuous when the sensor is the altitude, even with discretization, the Q-table is still the bottleneck as discussed in this chapter. Solutions for such a problem have been presented, like using FSR or moving completely away from Q-learning to deep Q-learning (DQN) which solves the issue of Q-table, but the approach might introduce other problems of its own with regards to building a deep neural network, like defining the network structure (layers) and activation functions. In the next section, an approach for solving the Q-learning bottleneck, by limiting the expansion of the Q-table using the Sigmoid function is presented.

3 | Tuning Quadcopter PID Controller Using Q-learning and Sigmoid

Following the MDP for sequential decision-making problem formulation. The agent senses the state S from the environment, and based on the state-action pair (s, a) taken, it receives a reward r. Using the reward, it updates the Q-table by using the action-value function $q(s, a)$, then the agent should decide which action a_{t+1} to take in the next time step based on its current state using $\epsilon - greedy$ strategy. We have to face the problem of tuning the PID controller using a Q-learning algorithm to reach the desired setpoint. In this case, the quadcopter is the agent, that senses the altitude from the environment using sensors such as sonar, lidar, camera, etc. The set of states S has a cardinality of 100, i.e. $|S| = 100$. The state of the quadcopter will be decided by the output of the state function, see Algorithm 3.1. The state function has the error as an input, see Algorithm 3.2. The set of actions has been carefully designed to resemble the natural behavior of adjusting values of a knob. Knobs are limited by three actions, increase [+1], decrease [-1], or leave the knob the as is [0]. Therefore, the set of actions A has three elements $A = \{0, 1, -1\}$. Moreover, the reward as shown in the reward function (see Algorithm 3.3) will be assigned based on the error produced by the error function in Algorithm 3.2.

Algorithm 3.1 State Function

$S \leftarrow \sigma(err) \times 100$

return S

Algorithm 3.2 Error Function

$\Delta alt \leftarrow alt - alt'$

$\Delta d \leftarrow \Delta alt - \Delta alt\_prev$

$v \leftarrow d\Delta d/dt$

$E \leftarrow \Delta alt + v$

return E

Algorithm 3.3 Reward Function

```
if err < 0.01 then
    R ← 1
else if err < 0.1 then
    R ← 0
else if err < 1 then
    R ← −1
else
    R ← −2
end if
return R
```

To know how good for an agent to take a given action in a given state, a value function is computed as in Eq. 3.1. The function is called Q-function and produces a Q-value that will be placed in the Q-table.

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}\left[G_t | S_t = s, A_t = a\right] \\ &= \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a\right] \end{aligned} \tag{3.1}$$

The optimal action-value function gives the evaluation of how good it is following a policy π considering all the policies.

$$q_*(s,a) = \max_\pi q_\pi(s,a) \tag{3.2}$$

It can be seen that the Bellman optimality equation (1.9) considers the expected return given the state-action pair. The return is composed of the reward for taking an action a in state s in combination with the optimal action-value function multiplied by γ, the discount rate. Finally, the Q-value is computed using equation (3.3) and the Q-table (3.4) [66].

$$q(s,a) = (1-\alpha)q(s,a) + \alpha(R_{t+1} + \gamma \max_{a'} q(s',a')) \tag{3.3}$$

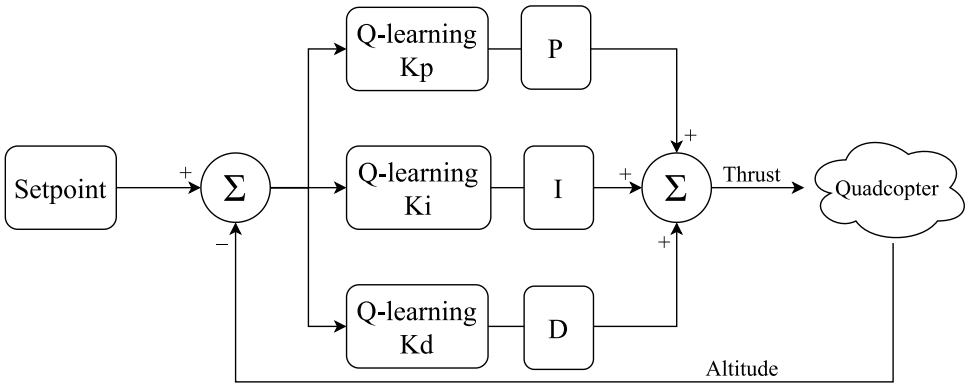


Figure 3.1: How Q-learning will fit in the standard PID controller loop. [3].

3.1. Problem Formulation

To adapt equation (1.10) to the problem of quadcopter altitude stabilization, values for parameters ϵ, γ, and α need to be defined. As the agent is learning, the ϵ is reduced from 1.0 to 0.001 by 0.001 each time step to give the agent the ability to explore and then exploit what has been learned at a later stage. To show that the agent does not only care about the immediate rewards but also the future ones, a choice of $\gamma = 0.99$ has been used. By the same token, to make it a little slower for the agent to replace the Q-value in the Q-table with the new value, the learning rate α has been chosen as 0.1. The Q-table used is a 100×3 matrix. The rows represent the states that the agent might find itself in, while the columns represent the three actions. The cells of the table correspond to the state-action pairs that are computed by equation (3.3).

$$Q_t = \begin{bmatrix} q_{s_1,a_1} & q_{s_1,a_2} & q_{s_1,a_3} \\ q_{s_2,a_1} & q_{s_2,a_2} & q_{s_2,a_3} \\ \vdots & \vdots & \vdots \\ q_{s_{100},a_1} & q_{s_{100},a_2} & q_{s_{100},a_3} \end{bmatrix} \tag{3.4}$$

At time $t = 0$, the agent does not know anything about which action to take as the learning process just started. Therefore, Q-table is initialized to 0.

$$Q_0 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \end{bmatrix} \tag{3.5}$$

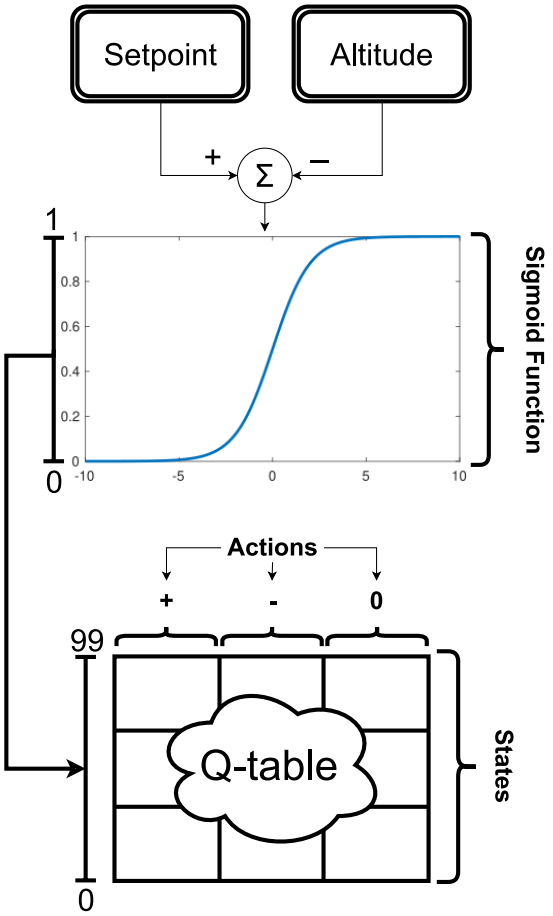


Figure 3.2: The use of Sigmoid function in the proposed algorithm [3].

The use of the Sigmoid function is to overcome the limitation of the Q-table, and to make the algorithm generalize to different setpoints or quadcopter settings without modifying the algorithm or its parameters. As demonstrated by Figure 3.2, the quadcopter senses the environment for the altitude. The sensed altitude is the distance between the quadcopter and the ground. The difference between the setpoint and altitude will be passed to the Sigmoid function to produce a value between 0 and 1. The produced value will be multiplied by 100 and then converted to an integer. Multiplication by 100 and taking the integer will allow the algorithm to get the row index that is 0 to 99 (100 rows). The columns on the other hand represent the actions, and also to generalize the algorithm, only three general actions are taken. Actions are either positive, that is, increase throttle, or negative which means the opposite, decreasing the throttle or lastly stay neutral.

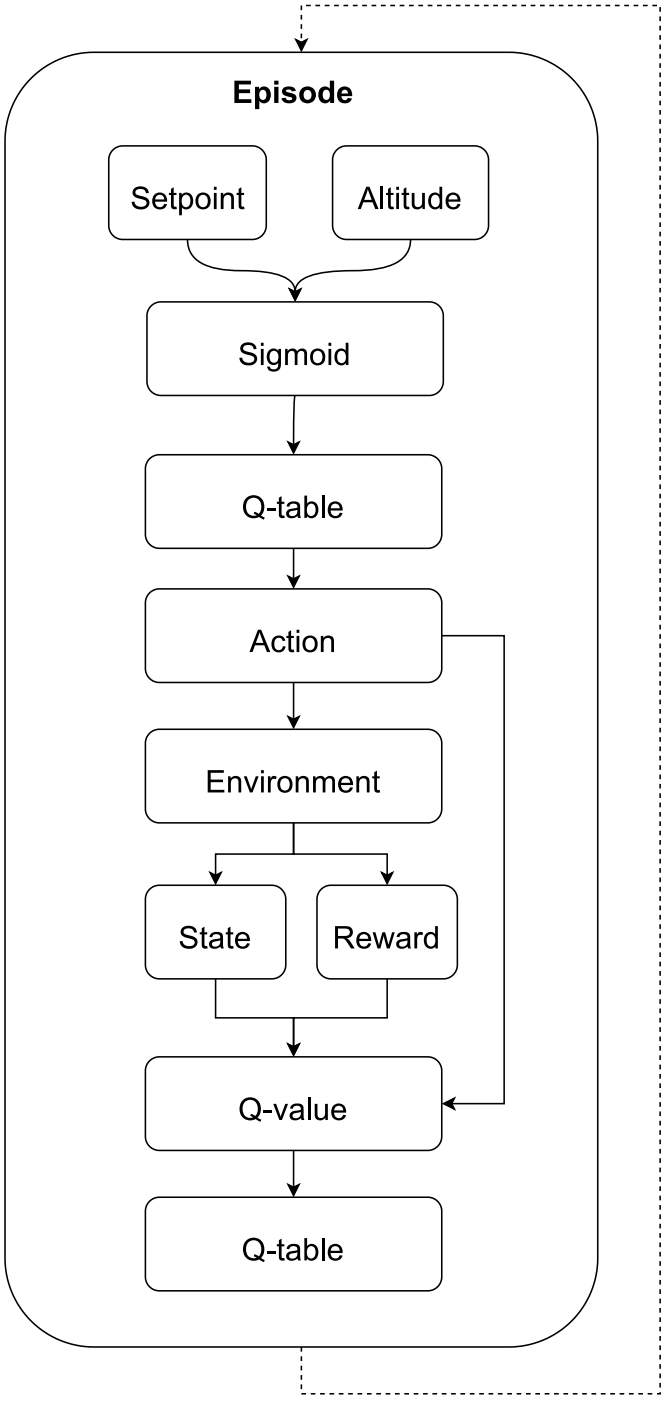


Figure 3.3: An episode of the proposed algorithm [3].

