POLITECNICO DI TORINO

Master's Degree in Mechatronics Engineering

Master's Thesis

Integration of an advanced GNSS system for Autonomous UAV Navigation

Supervisor:

Prof. Giorgio Guglieri

Co-supervisor:

Simone Godio

Candidate:

HU YAODONG

Academic Year 2021/2022

Acknowledges

I wish to show my appreciation to all the people who have supported me. In particular, I wish to show my appreciation to the co-supervisor Simone Godio who enthusiastically supported and guided my work. I wish to show my appreciation to Prof. Giorgio Guglieri who provided me with this topic. I also wish to show my appreciation to my family for their selfless support, encouragement, and trust during this pandemic.

Abstract

Over the last few decades, unmanned aerial vehicles (UAVs) have made great strides through the development of electronics, mechanics, aerodynamics, sensors, and control theories. UAVs are now increasingly attracting the attention of researchers and are becoming more common in everyday life. To increase the precision and autonomy of UAVs in outdoor activities, precise outdoor positioning technology is required. Many UAVs require the use of the Global Navigation Satellite System (GNSS) to provide accurate position information for applications like mapping, surveying, precision agriculture, and search and rescue.

This thesis focuses on autonomous flight controls and GNSS integration of the quadcopter. The quadcopter is a type of UAV lifted and propelled by four motor propellers. In this work, a cascade inner-outer loop flight controller is designed taking into consideration the under-actuated property of the quadcopter. Furthermore, an open-source quadcopter based on STM32, including different sensors such as IMU, compass, barometer, and GNSS receiver, with remote communication and data storage functions, is designed. The flight controller is deployed on this quadcopter to perform functions such as manual flight, autonomous flight, GNSS/GPS position hold, waypoint flight, and so on.

First, the non-linear mathematical model of the quadcopter is derived according to the translational and rotational dynamics of the quadcopter. A linearized mathematical model of the quadcopter is obtained by linearizing the nonlinear model around an equilibrium point. Then, based on the linear mathematical model, the quadcopter autonomous flight controller is designed with the Linear Quadratic Regular (LQR) control strategy. The controller parameters are tuned in a MATLAB/Simulink simulation environment, taking into account the time and material consumption. Finally, a quadcopter platform is developed and an STM32 development board is used as the microcontroller of this quadcopter. To measure the attitude of the quadcopter, an Inertia Measurement Unit (IMU) is used, and the method based on Euler angles & rotation matrix is adopted to compute the attitude with IMU measurements. A complementary filter is implemented to avoid the noise and drift of the IMU. To measure the heading to the North, a tilt-compensated compass is developed and integrated into the quadcopter. A GNSS receiver and a barometer are integrated into the quadcopter to measure its horizontal position and attitude during flight. To deploy the designed LQR controller on the quadcopter, the flight control program is developed based on the Arduino STM32. The programming language is C/C++. The quadcopter is programmed to fly in manual flight mode or autonomous flight mode. Experiments were conducted to test the manual flight performance of the quadcopter, and excellent results were achieved.

Contents

List of Figures

List of Tables

Chapter 1: Introduction

An Unmanned Aerial Vehicle (UVA), also known as a drone, is an aircraft that can fly remotely without any human pilot on board or fly autonomously with an onboard flight controller that works in conjunction with sensors and a Global Navigation Satellite System (GNSS) module. Over the past years have witnessed great technological advances in electronics, mechanics, sensors, embedded control systems, aerodynamics, and control theories, which make unmanned aerial vehicles (UVAs) more widespread in people's daily life and gaining more and more attention in the field of science and technology.

A variety of UAVs have been developed and according to their configurations, they are grouped into three categories: fixed-wing UAVs, multi-rotor UAVs, and fixed-wing hybrid VTOL UAVs. Fixed-wing UAVs generate lift through the wings themselves due to the forward airspeed provided by engines or electric motor propellers. They can cover very long distances, map very large areas, and perform long-time monitoring because they are energy efficient. But the fixed-wing UAVs can't take off vertically or hover in the air. Multi-rotor UAVs generate lift by multiple electronic motor propellers, they are very maneuverable, can take off and land vertically, can hover in the air, and achieve great control performance during flight. According to the number of motors, multi-rotor UAVs can be categorized into Tricopters (3 motors), Quadcopters (4 motors), Hexacopters (6 motors), and Octocopters (8 rotors). By far, quadcopters are the most popular multi-rotor UAVs all over the world. Fixed-wing hybrid VTOL UAVs merge the benefits of both fixed-wing and rotor-based designs. These types of UAVs have rotors attached to the fixed wings, allowing them to hover during flight and take off and land vertically, while at the same time retaining all the advantages of the fixed-wing type. Only a handful of fixed-wing hybrid VTOLs are currently on the market, and the technology used in these UAV types is still in the nascent stage.

This work pays attention to the quadcopter, which is one type of helicopter lifted and propelled by four motors placed in a square format with an equal distance from the center of mass. The two arms of the quadcopter are in cross configuration and linked to its central body where the sensors, microcontrollers, and other electronic components are located. The quadcopter is an under-actuated system since only four motors are utilized to control its 6 degrees of freedom (6DoF) motion in space (four control inputs, six control outputs). This means the quadcopter's translational and rotational motions are coupled, leading to highly nonlinear dynamics in the system. Furthermore, quadcopters are inherently unstable and require constant corrections from the onboard microcontroller hundreds of times a second to maintain stability. Despite these disadvantages, quadcopters feature various advantages such as cheap, lightweight, strong maneuverability, vertical take off and landing, and the capability of hovering.

This work focuses on the design of an autonomous flight controller for the quadcopter based on the GNSS positioning. It is difficult for a quadcopter without a GNSS module to achieve a stable flight outdoors. When the quadcopter performs outdoor flight tasks such as fixed-point hovering, some unpredictable interference in the surrounding environment might cause the quadcopter to deviate from its hovering position. It is difficult for the quadcopter to return to its original position only by

the pilot's remote control on the ground. Also, due to the interference of the surrounding environment, the pilot has to spend a lot of effort to make the quadcopter fly along a specific flight path in the air. Therefore, it is particularly important to add a GNSS module to the quadcopter to enable it to achieve self-positioning and autonomous flight. Adding a GNSS module to a quadcopter can not only make it fly more stable but also add more advanced features such as position hold, autonomous flight, return to home, and waypoint navigation. In this paper, a quadcopter autonomous flight controller based on the LQR control strategy is developed. The controller is first tuned and validated in a simulation environment. Then an open-source quadcopter platform based on STM32 is developed. This quadcopter platform consists of an IMU to estimate its attitude in three-dimensional space, a compass to estimate its heading, a barometer to estimate its altitude, as well as a GNSS module to position itself. The designed LQR controller will be tested on this quadcopter platform and the control performance will be evaluated.

1.1 History of Quadcopter

The history of quadcopters can date back to the early 19th century. In 1898, Nikola Tesla tested his radio-controlled boat for the first time in a New York Pond in Madison Square Garden, which is the beginning of every radio-controlled aircraft as we know it today [1]. Around 1907, the world's first quadcopter was created by inventor brothers Jacques and Louis Bréguet. But it had big limitations and its design was proved to be very unstable.

It was until 1924 that the world's first working quadcopter, the Ominichen 2 (Figure 1.1), was invented by French engineer Etienne Omnichen. The Ominichen 2 had a cross-shaped structure built of metal tubing and lifted by four two-bladed, counter-rotating main rotors. The pitch angle of these blades was controlled by warping. The quadcopter also had another five rotors positioned in the horizontal plane to provide lateral control, one of which at the front was used to steer the quadcopter [2]. In 1924, Ominichen flew this quadcopter a distance of 360 meters and in the same year, he flew a one kilometer closed circle in 7m and 40s, which established a record for helicopters. Around the same time, George de Bothezat built and tested his quadcopter for the US army, completing a number of flight tests before the program was scrapped [3]. The de Bothezat quadcopter (Figure 1.2) had an X-shaped structure supported by four six-blade rotors at each end of the arms, and at the ends of the lateral arms, two small propellers with variable pitch were used for thrusting and yaw control [4]. This quadcopter made its first flight in October 1922, and about 100 flights were made by the end of 1923.

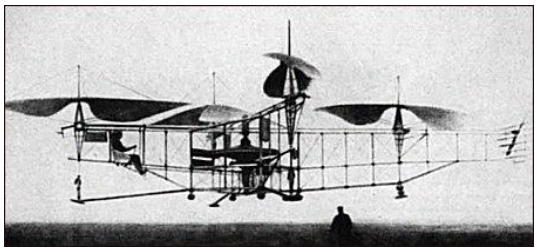


Figure 1.1 Ominichen 2 [3]

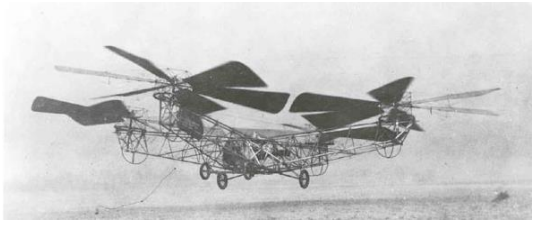


Figure 1.2 de Bothezat quadcopter [3]

In the middle of the 1950s, Convertawings Model A Quadcopter was intended. The four rotors of this quadcopter were mounted laterally on outriggers in two tandem pairs. The control mechanism was extremely simple and obtained by differential change of thrust between the rotors [4]. It was one of the first quadcopters to use the varying thrusts of the four rotors to achieve control in flight. But it was very hard for the pilot to maneuver during flight because of the workload of trying to control the thrusts of all four rotors at once. Despite successful testing and development, military support for this quadcopter ceased after cutbacks in defense spending [4]. However, the design, particularly its control mechanism, was a forerunner of the modern quadcopters.

Figure 1.3 Convertawings model A quadcopter [4]

By the end of the 20th century, with the development of electric motors, microelectronics, and micromechanical devices, it became possible to build small size, lightweight, reliable, and cheap modern quadcopters. The modern quadcopter first emerged in the late 1990s and early 2000s as hobbyist kits. In 2006, the first industrial and commercial uses of quadcopters and other UAVs had their infancy with the first commercial drone permit released by the FAA (Federal Aviation Administration), and in the same year, Frank Wang found DJI, whose idea was to revolutionize the use of quadcopters to the mainstream population [1]. Up to now, DJI has released a series of quadcopter products and achieved great achievements in civilian consumer-grade quadcopters and aerial photography. Later in 2010, the French drone manufacturer Parrot unveiled its AR Drone (Figure 1.4 (a)), the first commercially successful ready-to-fly consumer drone, and the first able to

be controlled solely by a Wi-Fi connection [1]. In 2013, Amazon and other delivery companies declared their intention to use quadcopters for delivering their products in record times through airborne means [1]. In 2016, DJI released the Phantom 4 (Figure 1.4 (c)), which had computer vision and machine learning ability to track a person or an object on the ground without simply following a GPS track. In 2018, DJI and its competitors, like Autel and Parrot continued to release new models and grew both in the hobby sector as well as the professional sector [1].

(a) Parrot AR drone

(b) Amazon delivery drone

(c) DJI Phantom 4

(d) DJI Mavic Mini

Figure 1.4 Modern quadcopters

1.2 Modern quadcopters applications

As one of the most popular UAVs nowadays, modern quadcopters feature the advantages of less power consumption, less risk to human life, ease of operation, simple structure, vertical take off and landing (VTOL), strong maneuverability, and stable hovering performance, which makes them more and more concerned and their applications more and more extensive. The potential application fields of modern quadcopters include civil, environmental, and defense sectors [5].

The civil application of modern quadcopters includes photography, construction, mining, delivery, agriculture, disaster management, surveillance, and so on. Modern quadcopters make aerial photography become very easy, cheap, and no longer just for professional photographers. Without the need for helicopters or crews on the ground or on board, only with a quadcopter pilot and a cameraman, it is possible to obtain affordable aerial photographs for ordinary people (Figure 1.5 (a)). One of the quadcopter's applications in agriculture is precision farming. Precision farming is based on data collection and variability mapping of agricultural lands, and data analysis. Then farming management such as pesticide spray, irrigation, and fertilizer can be implemented based on the results of the collected and analyzed data. The traditional satellite-based mapping technology is costly, and the data collection can be easily influenced by weather conditions [5]. Using a quadcopter equipped with special devices such as thermal sensors, RGB cameras, and LIDAR systems, the mapping can be implemented more accurately and cheaply. Also, a quadcopter can be

equipped with a sprayer, fertilizer, or irrigation devices to perform more precise crop spraying, fertilizing, and irrigating (Figure 1.5 (b)). In the field of aerial surveillance, quadcopters with high-resolution cameras and various security equipment can be widely used in the security system of families and companies. This can not only realize a wide range of real-time monitoring from multiple precepts but also make security work safer and easier. As for the application of quadcopters in delivery (Figure 1.5 (d)), it will greatly reduce the delivery time by skipping roads, traffic lights, and buildings, and lower the human cost. But the power supply limitation due to the battery capacity will limit the delivery range and working time, and the load capacity of the quadcopter will also limit the delivery capacity. The quadcopter delivery is still in development, researchers are working to overcome issues associated with the delivery scope limitation.

(a) Aerial photography

(b) Agriculture application

(c) Aerial surveillance

(d) quadcopter used for delivery

Figure 1.5 Modern quadcopter applications in the civil sector

In the environmental field, applications of quadcopters include air quality monitoring, soil monitoring, mountain inspection, mapping & surveying, and so on. Future air quality monitoring can be implemented by using a remotely controlled quadcopter equipped with a telemetric device (Figure 1.6 (a)). And the measurements can be implemented almost in real time and in different geometries – vertical and horizontal. The applications of quadcopters to mountain inspection can effectively implement forest patrol, fire prevention, and wildlife protection. As for mapping & surveying, the quadcopter-based mapping technique can overcome the shortcomings of traditional aircraft mapping operations. The remotely controlled quadcopters are the most powerful tools for safely capturing accurate aerial data quickly. Importantly, they can also create aerial maps in real-time, even before they have landed (Figure 1.6 (b)).

(a) Micro station in UAV Air Quality Monitoring

(b) image of drone mapping for the smart city

Figure 1.6 Quadcopter applications in the environmental sector

As for the applications in defense, quadcopters can be utilized for anti-terror, border security monitoring, aerial reconnaissance, bomb-dropping, and so on. USA, UK, Russia, India, and Israel are the leading countries in the development and deployment of military drones. The growing application of drones in the military and defense sector will propel the growth of the global drone market by 2027.

1.3 State-of-the-art control strategy

As mentioned above, the quadcopter is inherently unstable and under-actuated, this brings nonlinear dynamics to the system that make the control very difficult. Hence, the control strategy of the quadcopter is of crucial importance to achieve safe operation, reliable stabilization, autonomous flight, trajectory tracking, and robustness to unpredictable changes in the environment. Over the past years, various quadcopter control approaches have been proposed by researchers. These control methods can be categorized into three categories: linear control method, nonlinear control method, and learning-based control method.

Even though the quadcopter is a nonlinear system, linear control strategies are still useful and good control performance has been achieved now. To control the quadcopter with linear control strategies, the non-linear quadcopter mathematical model should be first linearized around an equilibrium point. The most used linear control strategies are the Proportional – Integrative – Derivative (PID) control, the Linear Quadratic Regulator (LQR) control, the H-infinity control, and the gain schedule. In [6], a classic PID controller is implemented to control the attitude of the quadcopter and good control performance is achieved. In [7, 8], a cascade PID feedback controller is proposed to stabilize the attitude of the quadcopter, and the cascade PID controller is proved to be more effective and robust compared to the classic one. [9, 10] exploits the cascade PID controller to track the given trajectory, simulations considering the disturbances are implemented and the results are very satisfactory. The LQR-based control methods are also showing very good control performance. In [11], the author utilizes an LQR controller to implement the attitude, position, and altitude control of the quadcopter. [12, 13] proposed two LQR controllers with integration action to control the position and yaw orientation of the quadcopter, the integration term is used to improve the quadcopter tracking performance. In [12] the controller is in an inner-outer loop structure and the testing results on a real quadcopter platform show that the controller can achieve good performance in trajectory tracking and is very robust to perturbations. In general, linear control methods can achieve the stability of

the quadcopter around the chosen operating point, usually the hovering state, but when the quadcopter's state is far from the equilibrium, the control performance will be poor, leading to the instability of the quadcopter. To overcome this limitation, a group of linear controllers is designed based on several operating points, this is the known gain schedule approach. The gain schedule method can incorporate both linear and nonlinear controllers [14, 15, 16] to improve the control performance. In [17], a gain scheduled LQR controller is exploited taking into consideration the different yaw angles of the quadcopter during flight and the linearization is performed based on the current yaw angle, the designed control eliminates the limitation of the quadcopter during the trajectory tracking and improves the tracking performance. As for the H-infinity control, [18, 19] have shown great control performance of implementing this approach to the quadcopter.

Liner control methods are not able to handle all the system behaviors since the controller is designed based on the linearization of the model around an equilibrium point. Better control performances can be achieved by using nonlinear control methods that consider more general quadcopter dynamics. Nonlinear control methods like backstepping, sliding mode (SMC), feedback linearization (FL), adaptive control, and model predictive control (MPC) have been proved to be very effective in quadcopter control. The state feedback linearization approach is implemented in [20, 21, 22, 23, 24] to get a linear model of the quadcopter, the linearized system can be controlled with a linear control approach like PID, LQR, MPC, etc., and works of literature show that this configuration achieves very good performance in quadcopter trajectory tracking. [25, 26, 27] present several cases where simple sliding mode control (SMC) has been successfully implemented. Even though the SMC has shown acceptable performance, it is often accompanied by the chattering issue. In [28, 29], the author attempts to avoid the undesired chattering effect by using 2-order sliding mode control (2-SMC), which acts directly on the second-derivative of the sliding surface, and the results demonstrate that the 2-SMC is superior in comparison with the simple SMC. Model predictive control shows the capability of working with constraints and disturbances. In [30, 31], researchers have implemented a linear MPC to the quadcopter trajectory tracking under disturbances in a simulation environment. In [31], the linear MPC is used in combination with the feedback linearization approach, and the simulation results demonstrate that the designed controller can effectively ensure trajectory tracking under constraints and continuous disturbances. Despite the linear MPC features above advantages, it can't handle the nonlinearity of the quadcopter, leading to a degree of performance degradation. In [32, 33], researchers have implemented the nonlinear MPC for quadcopter trajectory tracking. The proposed controller can not only handle the nonlinearity of the quadcopter but also make use of the inherent capabilities of MPC. As for the adaptive control, it's usually coupled with other controllers like PID, LQR, back-stepping, and SMC. This kind of controller usually includes two loops where one is the normal feedback loop, and another one is used for parameter adjustment [34].

learning-based control methods include fuzzy logic control and neural network control. The fuzzy logic control has achieved popularity on quadcopter platforms both in standalone approach or combined with other control approaches [34]. In [35], the author proposes a Fuzzy-PID controller to study the roll and pitch angles stability on a circular trajectory, simulations with both the Fuzzy-PID controller and the PID controller are implemented, and the Fuzzy-PID controller has relatively smaller errors and better robustness than the PID one. In [36] the Fuzzy-PID controller is implemented in the quadcopter trajectory tracking and shows better performance than the PID controller. Also, neural network control has been extensively used in quadcopter control. In [37],

the author proposes to use the neural network to continually adjust the PID parameters for minimizing the tracking error. Furthermore, in [38] the reinforcement learning approach is implemented in the quadcopter's automatic landing.

1.4 Outline

The rest of this thesis is organized as follows:

Chapter 2. This chapter first describes the flight mechanism of a quadcopter and introduces its four basic movements. Then, the NED coordinate frame and body coordinate frame that are required for describing the quadcopter's position and attitude are introduced. The concept of Euler angles is introduced to describe the attitude of the quadcopter. Finally, two nonlinear quadcopter mathematical models are derived based on Newton's law and the Euler equation, and the nonlinear mathematical model is linearized around the equilibrium point.

Chapter 3. This chapter demonstrates an autonomous flight controller with an LQR control strategy. The flight controller structure is firstly introduced. Then the altitude, attitude, and position controller are designed with the LQR control strategy. Finally, a simulation is performed to tune and validate the parameters of the designed controller. The performance of the designed controller is evaluated considering the step signal response and trajectory tracking performance.

Chapter 4. This chapter demonstrates the navigation module of the real quadcopter platform which includes an IMU, an electronic compass, a barometer, and a GNSS module. This chapter elaborates on how to use these sensors to estimate the quadcopter's attitude, heading, altitude, and position during flight.

Chapter 5. The hardware components for developing a real quadcopter platform as well as their specifications and characteristics are first introduced. Then the quadcopter schematic and the flowchart of the flight controller program are presented.

Chapter 6. In this chapter, the designed quadcopter is first introduced. Then, the experimental results are analyzed and presented.

Chapter 7. In this chapter, conclusions and possible future works are presented.

Chapter 2: Mathematical model

To analyze the translational and rotational dynamics of the quadcopter and design an autonomous flight controller for it, the mathematical model of the quadcopter needs to be firstly derived. In this section, the preliminary quadcopter flight mechanism and its mathematical model are presented. Firstly, the configuration structure and flight mechanism of the quadcopter are expounded, then the nonlinear mathematical model of the quadcopter is derived and represented in the state-space format, and finally, the nonlinear mathematical model is linearized in order to the subsequent flight controller design.

2.1 Quadcopter flight mechanism

The quadcopter is a multi-rotor helicopter that is lifted and propelled by four motors which are driven by electronic speed controllers. Four motors with the same structure and radius are symmetrically located on the four edges of a cross formed frame. The four motors are located on a plane at the same height. According to the direction of the nose, the quadcopter has two configurations: the X (cross) configuration and the + (plus) configuration (Figure 2.1). The X (cross) configuration is mostly used, this type of configuration maximizes the moments generated by the motor propellers and is more flexible compared to the + (plus) configuration. Due to the above advantages, the X (cross) configuration is adopted in this project.

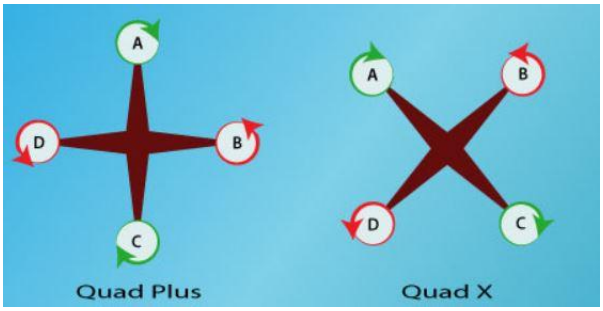


Figure 2.1 Two types of quadcopter configuration

To fly in three-dimensional space, the quadcopter must be capable of three different types of movement: vertical movement, lateral movement, and rotational movement. Based on Newton's third law, each of these can be achieved using the quadcopter's four propellers. The basic motions of a quadcopter are demonstrated in Figure 2.3 below.

The vertical movement is achieved by thrust generated by the four motors, when the propellers spin, each of the propellers will create a thrust, and the total thrust will be the sum of the four propeller's thrusts. As Figure 2.3 (c) shows, when the power of each motor is increased at the same time, the

increase of the rotor speed increases the total thrust. When the total thrust is enough to overcome the gravity, the quadrotor will take off vertically from the ground; otherwise, the power of each motor will be reduced at the same time, and the quadrotor will descend vertically. The quadcopter can hover in space with no vertical movement by having the thrust and gravity equal. The key to vertical movement is to ensure that the rotational speed of the four motors increases or decreases synchronously.

The rotational movements include roll motion, pitch motion, and yaw motion, both can be controlled to desired values by changing the speeds of the four motors. The rotational movements are shown in Figure 2.3(a) (b). The pitch motion and roll motion are caused by the unevenness of the motors' thrusts on the quadcopter's two sides, the aerodynamic torque effect, and the gyroscopic effect. The unbalanced thrusts on the quadcopter's two sides generate a moment, which pitches or rolls the quadcopter. The gyroscopic effect is due to the rotation of the rotors and is only taken into consideration in the lightweight construction quadcopter. The yaw motion is performed by the reactive torque produced by the motor. When a motor propeller spins, it produces a torque, according to Newton's third law of motion, an equal and opposite torque is acting on the quadcopter, which is the reactive torque. The sum of reactive torques generated by the four motors is the yaw moment, which causes the yaw motion of the quadcopter. As is shown in Figure 2.3 (d), to control the yaw motion, two motors locate on the same arm rotate in a clockwise direction, while the left ones rotate in a counterclockwise direction to cancel out the reactive torque. The yaw motion control can be achieved by increasing the speeds of the motors in one direction and/or decreasing the speeds of the motors in opposite direction. Even though having pairs of motors rotating in opposite directions effectively controls the yaw moment acting on the quadcopter, it causes a new problem. If motors spin in opposite directions, two of the propellers push air upward while two push air downward. When combining these forces, the total lift is zero and the quadcopter cannot take off. To overcome this issue, two different types of propellers are used. In Figure 2.2, the leftmost propeller has the left edge higher in the front while the other has the right edge higher. This design eventually makes sure that all the propellers push the air in the same direction.

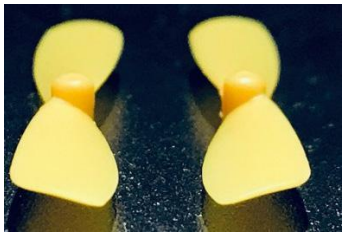
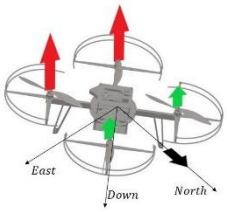


Figure 2.2 Two propellers with different blade angles

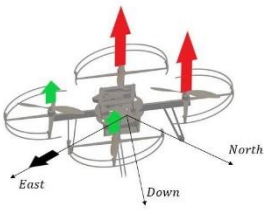
As discussed in chapter 1, the quadcopter is an under-actuated system. The lateral movement is strictly coupled with the rotational motion. This means that for the quadcopter to move forward or to move sideways, it has to tilt forward or sideways first. When the quadcopter tilts, the total thrust acts at an angle, in this case, part of the force of the thrust is upward and part of it is to the side, resulting in lateral movement that can be from side to side or forward and backward. As is shown in Figure 2.3 (a) (b), lateral movement occurs by varying the speed of the propellers, increasing the speed of the two propellers on one side, and/or decreasing the speed of the two propellers on the other side creating uneven amounts of lift on the two sides, the lift created on the side with the faster

spinning propellers is greater than the lift created on the opposite side. The result is that the quadcopter moves in the direction of the side where less lift is created.

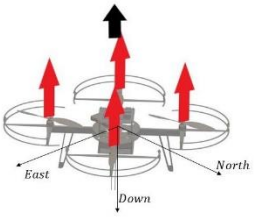
The quadcopter is an inherently unstable system, when a disturbance occurs, it will lose its stability quickly. Hence, a flight controller is required to continuously control and adjust the four motors' speeds to maintain stability. Thanks to the quadcopter's configuration, the propeller's design, and the motor's spinning direction, each basic motion can be performed separately from the others. Engineers can design a corresponding controller for one basic motion individually, which makes the design and tuning of the flight controller simple.



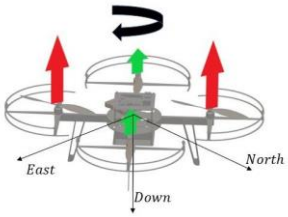
(a) Pitch motion and forward movement



(b) Roll motion and lateral movement



(c) Vertical movement



(d) Yaw motion

Figure 2.3 Basic maneuvers of a quadcopter

2.2 Quadcopter mathematical model

Considering a quadcopter as a rigid body aircraft, its dynamics can be divided into two parts: translational movement and rotational movement. The movement of the body's center of mass is related to translational dynamics and the rotational movement of the body is related to the rotational dynamics. In this project, Newton's second law and Euler moment equation are used to derive the translational and rotational dynamics, respectively. To describe the position and attitude of the quadcopter in 3D space and the torque and thrust acting on it, an appropriate coordinate system should be defined firstly, then kinematics equations and dynamic equations are implemented based on the coordinate system. Finally, the mathematical model of the quadcopter kinematics and dynamics is derived, represented in the State-Space format and a linearization of the model is performed. The State-Space representation of the linearized model plays a key role in designing the autonomous flight controller.

2.2.1 Assumptions

Some assumptions must be done before developing the mathematical model:

1. The quadcopter and all its components are assumed to be a rigid body thus vibrations and deformations during the flight are neglected.
2. The quadcopter is supposed to be perfectly symmetric and the four motors are located on the same plane and perfectly equally spaced from the quadcopter's geometric center.
3. The Center of Gravity of the quadcopter coincides with its geometric center.
4. Forces and torques generated by the wind are neglected
5. The thrust provided by motors is supposed to be constant concerning the environmental conditions (air density, temperature, and altitude).
6. The Earth is considered as flat, and its rotation is negligible concerning body angular speeds.

2.2.2 Coordinate frames

Given an aircraft, the positive x direction in the body coordinate frame is usually defined as the moving direction, the positive y direction as the right side of the moving direction, and the positive z direction as the vertically downward direction; this is also termed forward-right-downward coordinate frame. The body coordinate frame follows the movement of the aircraft, and the origin coincides with the aircraft's center of mass.

To facilitate navigation and waypoint tracking, the axis directions of the inertial coordinate frame are chosen as North-East-Down (NED) navigation frame directions. As is shown in Figure 2.4: the inertial coordinate frame is identified by $x - y - z$ axes, where the z axis is perpendicular to the ground and points toward the center of the earth, the x axis points toward the geometric north, and the y axis points toward the East, $\hat{e}_x, \hat{e}_y, \hat{e}_y$ are three unit vectors used to identify the direction of the three axes. The origin of the body coordinate frame coincides with the quadcopter's Center of Gravity and is identified by $x_b - y_b - z_b$ axes, $\hat{e}_1, \hat{e}_2, \hat{e}_3$ are three the unit vectors used to describe the direction of the three axes.

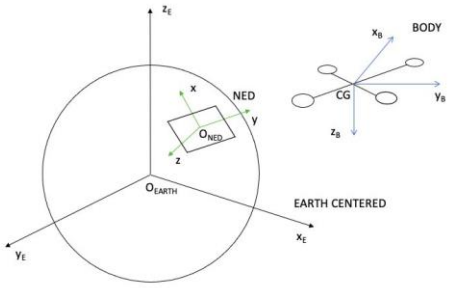


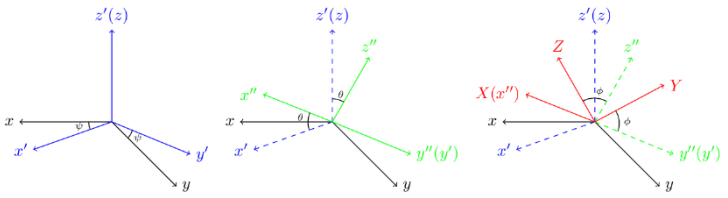
Figure 2.4 NED coordinate frame and body coordinate frame

2.2.3 Rotation and Euler angles

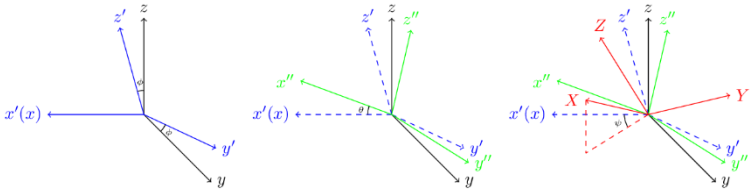
Euler angles and Quaternion are mostly used to describe the rotation and orientation of a plane in three-dimensional space. The quaternion uses four parameters to stand for a rotation, it's a very efficient rotation operator and is mostly used in cases when the aircraft needs to do a full rotation. The Euler angles are simpler and more intuitive compared to the quaternion, but they are limited by a phenomenon called "Gimbal Lock". In this work, Euler angles are chosen to represent the rotation of the quadcopter since the quadcopter's pitch and roll motions are limited and the "Gimbal Lock" issue can be avoided.

Euler angles are three angles introduced by Leonhard Euler to describe the rotation of a rigid body in three-dimensional space [39]. Any three-dimensional rotation can be decomposed by a sequence of three elementary rotations around three axes (consecutive rotation must be performed around two different axes), and the rotation angles around three axes are Euler angles. Without considering the possibility of using two different conventions for the definition of the rotation axes (intrinsic or extrinsic), there exist twelve possible sequences of rotation axes, divided into two groups: Proper Euler angles and Tait-Bryan angles [39]. The three elemental rotations may occur either about the axes of the original coordinate system (extrinsic rotations) or about the axes of the rotating coordinate system (intrinsic rotation). Tait-Bryan 321 angles Figure 2.5, following $z-y'-x''$ (intrinsic rotation) or $x-y-z$ (extrinsic rotation) convention, are also known as yaw, pitch, and roll angles, and are mostly used in aviation to describe the orientation of a ship or aircraft. In this work, Tait-Bryan 321 angles are used to describe the rotation of the quadcopter body coordinate frame relative to the inertial coordinate frame. The rotation matrix of the quadcopter body coordinate frame can be derived by multiplying three elementary matrixes.

The roll pitch and yaw angles of an aircraft are demonstrated in Figure 2.6. The roll angle values range from -90° to 90° where zero means horizontal, 90° means full roll right and -90° means full roll left. The pitch angle values also range from -90° to 90° where zero means horizontal, 90° means straight up and -90° means straight down. The yaw angle values range from 0 to 360°, which gives information about the quadcopter's direction in space. In the attitude and heading reference system (AHRS), the yaw angle represents the direction of the aircraft to the north, where 0° means the aircraft pointing North, 90° means pointing East, 180° means pointing South, and 270° means pointing West.



(a) Intrinsic rotation axes in order $z-y'-x''$



(b) Extrinsic rotation axes in order $x-y-z$

Figure 2.5 Tait-Bryan angles

For convenience, the following notation is introduced:

$$\cos(*) = c_{(*)} \quad \sin(*) = s_{(*)} \quad \tan(*) = t_{(*)}$$

The basic elementary rotations around x, y, and z axes can be described as:

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\phi & -s_\phi \\ 0 & s_\phi & c_\phi \end{bmatrix} \quad R_y = \begin{bmatrix} c_\theta & 0 & s_\theta \\ 0 & 1 & 0 \\ -s_\theta & 0 & c_\theta \end{bmatrix} \quad R_z = \begin{bmatrix} c_\psi & -s_\psi & 0 \\ s_\psi & c_\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2-1}$$

The notation ϕ, θ, ψ represents roll pitch and yaw angle around three axes respectively. By multiplying the three matrixes, the rotation matrix from the body coordinate frame to the inertial coordinate frame can be described as:

$$R_{body}^{NED} = R_z R_y R_x = \begin{bmatrix} c_\theta c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi \\ c_\theta s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{bmatrix} \tag{2-2}$$

The rotation matrix R_{body}^{NED} is orthogonal, thus $R_{body}^{NED\ -1} = R_{body}^{NED\ T}$, which is the rotation matrix from the inertial coordinate frame to the body coordinate frame.

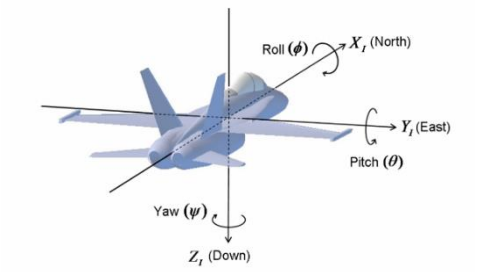


Figure 2.6 Roll pitch and yaw angles of an aircraft

2.2.4 Kinematics equations

Kinematics describes the motion of the quadcopter without considering the forces that cause it to move. Defining $V = [\dot{x} \quad \dot{y} \quad \dot{z}]^T$ the velocity of the quadcopter in the inertial coordinate frame and $V_b = [u \quad v \quad w]^T$ the velocity of it represented in the body frame, the rotation matrix transfers the velocity represented in the body coordinate frame to the inertial coordinate frame:

$$V = R_{body}^{NED} \cdot V_b \tag{2-3}$$

Denoting the angular velocity of the quadcopter represented in the inertial coordinate frame as $\omega = [\dot{\phi} \quad \dot{\vartheta} \quad \dot{\psi}]^T$ and the angular velocity represented in the body coordinate frame as $\omega_b = [p \quad q \quad r]^T$, it's possible to find these two angular velocities are linked by the following relations:

$$\omega_b = T \cdot \omega \tag{2-4}$$

Where T denotes the transformation matrix for angular velocities from the inertial coordinate frame to the body coordinate frame:

$$T = \begin{bmatrix} 1 & 0 & -s_\theta \\ 0 & c_\phi & s_\phi c_\theta \\ 0 & -s_\phi & c_\phi c_\theta \end{bmatrix} \tag{2-5}$$

In the case of $\theta \neq (2k-1)\pi/2, (k \in Z)$, T is invertible and T^{-1} denotes transformation matrix of angular velocities from the body coordinate frame to the inertial coordinate frame:

$$\omega = T^{-1} \cdot \omega_b \tag{2-6}$$

$$T^{-1} = \begin{bmatrix} 1 & s_\phi t_\theta & c_\phi t_\theta \\ 0 & c_\phi & -s_\phi \\ 0 & \frac{s_\phi}{c_\theta} & \frac{c_\phi}{c_\theta} \end{bmatrix} \tag{2-7}$$

Combining Equation 2-3 and Equation 2-6, the kinematic equation of the quadcopter can be written as:

$$\begin{cases} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = R_{body}^{NED} \cdot \begin{bmatrix} u \\ v \\ w \end{bmatrix} \\ \dot{\phi} = p + r[\cos(\phi)\tan(\theta)] + q[\sin(\phi)\tan(\theta)] \\ \dot{\theta} = q[\cos(\phi)] - r[\sin(\phi)] \\ \dot{\psi} = r\frac{\cos(\phi)}{\cos(\theta)} + q\frac{\sin(\phi)}{\cos(\theta)} \end{cases} \tag{2-8}$$

2.2.5 Dynamics equations

Dynamics equations describe the movement of the quadcopter under the action of external forces and torques. The solution of these equations supplies a description of the position, the motion, and the acceleration of the quadcopter as a function of time. In this section, Newton's second law and the Euler moment equation are used to derive the translational and rotational dynamics equations of the quadcopter, respectively. The two equations are written as follows:

$$F_b = m(\omega_b \wedge V_b + \dot{V}_b) \tag{2-9}$$

$$M_b = I \cdot \dot{\omega}_b + \omega_b \wedge (I \cdot \omega_b) \tag{2-10}$$

In the work, F_b is the vector containing the total force applied to the quadcopter in the body frame, m is the total mass of the quadcopter, $\omega_b \wedge V_b$ represents the centrifugal acceleration, I is the inertial matrix of the quadcopter and $M_b = [M_x \quad M_y \quad M_z]^T$ is the vector containing the total torques applied to the quadcopter in the body frame. Notation $\wedge$ represents cross-product. The inertial matrix I is defined as follows:

$$I = \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix}$$

Where I_x, I_y and I_z are the moment of inertia along the axes of the body coordinate frame, the inertia matrix is assumed to be diagonal due to the quadcopter being assumed to be perfect symmetry to the body coordinate frame.

The dynamics equations of the quadcopter can be described as:

$$\begin{cases} F_x = m(\dot{u} + qw - rv) \\ F_y = m(\dot{v} + ru - pw) \\ F_z = m(\dot{w} + pv - qu) \\ M_x = I_x\dot{p} + (I_z - I_y)qr \\ M_y = I_y\dot{q} + (I_x - I_z)pr \\ M_z = I_z\dot{r} + (I_y - I_x)pq \end{cases} \tag{2-11}$$

It should be noted that all quantities in dynamics equations are expressed in the body coordinate frame, and kinematics equations in Equation 2-8 must be used to derive the position and attitude of the quadcopter in the inertial coordinate frame.

The total force applied to the quadcopter is made up of two components: the gravity and the thrust generated by the four motors. The total force represented in the body coordinate frame can be given by:

$$F_b = mgR_{body}^{NED\ T} \cdot \hat{e}_z + F_t \tag{2-12}$$

Where $\hat{e}_z$ is the unit vector of the z axis of the inertial coordinate frame, g is the gravitational acceleration, and F_t is the total thrust provided by the four motors in the body coordinate frame. Note that in this work the forces due by the wind and other disturbances are neglected. The total thrust generated by the four motors can be written as:

$$F_t = \begin{bmatrix} 0 \\ 0 \\ -F_t \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -(T_1 + T_2 + T_3 + T_4) \end{bmatrix} \tag{2-13}$$

Where T_1, T_2, T_3, T_4 are thrusts generated by the four motors, respectively. It should be noted that the four motors are assumed to produce force only in the z axis direction of the body coordinate frame.