In every episode as illustrated in Figure 3.3, the quadcopter senses the environment for the altitude. Then, the altitude is passed to the Sigmoid function to locate the row in which the Q-value will be placed in the Q-table. Based on the action taken and the state of the previous step and the reward, the Q-value is calculated using the Equation (3.3).

In the classical approach of tuning the PID controller using the trial and error approach, the parameters are set to zero, and then start tuning one by one. That is, starting with proportional part K_p, then moving to the other two parameters, namely K_i and K_d [40]. Unlike the classical approach, the RL agent is given the ability to adjust all three K_p, K_i, K_d, in the same or different order as it sees fit to make the quadcopter reach the required setpoint altitude and continue adapting for new circumstances and conditions. Also, initializing the PID controller parameters to 0.1 instead of zero helps the tuning process to be faster, because setting PID parameters to zero, will make the propellers thrust equal to zero which will take the agent longer time to see any reduction on the error returned by Algorithm 3.2.

3.2. Experiment Setup

The experiments have been done in a simulated environment that has a physics engine. Furthermore, the time reported in the experiments is a simulation time. The quadcopter has been designed to embed the real-world physical properties like mass and materials of each component, see Figure 3.4. The physical properties that have been used are mass, gravity, drag, angular drag, and colliders. Since the algorithm is indeterministic in nature, each experiment illustrated in this study has been executed 100 times and an average has been reported in tables (4.1, 4.2), along with the upper and lower bounds, first, second, and third quartiles. In addition, the speed at which the PID controller parameters are learned by the agent and reach its setpoint has been outlined in Figure 4.2. The starting condition of the quadcopter is on the ground plane at x=0, y=0 and z=0 with the thrust set to zero for all the motors. In addition, the stopping criteria is set to an error of $\pm 0.1\%$ to the target altitude.

Mass Variation

In this variation, the aim is to examine the effect of changing the mass on the speed of convergence of the algorithm. Drag and angular drag properties have been fixed to 1N and 0.05N respectively, and only mass is changing. For every 100 experiments, an addition of 500 grams is added to the mass of the quadcopter. The experiment is terminated at 500 experiments which are equivalent to the addition of 2kg in total.

Drag Variation

In this variation, the objective is to examine the effect of changing the drag on the speed of convergence of the algorithm. Mass and angular drag properties have been fixed and only mass is changing. For every 100 experiments, an additional force of 0.5N is added to the drag that affects the quadcopter. The experiment is terminated at 500 trials which is equivalent to an addition of 5N of drag in total.

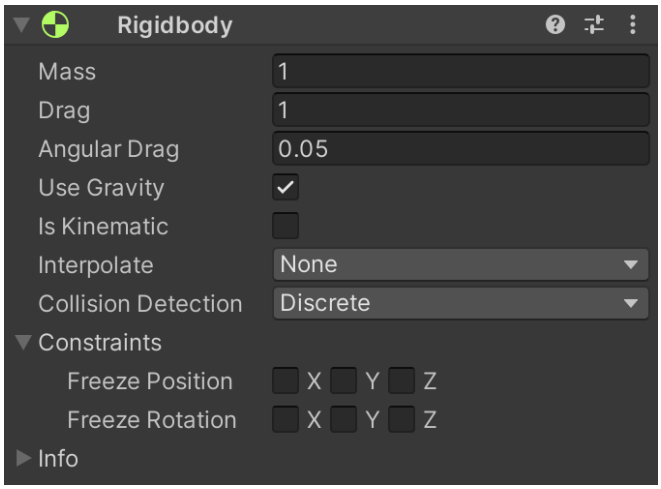
Disturbance Variation

As can be seen in Figure 3.5, random rotation (Figure 3.5a) and push (Figure 3.5b) forces have been applied around the quadcopter during the learning phase. The objective is to examine the effect of introducing disturbances to the quadcopter and investigate how that might affect the outcome of the algorithm. The torque (Newton-metre) is added to the pitch, yaw, and roll angles by 100 Newton-metres. The torque is applied every 1.5 seconds to all of the angles sequentially, that is roll, pitch then yaw, then applied all at once, and the procedure is repeated for the whole learning process. Please, refer to Appendix B.2 to see the actual code for the added disturbances.

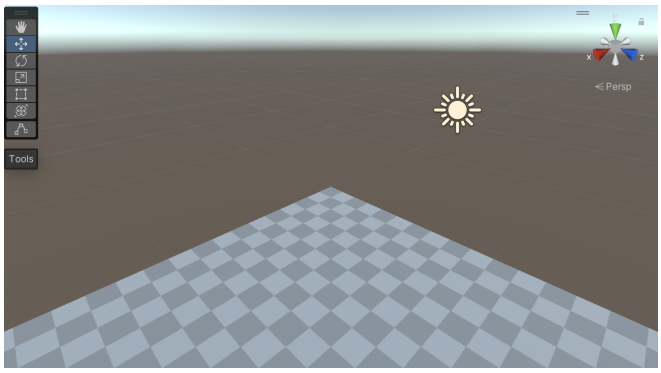
Setpoint Variation

In this variation, the objective is to examine the effect of changing the setpoint through the learning phase and investigate how that might affect the outcome of the algorithm. The setpoint has been set to 5m, 7m, and 9m.

(a)

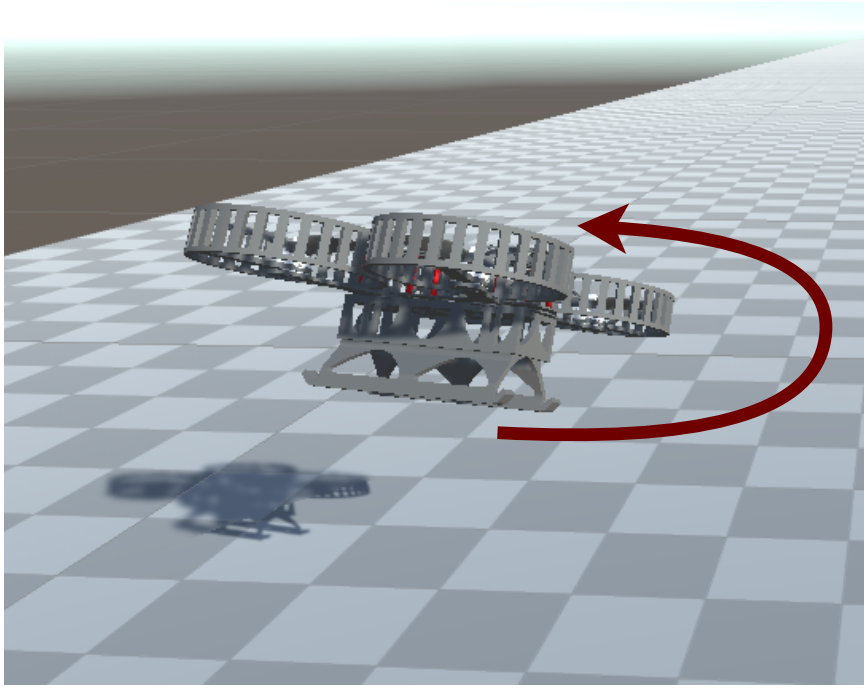


(b)

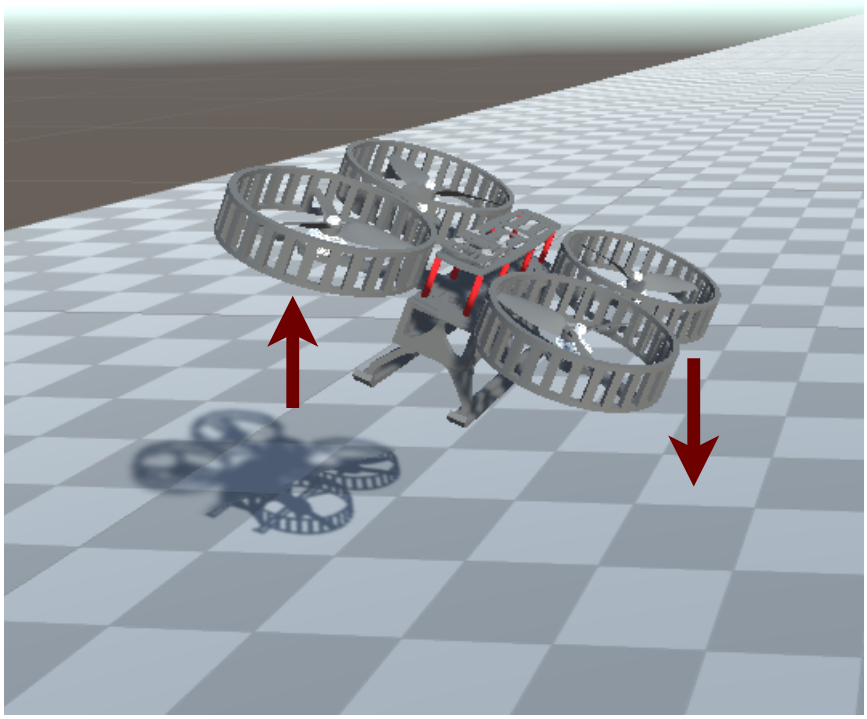


(c)

Figure 3.4: (a) 3D model of quadcopter (b) Simulation settings [Mass, Drag] (c) Simulation viewport of a ground.



(a)



(b)

Figure 3.5: (a) Apply random rotational force. (b) Apply random force pushing on different angles around the quadcopter.

4 | Results and Discussion

In Table 4.1 is reported the result of an experiment that consists of 500 trials, that is, 100 trials for each mass variation. A trial consists of the period taken by the algorithm to converge and tune the PID controller. Further, there are five mass variations, one to three kilograms as the quadcopter mass. Based on the results shown in the table, the median has increased by 10 seconds for every half a kilogram added. This is because when the mass increased, the algorithm takes more time to add force to the motors to tune the PID controller to reach the setpoint altitude. That is, the change does not happen immediately after a change in one of the PID controller gains, therefore, more time is needed for the agent to learn the policy required to reach the stability condition.

Table 4.1: Descriptive statistics for learning time in seconds (variable = Mass)

Variable*	**Mean**	**Min**	**Max**	**25th Percentile**	**Median**	**75th Percentile**
1.0 kg	76.1	14.1	266.9	32.9	53.8	114.2
1.5 kg	96.2	17.8	437.3	39.1	67.7	127.2
2.0 kg	102.1	15.0	489.6	41.2	75.1	114.0
2.5 kg	100.6	10.5	754.0	40.3	66.5	110.2
3.0 kg	115.1	22.9	340.0	50.9	91.1	159.3

*Note: 100 trials are taken for each change in the variable value.