Grouping Equation 2-11, Equation 2-12, Equation 2-13 and isolating $\dot{u}, \dot{v}, \dot{w}$ on the left side of the equation, the result is:

$$\dot{V}_b = \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} rv - qw - gsin(\theta) \\ pw - ru + gsin(\phi)\cos(\theta) \\ qu - pv + gcos(\phi)\cos(\theta) - \frac{F_t}{m} \end{bmatrix} \tag{2-14}$$

Lastly, isolating $\dot{p}, \dot{q}, \dot{r}$ on the left side of Equation 2-11, we can obtain:

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \frac{I_y - I_z}{I_x} qr + \frac{M_x}{I_x} \\ \frac{I_z - I_x}{I_y} pr + \frac{M_x}{I_y} \\ \frac{I_x - I_y}{I_z} pq + \frac{M_z}{I_z} \end{bmatrix} \tag{2-15}$$

The total torque $M_b = [M_x \quad M_y \quad M_z]^T$ applied to the quadcopter in the body coordinate frame is given by:

$$M_b = \tau_b + g_m \tag{2-16}$$

Where $\tau_b = [\tau_x \quad \tau_y \quad \tau_z]^T$ is the vector containing the control torques provided by the quadcopter's four motors, and $g_m = [g_{mx} \quad g_{my} \quad g_{mz}]^T$ is the vector containing the gyroscopic

torque caused by the combined rotation of the four motors and the quadcopter body. The vector τ_b can be calculated using the following equation referring to Figure 2.7:

$$\begin{bmatrix}\tau_x\\ \tau_y\\ \tau_z\end{bmatrix} = \begin{bmatrix}(T_1 - T_2 - T_3 + T_4)Lsin(\alpha)\\ (T_1 + T_2 - T_3 - T_4)Lcos(\alpha)\\ -C_1 + C_2 - C_3 + C_4\end{bmatrix} \quad (2\text{-}17)$$

Where L is the distance between the quadcopter's center of gravity and the motor, α is defined as in Figure 2.7, and C_1, C_2, C_3, C_4 are the reactive torques generated by the four motors in z axis direction of the body coordinate frame, respectively. The gyroscopic torque g_m is given by the following relation:

$$g_m = \sum_{i=1}^{4} J_p(\omega_b \wedge \hat{e}_3)(-1)^{i+1}\Omega_i \quad (2\text{-}18)$$

Where J_p is the moment of inertia of the propellers, $\hat{e}_3$ is the unit vector in z axis of the body coordinate frame, and Ω_i is the angular velocity of the i^{th} motor. The gyroscopic effect is only taken into consideration in the lightweight construction quadcopter. In this project, the torques due to the wind or any other disturbances and the gyroscopic effect are not taken into consideration, as a result, the total torque $[M_x \quad M_y \quad M_z]^T$ is equal to the control torques generated by the motor $[\tau_x \quad \tau_y \quad \tau_z]^T$.

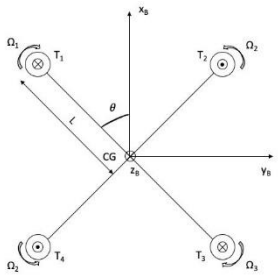


Figure 2.7 Motors configuration of the quadcopter in this project

2.2.6 State-space representation and linearization

Organizing all the equations presented in the previous section, the state-space representation of the quadcopter mathematical model can be written as a nonlinear differential equation of this form:

$$\dot{x} = f(x,u) = f(x) + g(x) \cdot u \quad (2\text{-}19)$$

This kind of representation is convenient when dealing with control theories like feedback linearization (FL) and sliding mode control (SMC). x and u are the state variable and input of the system, respectively. Defining x and u as:

$$x = [x \quad y \quad z \quad \phi \quad \theta \quad \psi \quad u \quad v \quad w \quad p \quad q \quad r]^T \in \mathbb{R}^{12}$$
$$u = [-F_t \quad M_x \quad M_y \quad M_z]^T \in \mathbb{R}^4 \quad (2\text{-}20)$$

Grouping Equation 2-8, Equation 2-14, and Equation 2-15 together and rewriting the result following Equation 2-19, it is possible to express the dynamical model in the state-space format like this:

$$\dot{x} = f(x) + g(x) \cdot u$$

$$f(x)=\begin{bmatrix} R_{body}^{NED}\begin{bmatrix}u\\v\\w\end{bmatrix} \\ p+\tan(\theta)\,[qsin(\phi)+rcos(\phi)] \\ qcos(\phi)-rsin(\phi) \\ \frac{1}{\cos(\theta)}[qsin(\phi)+rcos(\phi)] \\ rv-qw-gsin(\theta) \\ pw-ru+gsin(\phi)\cos(\theta) \\ qu-pv+gcos(\phi)\cos(\theta) \\ \frac{I_y-I_z}{I_x}rq \\ \frac{I_z-I_x}{I_y}pr \\ \frac{I_x-I_y}{I_z}pq \end{bmatrix} \quad g(x)=\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{m} & 0 & 0 & 0 \\ 0 & \frac{1}{I_x} & 0 & 0 \\ 0 & 0 & \frac{1}{I_y} & 0 \\ 0 & 0 & 0 & \frac{1}{I_z} \end{bmatrix} \tag{2-21}$$

Another kind of mathematical model of the quadcopter is presented below. The procedure of the Newton-Euler formalism modeling is to project thrust forces acting on the quadcopter to the inertial coordinate frame and analyze the translational dynamics in the inertial coordinate system and the rotational dynamics in the body coordinate systems. In the inertial coordinate frame, the centrifugal force is nullified, only the gravitational force and the thrust are contributing to the acceleration of the quadcopter. The translational dynamics equations can be written as:

$$m\ddot{\xi}=mg+R_{body}^{NED}F_t$$

$$\begin{bmatrix}\ddot{x}\\\ddot{y}\\\ddot{z}\end{bmatrix}=g\begin{bmatrix}0\\0\\1\end{bmatrix}-\frac{F_t}{m}\begin{bmatrix}\cos(\phi)\sin(\theta)\cos(\psi)+\sin(\phi)\sin(\psi)\\\cos(\phi)\sin(\theta)\sin(\psi)-\sin(\phi)\cos(\psi)\\\cos(\phi)\cos(\theta)\end{bmatrix} \tag{2-22}$$

Grouping Equation 2-8, Equation 2-15, and Equation 2-22 together, the dynamics equations can be written as:

$$\begin{cases} \ddot{x}=-\frac{F_t}{m}[cos(\phi)\,sin(\theta)\,cos(\psi)+si\,n(\phi)\,si\,n(\psi)] \\ \ddot{y}=-\frac{F_t}{m}[cos(\phi)\,sin(\theta)\,sin(\psi)-sin(\phi)\,co\,s(\psi)] \\ \ddot{z}=g-\frac{F_t}{m}cos(\phi)\,cos\,(\theta) \\ \dot{\phi}=p+r[cos(\phi)\,tan(\theta)]+q[sin(\phi)\,tan(\theta)] \\ \dot{\theta}=q[cos(\phi)]-r[sin(\phi)] \\ \dot{\psi}=r\frac{cos\,(\phi)}{cos\,(\theta)}+q\frac{sin\,(\phi)}{cos\,(\theta)} \\ \dot{p}=\frac{I_y-I_z}{I_x}qr+\frac{M_x}{I_x} \\ \dot{q}=\frac{I_z-I_x}{I_y}pr+\frac{M_x}{I_y} \\ \dot{r}=\frac{I_x-I_y}{I_z}pq+\frac{M_z}{I_z} \end{cases} \tag{2-23}$$

Defining the state vector and input as:

$$x=[x\quad y\quad z\quad \phi\quad \theta\quad \psi\quad \dot{x}\quad \dot{y}\quad \dot{z}\quad p\quad q\quad r]^T\in\mathbb{R}^{12}$$

$$u=[-F_t\quad M_x\quad M_y\quad M_z]^T\in\mathbb{R}^4$$

It is possible to write the equations in state-space form:

$$\dot{x}=f(x)+g(x)\cdot u$$

$$f(x) = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ p + tan(\theta)\,[qsin(\phi) + rcos(\phi)] \\ qcos(\phi) - rsin(\phi) \\ \frac{1}{cos(\theta)}[qsin(\phi) + rcos(\phi)] \\ 0 \\ 0 \\ g \\ \frac{I_y - I_z}{I_x} qr \\ \frac{I_z - I_x}{I_y} pr \\ \frac{I_x - I_y}{I_z} pq \end{bmatrix} \quad g(x) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ U_x & 0 & 0 & 0 \\ U_y & 0 & 0 & 0 \\ U_z & 0 & 0 & 0 \\ 0 & \frac{1}{I_x} & 0 & 0 \\ 0 & 0 & \frac{1}{I_y} & 0 \\ 0 & 0 & 0 & \frac{1}{I_z} \end{bmatrix} \tag{2-24}$$

Where:

$$U_x = \frac{1}{m}[cos(\phi)\,sin(\theta)\,cos(\psi) + si\,n(\phi)\,si\,n(\psi)]$$

$$U_y = \frac{1}{m}[cos(\phi)\,sin(\theta)\,sin(\psi) - sin(\phi)\,co\,s(\psi)]$$

$$U_z = \frac{1}{m}[cos(\phi)\,co\,s(\theta)]$$

The mathematical model of the quadcopter is a quite complex system due to its nonlinearity, which makes it difficult to gain an accurate insight into the quadcopter's behavior and to troubleshoot control systems in simulation. To develop basic control strategies like linear control, several assumptions that reduce the complexity of the non-linear equations are required. Based on these assumptions, the linearized mathematical model of the quadcopter is derived, and then its control strategy can be formulated. In this project, the Linear Quadratic Regulator (LQR) control strategy is adopted, and the controller parameters are tuned based on the linearized model. The designed controller can achieve acceptable control performance to some extent when deployed on the nonlinear model. In future works, the complexity of the system model can be increased, and then more advanced nonlinear control strategies such as Feedback Linearization Control, Sliding Mode Control (SMC), and Nonlinear Model Predictive Control (NMPC) can be deployed to implement much more precise control of the quadcopter.

The assumption made to simplify the model is that the quadcopter will be operated around a stable hover condition with small attitude angles and minimal rotational and translational velocities and accelerations. These assumptions are described by the mathematically functions below:

$$\begin{gathered} \dot{x} = \dot{y} = \dot{z} = 0 \\ \dot{\phi} = \dot{\theta} = \dot{\psi} = 0 \\ \phi = \theta = 0 \\ \psi = 0 \end{gathered} \tag{2-25}$$

The linearization is made by approximating the sine function with its argument and the cosine function with unity. The resulting simplified and linearized equations are:

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} \approx \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \approx \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{2-26}$$

$$\begin{bmatrix}\ddot{x}\\ \ddot{y}\\ \ddot{z}\end{bmatrix}=\begin{bmatrix}-\frac{F_t}{m}[\theta\cos(\psi)+\phi\sin(\psi)]\\ -\frac{F_t}{m}[\theta\sin(\psi)-\phi\cos(\psi)]\\ g-\frac{F_t}{m}\end{bmatrix}=\begin{bmatrix}-\frac{F_t}{m}\theta\\ \frac{F_t}{m}\phi\\ g-\frac{F_t}{m}\end{bmatrix}\approx\begin{bmatrix}-g\theta\\ g\phi\\ g-\frac{F_t}{m}\end{bmatrix} \tag{2-27}$$

$$\begin{bmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{bmatrix}=\begin{bmatrix}\frac{1}{I_x}M_x\\ \frac{1}{I_y}M_y\\ \frac{1}{I_z}M_z\end{bmatrix} \tag{2-28}$$

It should be noted that in Equation 2-27 the translational movement has been decoupled from the attitude assuming the yaw angle remains around zero degrees. With this assumption, movement along the global x and y axis can be controlled by independently controlling the pitch and roll angles, respectively. Also, the total thrust F_t is almost equal to the gravity for a quadcopter to maintain a stable altitude during hovering. The simplified equations of the quadcopter model by linearization are:

$$\begin{cases}\dot{x}=\dot{x}\\ \dot{y}=\dot{y}\\ \dot{z}=\dot{z}\\ \dot{\phi}=p\\ \dot{\theta}=q\\ \dot{\psi}=r\\ \ddot{x}=-g\theta\\ \ddot{y}=g\phi\\ \ddot{z}=g-\frac{F_t}{m}\\ \dot{p}=\frac{M_x}{I_x}\\ \dot{q}=\frac{M_y}{I_y}\\ \dot{r}=\frac{M_z}{I_z}\end{cases} \tag{2-29}$$

Once a linear model of the quadcopter dynamics is obtained, and letting matrix $C=I_{12}$ be the output matrix, the model can be written in the state-space form as follows:

$$\begin{cases}\dot{x}=Ax+Bu\\ y=Cx\end{cases} \tag{2-30}$$

The matrices associated with the linear model are given by relations:

$$A=\begin{bmatrix}0&0&0&0&0&0&1&0&0&0&0&0\\ 0&0&0&0&0&0&0&1&0&0&0&0\\ 0&0&0&0&0&0&0&0&1&0&0&0\\ 0&0&0&0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&-g&0&0&0&0&0&0&0\\ 0&0&0&g&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0\end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{m} & 0 & 0 & 0 \\ 0 & \frac{1}{I_x} & 0 & 0 \\ 0 & 0 & \frac{1}{I_y} & 0 \\ 0 & 0 & 0 & \frac{1}{I_z} \end{bmatrix} \tag{2-31}$$

It should be noted that the input vector u for Equation 2-30 should be defined as:

$$u = [mg - F_t \quad M_x \quad M_y \quad M_z]^T \in \mathbb{R}^4$$

Chapter 3: Controller design and simulation

In this chapter, the flight controller structure and the control strategy are discussed. The control strategy utilized to design the flight controller is the Linear Quadratic Regular (LQR) control. The linearized mathematical model derived above is used in the LQR controller design. From the mathematical model derived above, the rotational motions of the quadcopter are independent of translational motions and full actuated, while translational motions are underactuated and depend heavily on the rotational motions. Therefore, the cascade controller with an inner-outer loop structure can better control the quadcopter. The inner loop is related to the fast dynamics of the quadcopter, which generates control signals to control the attitude and altitude, while the outer loop is related to the slow dynamics of it, which controls the position in the $X-Y$ horizontal plane. The structure of this cascade controller is shown in Figure 3.1 below.

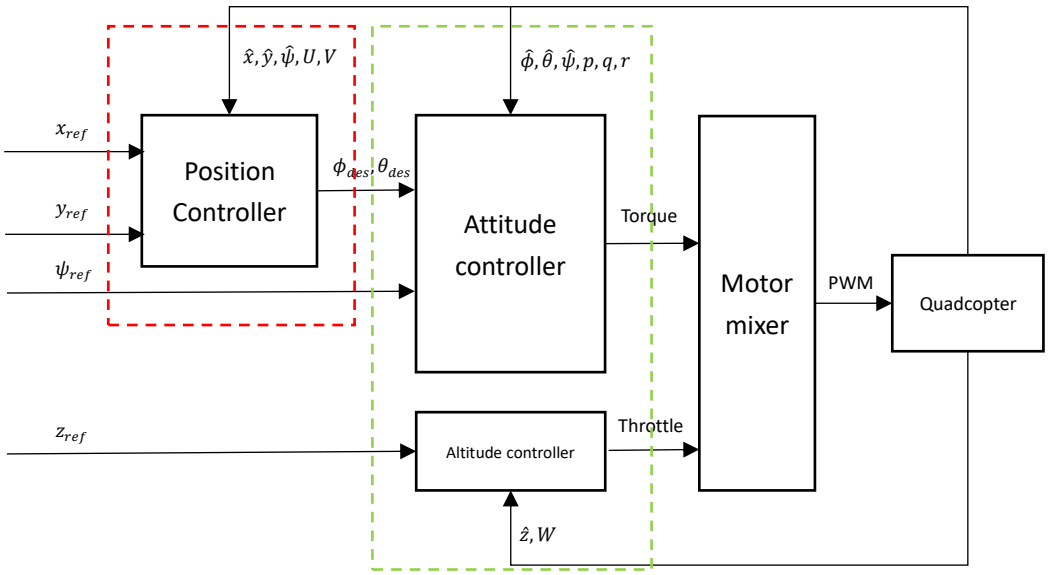


Figure 3.1 Structure of the flight controller

3.1 Linear Quadratic Regulator Control

The Linear Quadratic Regular (LQR) control, in the context of optimal control, uses full state feedback to determine the control signal to bring the system's state $x(t)$ to the desired reference $r(t)$ while at the same time minimizing some cost index. Furthermore, LQR control can minimize the control inputs, thus reducing the use of actuators. For the Linear Time-Invariant system described in the state-space representation:

$$\begin{cases} \dot{x} = Ax + Bu \\ y = Cx \end{cases} \tag{3-1}$$

The cost index is a function of the state variable $x(t)$ and controller signal $u(t)$.

$$J(x,u) = \frac{1}{2}\int_0^\infty (x(t)^T Q x(t) + u(t)^T R u(t))dt \tag{3-2}$$

Where matrix Q weights the cost of the system state and matrix R weights the cost of actuators. These weighting matrices determine the relative importance of the existing state error as well as the energy expenditure of the system.

The full state feedback control law $u(t)$ which minimizes the cost index function is:

$$u(t) = -Kx(t) \tag{3-3}$$

The computation of the optimal gains is performed through the function:

$$K = R^{-1}B^T P \tag{3-4}$$

Where the positive-definite matrix P results from the steady-state Riccati equation:

$$A^T P + PA + Q - PBR^{-1}B^T P = 0 \tag{3-5}$$

The closed-loop system represented as $\dot{x} = (A - BK)x$ is asymptotically stable with the optimal gain computed by Equation 3-4. Under MATLAB, the algebraic Riccatic equation can be solved by the *lqr* function:

$$K = lqr(A, B, Q, R) \tag{3-6}$$

LQR requires a linear model to get an adequately controlled system, and it can handle multiple inputs and outputs simultaneously and offer a fast response, but unlike PID control, LQR control often provides static error during tracking due to the lack of an integral part. To overcome this problem, an integrator can be included in the control structure to eliminate the static error and stabilize the system.

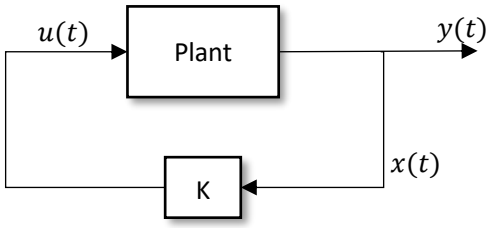


Figure 3.2 LQR control scheme

One of the key questions in LQR controller design is how to choose the weighting matrices Q and R. $Q \in \mathbb{R}^{n\times n}$ is a positive semi-definite matrix that penalizes the state variables and $R \in \mathbb{R}^{m\times m}$ is a positive definite matrix that penalizes the control signals. A simple choice is to use a diagonal weighting matrix:

$$Q = \begin{bmatrix} q_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & q_n \end{bmatrix} \quad R = \begin{bmatrix} \rho_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \rho_m \end{bmatrix} \tag{3-7}$$

For choosing Q and R, the individual diagonal elements describe how much each state and input should contribute to the overall cost index. The larger these values are, the more the state variables and input signals are penalized. Choosing a large value for R means stabilizing the system with less (weighted) energy. This is usually called an expensive control strategy. On the other hand,

choosing a small value for R means the energy consumption is little considered (cheap control strategy). Similarly, choosing a large value for Q means that the state variables should remain small and a small value for Q implies less concern about the state variables. Hence, the tuning of weight matrices Q and R is a trade-off between the control performance and the energy consumption.
The simplest method for choosing the elements of Q and R matrix is to make $Q = I, R = \rho I \Rightarrow J = \|x\|^2 + \rho\|u\|^2$, and vary ρ to get something that has a good response. Another method for choosing the individual weights q_i and ρ_i can be done by deciding on a weighting of the errors from the individual terms. Bryson and Ho [40] have suggested the following method for choosing the matrices Q and R: (1) choosing q_i and ρ_i as the inverse of the square of the maximum value for the corresponding x_i or u_i; (2) modify the q_i and ρ_i to obtain a compromise among response time, overshoot, damping, and control effort. The second step can be performed by trial and error.