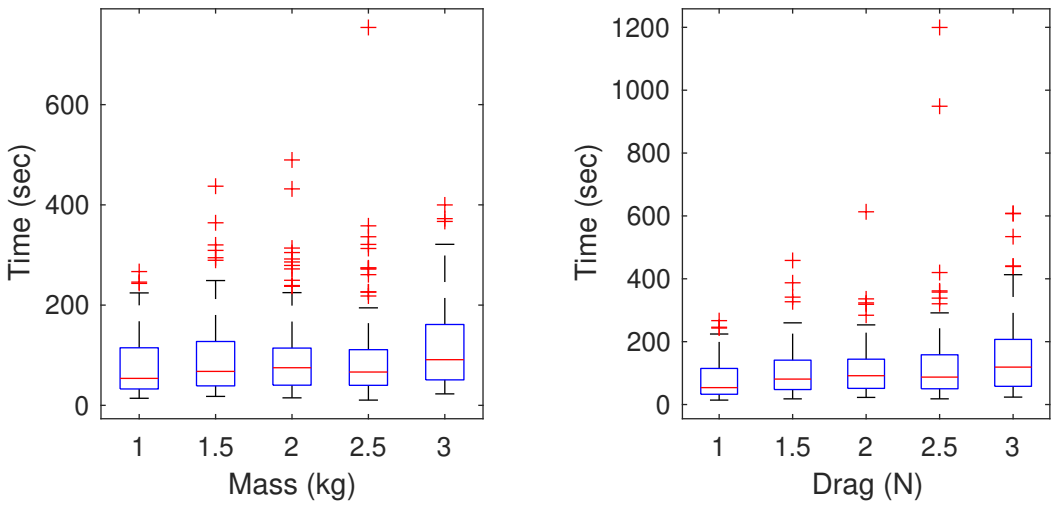
The same goes for Table 4.2 that represents applying drag variation on the quadcopter. Drag simulates a force acting opposite to the relative motion of the quadcopter. As the drag increased the time to converge also increased. This experiment added drag of 5 different magnitudes, starting from 1N to 3N by adding 0.5N at every drag experiment variation. The cause of increased time when drag increases time to convergence is due to the agent needing more time to learn the correct policy that leads to correct tuning of the PID controller. The change to one of the gains does not result in an immediate outcome, instead, the system needs more time to figure out the correct sequence of actions that results in a higher reward, that is, reaching stability at the setpoint altitude.

Table 4.2: Descriptive statistics for for learning time in seconds (variable = Drag)

Variable*	**Mean**	**Min**	**Max**	**25th Percentile**	**Median**	**75th Percentile**
1.0 N	76.1	14.1	266.9	32.9	53.8	114.2
1.5 N	106.9	18.1	458.4	47.9	81.0	140.9
2.0 N	113.2	22.7	613.2	51.4	91.7	142.2
2.5 N	130.3	18.2	1199.6	50.3	87.3	157.6
3.0 N	154.6	23.6	608.4	58.6	119.1	206.9

*Note: 100 trials are taken for each change in the variable values.

In Figure 4.1 a box plot is shown to visualize the data points; the outliers which are represented by a red cross (+). Either the mass or drag variations can be seen on the abscissas and it can be observed that the outliers are randomly distributed. Starting with the mass variation in Figure 4.1a, at one kilogram, the outliers are around the maximum value of the box plot which is around 250 seconds which is the time took the algorithm to converge and find the PID controller gains. Thereafter, the outliers start to drift away from the maximum of the box plot, but when the value of the quadcopter's mass reached 3 kilograms the outliers are back to around the maximum, which is something expected from a non-deterministic algorithm like the one proposed. Also, Figure 4.1b represents the same phenomenon, that is, the randomness of convergence times concerning the outliers.



(a) (b)

Figure 4.1: (a) Box plot for each increase in mass on the horizontal axis, and corresponding time of convergence on the vertical axis. (b) Box plot that represents the change in drag on the horizontal axis, with the time of convergence along the vertical axis.

Another experiment has been done to measure the effect of changing the setpoint on the number of episodes the proposed algorithm needs to converge. The result of the experiment consists of 300 trials, that is, 50 trials for each setpoint variation. Mass is kept at 1kg, 1N for the drag. In Table 4.3 it can be seen that randomness again shows up as expected, from 5 meters setpoint altitude the number of episodes was under 900 and again on 9 meters. On the other hand, for 6 and 10 meters was around 1500 episodes and 1000 episodes in the case of 7 meters. Furthermore, at 8 meters mark, the algorithm needed over 2500 episodes to tune the PID controller altitude gains.

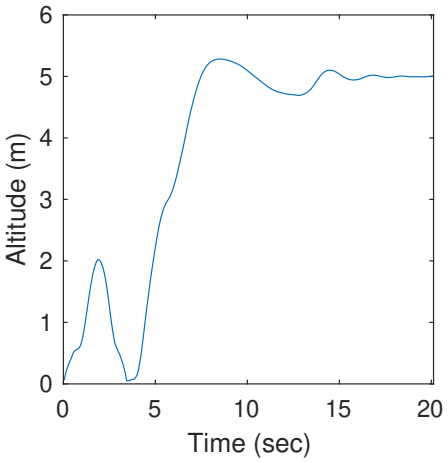
Table 4.3: Episodes needed to tune the PID controller given the altitude.

Altitude (m)	**Episodes**
5	867
6	1632
7	1001
8	2644
9	812
10	1800

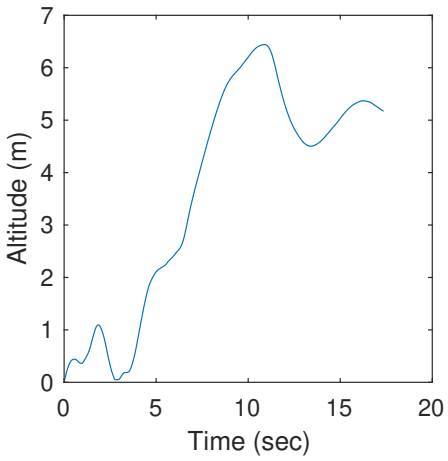
First, Figure 4.2a represents the phase where the agent fills up the Q-table and tries different values to make the quadcopter stable at the setpoint altitude. From 0 (ground) to just before 5 seconds, the quadcopter reaches 2 meters and falls back to 0, then rises back again without returning to 0. That also happens in Figure 4.2e but this time it reaches just before 5 meters to go back again to 0. On the other hand, the experiment results shown in Figure 4.2c show that the quadcopter stays around 0 then jumps to around 4 meters to go back to just above 3 meters and then up to 5 meters where it settles. In the experiment shown in Figures 4.2b, 4.2d, and 4.2f, the chosen setpoints for altitude were picked to be 5, 7, and 9 meters. Figures 4.2b, 4.2d, and 4.2f show the training phase for different altitudes, 5m, 7m, 9m respectively. Both sides of Figure 4.2 were plotted with time steps on the horizontal axis and the altitude on the vertical axis.

From an initial initialization of the PID controller parameters to a fully tuned PID controller in under 1.5 minutes on average. The box plots seen in Figures 4.1a, 4.1b with 100 trials for each variation in mass and drag, show that a minimum of 14 seconds is needed to reach the required setpoint altitude. Furthermore, the quadcopter reaches the setpoint quickly as shown in Figures 4.2a, 4.2c, and 4.2e as the agent tuned the PID controller parameters in under 30 seconds. It is worth noting that the 75% percentile, are ranging from 2 to 4 minutes across the thousand trials.

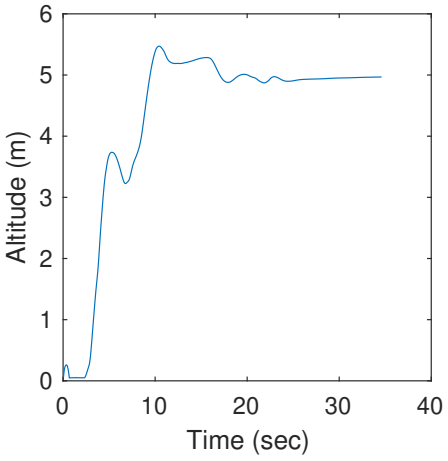
On the other hand, due to the non-deterministic nature of the RL algorithms, more experiments are needed as the results are far from perfect. As seen from the handpicked experiments in Figure 4.2 the algorithm needs to optimize the learned PID controller parameters. It can also be seen in Tables 4.1-4.2 that increasing the mass or drag will increase the convergence time to reach the setpoint. It is easily inferred from Tables 4.1-4.2 that the average convergence time increases whenever mass or drag increases on the system.



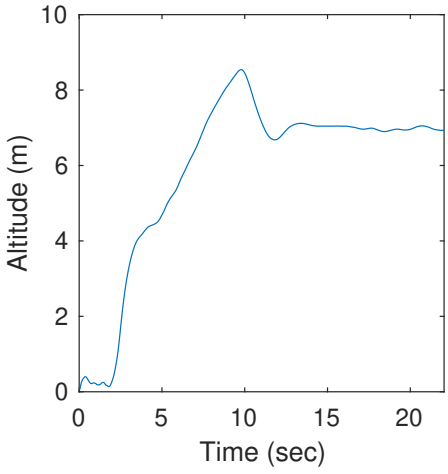
(a) Setpoint Altitude: 5m



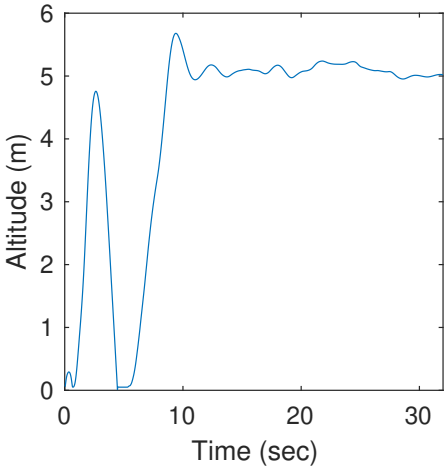
(b) Setpoint Altitude: 5m



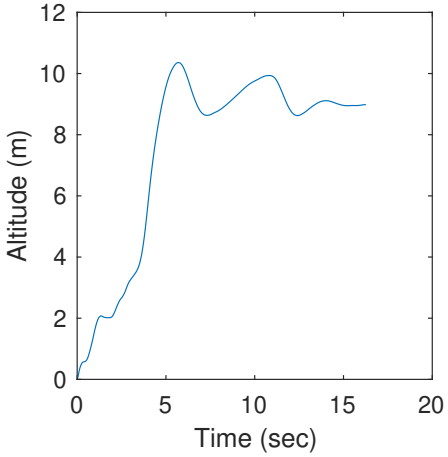
(c) Setpoint Altitude: 5m



(d) Setpoint Altitude: 7m



(e) Setpoint Altitude: 5m



(f) Setpoint Altitude: 9m

Figure 4.2: Handpicked experiments to show the nondeterministic behavior to be expected from such an algorithm. On the other hand, all converge to the desired setpoint altitude.

5 | Conclusions

In this thesis, a reinforcement learning, model-free, $\epsilon - greedy$ policy algorithm is implemented through the use of Q-learning with a modification of the Q-table to make it appropriate for quadcopters to reach altitude stability in less than half a minute on average as evident from experiments presented. These results are achievable without the need for extensive simulations and modeling of the environment. Although minimal proper knowledge of PID controller and the application at hand is needed to aid the algorithm to reach desired altitude faster and smoother. Hence, making it applicable to a wide range of quadcopter sizes and weights. Furthermore, the algorithm outperforms the latest work found in the literature on the same problem that is discussed in Chapter 2 by [36] where our Q-table has a smaller memory footprint, 500 fewer cells with 300 cells only. Furthermore, the algorithm is built to be adaptive in terms of motors used by the use of general actions, increment, decrement, or staying neutral. Unlike the work of [36] where the actions depend on the PWM of the motors. Also, based on the empirical data presented, the algorithm reaches a setpoint altitude at a thousand epsiods on average, unlike the work that has been done by [36] where the number of episodes always exceeds 2000.

5.1. Future Developments

The experimental validation of the proposed adaptive controller is the subject of a future study. Also, further investigation of other RL algorithms could be considered for use with PID controller. Algorithms like Deep Q-learning that uses a deep neural network to replace the Q-table [63]. In addition, optimization of the algorithm used in this thesis could result in more stable and robust results. Only the altitude stabilization has been investigated in this thesis, but the pitch, yaw, and roll of the quadcopter were tuned using a trial and error approach. Therefore, future investigation is necessary to have a solution to replace the class PID controller with an intelligent adaptive solution. Furthermore, an investigation of Micro Areia Vehicle (MAV) with the presented algorithm should be tested as micro quadcopters tend to be sensitive to disturbances [49]. Numerical simulations were performed and demonstrated the effectiveness of the proposed method.

Bibliography

[1] S. Abdelhay and A. Zakriti. Modeling of a quadcopter trajectory tracking system using pid controller. *Procedia Manufacturing*, 32:564–571, 2019.

[2] A. Alkamachi and E. Erçelebi. Modelling and genetic algorithm based-pid control of h-shaped racing quadcopter. *Arabian Journal for Science and Engineering*, 42(7): 2777–2786, 2017.

[3] Y. Alrubyli and A. Bonarini. Using q-learning to automatically tune quadcopter pid controller online for fast altitude stabilization. In *2022 IEEE International Conference on Mechatronics and Automation (ICMA)*, pages 514–519, 2022. doi: 10.1109/ICMA54519.2022.9856292.

[4] W. An, H. Wang, Q. Sun, J. Xu, Q. Dai, and L. Zhang. A pid controller approach for stochastic optimization of deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8522–8531, 2018.

[5] B. Anderson, B. Fidan, C. Yu, and D. Walle. Uav formation control: Theory and application. In *Recent advances in learning and control*, pages 15–33. Springer, 2008.

[6] ARCELI. Hrb lipo battery, 2017. URL `https://www.amazon.it/gp/product/B072KH76JN`.

[7] A. T. Azar, A. Koubaa, N. Ali Mohamed, H. A. Ibrahim, Z. F. Ibrahim, M. Kazim, A. Ammar, B. Benjdira, A. M. Khamis, I. A. Hameed, et al. Drone deep reinforcement learning: A review. *Electronics*, 10(9):999, 2021.

[8] N. Bernini, M. Bessa, R. Delmas, A. Gold, E. Goubault, R. Pennec, S. Putot, and F. Sillion. A few lessons learned in reinforcement learning for quadcopter attitude control. In *Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control*, pages 1–11, 2021.

[9] R. P. Borase, D. Maghade, S. Sondkar, and S. Pawar. A review of pid control, tuning methods and applications. *International Journal of Dynamics and Control*, 9(2): 818–827, 2021.

[10] H. Boubertakh, M. Tadjine, P.-Y. Glorennec, and S. Labiod. Tuning fuzzy pd and pi controllers using reinforcement learning. *ISA transactions*, 49(4):543–551, 2010.

[11] I. Carlucho, M. De Paula, S. A. Villar, and G. G. Acosta. Incremental q-learning strategy for adaptive pid control of mobile robots. *Expert Systems with Applications*, 80:183–199, 2017.

[12] I. Carlucho, M. De Paula, and G. G. Acosta. An adaptive deep reinforcement learning approach for mimo pid control of mobile robots. *ISA transactions*, 102:280–294, 2020.

[13] M. S. Chehadeh and I. Boiko. Design of rules for in-flight non-parametric tuning of pid controllers for unmanned aerial vehicles. *Journal of the Franklin Institute*, 356 (1):474–491, 2019.

[14] J. J. Chiew and M. J. Aftosmis. Efficient simulation of multi-rotor vehicles with low reynolds number propellers. In *2018 Applied Aerodynamics Conference*, page 4119, 2018.

[15] DFRobot. Df srf02 ultrasonic sensor, 2021. URL `https://www.amazon.it/gp/product/B00BIVDAM0`.

[16] O. Dogru, K. Velswamy, F. Ibrahim, Y. Wu, A. S. Sundaramoorthy, B. Huang, S. Xu, M. Nixon, and N. Bell. Reinforcement learning approach to autonomous pid tuning. *Computers & Chemical Engineering*, 161:107760, 2022.

[17] P. Doherty and P. Rudol. A uav search and rescue scenario with human body detection and geolocalization. In *Australasian Joint Conference on Artificial Intelligence*, pages 1–13. Springer, 2007.

[18] K. El Hamidi, M. Mjahed, A. El Kari, and H. Ayad. Neural network and fuzzy-logic-based self-tuning pid control for quadcopter path tracking. *Stud. Inform. Control*, 28(4):401–412, 2019.

[19] EMAX. Rs2205-s 2300kv brushless cw motors, 2017. URL `https://www.amazon.it/gp/product/B06XPRX4YV`.

[20] M. Euston, P. Coote, R. Mahony, J. Kim, and T. Hamel. A complementary filter for attitude estimation of a fixed-wing uav. In *2008 IEEE/RSJ international conference on intelligent robots and systems*, pages 340–345. IEEE, 2008.

[21] P. G. Fahlstrom, T. J. Gleason, and M. H. Sadraey. *Introduction to UAV systems*. John Wiley & Sons, 2022.

[22] C. Flener, M. Vaaja, A. Jaakkola, A. Krooks, H. Kaartinen, A. Kukko, E. Kasvi,

H. Hyyppä, J. Hyyppä, and P. Alho. Seamless mapping of river channels at high resolution using mobile lidar and uav-photography. *Remote Sensing*, 5(12):6382–6407, 2013.

[23] U. E. Franke. The global diffusion of unmanned aerial vehicles (uavs) or 'drones. *Precision Strike Warfare and International Intervention: Strategic, Ethico-Legal and Decisional Implications*, page 27109, 2014.

[24] S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[25] H. Gao, C. Liu, D. Guo, and J. Liu. Fuzzy adaptive pd control for quadrotor helicopter. In *2015 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)*, pages 281–286. IEEE, 2015.

[26] Gemfan. 5055 3-blade propeller 5inch, 2018. URL `https://www.amazon.it/gp/product/B0792Q74RT`.

[27] M. A. Goodrich, B. S. Morse, D. Gerhardt, J. L. Cooper, M. Quigley, J. A. Adams, and C. Humphrey. Supporting wilderness search and rescue using a camera-equipped mini uav. *Journal of Field Robotics*, 25(1-2):89–110, 2008.

[28] J. Gu, T. Su, Q. Wang, X. Du, and M. Guizani. Multiple moving targets surveillance based on a cooperative network for multi-uav. *IEEE Communications Magazine*, 56 (4):82–89, 2018.

[29] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[30] HRB. Mpu6050 inertial measurement unit, 2018. URL `https://www.amazon.it/ARCELI-giroscopio-accelerometro-Accelerometer-Convertitore/dp/B07BVXN2GP`.

[31] S. Islam, P. X. Liu, and A. El Saddik. Robust control of four-rotor unmanned aerial vehicle with disturbance uncertainty. *IEEE Transactions on Industrial Electronics*, 62(3):1563–1571, 2014.

[32] N. Iversen, O. B. Schofield, L. Cousin, N. Ayoub, G. Vom Bögel, and E. Ebeid. Design, integration and implementation of an intelligent and self-recharging drone system for autonomous power line inspection. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4168–4175. IEEE, 2021.

[33] F. Jiang, F. Pourpanah, and Q. Hao. Design, implementation, and evaluation of a neural-network-based quadcopter uav system. *IEEE Transactions on Industrial Electronics*, 67(3):2076–2085, 2019.

[34] M. A. Johnson and M. H. Moradi. *PID control*. Springer, 2005.

[35] M. I. Jordan and T. M. Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

[36] P. Karthik, K. Kumar, V. Fernandes, and K. Arya. Reinforcement learning for altitude hold and path planning in a quadcopter. In *2020 6th International Conference on Control, Automation and Robotics (ICCAR)*, pages 463–467. IEEE, 2020.

[37] J.-S. Kim, J.-H. Kim, J.-M. Park, S.-M. Park, W.-Y. Choe, and H. Heo. Auto tuning pid controller based on improved genetic algorithm for reverse osmosis plant. *World Academy of Science, Engineering and Technology*, 47(2):384–389, 2008.

[38] P. Klaer, A. Huang, P. Sévigny, S. Rajan, S. Pant, P. Patnaik, and B. Balaji. An investigation of rotary drone herm line spectrum under manoeuvering conditions. *Sensors*, 20(20):5940, 2020.

[39] E. Kuantama, T. Vesselenyi, S. Dzitac, and R. Tarca. Pid and fuzzy-pid control model for quadcopter attitude with disturbance parameter. *International journal of computers communications & control*, 12(4):519–532, 2017.

[40] N. Kuyvenhoven. Pid tuning methods an automatic pid tuning study with mathcad. *Calvin college ENGR*, 315, 2002.

[41] K. B. Laksham. Unmanned aerial vehicle (drones) in public health: A swot analysis. *Journal of family medicine and primary care*, 8(2):342, 2019.

[42] K. S. Lee, M. Ovinis, T. Nagarajan, R. Seulin, and O. Morel. Autonomous patrol and surveillance system using unmanned aerial vehicles. In *2015 IEEE 15th International Conference on Environment and Electrical Engineering (EEEIC)*, pages 1291–1297. IEEE, 2015.