3.1.1 Altitude controller

The altitude controller belongs to the inner loop part of the flight controller as shown in Figure 3.1. The altitude controller generates the thrust command to control the quadcopter's position along z axis of the inertial coordinate frame (altitude). The unit of the thrust command is Newton [N], in this way, the designing and the turning of the controller are more intuitive. LQR control strategy is used to design the altitude controller, and very good control performance is achieved. The linearized mathematical model (Equation 2-29) in chapter 2 is used for the LQR controller design. For the altitude subsystem, the related differential equation is:

$$\ddot{z} = g - \frac{F_t}{m} \tag{3-8}$$

To write the differential equation for altitude to the format $\dot{x} = Ax + Bu$, the state variables and the input of the subsystem are defined as:

$$x_z = [z \quad \dot{z}]^T \quad u_z = mg - F_t \tag{3-9}$$

Then the state-space representation of the altitude subsystem is:

$$\dot{x}_z = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} x_z + \begin{bmatrix} 0 \\ 1/m \end{bmatrix} u_z$$

$$z = [1 \quad 0] x_z \tag{3-10}$$

After determining the weighting matrix $Q_z \in \mathbb{R}^{2\times2}$ and $R_z \in \mathbb{R}$, the feedback gain $K_z \in \mathbb{R}^{1\times2}$ for the altitude control can be computed by MATLAB command *lqr*, and the control law for the altitude can be expressed as:

$$F_t = -1 \cdot (K_z \cdot \begin{bmatrix} z_r - z \\ \dot{z}_r - \dot{z} \end{bmatrix} - mg) \tag{3-11}$$

The block diagram of the altitude controller is shown in Figure 3.3 below:

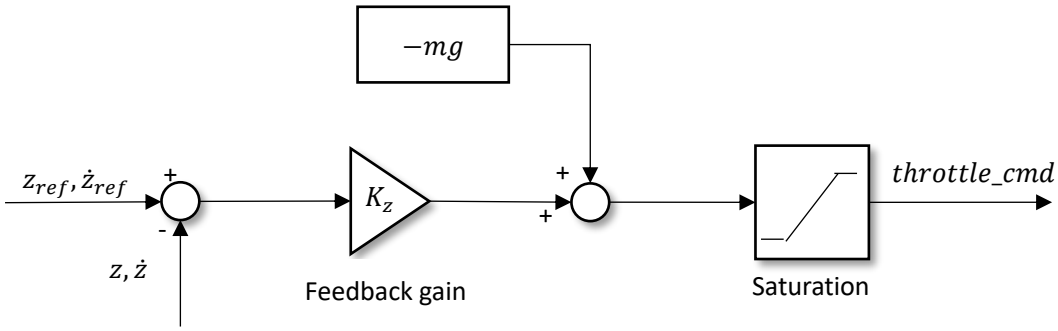


Figure 3.3 Block diagram of altitude controller

Where the $-mg$ is the feedforward gravity term, this is the amount of thrust needed to offset the weight of the quadcopter so that the LQR controller just generates a positive thrust to make the quadcopter go down and a negative thrust to make it go up (The sign is opposite here since we use NED coordinate frame, up is the opposite direction of the coordinate frame). To limit the control command to the feasible values the quadcopter's motor can generate, a saturation block is included in the controller. The limitations are set according to the characteristics of the actuator, and they are listed in Table 3.2 below.

3.1.2 Attitude controller

The Attitude controller is related to the rotational dynamics of the quadcopter, which belongs to the inner loop part of the flight controller. The attitude controller computes the desired torques implied to the quadcopter in the body coordinate frame to control the roll, pitch, and yaw angles for tracking the desired references. The unit of the torque commands is Newton-meters [Nm], in this way the designing and tuning of the controller will be more intuitive. The LQR control strategy is used, and the designed controller can achieve zero static error. The linearized mathematical model Equation 2-29 derived in chapter 2 is used for controller design. For the attitude subsystem, the related differential equations are:

$$\begin{cases} \dot{\phi} = p \\ \dot{p} = \frac{M_x}{I_x} \\ \dot{\theta} = q \\ \dot{q} = \frac{M_y}{I_y} \\ \dot{\psi} = r \\ \dot{r} = \frac{M_z}{I_z} \end{cases} \tag{3-12}$$

To write the equations in state-space format, the state vector x_a and input vector u_a for this subsystem are defined as:

$$x_a = [\phi \quad \theta \quad \psi \quad p \quad q \quad r]^T \quad u_a = [M_x \quad M_y \quad M_z]^T \tag{3-13}$$

The state-space representation of the subsystem is:

$$\dot{x}_a = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} x_a + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1/I_x & 0 & 0 \\ 0 & 1/I_y & 0 \\ 0 & 0 & 1/I_z \end{bmatrix} u_a \tag{3-14}$$

$$y_a = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} x_a \tag{3-15}$$

After determining the weighting matrix $Q_a \in \mathbb{R}^{6\times6}$ and $R_a \in \mathbb{R}^{3\times3}$, the feedback gain $K_a \in \mathbb{R}^{3\times6}$ can be computed by solving the algebraic Riccatic equation with MATLAB command *lqr*. The feedback gain K_a is a 3×6 matrix, and the control law for the attitude subsystem can be expressed by:

$$u_a = \begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix} = K_a \begin{bmatrix} \phi_r - \phi \\ \theta_r - \theta \\ \psi_r - \psi \\ -p \\ -q \\ -r \end{bmatrix} \tag{3-16}$$

The block diagram of the attitude controller is shown below:

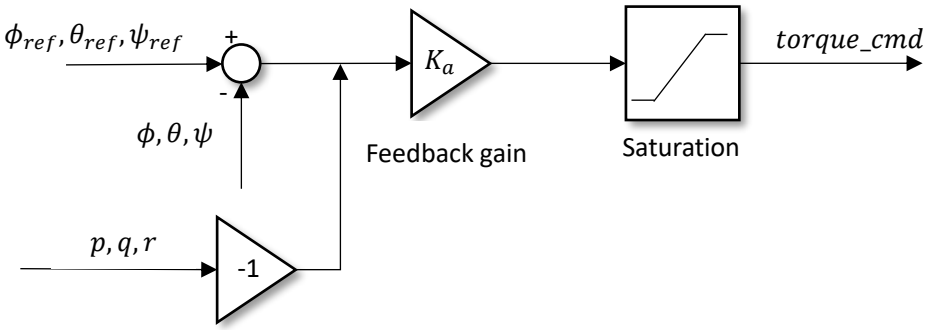


Figure 3.4 Block diagram of the attitude controller

A saturation block is included in the controller to limit the computed torque command to the feasible values the quadcopter's motor can generate. The maximum and minimum values are set according to the characteristics of the motors and the arm length of the quadcopter, and they are listed in Table 3.2 below.

3.1.3 Position controller

The outer loop part of the controller refers to the position control of the quadcopter on the $X-Y$ horizontal plane. This loop takes the quadcopter's current heading angle ψ, current position, velocity, and the position reference signal x_r and y_r as inputs and computes the desired roll angle

ϕ and pitch angle θ for tracking the reference position. The unit of the position controller outputs is the radian. In this case, the turning of the controller is more intuitive. The linearized mathematical model derived in chapter 2 is used for designing the LQR controller and the differential equations related to the x and y positions are:

$$\begin{cases} \ddot{x} = -g\theta \\ \ddot{y} = g\phi \end{cases} \tag{3-17}$$

Defining the state variables and the input of the subsystem as:

$$x_p = [x \quad y \quad \dot{x} \quad \dot{y}]^T \quad u_p = [\theta \quad \phi]^T \tag{3-18}$$

The subsystem represented in state-space form is:

$$\dot{x}_p = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} x_p + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ -g & 0 \\ 0 & g \end{bmatrix} u_p \tag{3-19}$$

$$y_p = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} x_p \tag{3-20}$$

After determining the weighting matrix Q_p and R_p, the full state feedback gain K_p for the position control can be computed by MATLAB *lqr* command. The control law for the position control can be expressed by:

$$\begin{bmatrix} \theta_r \\ \phi_r \end{bmatrix} = K_p \begin{bmatrix} x_r - x \\ y_r - y \\ \dot{x}_r - \dot{x} \\ \dot{y}_r - \dot{y} \end{bmatrix} \tag{3-21}$$

The output of the out-loop controller θ_r and ϕ_r are the desired pitch and roll angles for the quadcopter to track the desired reference, which are the input of the attitude controller part in the inner loop.

The position controller is derived from the model linearizing around a hover point where the yaw angle ψ is assumed to be zero. The movement of the quadcopter in the x axis is decoupled from the roll angle and the movement in the y axis is decoupled from the pitch angle under this assumption, which means giving a roll angle to the quadcopter, it only has a movement in y axis, and giving a pitch angle, it only has a movement in x axis. This simplifies the computation of the desired roll and pitch angles for tracking the reference position. In case when the yaw angle ψ is not zero, the movements of the quadcopter in the xy-plane are related to both roll and pitch angles. A roll angle will not only force the quadcopter to move in y axis but also x axis, and the same is for the pitch angle. To decouple the movement with the attitude as if the yaw angle ψ is zero, a rotation must be considered. The rotation matrix applied to the x and y axes of the body coordinate frame is defined as:

$$R_{body-hover} = \begin{bmatrix} \cos(\psi) & \sin(\psi) \\ -\sin(\psi) & \cos(\psi) \end{bmatrix} \tag{3-22}$$

The block diagram of the position controller is shown below:

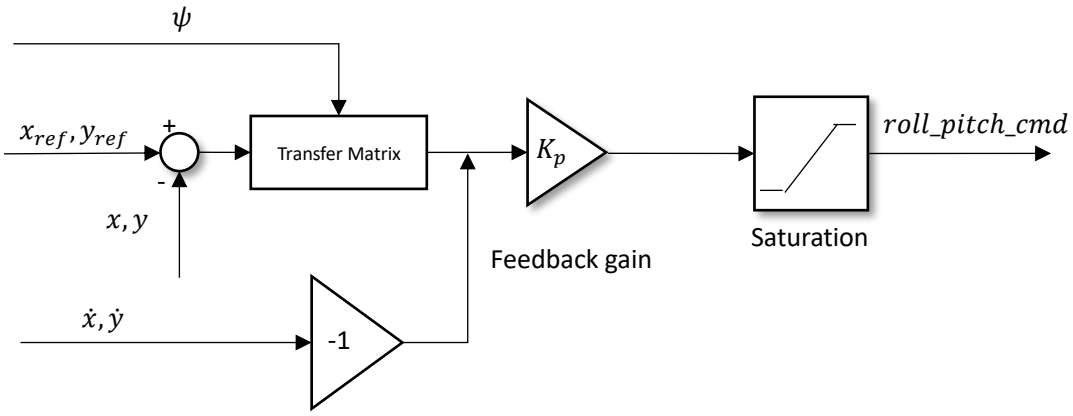


Figure 3.5 Block diagram of the position controller

A saturation is included in the controller to limit computed roll and pitch angle since too large angles make the quadcopter away from the equilibrium point at which the linearization is performed. The control performance of the linear controller will be less effective, leading to the instability of the quadcopter. The limit values are set to $\pm 30°$ when performing the simulation.

3.2 Motor mixer

Grouping Equation 3-11 and Equation 3-16 together, we can get the physical control commands (thrust and moments). But such physical control commands are not enough for implementing the control action to a hardware platform since the real quadcopter platform takes the Power-Width Modulation (PWM) signal as input. The Electronic Speed Controllers of the quadcopter take the PWM signals as input to control and regulate the speeds of the motors, leading to the control action to the quadcopter. To generate the control command that can be used by motor ESC, the transformation between the physical values (thrust and moments) and PWM pulse width values must be implemented.

In this project, the gyroscopic effect of the motor and the torque generated by wind, or any other disturbance are not taken into consideration, so the total torques implied to the quadcopter are completely supplied by the motors. Considering the motor configuration as shown in Figure 3.6, the relationship between physical control commands F_t, M_x, M_y, M_z and the variation of the thrust vector $\Delta T = [\Delta T_f \quad \Delta T_x \quad \Delta T_y]^T$ and torque ΔC can be expressed by:

$$\begin{cases} \Delta T_f = \frac{F_t}{4} \\ \Delta T_x = \frac{M_x}{4Lsin(\alpha)} \\ \Delta T_y = \frac{M_y}{4Lcos(\alpha)} \\ \Delta C = \frac{M_z}{4} \end{cases} \tag{3-23}$$

The first component of the variation vector ΔT_f represents the variation of the thrust required by

each motor to provide the quadcopter with a control force along the z axis of the body coordinate frame, the second component ΔT_x represents the variation of the thrust required by each motor to provide the quadcopter with a control torque along the x axis of the body coordinate frame, and the last component ΔT_y represents the variation of the thrust required by each motor to provide the quadcopter with a control torque along the y axis of the body coordinate frame. The torque variation ΔC represents the variation of the reactive torque required by each motor to provide the quadcopter with a control moment along the z axis of the body coordinate frame.

The relationship between the PWM signal pulse width variations which defined as $\Delta PWM = [\Delta PWM_f \quad \Delta PWM_x \quad \Delta PWM_y \quad \Delta PWM_z]^T$ and the thrust and torque variations ΔT, ΔC need to be found. The first component ΔPWM_f represents the change of the PWM signal pulse width required by each motor to generate control thrust, the second component ΔPWM_x represents the change of the PWM signal pulse width required by each motor to generate the control torque around the x axis, the third component ΔPWM_y represents the PWM pulse width change of each motor to generate control torque around y axis, and the last component ΔPWM_z is the requested PWM signal pulse width change for each motor to generate control torque around z axis.

To find the relationship between ΔPWM, ΔT, and ΔC, the propeller test to find the thrust and torque generated by one motor with respect to the PWM signal pulse width is required. The propeller test can be deployed on a test stand where the torque and thrust generated by the motor can be measured. More detailed information about the propeller test can be found in Appendix A Experimental measurements. The propeller test shows that the thrust and torque generated by the motor have a linear relationship with the PWM pulse width value. Hence the relationship between ΔT, ΔC, and ΔPWM can be expressed by:

$$\begin{cases} \Delta T = n_F \cdot \Delta PWM \\ \Delta C = n_T \cdot \Delta PWM \end{cases} \tag{3-24}$$

Where n_F is the slope of the linear relationship between the thrust generated by each motor and the PWM pulse width value, and n_T is the slope of the linear relationship between the reactive torque of each motor and the PWM pulse width value. Thus, the relationship between the physical control command (thrust and torque) and the PWM pulse width value can be expressed as follows:

$$\begin{cases} F_t = 4n_F \Delta PWM_f \\ M_x = 4n_F L sin(\alpha) \Delta PWM_x \\ M_y = 4n_F L cos(\alpha) \Delta PWM_y \\ M_z = 4n_T \Delta PWM_z \end{cases} \tag{3-25}$$

The motor mixer block determines how the four motors of the quadcopter work together to generate the control action. Figure 3.6 presents the motor configuration of the quadcopter considering the body coordinate frame. According to the motor configuration, the 1^{th} motor contributes to the positive thrust, positive torque around x axis, positive torque around y axis, and negative torque around z axis. The motor mixer equation can be simply derived by a linear combination of the required thrust and torques. The same analytical method can be used to derive the motor mixer equations of the other motors. Denoting $PWM_{(i)}$ the PWM pulse width of the i^{th} motor, the motor mixer equations are:

$$\begin{aligned} PWM_{(1)} &= \Delta PWM_f + \Delta PWM_x + \Delta PWM_y - \Delta PWM_z + idel \\ PWM_{(2)} &= \Delta PWM_f - \Delta PWM_x + \Delta PWM_y + \Delta PWM_z + idel \\ PWM_{(3)} &= \Delta PWM_f - \Delta PWM_x - \Delta PWM_y - \Delta PWM_z + idel \\ PWM_{(4)} &= \Delta PWM_f + \Delta PWM_x - \Delta PWM_y + \Delta PWM_z + idel \end{aligned} \tag{3-26}$$

Where $idel$ is the pulse width set when the quadcopter is armed (usually equal to 1000). The MATLAB Simulink block diagram of the motor mixer is shown in Figure 3.7 below. The value of the PWM pulse width used by the ESC ranges between 1000 (minimum when armed) and 2000 (maximum), and a saturation block is added to limit the output of the motor mixer.

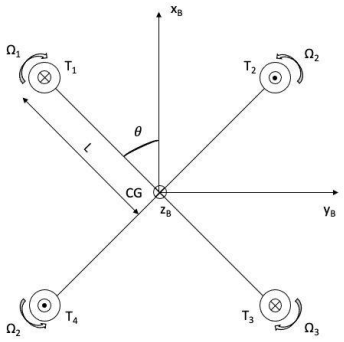


Figure 3.6 Quadcopter's motor configuration

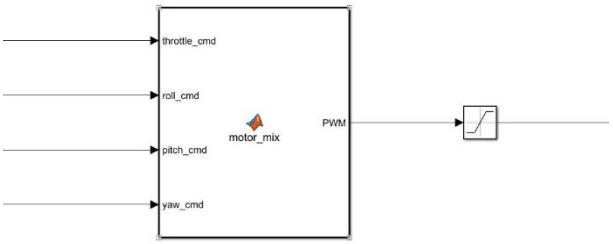


Figure 3.7 Simulink block diagram of motor mixer

3.3 Simulation results

In this section, the complete simulation model is implemented, and the simulation results are analyzed. Simulation is performed to turn and validate the LQR controllers without implementing them directly on the real quadcopter platform, leading to a significant saving on time and material. Firstly, the controller parameters are tuned, and the control performance is evaluated considering the rising time, the settling time, and the overshoot in response to the step input. Then, the trajectory tracking performance of the designed controller is validated and the results are reported and analyzed. In this project, the software used for deploying the simulation model is MATLAB/Simulink R2021b.

3.3.1 Preliminary

To perform the simulation, some parameters such as the mass, and the moment of inertia of the quadcopter must be known. These mechanical parameters can be measured with the method

proposed in [41], and they are listed in Table 3.1 below.

Quadcopter's mechanical parameters	
Parameters	Values
m	$1.7kg$
L	$0.15m$
I_x	$0.023kg \cdot m^2$
I_y	$0.028kg \cdot m^2$
I_z	$0.02kg \cdot m^2$

Table 3.1 Quadcopter's mechanical parameters

The maximum torque and thrust that the quadcopter's motor can generate should be taken into consideration in the simulation. The computed torque and thrust by the controller must be limited to the maximum values that the quadcopter's motor can achieve. Referring to the propeller test in Appendix A Experimental measurements, as well as the quadcopter mechanical parameters, these maximum values are determined and listed in Table 3.2 below.

	Limitation values
Thrust	30N
Torque around x axis	$\pm 1.5Nm$
Torque around y axis	$\pm 1.5Nm$
Torque around z axis	$\pm 0.02Nm$

Table 3.2 Limitations to the thrust and torque

3.3.2 Model implementation

Some assumptions have to be made to implement the mathematical model in the simulation environment:

1. All quadcopter states (attitude, position, linear velocity, and angular velocity) are assumed to be directly known from the mathematical model (the sensor block and the estimation block are not implemented).
2. The aerodynamics due to wind and any other disturbance are not taken into account (the environment simulation is not implemented and the impact of the environment on the quadcopter is not considered during simulation).
3. The control action is performed at 250Hz frequency, which is the same refresh frequency of the flight controller developed to implement on the quadcopter.

The structure of the Simulink model follows the cascade flight controller structure in Figure 3.1, and it mainly consists of five blocks:

1. *Position controller*: this block takes the reference position, actual position, current velocity, and current yaw angle as input, and outputs the roll and pitch angle references, which are the desired roll and pitch angles required by the system for tracking the reference position.

2. *Altitude controller*: this block takes the reference, altitude, and vertical velocity as input, and the throttle command to control the system is given as output.
3. *Attitude controller*: this block takes the reference, attitude angles, and angular velocity as input and computes desired control torques to ensure that the system can track reference roll pitch and yaw angles.
4. *Motor mixer*: the motor mixer block transforms the physical control commands (thrust and torques) computed by the controllers into the Power-Width Modulation (PWM) Duty Cycle (DC).
5. *Plant*: the mathematical models of the actuator and the nonlinear quadcopter model are implemented in this block.

The controller and motor mixer blocks have been previously discussed in section 3.1 and section 3.2, respectively. The plant block includes the nonlinear mathematical model of the quadcopter represented by Equation 2-24 and the actuator model. The block diagram of the plant block is presented in Figure 3.8 below, which takes the PWM signal as input. The actuator block simulates the motors of the quadcopter, which transfers the PWM signal to physical control actions (thrust and torque). To model the actuator, an experimental test is performed on the test stand to find the equations describing the thrust and torque that one brushless motor can generate with respect to the PWM pulse width. More detailed information about the experimental test is presented in Appendix A Experimental measurements. Afterward, the actuator is simulated by a MATLAB function block, and the resulting model provides a relationship between the PWM signal computed by the motor mixer and the thrust and torques the actuator can generate.

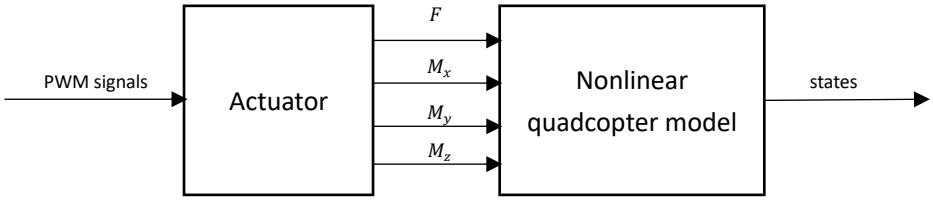


Figure 3.8 Block diagram of the plant

3.3.3 Step response performance

In this section, the controller performance in response to the step signal is simulated. The parameters of the LQR controller are tuned and determined by trial and error, and the designed controller achieves a fast response to commands with a low overshoot and a very small steady-state error. The parameters used in the simulation are listed in Table 3.3 below, and the x, y, z, roll, pitch, and yaw step signal responses are shown in Figure 3.9. The limitations to the minimum and maximum torque and thrust that the actuator can generate are required considering its capability. The limitation values are listed in Table 3.2 above.

Subsystem	Q matrix	R matrix
Attitude control	$diag(0.05,0.07,0.1,0.001,0.001,0.1)$	$diag(1,1,1) * 10^{-6}$
Altitude control	$diag(10,0.5)$	0.01
Position control	$diag(15,15,1,1)$	$diag(1,1) * 500$

Table 3.3 Parameters of LQR controller

The control performance in response to the step signal using the parameters listed in Table 3.3 are depicted in Figure 3.9 below.