[43] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[44] R. Mao, B. Du, D. Sun, and N. Kong. Optimizing a uav-based emergency medical service network for trauma injury patients. In *2019 IEEE 15th International*

Conference on Automation Science and Engineering (CASE), pages 721–726. IEEE, 2019.

[45] N. Metni and T. Hamel. A uav for bridge inspection: Visual servoing control law with orientation limits. *Automation in construction*, 17(1):3–10, 2007.

[46] K. Muderrisoğlu, D. O. Arisoy, A. O. Ahan, and E. Akdogan. Pid parameters prediction using neural network for a linear quarter car suspension control. *International Journal of intelligent systems and applications in engineering*, 4(1):20–24, 2016.

[47] J. Nikolic, M. Burri, J. Rehder, S. Leutenegger, C. Huerzeler, and R. Siegwart. A uav system for inspection of industrial facilities. In *2013 IEEE Aerospace Conference*, pages 1–8. IEEE, 2013.

[48] N. S. Nise. *Control systems engineering*. John Wiley & Sons, 2020.

[49] A. Noordin, M. Basri, and Z. Mohamed. Simulation and experimental study on pid control of a quadrotor mav with perturbation. *Bulletin of Electrical Engineering and Informatics*, 9(5):1811–1818, 2020.

[50] NVIDIA. Jetson nano developer kit, b01, 2020. URL `https://developer.nvidia.com/embedded/jetson-nano-developer-kit`.

[51] P. Parihar, P. Bhawsar, and P. Hargod. Design & development analysis of quadcopter. *Compusoft*, 5(6):2128, 2016.

[52] R. Parvathy and R. R. Devi. Gradient descent based linear regression approach for modeling pid parameters. In *2014 International conference on power signals control and computations (EPSCICON)*, pages 1–4. IEEE, 2014.

[53] P. Poksawat, L. Wang, and A. Mohamed. Automatic tuning of attitude control system for fixed-wing unmanned aerial vehicles. *IET Control Theory & Applications*, 10(17):2233–2242, 2016.

[54] V. Praveen, S. Pillai, et al. Modeling and simulation of quadcopter using pid controller. *International Journal of Control Theory and Applications*, 9(15):7151–7158, 2016.

[55] A. Puente-Castro, D. Rivero, A. Pazos, and E. Fernandez-Blanco. A review of artificial intelligence applied to path planning in uav swarms. *Neural Computing and Applications*, pages 1–18, 2021.

[56] A. Salem, M. M. Hassan, and M. Ammar. Tuning pid controllers using artificial intel-

ligence techniques applied to dc-motor and avr system. *Asian Journal of Engineering and Technology*, 2(2), 2014.

[57] A. M. Samad, N. Kamarulzaman, M. A. Hamdani, T. A. Mastor, and K. A. Hashim. The potential of unmanned aerial vehicle (uav) for civilian and mapping application. In *2013 IEEE 3rd International Conference on System Engineering and Technology*, pages 313–318. IEEE, 2013.

[58] G. Scott, J. Shavlik, and W. Ray. Refining pid controllers using neural networks. *Advances in Neural Information Processing Systems*, 4, 1991.

[59] E. Semsch, M. Jakob, D. Pavlicek, and M. Pechoucek. Autonomous uav surveillance in complex urban environments. In *2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, volume 2, pages 82–85. IEEE, 2009.

[60] S. Sharma, S. Sharma, and A. Athaiya. Activation functions in neural networks. *towards data science*, 6(12):310–316, 2017.

[61] J.-C. Shen. Fuzzy neural networks for tuning pid controller for plants with underdamped responses. *IEEE Transactions on Fuzzy Systems*, 9(2):333–342, 2001.

[62] W. J. Shipman and L. C. Coetzee. Reinforcement learning and deep neural networks for pi controller tuning. *IFAC-PapersOnLine*, 52(14):111–116, 2019.

[63] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[64] T. Tomic, K. Schmid, P. Lutz, A. Domel, M. Kassecker, E. Mair, I. L. Grixa, F. Ruess, M. Suppa, and D. Burschka. Toward a fully autonomous uav: Research platform for indoor and outdoor urban search and rescue. *IEEE robotics & automation magazine*, 19(3):46–56, 2012.

[65] T. Wang, R. Qin, Y. Chen, H. Snoussi, and C. Choi. A reinforcement learning approach for uav target searching and tracking. *Multimedia Tools and Applications*, 78(4):4347–4364, 2019.

[66] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

[67] G. Welch, G. Bishop, et al. *An introduction to the Kalman filter*. Chapel Hill, NC, USA, 1995.

[68] XTVTX. 35a blheli-s brushless electronic speed controller esc, 2021. URL `https://www.amazon.it/gp/product/B019IK73A2`.

[69] P. Yedamale. Brushless dc (bldc) motor fundamentals. *Microchip Technology Inc*, 20 (1):3–15, 2003.

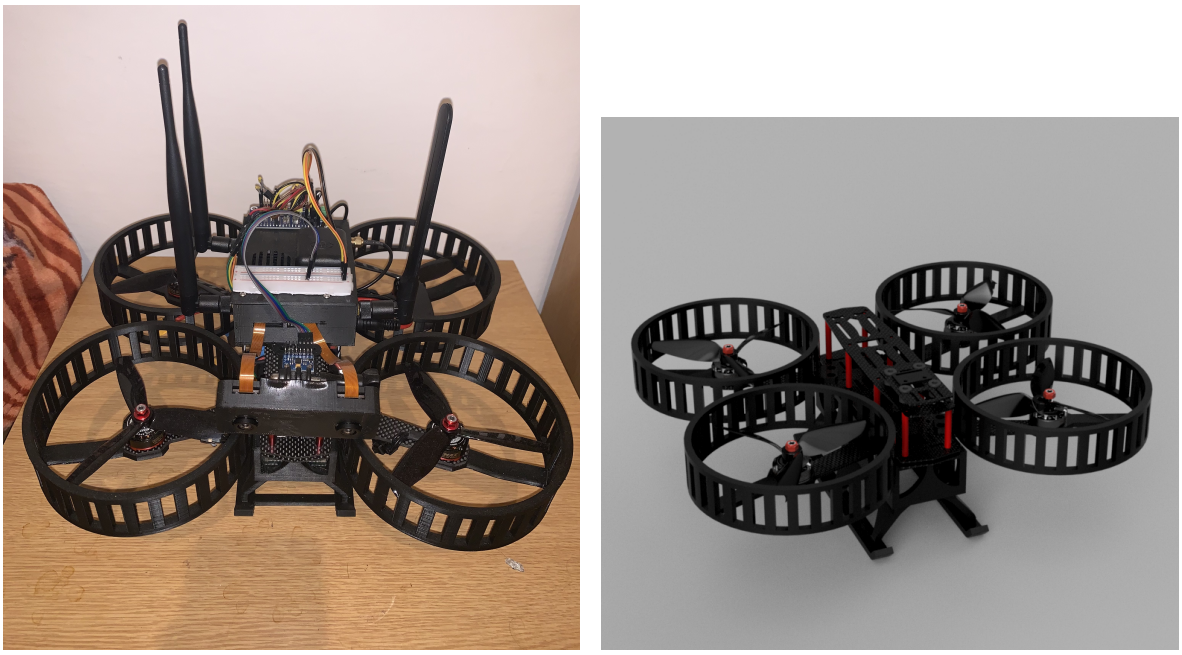
[70] J. Zhang, N. Wang, and S. Wang. A developed method of tuning pid controllers with fuzzy rules for integrating processes. In *Proceedings of the 2004 American Control Conference*, volume 2, pages 1109–1114. IEEE, 2004.

[71] Z. Zhang and M. Cong. Controlling quadrotors based on linear quadratic regulator. *Applied Science and Technology*, 5:38–42, 2011.

[72] ZHITING. Power distribution board for quadcopters, 2019. URL `https://www.amazon.it/gp/product/B08212TCF4`.

[73] ZMR250. Quadcopter 250mm frame from carbon fiber, 2015. URL `https://www.amazon.it/gp/product/B019IK73A2`.

[74] A. Zulu. Towards explicit pid control tuning using machine learning. In *2017 IEEE AFRICON*, pages 430–433. IEEE, 2017.



(a) (b)

Figure A.1: (a) Quadcopter (b) 3D model 1-to-1 to the quadcopter in **(a)** with embedded physical properties.

A | Appendix A: Quadcopter Assembly and Simulation

To have a result that can be reliable in a simulation, the exact quadcopter built using the components presented in this appendix has a 1-to-1 3D model with the exact dimensions. Also, every 3D model encodes the weight and physical operation, for example, for the motors and propellers. Furthermore, the 3D models have also encoded the material of the physical characteristics of the component. Also, the colors have been added to give it a 1-to-1 replica as shown in Figure A.1.

Figure A.2: ZMR250 Quadcopter Frame [73].

A.1. Off-the-shelf Components

Frame

The frame was chosen based on the durability, weight, size, price, and availability of the component in case of future damage. The frame is made of carbon fiber which gives it durability and lighter weight.

Table A.1: ZMR250 Quadcopter Frame Specifications [73].

Input voltage	Carbon Fiber (1.5mm)
Size	250mm
Weight	146g
Price	16.99 €

Figure A.3: EMAX RS2205-S 2300KV Brushless CW Motors [19].

Motors

A quadcopter is a multirotor vehicle with four motors placed in a certain configuration to make the quadcopter able to float and maneuver. A high-quality, low-weight motor and powerful for future research and matching the frame chosen is EMAX RS2205-S 2300KV brushless Clockwise (CW) motor. A Brushless motor is chosen over a brushed motor due to its high power-to-weight ratio, high speed, high efficiency, low maintenance, and nearly instantaneous control of speed (rpm) and torque [69].

Table A.2: EMAX RS2205-S 2300KV Brushless CW Motors Specifications [19].

Input voltage	3S-4S
KV	2300kv
Weight	30g
Maximum Thrust	1024g
Price	70.98 €

Figure A.4: Gemfan 5-inch 3-Blade Propeller [26].

Propellers

Propellers are attached to the motors of the quadcopter in cretin configuration, that is, Counterclockwise (CW) and Counterclockwise (CCW). These configurations make lifting and flying the quadcopter possible. These 3-blades provide more torque and higher maneuverability, but are less efficient, which in return means lower flight time [38].

Table A.3: Gemfan 5-inch 3-Blade Propeller Specifications [26].

Size	5 inch
Material	Plastic
Weight	4.90g
Other	4 CW - 4CCW
Price	8.99

Figure A.5: HRB LiPo Battery [30].