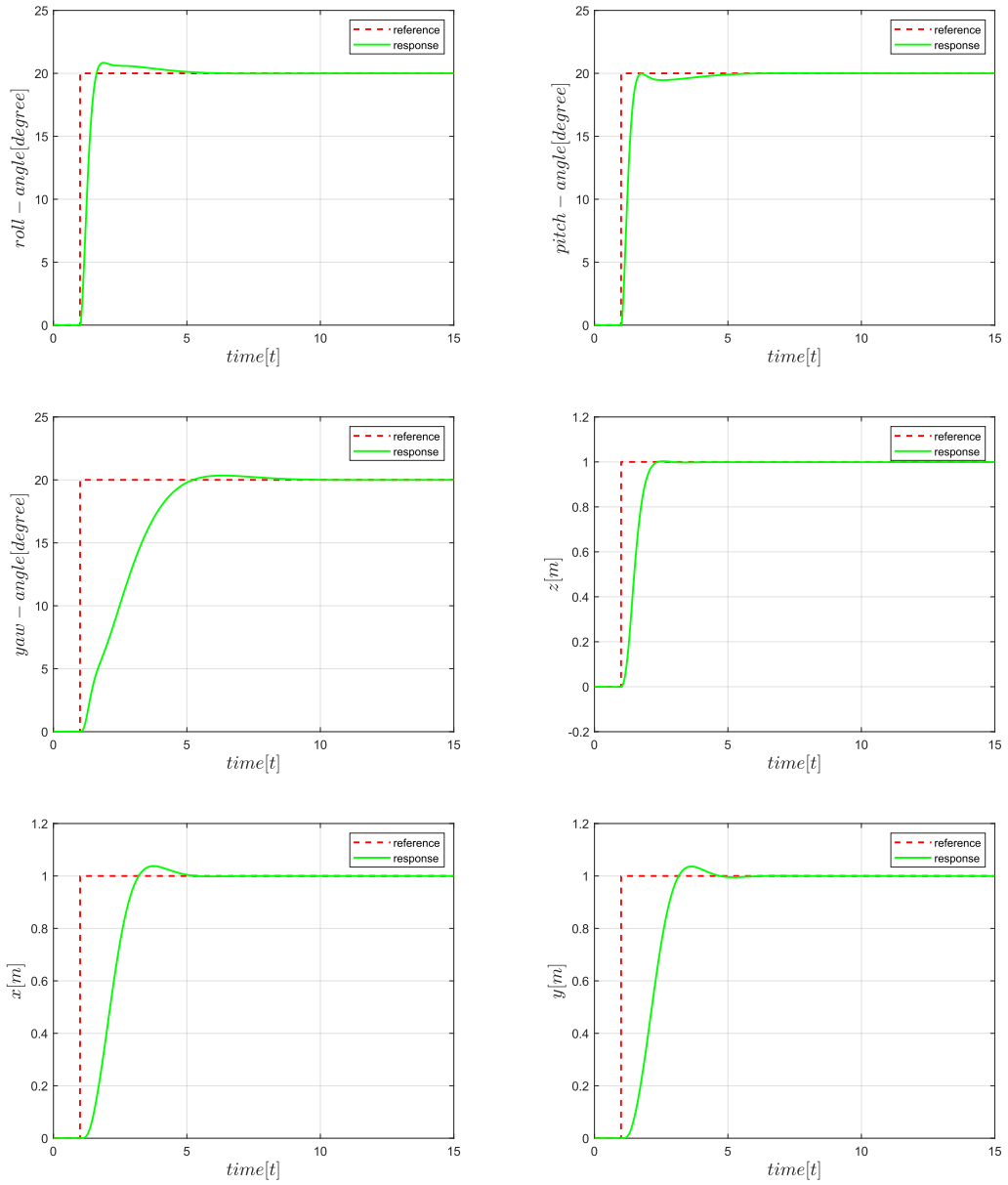


Figure 3.9 Step signal response of the designed controller

The performance of the designed controller is evaluated considering the rising time, settling time, overshoot, and steady-state error. The rising time is defined as the time to reach for the first time the steady-state value of the response. It reflects the responsiveness of the controller. The overshoot is

the manifestation of the response that exceeds the reference. The settling time is defined as the time to reach and stay within $\pm\alpha\%$ of the steady-state value of the response. Typical values for α are 1, 2, and 5. In this project, we choose 2 for α. The steady-state value is defined as the difference between the reference and the steady-state value of the response. The response performance of the controller is shown in Table 3.4 below.

	Roll	Pitch	Yaw	X	Y	Z
Rising time	0.62s	0.76s	4.23s	2.20s	2.16s	1.38s
Overshoot	3.90%	0%	1.72%	3.76%	3.62%	0%
Settling time	2.72s	2.43s	3.84s	3.42s	3.13s	1.16s
Steady-state error	0.0002°	0.0002°	0.0018°	0.00007m	0.000037m	0.0011m

Table 3.4 Controller performance in response to the step signal

The designed controller demonstrates a small rising time and settling time in response to roll and pitch commands, leading to a fast roll and pitch response speed, which benefits the position control. The rising time and settling time of the yaw response are much larger compared to the ones of roll and pitch response. The yaw response speed is much slower since the moment around the z axis generated by the actuator is smaller than the moment around the x or y axis. But it's still enough for autonomous flight. In general, the attitude controller shows a fast response speed, small overshoot, and small steady-state error. The altitude controller also shows a fast response with no overshoot. As for the position controller, it belongs to the out-loop controller and the response speed is slower than the one of the inner-loop controller. Also, it has an overshoot. But the overall control performance is acceptable.

Despite being very small, the designed LQR controller always presents a steady-state error due to the lack of integrator action. The LQR controller takes the state and the state derivative as input to compute the control command, which is similar to a classical PD controller. Unlike the PID controller, the lack of integrator action in the LQR controller leads to a non-zero steady-state error issue. Hence, under the influence of noise and environmental uncertainties, the control performance of the LQR will decrease and even lead to system instability.

3.3.4 Trajectory tracking performance

To evaluate the performance of the proposed controller, a trajectory is given as the reference signal for the controller. The trajectory of the quadcopter can be defined in the four-dimensional space:

$$(x(t), y(t), z(t), \psi(t)) \tag{3-27}$$

Where $x(t)$, $y(t)$, $z(t)$ represent the position of the quadcopter in the inertial coordinate frame over time, $\psi(t)$ represents the yaw angle or heading of the quadcopter over time. In this section, the trajectory is defined with a constant yaw angle equal to zero, hence $\psi(t) = 0$ and the trajectory generation problem can be simplified.

A series of waypoints and corresponding times of arrival are required to generate a trajectory for the quadcopter. With the given information, many methods such as trapezoidal velocity profile,

minimum jerk trajectory generation [42], and minimum snap trajectory generation [43] can be used to generate a trajectory for the quadcopter. In this section, the trapezoidal speed profile is adopted as the trajectory generation method. The trapezoidal velocity profile imposes a constant acceleration in the start phase, a cruise velocity, and a constant deceleration in the arrival phase [44]. The trapezoidal velocity profile can be described in Figure 3.10 below. The segment between two waypoints has three phases. In the start phase, The speed increases steadily until it reaches the maximum value. Then, In the constant velocity phase, the acceleration is zero and the velocity remains constant with maximum value. In the arriving phase, the velocity decreases to zero with constant deacceleration.

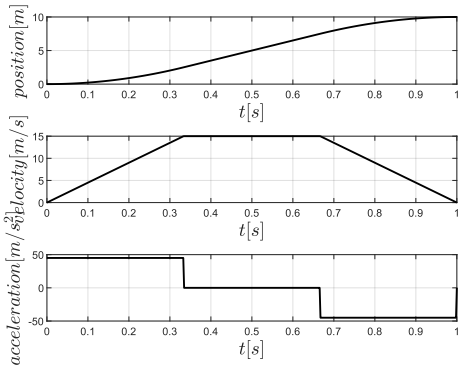


Figure 3.10 Trapezoidal velocity profile

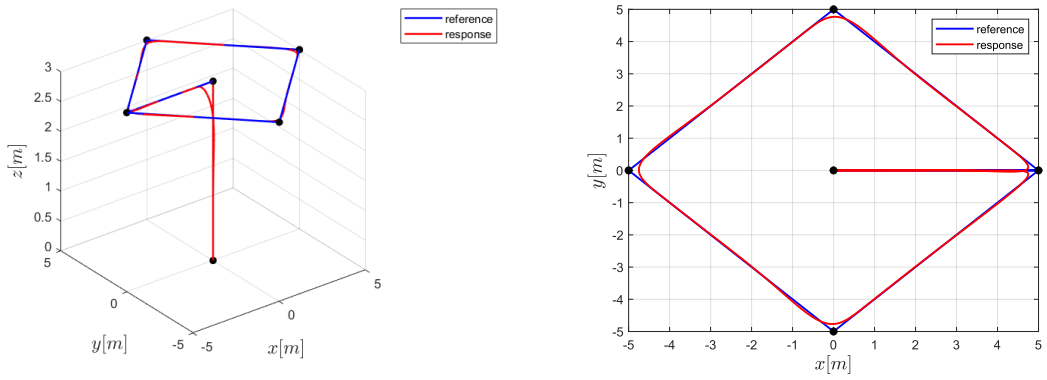
The square pattern trajectory is given as the reference to test tracking performance and Table 3.5 below lists the coordinates of the waypoints in the inertia coordinate frame and the corresponding times of arrival. The acceleration is set to $\pm g/2$, and the maximum velocity is determined such that the time of arrival can be respected.

Sequence	Coordinates [m]	Time of arrival [s]
#1	(0, 0, 0)	0
#2	(0, 0, -3)	5
#3	(5, 0, -3)	10
#4	(0, -5, -3)	15
#5	(-5, 0, -3)	20
#6	(0, 5, -3)	25
#7	(5, 0, -3)	30
#8	(0, 0, -3)	35
#9	(0, 0, 0)	40

Table 3.5 Waypoint list of square pattern trajectory

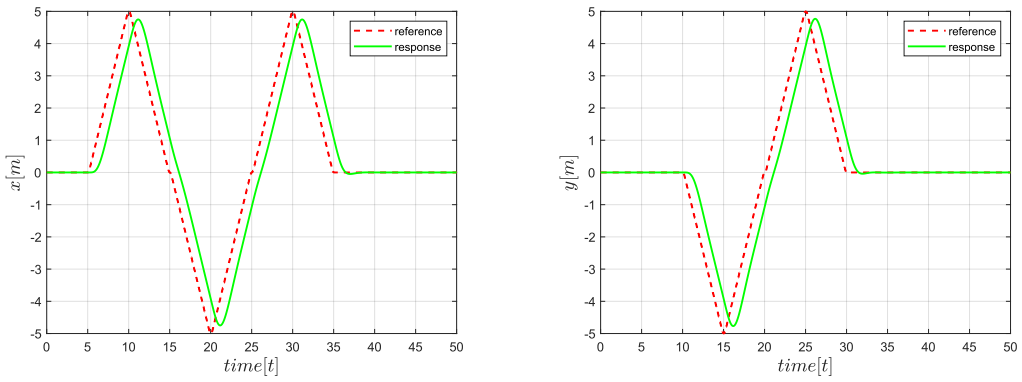
The trajectory tracking performance of the designed LQR controller is shown in Figure 3.11. The simulation results demonstrate that the designed controller allows an acceptable following of the trajectory. The $X-Y$ view of the trajectory in Figure 3.11 (b) shows that the real response doesn't pass through the defined waypoint, it will go to the next waypoint when it is closest to the current waypoint. This is the drawback of the designed controller. Figure 3.12 (a) and Figure 3.12 (b) demonstrate a 1.1s delay and a tracking error in x, y direction between the reference and the

response. As for the altitude control performance, Figure 3.12 (c) shows a good tracking of the position in z axis with small error and delay. (Figure) below compares the velocity calculate by the trajectory and the real quadcopter velocity. Figure 3.13 (a) and Figure 3.13 (b) show a large rising/settling time for tracking the velocity in x, y direction, which leads to the delay in position tracking. The velocity tracking in z direction performs much better than the ones in x, y direction. In general, the designed LQR controller shows excellent altitude control performance, but the x, y position control has the problem related to steady-state error and delay.

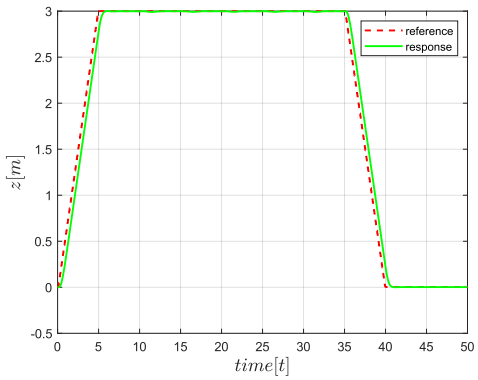


(a) Three-dimensional view (b) $X-Y$ view

Figure 3.11 Square pattern trajectory tracking performance

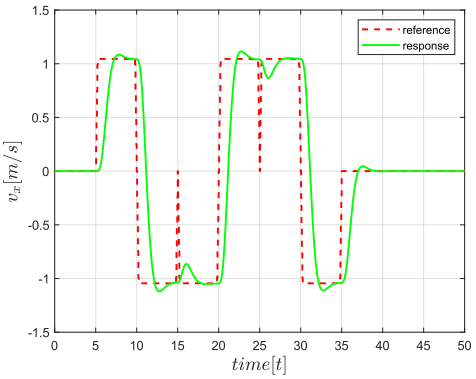


(a) Response and reference signal in x (b) Response and reference signal in y

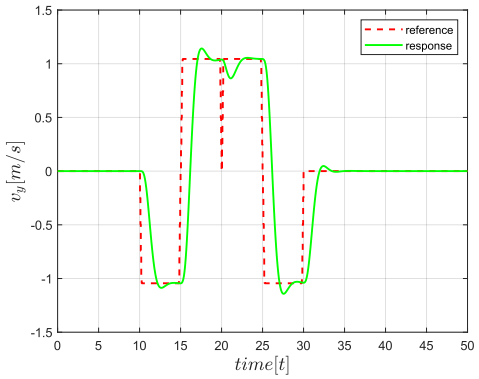


(c) Response and reference signal in z

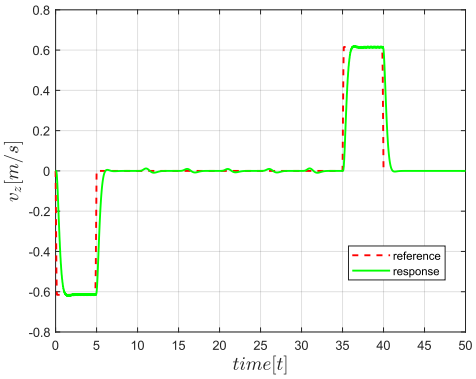
Figure 3.12 x, y, z step response



(a) v_x reference and response



(b) v_y reference and response



(c) v_z reference and response

Figure 3.13 Velocity behavior

Chapter 4: Sensors and GNSS module

A real quadcopter platform mostly consists of the flight controller, sensors, actuators, radio communication module, and so on. To control and stabilize the quadcopter's state such as position and attitude in flight, the flight controller needs accurate and reliable information on the real-time and accurate quadcopter state. The flight controller takes the actual quadcopter state as input, generating the desired control commands according to the difference between the reference and actual state to control and stabilize the quadcopter to the reference state. The sensors on the quadcopter in combination with the advanced sensor fusion algorithms can measure the quadcopter's state very accurately in real-time.

This chapter focuses on the sensors and the methods for using these sensor measurements to get accurate quadcopter position, attitude, and velocity. As mentioned above, the quadcopter's motion in three-dimensional space includes translational and rotational motions. To control the translational motion in the horizontal plane, a sensor for positioning is required, and the typical positioning sensor is a GNSS module. To control the vertical motion, a sensor is required to measure the quadcopter's altitude, which could be a barometer, a TOF (time of flight) based laser sensor, an ultrasonic sensor, or a camera. As for the rotational motion control, the accurate and reliable attitude of the quadcopter is required, which can be measured by sensors consisting of MEMS gyroscopes, accelerometers, and magnetometers.

4.1 Inertial Measurement Unit

An inertial measurement unit or IMU for short is an electronic device that typically consists of accelerometers, gyroscopes, and magnetometers (optimal) which measure linear acceleration, angular velocity, and magnetic field strength, respectively. IMUs are often incorporated into Navigation Systems which utilize the raw measurements to calculate attitude, linear velocity, and position of an aircraft relative to a global reference frame. Besides navigational purposes, IMUs also serve as orientation sensors in many consumers product such as smartphones and game equipment. No matter what, IMUs are essential tools for measuring the attitude of an object in space. To develop a flight controller for the quadcopter, the IMU is indispensable, it provides the microcontroller with the quadcopter's attitude, with which the control algorithm can generate the desired command to track an attitude signal.

In this project, the MPU-6050 contains a gyroscope and an accelerometer is used. And the method for computing attitude angles of the quadcopter by the gyroscope and accelerometer is expounded, respectively. Then, the complementary filter used for reducing noises and sensor fusion is introduced and implemented to get more accurate and reliable attitude angles.

4.1.1 Gyroscope

The gyroscope is an electronic sensor that measures the angular rate of the attached object, or how fast the object is turning relative to its body coordinate frame. By integrating the angular rate, it's possible to get the attitude of the object in space.

The gyroscope of MPU-6050 detects rotation about the X, Y, Z axis and outputs the angular rate in 16 bits two's complement format. The full-scale range of the gyroscope can be digitally programmed to ± 250, ± 500, ± 1000, or ± 2000. In this project, a ± 500 full-scale range is enough considering the quadcopter's rotational dynamics. By referring to the MPU-6050 datasheet [45], it is found that in the case of a ± 500 full-scale range, the gyroscope will output 65.5 when it is rotating at 1 degree per second. Hence the measured angular rate in degrees per second can be expressed as:

$$angular\ rate = \frac{raw\ gyro\ output}{65.5} \tag{4-1}$$

Since the gyroscope measures angular rates, in ideal conditions, the gyroscope should output zero when it's at rest, and output a constant value when it's rotating at a constant speed. But in reality, the output of the gyroscope is not zero when it's static. The gyroscope has a zero-rotation error due to its characteristics. The zero-rotation error should be subtracted from the output to center the gyroscope output to zero. The gyroscope calibration is a method to compute the zero-rotation error and it is implemented by collecting 2000 measurements and subtracting the mean value from the raw gyroscope output. The calibration starts every time the flight control program runs on the microcontroller, and it lasts a couple of seconds. During calibration, the gyroscope should be static to avoid affecting the calculation of the mean value. After the calibration process is finished, the raw gyroscope output is centered to zero and can be used to calculate the attitude angles.

As mentioned above, the MPU-6050 is used to calculate the attitude angles of the quadcopter, which are the roll pitch and yaw angles introduced in section 2.2.3. The attitude angles can be obtained by integrating the angular rate. Denoting the roll pitch and yaw angles at n^{th} moment as $r(n)$, $p(n)$, $y(n)$ respectively, and the angular rates measured by the gyroscope as g_x, g_y, and g_z, the attitude at the next moment needs to be calculated at the current moment. To calculate the attitude at $(n+1)^{th}$ moment, just add the corresponding attitude angle change based on the attitude at n^{th} moment. The amount of change in the attitude angle can be multiplied by the angular velocity and the adopted time interval. The renewal of the attitude angles can be expressed as:

$$\begin{cases} r(n+1) = r(n) + \frac{dr}{dt}\Delta t \\ p(n+1) = p(n) + \frac{dp}{dt}\Delta t \\ y(n+1) = y(n) + \frac{dy}{dt}\Delta t \end{cases} \tag{4-2}$$

Here, $\begin{bmatrix} \frac{dr}{dt} & \frac{dp}{dt} & \frac{dy}{dt} \end{bmatrix}^T$ is the angular velocity relative to the inertial coordinate frame, which is used for attitude update. The attitude update is based on the inertial coordinate frame, while the angular rate measured by the gyroscope at n^{th} moment is relative to its own IMU coordinate frame or the body coordinate frame of the quadcopter. Hence, the angular rate measured by the gyroscope needs to be transformed to the angular velocity relative to the inertial coordinate frame before updating the attitude. According to Equation 2-7, the transformation matrix is based on current attitude and

can be expressed as:

$$T = \begin{bmatrix} 1 & \frac{\sin(p)\sin(r)}{\cos(p)} & \frac{\cos(r)\sin(p)}{\cos(p)} \\ 0 & \cos(r) & -\sin(r) \\ 0 & \frac{\sin(r)}{\cos(p)} & \frac{\cos(r)}{\cos(p)} \end{bmatrix} \tag{4-3}$$

The angular velocity relative to the inertial coordinate frame can be expressed as:

$$\begin{bmatrix} \frac{dr}{dt} \\ \frac{dp}{dt} \\ \frac{dy}{dt} \end{bmatrix} = T \begin{bmatrix} g_x \\ g_y \\ g_z \end{bmatrix} = \begin{bmatrix} 1 & \frac{sin(p)sin(r)}{cos(p)} & \frac{cos(r)sin(p)}{cos(p)} \\ 0 & cos(r) & -sin(r) \\ 0 & \frac{sin(r)}{cos(p)} & \frac{cos(r)}{cos(p)} \end{bmatrix} \begin{bmatrix} g_x \\ g_y \\ g_z \end{bmatrix} \tag{4-4}$$

The updating of the attitude angles can be implemented by Equation 4-4 and Equation 4-2. It should be noted that Δt is the integration interval. Since the main loop frequency of the developed program is 250Hz, the integration interval is 0.004s.