Battery

To power the quadcopter and all the electronics onboard, a battery is needed. LiPo 11.1V battery is selected to power the motors at their full power. Also, it has 6000mAh which is enough power to operate not only the motors but also the onboard electronics Although, increasing the pack size will give longer flight times however will add weight. Furthermore, the battery has a 100C burst rate which allows the quadcopter to accelerate faster. The battery comes in a compact size, small size, and weight. Continuous discharge rating is often what distinguishes batteries, it refers to how quickly a battery can discharge the energy it has and is often the limiting factor in a high-performance application, at 50C is enough for the use cases planned for the quadcopter.

Table A.4: HRB LiPo Battery Specifications [30].

Capacity	6000mAh
Voltage	11.1V
Cells	3S
Continuous Discharge	50C
Burst Rate	100C
Weight	420g
Price	63.99 €

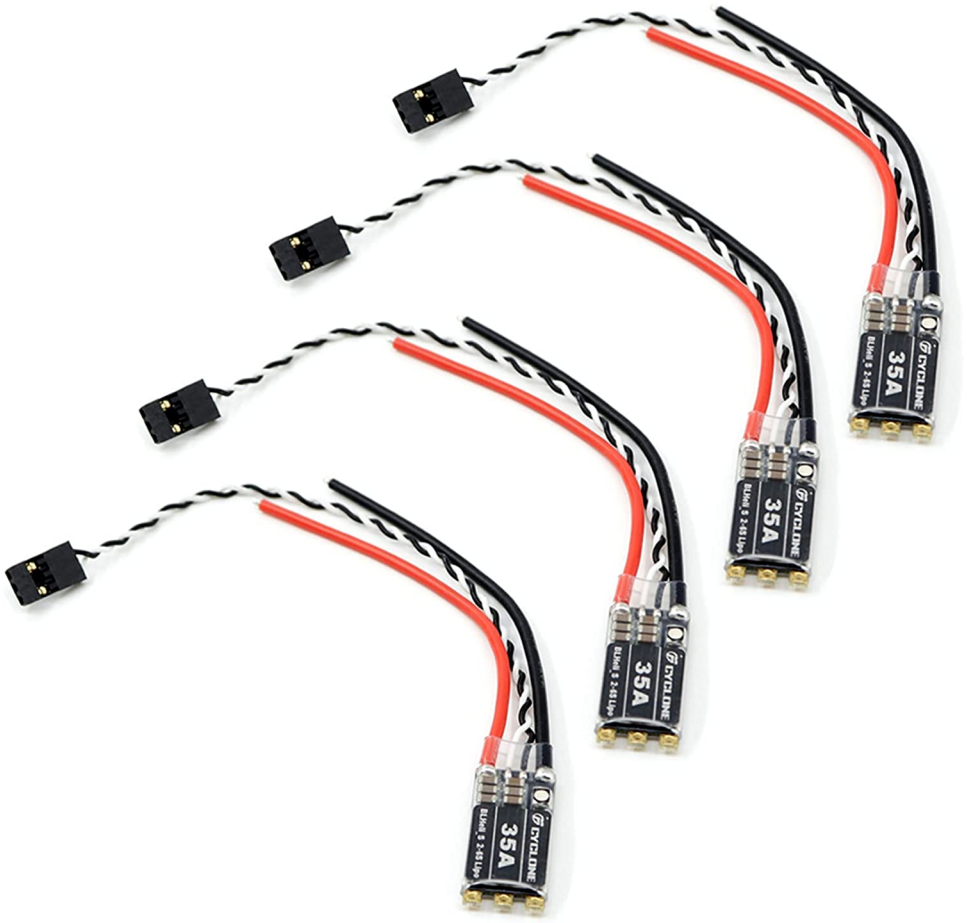


Figure A.6: XTVTX 35A BLHeli-S Electronic Speed Controller [68].

Electronic Speed Controller

Based on the choice of the battery and motors, XTVTX 35A BLHeli-S has been selected as the Electronic Speed Controller (ESC). ESC is used to regulate and control the speed of the motors [51]. 35A is enough for the motors and future-proof in case a stronger and faster motor becomes available.

Table A.5: XTVTX 35A BLHeli-S Electronic Speed Controller Specifications [68].

Voltage Support	2S-6S
Current	35A DC
Weight	6g
Price	89.99 €

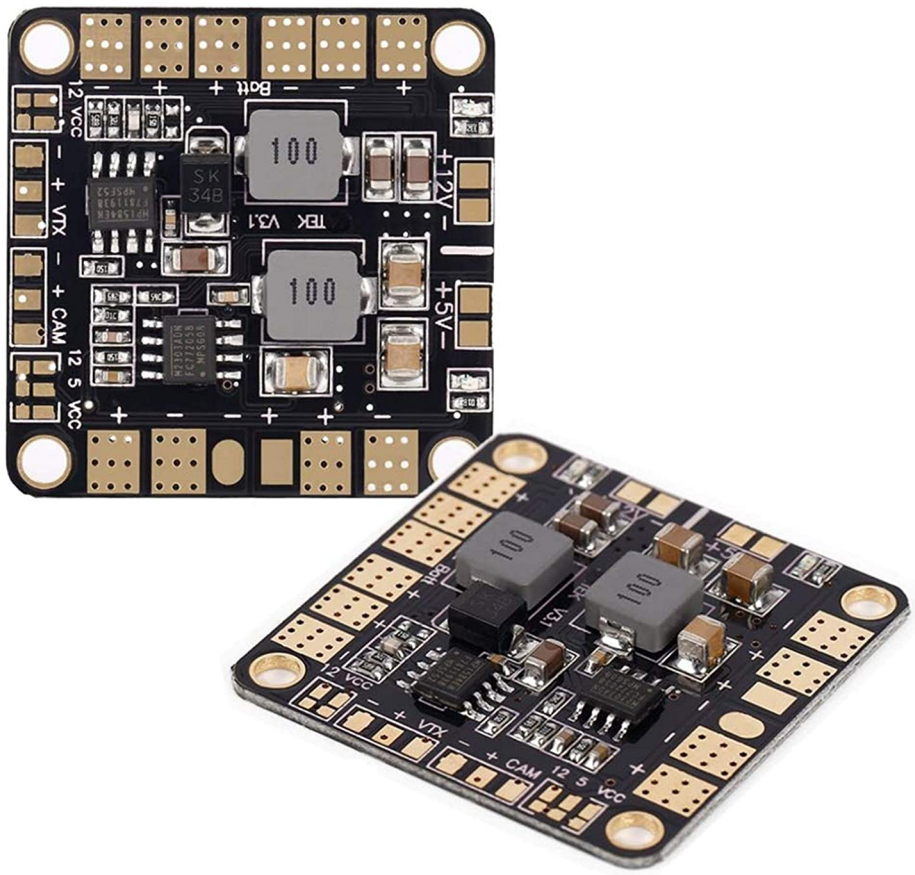


Figure A.7: ZHITING PDB [72].

Power Distribution Board

To distribute the power from the battery to the motors and other electronics onboard the quadcopter, a Power Distribution Board (PDB) is required [32]. Furthermore, different components need different voltage ratings and current flow. Also, a compact package is preferable. The specifications is shown in Table A.6.

Table A.6: ZHITING PDB Specifications [72].

Input Voltage	9V to 26V
Output Voltage	5V and 12V
ESC Output	4
Continuous Current	20A per output
Price	10.99 €

Figure A.8: Jetson Nano Developer Kit (Version: B01) [50].

Computer

The goal is to find a small compact computer to run the flight controller onboard the quadcopter, yet powerful enough for future research. The choice landed on Jetson Nano Developer Kit from NVIDIA. Specifically, version *B01* added a connector to support a dual camera setup for stereo vision. The specifications is shown in Table A.7.

Table A.7: Jetson Nano Developer Kit Specifications [50].

USB	(4x) USB 3.0 Type-A, USB 2.0 Micro-B
Camera	(2x) MIPI CSI-2 x2 (15-position Camera Flex Connector)
Display	HDMI 2.0, DisplayPort
Wireless	M.2 Key-E (PCIe x1)
Ethernet	Gigabit Ethernet (RJ45)
Storage	MicroSD card slot
Other	40-pin Header - (3x) I2C, (2x) SPI, UART, I2S, GPIOs
Power	Micro-USB (5V=2.5A) or DC barrel jack (5V=4A)
Price	400 €

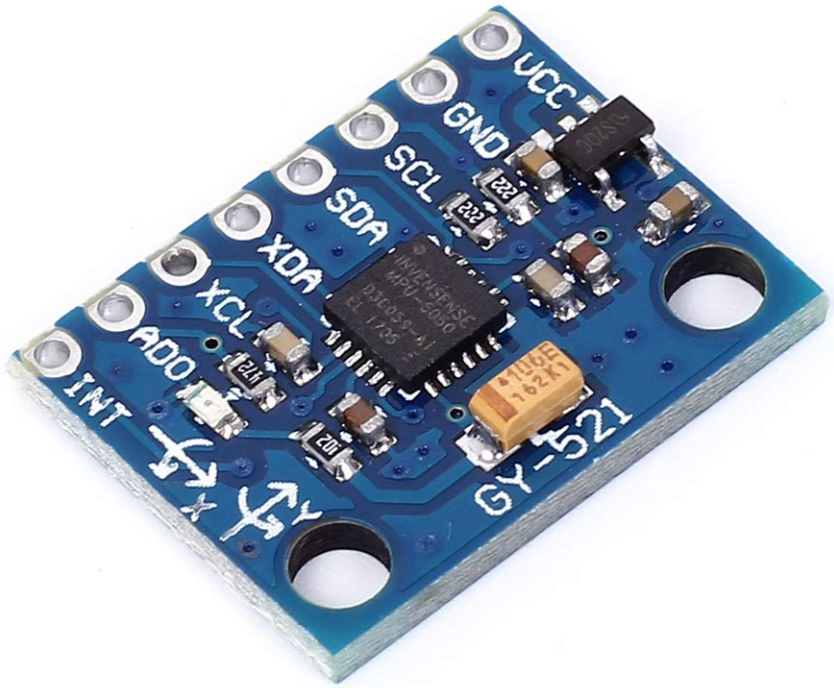


Figure A.9: MPU6050 Inertial Measurement Unit [6].

Inertial Measurement Unit

The Inertial Measurement Unit (IMU) is used to measure the pitch, yaw, and roll angles. These angles can help algorithms like PID controller to get the quadcopter to stabilize despite the existence of disturbances [54].

Table A.8: MPU6050 Inertial Measurement Unit Specifications [6].