4.1.2 Accelerometer

Accelerometer sensors can measure the linear acceleration and the gravitational acceleration along the axes of its coordinate frame. The measurement of the components of the gravitational acceleration on each coordinate axes can be used to determine the accelerometer roll and pitch attitude angles. The attitude angles are dependent on the order in which the rotations are applied, which are declared in section 2.2.3.

When the accelerometer is at rest and placed horizontally, that is when the z axis is straightly upright, the gravitational acceleration distribution along z axis is $1g$, and along both x axis and y axis, the distributed components are both 0. At this moment, the gravitational acceleration relative to the accelerometer's coordinate frame can be recorded as:

$$G = [0 \quad 0 \quad g]^T \tag{4-5}$$

When the accelerometer rotates to a certain attitude, the gravitational acceleration will produce corresponding components along the three axes of the accelerometer's coordinate frame. These are essentially the coordinates of the gravitational acceleration relative to the new accelerometer's coordinate frame. The rotation sequence applied to the accelerometer's coordinate frame is ZYX (yaw, then pitch and finally roll), and according to section 2.2.3, the rotation matrix from the accelerometer's coordinate frame to the inertial coordinate frame can be expressed as:

$$R = \begin{bmatrix} c_\theta c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi \\ c_\theta s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{bmatrix} \tag{4-6}$$

Denoting the coordinate of gravitational acceleration relative to the accelerometer's coordinate frame as $[a_x \quad a_y \quad a_z]^T$, then:

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = R^T \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} = \begin{bmatrix} -\sin(\theta)\, g \\ \sin(\phi)\cos(\theta)\, g \\ \cos(\phi)\cos(\theta)\, g \end{bmatrix} \tag{4-7}$$

By solving Equation 4-7, the roll and pitch attitude angles are obtained and can be expressed as:

$$\begin{cases} \phi = \arctan\left(\frac{a_y}{a_z}\right) \\ \theta = -\arctan\left(\frac{a_x}{\sqrt{a_y^2+a_z^2}}\right) \end{cases} \tag{4-8}$$

Since the gravitational acceleration along the z axis remains constant when the accelerometer rotates around the z axis, the yaw angle cannot be evaluated by the accelerometer. Another problem with the evaluated attitude angles is that the equations for the roll and pitch angles have mathematical instabilities when x or y axis happens to become aligned with the gravity and points upwards or downwards. In this project, the roll and pitch angles estimated by the accelerometer are limited considering the rotational dynamics of the quadcopter, hence this instability problem can be avoided.

The accelerometer calibration is required to determine the level position of the quadcopter during flight, this can make the quadcopter fly as level and stable as possible. The data read from the accelerometer should be $[0 \quad 0 \quad g]^T$ when the quadcopter is on a spirit-level surface. But in reality, there is a bias in the x, y axis of the accelerometer coordinate frame and it should be subtracted from the raw accelerometer data. The calibration procedure is the same as the one of the gyroscope. It's implemented in the program by placing the quadcopter on a spirit-level surface, collecting 2000 samples, and calculating the average. The accelerometer calibration is required after a crash.

The accelerometer of MPU-6050 outputs the measured accelerations in 16 bits two's complement format. And the full-scale range can be digitally programmed to $\pm 2g$, $\pm 4g$, $\pm 8g$, or $\pm 16g$. In this project, the full-scale range is set to $\pm 8g$ and by checking the MPU-6050 datasheet [45], an output of 4096 from the accelerometer corresponds to $1g$.

4.1.3 Complementary filter and sensor fusion

Although both the gyroscope and the accelerometer can be used individually to measure the roll and pitch angles of the quadcopter, neither of the measured angles can be directly used by the flight controller due to the characteristics of the sensors. Hence, it's necessary to integrate the results of the two sensor measurements to obtain more accurate and reliable results.

The gyroscope gives a good indication of the attitude angles in dynamic conditions. Due to the non-ideal offset compensation of the gyroscope, even after calibration, the raw outputs of the gyroscope are still not zero at rest. This makes the angles estimated by integrating the output of the gyroscope drift. When the quadcopter is in flight, the vibrations generated by the quadcopter will speed up the drift, which makes the attitude angles computed by the gyroscope different from the real attitude angles. The accelerometer gives a good indication of the tilt angles in static conditions. The accelerometer doesn't have a drift problem, but the always presented vibrations and small accelerations during flight make the attitude angles unreliable and useless to the flight controller. Only the average of the angles computed by the accelerometer is usable to the flight controller.

To overcome these problems, a complementary filter is utilized in the project to combine the two measurements obtained from the gyroscope and the accelerometer. The complementary filter is a convenient way to combine measurements from an accelerometer and a gyroscope into a better angle estimation than either could provide on its own. The idea behind the complementary filter is

to take slow-moving signals from the accelerometer and fast-moving signals from the gyroscope and combine them.

The basic structure of the complementary filter is shown in Figure 4.1, where x_1 and x_2 are noisy measurements of the same signal that mainly contain low-frequency noise and high-frequency noise, respectively. The transfer function $G(s)$ can be made into a low-pass filter to filter out the high-frequency noise in x_2 and $[1 - G(s)]$ can be made into a high-pass filter that can filter out the low-frequency noise in x_1. The output of the complementary filter can be expressed as:

$$x = x_1[1 - G(s)] + x_2 G(s) \tag{4-9}$$

And the first-order low-pass filter (LPF) and first-order high-pass filter (HPF) can be expressed as:

$$LPF = G(s) = \frac{1}{\tau s + 1} \tag{4-10}$$

$$HPF = 1 - G(s) = \frac{\tau s}{\tau s + 1} \tag{4-11}$$

Where $s = \sigma + j\omega$ is the complex variable in the frequency domain used for Laplace transform, and τ is the time constant that determines the filter cut-off frequency.

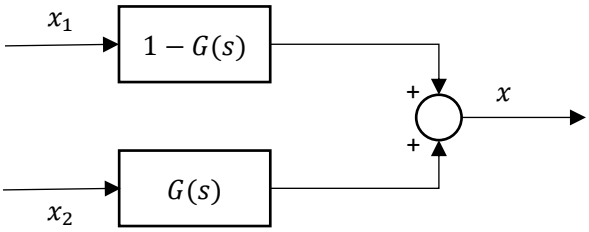
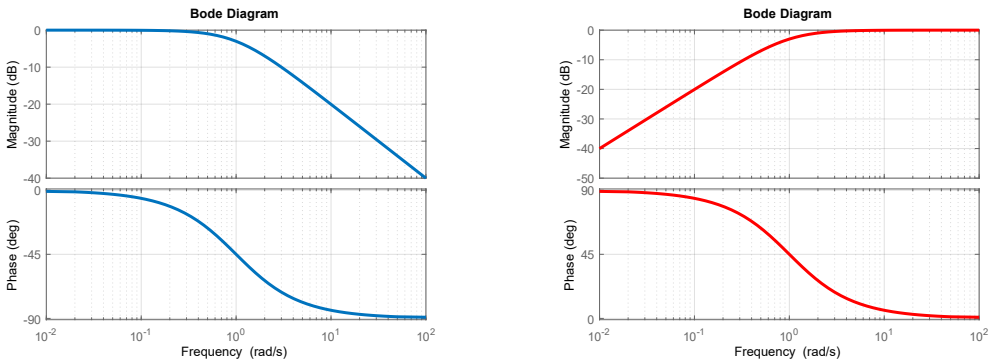


Figure 4.1 Basic complementary filter

In Figure 4.2, the bode diagrams of the first-order low-pass filter and high-pass filter are shown. The low-pass filter passes through signals with frequencies lower than the cut-off frequency and attenuates signals with frequencies higher than the cut-off frequency. Hence, the low-pass filter can be used to process the signals coming from the accelerometer which contain high-frequency noise (such as the accelerometer in the case of vibration). The high-pass filter behaves exactly in the opposite way. It allows short-duration signals to pass through while filtering out signals that are steady over time. This can be used to process signals coming from the gyroscope to cancel out the drift.



(a) LPF (b) HPF

Figure 4.2 Bode diagram of LPF and HPF (τ=1)

To implement the complementary filter to the microcontroller, the complementary filter represented in transfer function format must be written into a difference equation format. A straightforward method to implement the complementary filter to a microcontroller is to make the complementary filter in the form like this:

$$x = \alpha \cdot x_1 + (1 - \alpha) \cdot x_2 \tag{4-12}$$

Where x_1 and x_2 represents signals coming from the gyroscope and accelerometer, the parameter α can be calculated by:

$$\alpha = \frac{\tau}{\tau + \Delta t} \tag{4-13}$$

Where $\alpha \in \mathbb{R}$ and $0 < \alpha < 1$, τ is the time constant of the low-pass filter and the high-pass filter, Δt is the sampling interval. Since the accelerometer doesn't estimate yaw angle, the complementary can only be used to integrate roll and pitch angles estimated from different sensors. The attitude roll and pitch angles estimated by the IMU can be expressed as:

$$\begin{cases} roll = \alpha \cdot roll_{gyro} + (1 - \alpha) \cdot roll_{acc} \\ pitch = \alpha \cdot pitch_{gyro} + (1 - \alpha) \cdot pitch_{acc} \end{cases} \tag{4-14}$$

And the block diagram of the complement filter utilized in this project is shown in Figure 4.3 below:

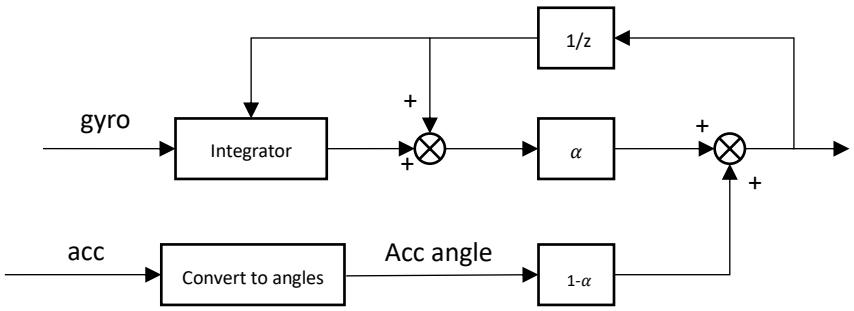
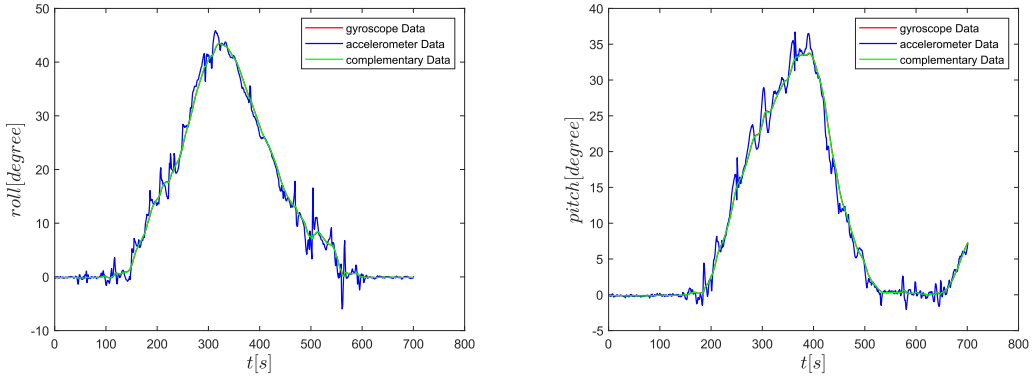


Figure 4.3 Block diagram of the discrete complementary filter

The roll and pitch angles estimated by the gyroscope, the accelerometer, and the complementary filter are shown in Figure 4.4 below. The blue curves represent the accelerometer estimation, which are rough and spiky, indicating that the angles estimated by the accelerometer are affected by the noise and vibration. The angles estimated by the gyroscope are less affected by the noise and vibration, but they have drift problems, and over time, the difference between them and the true values will become larger and larger. The green curves represent the complementary filter estimation, and the results show that the angles estimated by the complementary filter are more accurate, less affected by noise and vibration, and there is no drift phenomenon.



(a) Roll angle $(\alpha = 0.995)$ (b) Pitch angle $(\alpha = 0.995)$

Figure 4.4 Comparison between roll and pitch angles estimated by gyroscope, accelerometer, and complementary filter

4.2 Electronic compass

The yaw angle estimated by the IMU by integration can't be used to represent the heading angle of the quadcopter to the North, and the inherent drift problem makes the estimated yaw angle away from the real one. Hence, it can't be used directly by the flight controller. To get an accurate and reliable heading angle of the quadcopter, an electronic compass module is required. Implementing an electronic compass module to the quadcopter allows it to know its heading angle or direction in space. With the direction information, it's possible to add features like head locking, GPS position control, and waypoint fly to the quadcopter.

The electronic compass module calculates the heading angle by measuring the strength of its surrounding geomagnetic field. In this section, the Earth's magnetic field and its basic features are introduced. Then, the formula to calculate the heading angle using the measurements of the electronic compass module is derived. Finally, the method to use the electronic compass module in combination with the IMU to design a more accurate, reliable, and tilt-compensated compass is presented.

4.2.1 Earth's magnetic field

The Earth's magnetic field can be approximated with the dipole model shown in Figure 4.5 below. As the figure shows, the Earth's magnetic field points down toward the north in the northern hemisphere, is horizontal and points toward the north at the equator and points up toward the north in the southern hemisphere. In all cases, the Earth's magnetic field has a component parallel to the surface and points toward the magnetic north [46]. This horizontal component can be measured by the electronic compass module and the heading angle can be calculated by trigonometry. This is the principle for using the electronic compass to estimate the heading angle.

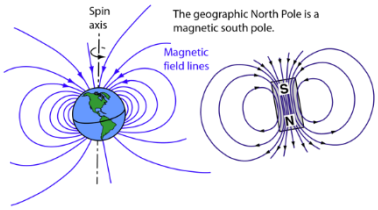


Figure 4.5 Earth's magnetic field

4.2.2 Heading angle calculation

When the compass is at a leveled position (horizontal to the Earth's surface), then the roll and pitch angles would be zero and the heading angle can be determined as shown in Figure 4.6. The local Earth's magnetic field H has a fixed component H_h on the horizontal plane pointing to the Earth's magnetic north. Denoting the horizontal components measured by the electronic compass module as X_h and Y_h. Then the heading angle is calculated as:

$$heading = \arctan\left(\frac{Y_h}{X_h}\right) \tag{4-15}$$

To account for the tangent function being valid over 180° and not allowing the $Y_h = 0$ division calculation, the following equations can be used:

$$heading = \begin{cases} 180 - \arctan\left(\frac{Y_h}{X_h}\right), & for(X_h < 0) \\ -\arctan\left(\frac{Y_h}{X_h}\right), & for(X_h > 0, Y_h < 0) \\ 360 - \arctan\left(\frac{Y_h}{X_h}\right), & for(X_h > 0, Y_h > 0) \\ 90, & for(X_h = 0, Y_h < 0) \\ 270, & for(X_h = 0, Y_h > 0) \end{cases} \tag{4-16}$$

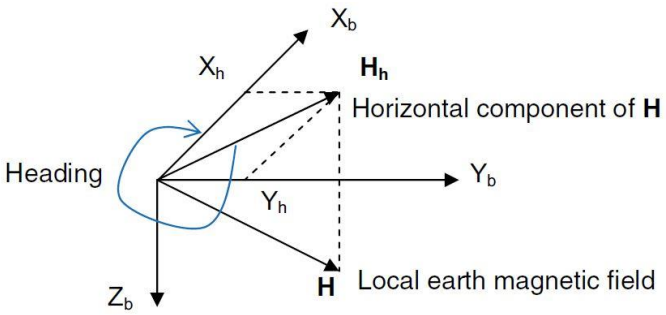


Figure 4.6 Heading angle calculation

As is shown in Figure 4.6, when X_b (X axis of the electronic compass coordinate frame) is parallel to H_h, which is the horizontal component pointing to the magnetic north, then the heading angle equals zero. Rotating the electronic compass clockwise or counterclockwise on the horizontal plane, the heading angle increases or decreases. After a full round 360° rotation, we can get a centered circle if plotting X_h and Y_h values measurements in $x - y$ plane. It should be noted that only the

horizontal components X_h and Y_h are used when computing the heading angle, the vertical component Z_h should be ignored.

In this project, the electronic compass module is mounted on the quadcopter to measure the heading angle. Hence, in most cases, the compass is not confined to a level plane (Figure 4.7). In this situation, the roll and pitch angles are not equal to zero, and the electronic compass measurements X and Y are not the horizontal components, and can't be directly used in Equation 4-16 for computing the heading angle. The conversion between the electronic compass measurements X, Y, Z, and the horizontal components X_h and Y_h is required. As is mentioned in section 2.2.3 above, the rotation to be performed is first rolled and then pitched. The electronic compass also follows this rotation sequence, and the following relationship can be obtained:

$$\begin{bmatrix} X_h \\ Y_h \\ Z_h \end{bmatrix} = R(\theta) \cdot R(\phi) \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

$$\begin{bmatrix} X_h \\ Y_h \\ Z_h \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\phi)\sin(\theta) & \cos(\phi)\sin(\theta) \\ 0 & \cos(\phi) & -\sin(\phi) \\ -\sin(\theta) & \cos(\theta)\sin(\phi) & \cos(\phi)\cos(\theta) \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{4-17}$$

Where the roll and pitch angles can be estimated by the MPU-6050 as mentioned in section 4.2. the relationship between the horizontal components and the electronic compass measurements can be expressed as:

$$\begin{cases} X_h = X \cdot \cos(\theta) + Y \cdot \sin(\phi)\cos(\theta) + Z \cdot \cos(\phi)\sin(\theta) \\ Y_h = Y \cdot \cos(\phi) - Z \cdot \sin(\phi) \end{cases} \tag{4-18}$$

After the horizontal components are determined by Equation 4-18, then the heading angle can be computed using Equation 4-16.

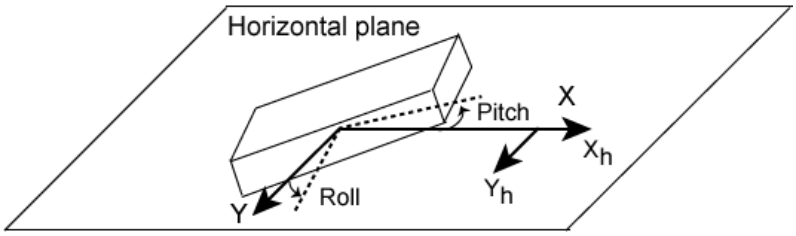


Figure 4.7 Tilt compensation of the compass

As is shown in Figure 4.8 below, the true north, also called the geographic north, is the intersection of the Earth's rotation axis and the Earth's surface, which is not at the same geographical location as the magnetic north (They are about 11.5° rotation from each other). The declination angle is used to describe the difference between them. By definition, the angle on the horizontal plane between magnetic north and geographic north is the declination angle. It's positive when the magnetic north is to the east of the geographic north, and negative when the magnetic north is to the west. The declination angle varies a lot depending on the position on the Earth's surface, and at some locations, the declination angle will even be as large as 25°. For a given location the declination angle can be found by using a geomagnetic declination map or by checking some official website. Equation 4-16 computes the heading angle relative to the magnetic north. To account for the difference between the magnetic north and the geographic north and get a more precise heading angle, the declination angle should be added or subtracted from the heading angle computed by

Equation 4-16. The declination used in this project is $+2.85°$.

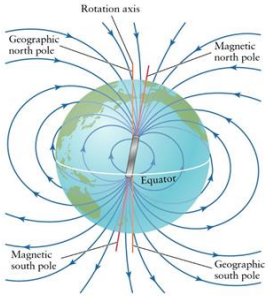


Figure 4.8 Magnetic North and Geographic North

In this project, the electronic compass module utilized is the QMC5883L. The output of this compass is in 16 bits two's complement format. In this project, this compass is configured to continuous mode, the output data rate is set to 200Hz, and the full-scale range is set to ± 8 Gauss. By checking the datasheet of QMC5883L [47], the sensitivity is 3000 LSB/G when the full-scale range is set to ± 8 Gauss.

4.2.3 Magnetic distortions

The reading from the compass will probably not be usable due to the magnetic distortions caused by nearby ferrous materials. Magnetic distortions can be categorized as hard iron distortion and soft iron distortion.

The hard iron distortion is produced by materials such as magnetized iron or steel that exhibit a constant, additive field to the Earth's magnetic field, thereby generating a constant additive value to the output of each axis of the compass. The hard iron distortion is the most important to be eliminated. The hard iron distortion causes an offset to the compass reading and the correction can be implemented by removing that bias. The formulas for calculating the bias and removing it from the compass reading can be expressed as:

$$\begin{cases} x_{offset} = (x_{max} + x_{min})/2 \\ y_{offset} = (y_{max} + y_{min})/2 \\ z_{offset} = (z_{max} + z_{min})/2 \end{cases} \tag{4-19}$$

$$\begin{cases} x_{correct} = x - x_{offset} \\ y_{correct} = y - y_{offset} \\ z_{correct} = z - z_{offset} \end{cases} \tag{4-20}$$

Where x_{max} and x_{min} are the maximum and minimum magnetic field strength measured in the x-axis direction when rotating the compass 360 degrees in space.

Soft iron distortion is the result of materials that distort a magnetic field but do not necessarily generate a magnetic field itself. The soft iron distortion depends on the orientation of the materials relative to the compass. The computationally cheap way of correcting the soft iron distortion is shown below:

$$\begin{cases} \Delta x = (x_{max} - x_{min})/2 \\ \Delta y = (y_{max} - y_{min})/2 \\ \Delta z = (z_{max} - z_{min})/2 \end{cases} \quad (4\text{-}21)$$

$$\Delta average = (\Delta x + \Delta y + \Delta z)/3 \quad (4\text{-}22)$$

$$\begin{cases} k_x = \Delta average/\Delta x \\ k_y = \Delta average/\Delta y \\ k_z = \Delta average/\Delta z \end{cases} \quad (4\text{-}23)$$

Where k_x, k_y, k_z are scale factors used for correcting the soft iron distortion. The formulas for correcting both hard iron distortion and soft iron distortion can be expressed as:

$$\begin{cases} x_{correct} = (x - x_{offset}) * k_x \\ y_{correct} = (y - y_{offset}) * k_y \\ z_{correct} = (z - z_{offset}) * k_z \end{cases} \quad (4\text{-}24)$$

4.3 GNSS module

The Global Navigation Satellite System, abbreviated GNSS, includes constellations of Earth-orbiting satellites that broadcast time and their location in space, networks of ground control stations, and receivers that calculate ground position by trilateration [48]. GNSS can provide users with three-dimensional coordinates, speed, and timing signals anywhere on the Earth's surface or near-Earth space. Hence, it now receives more and more attention from researchers and is widely used in positioning, navigation, transportation, military, surveying and mapping, agriculture, archaeology, and the Internet of Things. At present GNSS include two fully operational systems, the U.S. Global Positioning System (GPS) and the Russian Federation's Global Navigation Satellite System (GLONASS), as well as the developing global and regional systems such as Europe's European Satellite Navigation System (GALILEO) and China's COMPASS/Bei-Dou, India's Regional Navigation Satellite System (IRNSS) and Japan's Quasi-Zenith Satellite System (QZSS) [48]. In addition to these, GNSS also includes several satellite-based augmentation systems that can improve the positioning accuracy, integrity, and availability of the basic GNSS signals.

Over the past few years, the GNSS technology has advanced enough to make it both affordable and lightweight enough to be implemented in the average consumer quadcopter. A quadcopter in combination with a GNSS module can achieve autonomous flight, leading to an increasing number of outdoor applications such as mapping and surveying, reconnaissance surveillance, transportation, and so on. Implementing a GNSS module can add the following features to a quadcopter:

1. Position hold. The quadcopter with a GNSS module is able to identify and maintain its position and achieve a stable hover. It can hover in place even under a breeze or a gust. If it identifies that it has drifted away from the hover location, it will automatically correct and return to the same location. The GNSS module can also help the quadcopter to achieve a more stable and smooth flight.
2. Return to home. The quadcopter with a GNSS module can remember the take-off position and return to it. This is obviously only possible with the GNSS module to tell the quadcopter where it is at any given time and where it took off. When the quadcopter is in return to home mode,

its flight controller takes the take-off position as the reference and calculates desired control commands according to the current position measured by the GNSS module.

3. Waypoint fly. The principle of waypoint fly is the same as the one of return to home. A GNSS quadcopter can use the autopilot function to fly a series of predetermined waypoints. Some flight controllers can even direct the quadcopter to hover for a set amount of time at each waypoint.

In this section, the GNSS positioning principle is first introduced. Then how to use the GNSS measurement to calculate the state error that the designed LQR controller can use is expounded. Finally, the GNSS module implementation to the real quadcopter platform is elaborated.

4.3.1 GNSS positioning principle

The GNSS positioning involves the application of the mathematical principle called trilateration. The trilateration principle is quite simple: the position of a point can be calculated if the distances from that point to another three points whose positions are known can be obtained. As is shown in Figure 4.9 below, the locations of three satellites in space are known beforehand, These satellites send signals to the GNSS receiver while orbiting the Earth and the distances from the receiver to the three satellites can be computed by multiplying the signal transmission speed and transmission time, then the unknown receiver position can be computed by solving this trilateration problem:

$$c(\Delta t_m) = \sqrt{(x - x_m)^2 + (y - y_m)^2 + (z - z_m)^2} \qquad m = 1,2,3 \tag{4-25}$$

Where m is the satellite index, (x_m, y_m, z_m) is the known position of the satellite in space, Δt_m is the GNSS signal transmission time, $c(\Delta t_m)$ is the distance between the receiver and the satellite, and (x, y, z) is the unknown position of the receiver. In reality, the transmission time from each satellite to the receiver has a common unknown bias due to a common time error from the inaccurate receiver clock. Therefore, an additional clock bias term δt must be introduced as the fourth unknown, and (Equation) can be rewritten as:

$$c(\Delta t_m + \delta t) = \sqrt{(x - x_m)^2 + (y - y_m)^2 + (z - z_m)^2} \qquad m = 1,2,3,4 \tag{4-26}$$

The trilateration based positioning implies that a GNSS receiver actually needs to receive signals from at least four satellites to achieve a position fix. If the number of satellites is above four then every additional satellite can increase the positioning accuracy. Nowadays, many commercial GNSS receivers on market can receive signals from more than ten satellites at the same time, resulting in an accuracy of several meters in positioning.

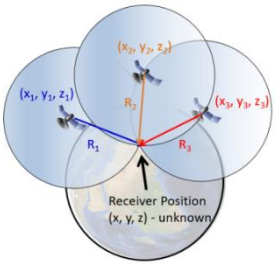


Figure 4.9 Principle of GNSS positioning [49]

4.3.2 Geographic coordinate frame

The GNSS module usually outputs the position in the form of geographic coordinates. A geographic coordinate system is a system that uses a three-dimensional spherical surface to determine locations on the Earth and any location can be referenced by a point with longitude and latitude coordinates [50]. For any given point on the Earth's surface represented with the geographic coordinate frame, it has two coordinate values: latitude φ and longitude λ, and both units are in degrees. Latitude is defined as the angle formed by the intersection of a line perpendicular to the Earth's surface at a point and the plane of the Equator [51]. At the equator, the latitude value is zero and from the equator to the North and South poles, the value of latitude gradually increases for 0° to 90°. Points In the northern hemisphere have positive latitude values, while points in the southern hemisphere have negative values. Longitude is defined as the horizontal angle between the point and the meridian that is defined as a circle line passing through the North and South poles, and the Greenwich, UK. The longitude value ranges from -180° to 180°. Positive longitude values are east of the meridian while negative ones are west. The geographic coordinate frame and the latitude and longitude definition are shown in Figure 4.10.

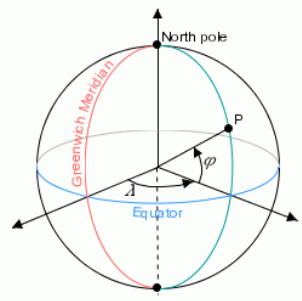


Figure 4.10 Geographical coordinate frame

4.3.3 Estimation of distance and speed based on GNSS data

The GNSS data should be converted from the geographic coordinate frame to the NED coordinate frame before it can be used by the LQR controller. To convert the position represented by the latitude and longitude to the x, y coordinates in the NED reference coordinate frame, the distance, and bearing from that point to the coordinate frame origin are required.

Considering the Earth as a sphere, the distance between two geographical coordinate points can be computed using spherical geometry and trigonometric functions. The shortest distance between two points on the Earth's surface is the great circle distance, corresponding to the arc linking two points on the sphere. The Haversine formula can be used to calculate an accurate great circle distance between two points. Denoting φ_1,φ_2 the latitude of two points, λ_1,λ_2 the longitude of two points, the Haversine formula can be expressed as:

$$a = sin^2\left(\frac{\Delta\varphi}{2}\right) + \cos(\varphi_1) \cdot \cos(\varphi_2) \cdot sin^2\left(\frac{\Delta\lambda}{2}\right) \tag{4-27}$$

$$c = 2 \cdot atan2(\sqrt{a}, \sqrt{1-a}) \tag{4-28}$$

$$d = R \cdot c \tag{4-29}$$

Where $\Delta\varphi$ and $\Delta\lambda$ represent the latitude and longitude difference, respectively, R is the Earth's radius (mean radius is 6371km).

The bearing is defined as the clockwise angle from the true North to the great circle arc. The bearing will vary as follows along the great circle arc. The final bearing and the initial bearing will be different. The bearing at the initial point can be computed by:

$$\theta = atan2(\sin(\Delta\lambda)\cos(\varphi_2), \cos(\varphi_1)\sin(\varphi_2) - \sin(\varphi_1)\cos(\varphi_2)\cos(\Delta\lambda)) \tag{4-30}$$

It should be noted that the $atan2$ function returns a value between -180° to 180°, to normalize it to the compass bearing (range between 0° and 360°), the negative values need to be transformed into the range 180° to 360°. The formula is expressed as:

$$\theta = \begin{cases} \theta, & \theta \geq 0 \\ \theta + 360^{\circ}, & x < 0 \end{cases} \tag{4-31}$$

As is shown in Figure 4.11, the great circle distance between two points is an arc on the surface of the Earth connecting these two points. The bearing angle is the clockwise angle from the geographic north to the great circle.

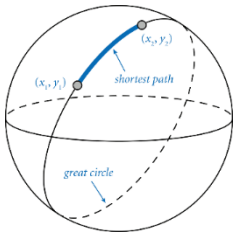


Figure 4.11 Great circle distance

Defining the first GNSS measurement as the origin of the NED coordinate frame, then the GNSS data represented by latitude and longitude can be converted to the x, y coordinate in the NED coordinate frame. With the great circle distance d and the initial bearing θ between two GNSS measurements. the transformation from the latitude and longitude degrees to x, y coordinates can be expressed as:

$$\begin{cases} x = d \cdot \cos(\theta) \\ y = d \cdot \sin(\theta) \end{cases} \tag{4-32}$$

Another method to compute the distance between two points and convert the latitude and longitude degrees to x, y coordinates is presented below, which is based on the flat-earth approximation. This method is valid only for short-range distances. Denoting the coordinates of two points that are close to each other on the Earth are (φ_1, λ_1) and (φ_2, λ_2), then the distance between them can be calculated with the formulas below:

$$\begin{cases} x = \pi R/180 \cdot (\varphi_2 - \varphi_1) \\ y = \pi R/180 \cdot (\lambda_2 - \lambda_1) \cdot \cos(\varphi_1) \end{cases} \tag{4-33}$$

$$d = \sqrt{x^2 + y^2} \tag{4-34}$$

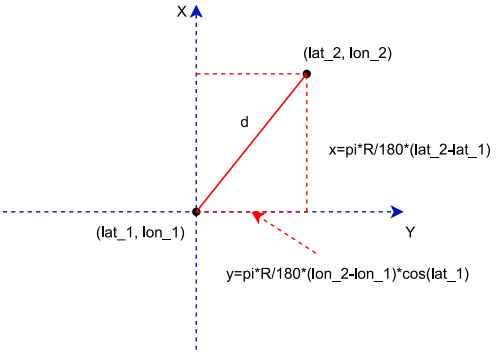


Figure 4.12 Flat-Earth approximation

This work uses Equation 4-33 and Equation 4-34 to convert GNSS data to x, y coordinate in NED coordinate frame. The quadcopter will fly only a short-range distance, hence using the method based on the flat-earth approximation to convert geographic coordinates to coordinates in NED coordinate frame will be accurate enough. Besides, compared to the Haversine method, the method based on the flat-earth approximation reduces programming and processing computation.

After converting the GNSS data represented by geographic coordinates to coordinates in the NED coordinate frame, the instantaneous speed in m/s can be replaced with the travel speed between the two consecutive GNSS coordinates:

$$\begin{cases} v_x = x/t \\ v_y = y/t \end{cases} \tag{4-35}$$

Where t is the travel time between two consecutive GNSS measurements and depends on the data output rate of the GNSS module.

4.3.4 GNSS module implementation

The GNSS module used to position the quadcopter is a U-Blox NEO-M8N GNSS receiver. This GNSS receiver can provide users with a positioning accuracy of up to 2 meters. It can also measure altitude but the altitude measurement accuracy is very poor compared to the positioning accuracy. In addition to measuring position and altitude, this GNSS receiver can also measure the speed over ground in kilometers per hour. The measurement results and other auxiliary messages are output in NMEA (National Marine Electronics Association) sentence format. Figure 4.13 below demonstrates a block of GNSS data outputs with NMEA sentence format.

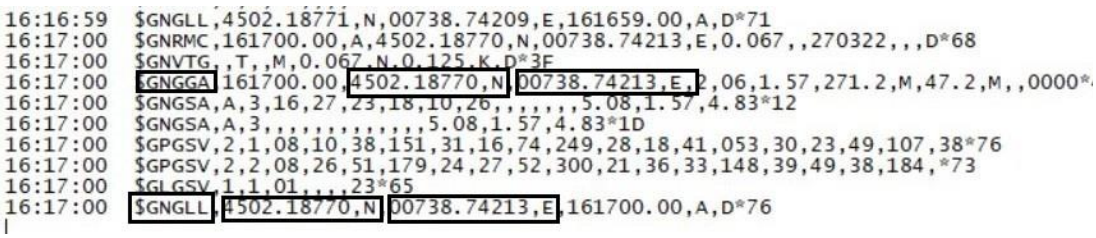


Figure 4.13 GNSS output in NMEA format

It can be seen every sentence begins with '$xxxxx'. The geographic coordinates (latitude and longitude) can be extracted from the sentence beginning with '$GNGGA' (Global Positioning System Fix Data) or '$GNGLL' (Geographic Position, Latitude / Longitude and time). The geographic coordinates are in degrees and minutes. For example, the latitude in Figure 4.13 is 45 degrees and 2.1877 minutes North, and the longitude is 7 degrees and 38.74213 minutes. Due to the property of the NMEA protocol, the GNSS receiver outputs signals in a fixed format, and the longitude and latitude coordinates are also at fixed positions in the sentence. Therefore, it is easy to read the output signal of the GNSS receiver through the serial port of the STM32 board and extract the longitude and latitude.

4.4 Altitude sensor

To enable the quadcopter to automatically adjust the flight altitude during autonomous flight, an altitude sensor is required to measure its flight altitude in real-time. The altitude of the quadcopter can be accurately measured with several different physical methods. These include optical (laser measuring instruments), electronic (microwave detectors), and barometric procedures. Many commercial quadcopters on the market utilize the barometer to measure altitude during flight, for example, most DJI quadcopters measure altitude using a barometric sensor with an AGL (Above Ground Level) reference. This section outlines the principle of using a barometer to measure the altitude of the quadcopter.

4.4.1 Air pressure and altitude

The barometer measures the air pressure and by means of the pressure equations, the altitude of the measured surface can be computed. Air pressure describes the pressure generated by the weight of the air surrounding the Earth. The air in the atmosphere is denser at the base, and the column of air is thus heavier than vice versa. Hence, the air pressure decreases with rising altitude from its zero point (sea level) at 1013 mbar. Measured from sea level, air pressure changes at approximately 1 mbar/8 m. There are two different equations for describing the relationship between air pressure and altitude. When the temperature change with height in the atmosphere is taken into consideration (temperature lapse rate is not equal to zero), the equation can be expressed as:

$$P = P_b[\frac{T_b+(h-h_b)L_b}{T_b}]^{\frac{-gM}{R\cdot L_b}} \quad (4\text{-}36)$$

Where P_b is the reference pressure (pressure at the sea level) [Pa], T_b is the reference temperature (the temperature at sea level) [K], L_b is the temperature lapse rate (-0.0065 $[\frac{K}{m}]$) in ISA, h is the height at which the air pressure is calculated [m], h_b is the height of reference level (height about sea level) [m], R is the universal gas constant (8.31432 $[\frac{J}{mol \cdot K}]$), g is the gravitational acceleration constant (9.80665 $[\frac{m}{s^2}]$), and M is the molar mass of the Earth's air (0.0289644 $[\frac{kg}{mol}]$). This equation can be arranged to calculate the altitude above sea level given the measured air pressure:

$$h = h_b + \frac{T_b}{L_b} \cdot [\left(\frac{P}{P_b}\right)^{\frac{-R \cdot L_b}{gM}} - 1] \tag{4-37}$$

When the temperature lapse rate is not taken into account, which means the temperature doesn't change through an altitude change, another equation can be used:

$$P = P_b \cdot \exp\left[\frac{-gM(h-h_b)}{R \cdot T_b}\right] \tag{4-38}$$

And given the measured pressure, the altitude can be computed according to:

$$h = h_b + \frac{R \cdot T_b \cdot \ln\left(\frac{P}{P_b}\right)}{-gM} \tag{4-39}$$

In this work, a barometer pressure sensor is mounted on the quadcopter to measure the absolute pressure during flight. Equation 4-37 is used to compute the altitude according to the measured air pressure. Substituting all parameters into the equation, the equation that calculates the altitude relative to the sea level is:

$$h = 44330.7 \cdot (1 - \left(\frac{P}{101325}\right)^{0.19}) \tag{4-40}$$

Figure 4.14 below is the plot of Figure 4-40 and the figure shows that there is a linear relationship between the air pressure and the altitude when the altitude is less than 600 meters.

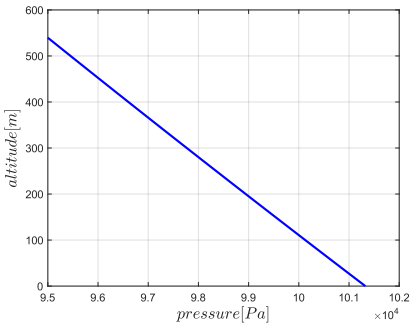


Figure 4.14 Relationship between air pressure and altitude

4.4.2 Barometer implementation

In this project, the MS5611 barometer is used for measuring the quadcopter altitude during flight. It's a very sensitive pressure sensor that can detect pressure differences with a 10cm accuracy. Every induvial barometer is factory calibrated at two temperatures and two pressures. As a result, six coefficients are calculated and stored in the memory of each induvial. These six coefficients must be read by the flight controller and used in the program converting the digital pressure value and digital temperature value into compensated pressure and temperature values. The MS5611 datasheet [52] has demonstrated the formulas for converting the digital pressure and temperature values into compensated pressure and temperature values.