Input Voltage	9V to 26V
Output Voltage	5V and 12V
ESC Output	4
Continuous Current	20A per output
Price	5.99 €

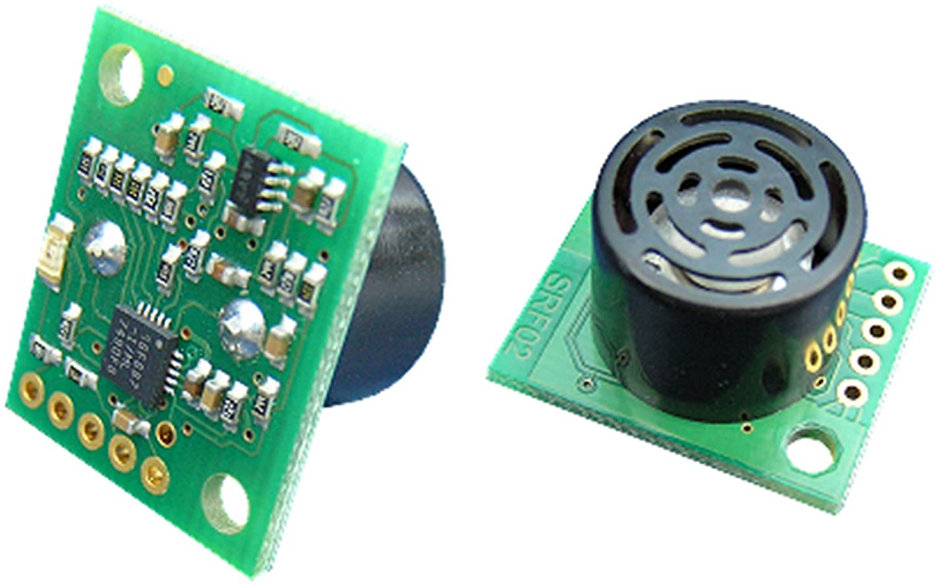


Figure A.10: DF SRF02 Ultrasonic Sensor [15].

Ultrasonic

For the quadcopter to sense how far it is from the ground, an ultrasonic can be used. The aim is to find a small compact ultrasonic sensor that can be attached to the bottom of the quadcopter to sense the distance to the ground. DF SRF02 ultrasonic sensor has been chosen for its quality, size, and price. The size competence is due to the ability to send and receive ultrasound single, these types of ultrasonic sensors are called transceivers. In addition, DF SRF02 uses the I2C bus for communication which is required for the underlying communication protocol used in the quadcopter between the sensors and actuators. Furthermore, two or more of these sensors can be attached at the bottom and fuse their measurements to reduce the effect of the noisy readings using complementary filter [20], or Kalman filter [67] to name a few.

Table A.9: DF SRF02 Ultrasonic Sensor Specifications [15].

Power	5V
Communication Interface	I2C - Serial
Range	16cm to 6m
Frequency	40KHz
Other	Full Automatic Tuning
Price	29.29 €

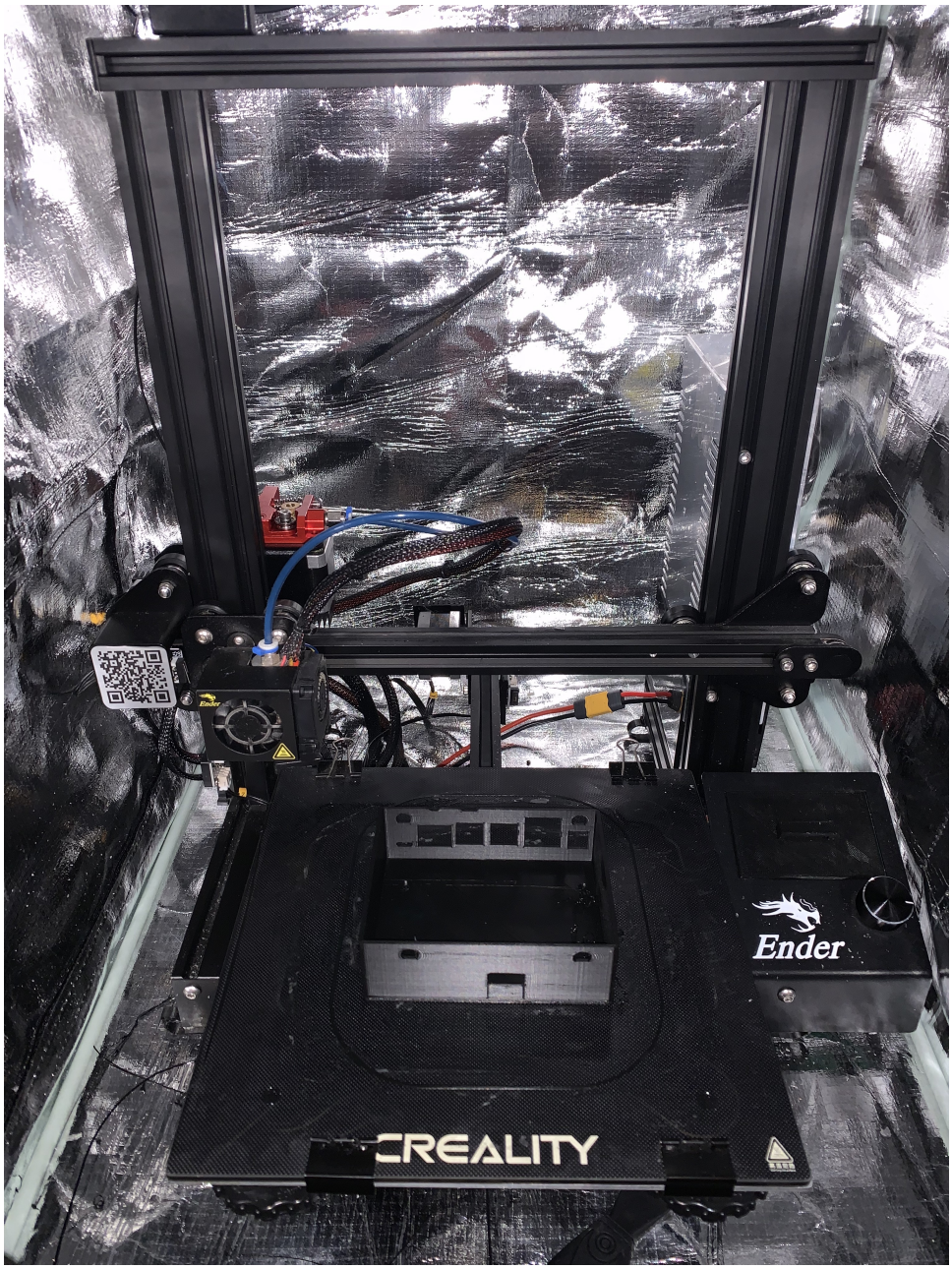
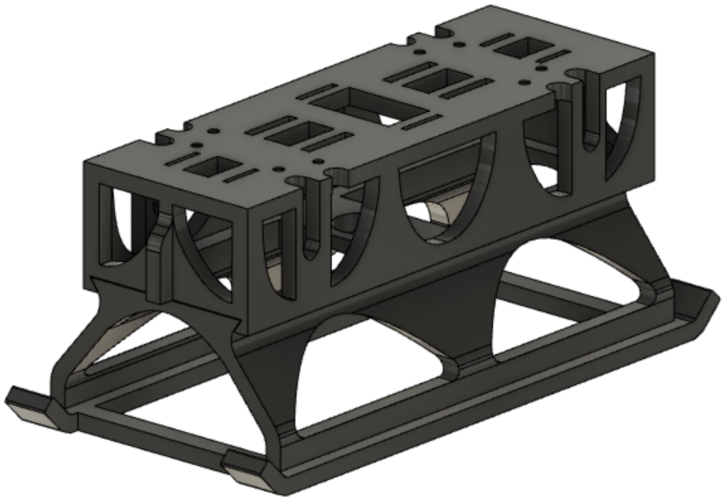


Figure A.11: 3D printer printing the computer protector.

A.2. 3D Printed Components

Some of the necessary parts needed to be designed to eliminate foreseen issues in the future. Issues like collisions and stable landing. A basic 3D printer is used with Polylactic Acid (PLA+) filament to print the designed parts. The goal is to make the part durable but also lightweight, therefore a choice of 40% infill is made.

(a)



(b)

Figure A.12: (a) Slid out the battery case from landing gear (b) Landing gear and battery box are securely locked.

Battery Holder and Landing Gear

Since the frame does not come with proper landing gear, one must be added to safely land the quadcopter without causing damage either to the frame or to the installed components. A place to house the battery is also needed, therefore the design makes it easy to reach the battery to be replaced by sliding on, and of the landing gear. The landing gear has also a place for the ultrasonic sensor. Furthermore, the design is a bridge-style to distribute the impact without breaking the landing gear.

(a)

(b)

Figure A.13: (a) Propeller Protector (b) Computer Protector.

Computer and Propeller Protector

To protect the quadcopter and above all the operation of the quadcopter, the cover of the propeller has to be installed. The propellers are sharp and very dangerous during the operation of the motors. Also, the propellers used are made of plastic and they easily break during any small collisions with walls and obstacles which also can be avoided by using such a solution. The dimensions have been chosen to also balance the quadcopter if, during landing operations, it can push back the quadcopter to be on its landing gear.

B | Appendix B: Simulation Code

B.1. Q-learning Algorithm with Sigmoid Function

```
public class QL {

    public struct RM{
        public int s { get; set; }
        public int a { get; set; }
        public float r { get; set; }
        public int s_ { get; set; }
    }

    RM mem = new RM();

    // the discount factor that applies to rewards
    float GAMMA = 0.99f;

    float ALPHA = 0.1f;

    public bool done = false;

    float[] ACTIONS = {0f, 0.1f, -0.1f};

    float[,] Q_table = new float[101,3];

    int r = -2;

    double epsilon = 1.0;

    System.Random RndB = new System.Random();
```

```
public float getAction(float k){
    mem.s = mem.s_;

    if(RndB.NextDouble() < epsilon) {
        // To avoid gains below zero
        if (k <= 0f) {
            mem.a = RndB.Next(0, 2);
        } else {
            mem.a = RndB.Next(0, 3);
        }
    }

    else {
        int maxIndex = 2;

        if (Q_table[mem.s,0] > Q_table[mem.s,1]){
            if (Q_table[mem.s,0] > Q_table[mem.s,2] || k <= 0.1f){
                maxIndex = 0;
            }
        }

        else {
            if (Q_table[mem.s,1] > Q_table[mem.s,2] || k <= 0.1f){
                maxIndex = 1;
            }
        }

        mem.a = maxIndex;
    }

    return ACTIONS[mem.a];
}

public void updateQTable(float err, float k){

    err = System.Math.Abs(err);
```

```
        if (err < 0.001f){
            done = true;
        }

        else if (err < 0.01f){
            mem.r = 1;
        }

        else if (err < 0.1f){
            mem.r = 0;
        }

        else if (err < 1f){
            mem.r = -1;
        }

        else {
            mem.r = -2;
        }

        mem.s_ = (int)(sigmoid(err) * 100.0);

        Q_table[mem.s,mem.a] = (1-ALPHA) * Q_table[mem.s,mem.a] + ALPHA *
            ↪ (mem.r + GAMMA * System.Math.Max(Q_table[mem.s_,0],System.
            ↪ Math.Max(Q_table[mem.s_,1],Q_table[mem.s_,2])));

        if (epsilon > 0.01){
            epsilon -= 0.001;
        }
    }

    double sigmoid(double x){
        return 1 / (1 + System.Math.Exp(-x));
    }
}
```

B.2. PID Controller

```
public class PID
{
    Rigidbody rb;

    float x = 0f;
    float y = 0f;
    float z = 0f;
    float w = 0f;

    float roll = 0f;
    float pitch = 0f;
    float yaw = 0f;
    float altitude = 0f;

    float err_pitch = 0f;
    float err_roll = 0f;
    float err_yaw = 0f;
    float err_altitude = 0f;

    float kp_roll = 0f;
    float ki_roll = 0f;
    float kd_roll = 0f;

    float kp_pitch = 0f;
    float ki_pitch = 0f;
    float kd_pitch = 0f;

    float kp_yaw = 0f;
    float ki_yaw = 0f;
    float kd_yaw = 0f;

    float kp_altitude = 0.1f;
    float ki_altitude = 0.1f;
    float kd_altitude = 0.1f;
```

```
float dt = 0f;

float d_err_pitch = 0f;
    float d_err_roll = 0f;
    float d_err_yaw = 0f;
float d_err_altitude = 0f;

    float pMem_roll = 0f;
    float pMem_pitch = 0f;
    float pMem_yaw = 0f;
float pMem_altitude = 0f;

    float iMem_roll = 0f;
    float iMem_pitch = 0f;
    float iMem_yaw = 0f;
float iMem_altitude = 0f;

    float dMem_roll = 0f;
    float dMem_pitch = 0f;
    float dMem_yaw = 0f;
float dMem_altitude = 0f;

float prev_time = 0f;
float prev_err_roll = 0f;
    float prev_err_pitch = 0f;
    float prev_err_yaw = 0f;
float prev_err_altitude = 0f;

QL altitudeKpQL = new QL();
QL altitudeKiQL = new QL();
QL altitudeKdQL = new QL();

bool flag = true;

int frames = 0;

int massExper = 0;
```

```
int dragExper = 0;
int angDragExper = 0;

// Start is called before the first frame update
void Start()
{
    rb = GetComponent<Rigidbody>();

    // Disturbances
    InvokeRepeating("pushRoll", 1.0f, 5f);
    InvokeRepeating("pushPitch", 1.5f, 5f);
    InvokeRepeating("pushYaw", 2.0f, 5f);
    InvokeRepeating("pushRoll", 2.5f, 5f);
    InvokeRepeating("pushPitch", 2.5f, 5f);
}

// Update is called once per frame
void FixedUpdate()
{
    ++frames;

    x = transform.rotation.x;
    y = transform.rotation.y;
    z = transform.rotation.z;
    w = transform.rotation.w;

    pitch = Mathf.Atan2(2*x*w - 2*y*z, 1 - 2*x*x - 2*z*z);
    roll = Mathf.Asin(2*x*y + 2*z*w);
    yaw = Mathf.Atan2(2*y*w - 2*x*z, 1 - 2*y*y - 2*z*z);
    altitude = transform.position.y;

    dt = Time.fixedDeltaTime;

    err_pitch = 0 - pitch;
        err_roll = 0 - roll;
        err_yaw = 0 - yaw;
    err_altitude = 5 - altitude;
```

```
if (flag){
    flag = false;
    return;
}

d_err_pitch = err_pitch - prev_err_pitch;
    d_err_roll = err_roll - prev_err_roll;
    d_err_yaw = err_yaw - prev_err_yaw;
d_err_altitude = err_altitude - prev_err_altitude;

// Train
if (!altitudeKpQL.done) {
    altitudeKpQL.updateQTable(err_altitude + (d_err_altitude / dt)
        ↪ , kp_altitude);
} else if (!altitudeKiQL.done){
    altitudeKiQL.updateQTable(err_altitude + (d_err_altitude / dt)
        ↪ , ki_altitude);
} else if (!altitudeKdQL.done){
    altitudeKdQL.updateQTable(err_altitude + (d_err_altitude / dt)
        ↪ , kd_altitude);
}

// PID - ROLL
kp_roll = 3f;
ki_roll = 0.001f;
kd_roll = 0.5f;

// PID - PITCH
kp_pitch = kp_roll;
ki_pitch = ki_roll;
kd_pitch = kd_roll;

// PID - YAW
kp_yaw = 1f;
ki_yaw = 0f;
kd_yaw = 0f;
```

```
// PID - ALTITUDE
kp_altitude = 1.0f;
ki_altitude = 0.0f;
kd_altitude = 0.0f;

        pMem_roll = kp_roll * err_roll;
        pMem_pitch = kp_pitch * err_pitch;
        pMem_yaw = kp_yaw * err_yaw;
pMem_altitude = kp_altitude * err_altitude;

        iMem_roll += err_pitch * dt;
        iMem_pitch += err_roll * dt;
        iMem_yaw += err_yaw * dt;
iMem_altitude += err_altitude * dt;

        dMem_roll = d_err_roll / dt;
        dMem_pitch = d_err_pitch / dt;
        dMem_yaw = d_err_yaw / dt;
dMem_altitude = d_err_altitude / dt;

roll = pMem_roll + ki_roll * iMem_roll + kd_roll * dMem_roll;
    pitch = pMem_pitch + ki_pitch * iMem_pitch + kd_pitch *
       ↪ dMem_pitch;
    yaw = pMem_yaw + ki_yaw * iMem_yaw + kd_yaw * dMem_yaw;
altitude = pMem_altitude + ki_altitude * iMem_altitude +
   ↪ kd_altitude * dMem_altitude;

transform.Find("BR").GetComponent<Rigidbody>().AddForce(transform.
   ↪ up * (altitude + roll + pitch + yaw));
transform.Find("BL").GetComponent<Rigidbody>().AddForce(transform.
   ↪ up * (altitude - roll + pitch - yaw));
transform.Find("FR").GetComponent<Rigidbody>().AddForce(transform.
   ↪ up * (altitude + roll - pitch - yaw));
transform.Find("FL").GetComponent<Rigidbody>().AddForce(transform.
   ↪ up * (altitude - roll - pitch + yaw));
```

```
        prev_err_roll = err_roll;
            prev_err_pitch = err_pitch;
            prev_err_yaw = err_yaw;
        prev_err_altitude = err_altitude;
    }



    void pushYaw()
    {
        rb.AddTorque(transform.up * 100);
    }

    void pushRoll()
    {
        rb.AddTorque(transform.right * 100);
    }

    void pushPitch()
    {
        rb.AddTorque(transform.forward * 100);
    }

}
```

List of Figures

1.1 Agent-environment relation in a Markov decision process [3]. 3
1.2 Sigmoid function [60]. 6
1.3 Quadcopter's PID controller example [3]. 7
1.4 (a) Without PID (b) P (c) PD (d) PID [34] 8

3.1 How Q-learning will fit in the standard PID controller loop. [3]. 17
3.2 The use of Sigmoid function in the proposed algorithm [3]. 18
3.3 An episode of the proposed algorithm [3]. 19
3.4 (a) 3D model of quadcopter (b) Simulation settings [Mass, Drag] (c) Simulation viewport of a ground. 22
3.5 (a) Apply random rotational force. (b) Apply random force pushing on different angles around the quadcopter. 23

4.1 (a) Box plot for each increase in mass on the horizontal axis, and corresponding time of convergence on the vertical axis. (b) Box plot that represents the change in drag on the horizontal axis, with the time of convergence along the vertical axis. 26
4.2 Handpicked experiments to show the nondeterministic behavior to be expected from such an algorithm. On the other hand, all converge to the desired setpoint altitude. 29

A.1 (a) Quadcopter (b) 3D model 1-to-1 to the quadcopter in **(a)** with embedded physical properties. 41

A.2 ZMR250 Quadcopter Frame [73]. 42
A.3 EMAX RS2205-S 2300KV Brushless CW Motors [19]. 43
A.4 Gemfan 5-inch 3-Blade Propeller [26]. 44
A.5 HRB LiPo Battery [30]. 45
A.6 XTVTX 35A BLHeli-S Electronic Speed Controller [68]. 46
A.7 ZHITING PDB [72]. 47
A.8 Jetson Nano Developer Kit (Version: B01) [50]. 48

A.9 MPU6050 Inertial Measurement Unit [6]. 49
A.10 DF SRF02 Ultrasonic Sensor [15]. 50
A.11 3D printer printing the computer protector. 51
A.12 (a) Slid out the battery case from landing gear (b) Landing gear and battery box are securely locked. 52
A.13 (a) Propeller Protector (b) Computer Protector. 53

List of Tables

2.1 The Q-table of the work in [36]. 12

4.1 Descriptive statistics for learning time in seconds (variable = Mass) 25
4.2 Descriptive statistics for for learning time in seconds (variable = Drag) . . 26
4.3 Episodes needed to tune the PID controller given the altitude. 27

A.1 ZMR250 Quadcopter Frame Specifications [73]. 42
A.2 EMAX RS2205-S 2300KV Brushless CW Motors Specifications [19]. 43
A.3 Gemfan 5-inch 3-Blade Propeller Specifications [26]. 44
A.4 HRB LiPo Battery Specifications [30]. 45
A.5 XTVTX 35A BLHeli-S Electronic Speed Controller Specifications [68]. . . 46
A.6 ZHITING PDB Specifications [72]. 47
A.7 Jetson Nano Developer Kit Specifications [50]. 48
A.8 MPU6050 Inertial Measurement Unit Specifications [6]. 49
A.9 DF SRF02 Ultrasonic Sensor Specifications [15]. 50

List of Symbols

Variable	Description
$\boldsymbol{s}$	state
$\boldsymbol{a}$	action
$\boldsymbol{t}$	discrete time step
$\boldsymbol{T}$	final time step in an episode
$\boldsymbol{S}$	set of states
$\boldsymbol{A}$	set of actions
$\boldsymbol{A(s)}$	set of actions in state s
$\boldsymbol{R}$	set of rewards
$\boldsymbol{S_t}$	state at t
$\boldsymbol{A_t}$	action at t
$\boldsymbol{R_t}$	reward at t
$\boldsymbol{G_t}$	cumulative discounted return following t
$\boldsymbol{\pi}$	policy
$\boldsymbol{\pi(s)}$	action take in state s under policy π
$\boldsymbol{\pi(a\|s)}$	probability of taking action a in state s following policy π
$\boldsymbol{p(s', r\|s, a)}$	probability of transitioning to state s' with reward r from s and a
$\boldsymbol{v_\pi(s)}$	expected return of state s under policy π
$\boldsymbol{q_\pi(s, a)}$	value of taking action a in state s
$\boldsymbol{\gamma}$	discount rate
$\boldsymbol{\lambda}$	decay rate
$\boldsymbol{\epsilon}$	probability of random action in ϵ-greedy