The MS5611 can be configured to work under five different oversampling ratios (256/512/1024/2048/4096). Different oversampling ratio corresponds to different pressure and temperature resolution, and different conversion time required by the Analog to Digital Converter to get the digital pressure and temperature values. In this project, the oversampling ratio is configured to 1024. According to the MS5611 datasheet [52], the corresponding temperature resolution is 0.005℃, and the pressure resolution is 0.027 mbar. According to Figure 4.14, 0.027 mbar pressure resolution corresponds to 0.2319 meters altitude resolution, which is accurate enough for this project. When the oversampling ratio is configured to 1024, the conversion time required by the ADC is 2.28 milliseconds. Since the flight controller main loop runs every 4 milliseconds (flight controller frequency is 250Hz), it's impossible to get the digital pressure and temperature value and calculate the compensated pressure and temperature values at one main loop. Therefore, the following program running process is proposed:

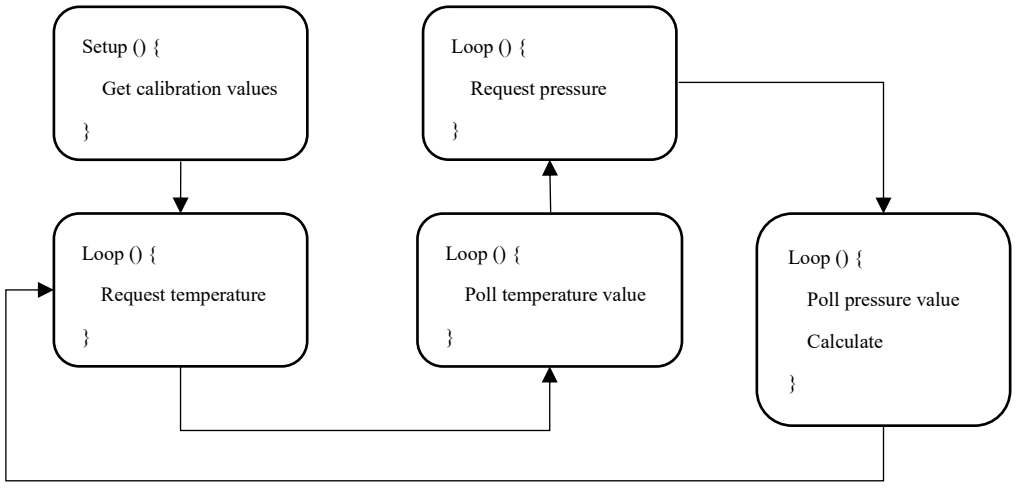


Figure 4.15 Flowchart of the program to read data from the barometer

In the first loop, the flight controller sends a command to the MS5611 barometer to request the digital temperature value, and the barometer ADC starts the conversion. Then in the second loop, the temperature conversion is already finished and the flight controller can directly access the digital temperature value from the barometer. In the third loop, the flight controller sends a command to

request the digital pressure value. Then in the fourth loop, the pressure conversion is finished and the flight controller accesses the digital pressure value from the barometer and calculated the compensated temperature and pressure values. With this approach, the barometer updates the pressure every four main program loops, which is 16 milliseconds.

The MS5611 is an absolute pressure sensor and the measured pressure can be used to compute the altitude relative to the sea level. Equation 4-40 is used to compute the altitude relative to the sea level and the results are shown in Figure 4.16 below. The red line represents the altitude estimated without using a filter. The result is pretty noisy and can't be used by the flight controller directly. The blue line represents the result after implementing a digital low pass filter to the barometer. It can be seen that the noise has become very small after using a low pass filter.

After knowing the altitude of the quadcopter, the instantaneous vertical velocity can be replaced by the travel speed between two consecutive barometer measurements:

$$v_z = \frac{h_1 - h_2}{t} \tag{4-41}$$

Where h_1, h_2 are two consecutive altitude measurements, t is the travel time between two measurements. As mentioned, the travel time between two measurements is 16ms.

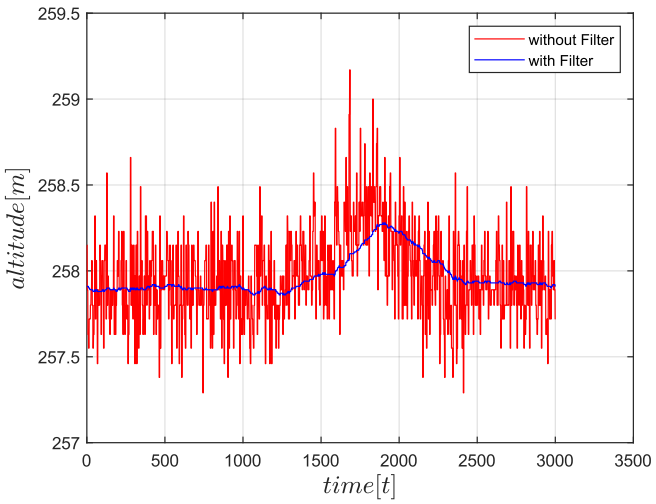


Figure 4.16 Barometer measurements

Chapter 5: Hardware components

In this chapter, the hardware components requested for developing a real quadcopter platform and their specifications and characteristics are first introduced. Then, according to the working voltage and interactive mode of these hardware components, the schematic diagram of this autonomous quadcopter is designed and presented.

5.1 hardware introduction

In this section, the hardware components such as IMU, electronic compass, barometer, GNSS module, actuators, and their features and specifications are introduced. The hardware components used to develop the quadcopter platform include:

1. STM32 development board
2. MPU-6050 gyroscope/accelerometer
3. QMC5883L compass module
4. FlySky FS-i6X transmitter and FS-IA6B receiver
5. Electronic speed controller
6. SD card adapter
7. APC220 radio communication module
8. NEO-M8N GNSS module
9. MS5611 barometer

5.1.1 STM32 development board

The STM32 development board used as the flight controller is the STM-32F103C8T6 (Figure 5.1). This development board has very low power consumption, very strong computing ability, and open-source hardware resources. Hence, it's ideal as the core of a flight control system. The MCU of this board is based on an ARM 32-bit Cortex-M3 CPU core, and the maximum CPU frequency is 72MHz, which makes this board very computational compared to an Arduino board (the CPU only runs at 8MHz or 16 MHz). The board works with 5V but the MCU works with 3.3V, hence this development board houses a 5V to 3.3V voltage regulator IC. Even though the MCU operates at 3.3V, most of its GPIO pins are 5V tolerant. This development board also has two header pins which can be used to toggle the MCU boot mode between programming mode and operating mode. In addition to the above features, this board also features 20 Kbytes of SRAM, up to 128Kbytes of Flash memory, and up to 9 communication interfaces. More detailed information about its specifications is shown in Table 5.1 below.

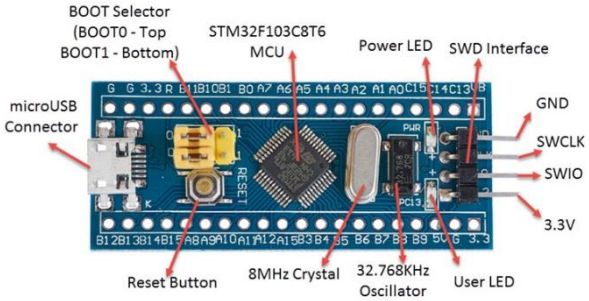


Figure 5.1 STM32F103C8T6

As mentioned above, two header pins (boot 0 and boot 1) are used to select the memory from which the MCU boots. When both boot 0 and boot 1 pins are set to low (0), then the internal flash memory acts as the main boot space and when boot 0 is set to high (1) and boot 1 is set to low (0), the system memory acts as the main boot space. To upload code to the flash memory of the MCU, the system memory must be selected as the main boot space. By booting into the system memory, the flash memory will be reprogrammable. This is the programming mode of this board. In programming mode, every reset or power-off will clear the code uploaded to the board. Once the program is uploaded to the flash memory, switch back the boot 0 to low (0), so that from the next reset or power-up, the MCU will boot from the flash memory, and the code uploaded will be preserved. This is the operating mode of this board.

STM32F103C8T6	
Operating Voltage	2.7V to 3.6V
CPU Frequency	72 MHz
Flash Memory	128 KB
RAM	20 KB
Number of GPIO pins	37
Number of PWM pins	12
Analog input pins	10 (12-bit)
USART peripherals	3
I2C peripherals	2
SPI peripherals	2
Can 2.0 peripheral	1
Timers	4

Table 5.1 STM32F103C8T6 specifications

In this project, we use the Arduino IDE to program this STM32 board. The code written on the Arduino IDE can be uploaded to the STM32 board by two methods. One is through the interface headers on the board, for this, the st-link debugger is required. The other one is through UART, and for this, a USB to TTL module is required. In this project, the FT232RL FTDI adapter (Figure 5.2) is used as the USB to TTL module, with which the Arduino code can be uploaded to the board with the serial method.

Figure 5.2 FT232RL FTDI adapter

5.1.2 MPU-6050 gyroscope/accelerometer

The IMU used to measure the attitude and rotation speed of the quadcopter during flight is MPU-6050 (Figure 5.3), which is a 6-axis Motion Tracking device that combines a 3-axis gyroscope, a 3-axis accelerometer, and a Digital Motion Processor (DMP) all in a very small package. This IMU features a small size, low power consumption requirements, high accuracy, and high reputability. Also, it's pretty simple to interface with microcontrollers and other sensors such as magnetometers. More detailed information about its specifications can be found in Table 5.2 below.

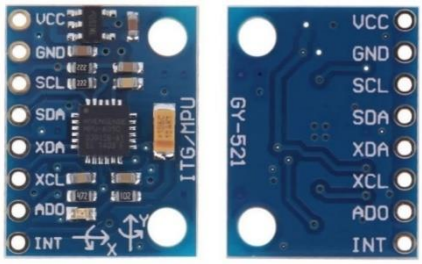


Figure 5.3 MPU-6050 with the GY-521 breakout board

Gyroscope Features	Accelerometer Features
3-Axis sensor with a user-programmable full-scale range of ±250, ±500, ±1000, and ±2000 degrees per second	3-Axis sensor with a user-programmable full-scale range of ±2g, ±4g, ±8g, and ±16g
16-bit ADCs	16-bit ADCs
Digitally programmable low-pass filter	Orientation detection and signaling
Factory calibrated sensitivity scale factor	User-programmable interrupts
Up to 8000Hz output data rate	Up to 1000Hz output data rate
Electrical and other common specifications	
9-Axis motion fusion by the on-chip Digital Motion Processor (DMP)	
VDD supply voltage range of 2.375V-3.46V	
1024-byte FIFO buffer reduces power consumption	
Digital-output temperature sensor	
400kHz Fast Mode I2C for communication	
Internal clock oscillator	

Table 5.2 MPU-6050 specifications

5.1.3 Compass module

The compass module used in this project is QMC5883L (Figure 5.4), which is a multi-chip three-axis magnetic sensor. This sensor is based on state-of-the-art magneto-resistive technology and has the advantage of low noise, high accuracy, low power consumption, offset cancellation, and temperature compensation [47]. The QMC5883L compass module can achieve 1° to 2° compass heading accuracy. More detailed information about its specifications can be found in Table 5.3 below.

Figure 5.4 QMC5883L compass module

QMC5883L	
Supply voltage	2.16V to 3.6V
Full-scale range	±8 Gauss
Resolution (ADC)	16 bits
Gauss resolution	±2mG to ±8G
Output data rate	10Hz, 50Hz, 100Hz, 200Hz
Interface	I2C
I2C rates (kHz)	100, 400
Operating temperature	-40°C to 85°C

Table 5.3 QMC5883L specifications

5.1.4 Transmitter and receiver

The transmitter and receiver make up the radio control system of the quadcopter. The transmitter is an electronic device that uses radio signals to transmit commands wirelessly to the radio receiver that is connected to the quadcopter flight controller. The transmitter reads the stick inputs (control commands) and sends them through a PWM or PPM signal to the receiver in near real-time.

The radio receiver is the device capable of receiving commands from the radio transmitter. A transmitter uses many frequencies like 27MHz, 72MHz, 433MHz, 900MHz, 1.3GHz, and 2.4GHz. Most transmitters work on a 2.4GHz radio frequency. A receiver's frequency must be compatible with the transmitter one to establish communication, which means a 2.4GHz transmitter can only work with the 2.4GHz radio receiver.

In this project, the FS-i6X transmitter with FS-iA6B receiver (Figure 5.5) is utilized as the radio

control module of the quadcopter. The FS-i6X transmitter features 6 channels and works in the frequency of 2.4GHz with PPM output and the FS-iA6B is the matched 6 channels receiver that can receive PPM signal. The specifications of the transmitter and receiver are summarized and listed in Table 5.4 and Table 5.5, respectively.

Figure 5.5 Flysky FS-i6X transmitter with IA6B Receiver

FS-i6X transmitter	
Channels	6-10 (Default 6)
RF range	2.408-2.475GHz
Bandwidth	500KHz
Range	500-1500m (in the air)
Stick resolution	4096
Low voltage warning	Less than 4.2V
Power supply	6V DC 1.5AA*4
Size	174×89×190mm
Weight	392 g
Temperature range	-10˚C to +60˚C

Table 5.4 Specifications of FS-i6X transmitter

IA6B receiver	
Channels	6
RF range	2.408-2.475GHz
RF power	Less than 20dBm
RF receiver sensitivity	- 105dBm
Range	500-1500m (in the air)
Power supply	4.0-8.4V
Temperature range	-10˚C to +60˚C
Size	47×26.2×15mm
Weight	10 g

Table 5.5 Specifications of IA6B receiver

5.1.5 Electronic Speed Controller

The electronic speed controller or ESC controls the motor of the quadcopter, it takes the control signal (PWM signal) generated by the flight controller as input and controls and regulates the rotational speed of the motor. The ESCs raise or lower the voltage to the motor according to the duty cycle of the input PWM signal, thus changing the speed of the motor. ESCs have a refresh rate in Hertz (Hz), which is how many times a second the motor speed can be regulated. The ESCs for quadcopters and other multirotor drones may have higher refresh rates, as their stability and maneuverability depend entirely on the balance of motor speeds, and as such, they require fine control over the motor speeds.

In this project, four XXD HW30A Brushless Motor ESCs (Figure 5.6) are utilized to control and regulate the motors of the quadcopter. More detailed information about the ESC's specifications is available in Table 5.6 below. The four ESCs receive the PWM signals from the PB6, PB7, PB8, and PB9 pins of the STM32 board respectively. The STM32 flight controller computes the desired PWM pulse width values and generates PWM signals with the STM32 timer. More detailed information about how to generate PWM signals with STM32 and how to set up the timer configurations can be found in Appendix B PWM signal generation with STM32 timer.

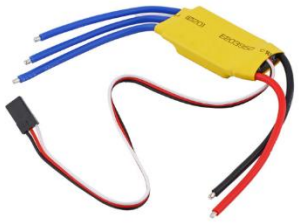


Figure 5.6 Electronic speed controller

HW30A Brushless Motor ESC	
BEC power	1.5A/5V
Cut off voltage	4V
Current	30A
Size	57mm×25mm×8mm
Net weight	27g

Table 5.6 ESC specifications

5.1.6 NEO-M8N GNSS module

In this project, the GNSS module used to position the quadcopter during outdoor flight is a u-blox GNSS module from the NEO-M8 series (Figure 5.7). The NEO-M8 modules utilize concurrent reception of up to three GNSS systems (GPS/Galileo together with BeiDou or GLONASS),

recognize multiple constellations simultaneously, and provide outstanding positioning accuracy in scenarios where urban canyon or weak signals are involved [53]. This GNSS module can measure the latitude, longitude, altitude, velocity, and also the date information of the quadcopter during flight. The horizontal position accuracy can be up to 2 meters with concurrent reception from two GNSS systems, but the altitude position accuracy is much worse compared to the horizontal one (almost ten times worse). Hence the altitude measured by this GNSS module can't be used by the flight controller, and a more precise altitude sensor such as the barometer or the LiDAR is required to get the accurate altitude of the quadcopter. The interface between this GNSS module and the microcontroller can be easily achieved by UART or SPI bus interface. Some basic information about this GNSS module is listed in Table 5.7 below.

Figure 5.7 NEO-M8N GNSS module

NEO-M8N GNSS module	
Power supply voltage	3.3V to 5V
Sensitivity	-167dBm
Update rate single GNSS	Up to 10Hz
Update rate 2 concurrent GNSS	Up to 5Hz
Horizontal position accuracy	2m
Time to first fix	26s for cold start and 1s for hot start
Operating temperature	-40℃ to +85℃
Altitude limit	50000m

Table 5.7 Specifications of the NEO-M8N

5.1.7 MS5611 barometer

Since the GNSS module can't provide accurate altitude information of the quadcopter during flight, an MS5611 barometer is used as the altitude sensor (Figure 5.8). The MS5611 is a new generation of high-resolution altimeter sensors from MEAS Switzerland with SPI and I2C bus interface and it's optimized for altimeters and variometers with an altitude resolution of 10 cm [52]. Table 5.8 below lists more detailed information about the specifications and features of this barometer. One important thing that should be noted when using this barometer is that it's very light sensitive and need to be protected from sunlight. In this project, a small black cage is utilized to protect this barometer from the sunlight during outdoor flight while at the same let the air flow in and out to

make sure this barometer can measure the air pressure and work properly.

Figure 5.8 MS5611 barometer

MS5611 Barometer	
Power supply	1.8 V to 3.6 V
Operating temperature	-40℃ to +85℃
ADC	24 bits
Pressure range	10 mbar to 1200 mbar
Oversample rate	256/512/1024/2048/4096
Pressure resolution	0.065/0.042/0.027/0.018/0.012 mbar
Response time	0.5/1.1/2.1/4.1/8.22 ms
Temperature range	-40℃ to +85℃
Temperature resolution	<0.01℃
Resolution RMS	0.012/0.008/0.005/0.003/0.002℃

Table 5.8 Specifications of MS5611 barometer

5.1.8 SD card and SD card adapter

To collect quadcopter data such as position, attitude, and velocity during flight for subsequent analysis, a data storage module is mandatory. The SD card is one of the most practical one among storage devices. With the corresponding SD card adapter, the flight controller can communicate with the SD card and write or read data on it. The SD card adapter interfaces with the flight controller through the Serial Peripheral Interface (SPI) bus and the communication between them can be easily implemented with the Arduino SD library. The pinout of the SD card adapter is shown in Figure 5.9 below. MISO is the Master In Slave Out line, this line transfers data from the SD card adapter (the slave device) to the flight controller (master device). MOSI is the Master Out Slave In line, this line transfers data from the flight controller to the SD card adapter. SCK is the clock line that synchronizes the data transmission between the master and slave. CS is the chip select line, which is used by the flight controller (master device) to enable and disable a specific device on the SPI bus. The specifications of the SD card adapter are listed in Table 5.9 below.

Figure 5.9 SD card adapter

The SD card used in this project is shown in Figure 5.10, the storage capacity is 2GB, which is more than enough for collecting data during flight.

Figure 5.10 SD card

SD card adapter	
Operating voltage	4.5V to 5.5V DC
Current requirement	0.2mA to 200mA
3.3V on-board voltage regulator Supports FAT file system Supports micro SD up to 2GB Supports Micro SDHC up to 16GB	

Table 5.9 Specifications of SD card adapter

5.1.9 radio communication module

To monitor the flight conditions and have real-time information on the key components of the quadcopter during outdoor flight, a radio communication module is required. The radio communication module allows real-time communication between the quadcopter and the PC in the form of a data stream. In this project, the radio communication module is utilized to send commands from the PC to the quadcopter or to receive signals back from the quadcopter for monitoring its conditions. The radio communication module used in this project is the APC220 radio communication module (Figure 5.11). Two APC220 modules are required for the data transmission, one is wired to the flight controller and interfaces with the flight controller with UART protocol, and the other is connected to the PC through a USB to TTL converter. More detailed information about the specifications of the APC220 module is in Table 5.10 below.

Figure 5.11 APC220 radio communication module and USB to TTL converter

APC220 radio communication module	
Working frequency	431 MHz to 478 MHz
Power supply	3.3V to 5.5V
Current	<25-35mA
Working temperature	-20˚C to 70˚C
Range	1200m line of sight (1200 bps)
Interface	UART/TTL
Baud rate	1200-19200 bps
Baud rate (air)	1200-19200 bps
Receiver buffer	256 bytes
Size	37mm × 17mm × 6.6mm
Weight	30g

Table 5.10 APC220 specifications

5.2 Schematic of the quadcopter

The schematic of the quadcopter is designed taking into account the operating voltages of the individual components and their interaction method with the STM32. The designed schematic is shown in Figure 5.12 below. The STM32 board accesses data from the IMU, the compass, and the barometer through the I2C (Inter Integrated Circuit) bus. Hence, the I2C interfaces of these sensors should be connected to the PB10 and PB11 pins of the STM32 board. The IMU and the barometer work at 5V, they should be wired to the 5V pin of the board, while the compass works at 3.3V and should be wired to the 3.3V pin. The GNSS module uses the UART (Universal Asynchronous Receiver-Transmitter) protocol to transmit data to the STM32 board, and its RX and TX pins should be wired to the PA2 and PA3 pins of the board. The telemetry module communicates with the STM32 board through the UART protocol too, and its RX and TX pins should be wired to the PA9 and PA10 pins of the board. The FTDI adapter is used to upload codes to the STM32 board, and it shares the same RX and TX pins of the board with the telemetry module. PB6, PB7, PB8, and PB9 correspond to the four channels of STM32 timer2, they are wired to the ESCs to transmit PWM signals generated by the STM32 timer. The four ESCs are powered by a 12V battery. One of the ESCs outputs 5V voltage and is used to power the STM32 board. The receiver has six channels and channel 1 is used to receive PPM control signals from the transmitter. The receiver is powered by

5V and the channel is wired to the PA0 pin of the board. The SD card adapter communicates with the STM32 through SPI (Serial Peripheral Interface) protocol. The CS (Chip Select) pin should be wired to PA4, the SCK (Serial Clock) pin should be wired to PA5, the MOSI (Master Out Slave In) pin should be wired to PA7, and the MISO (Master In Slave Out) pin should be wired to PA6. The SD card adapter must be wired to 5V otherwise it won't work.

All the sensors, receiver, telemetry module, SD card adapter, and STM32 board are mounted on a Printed Circuit Board (PCB) and connected with wires.

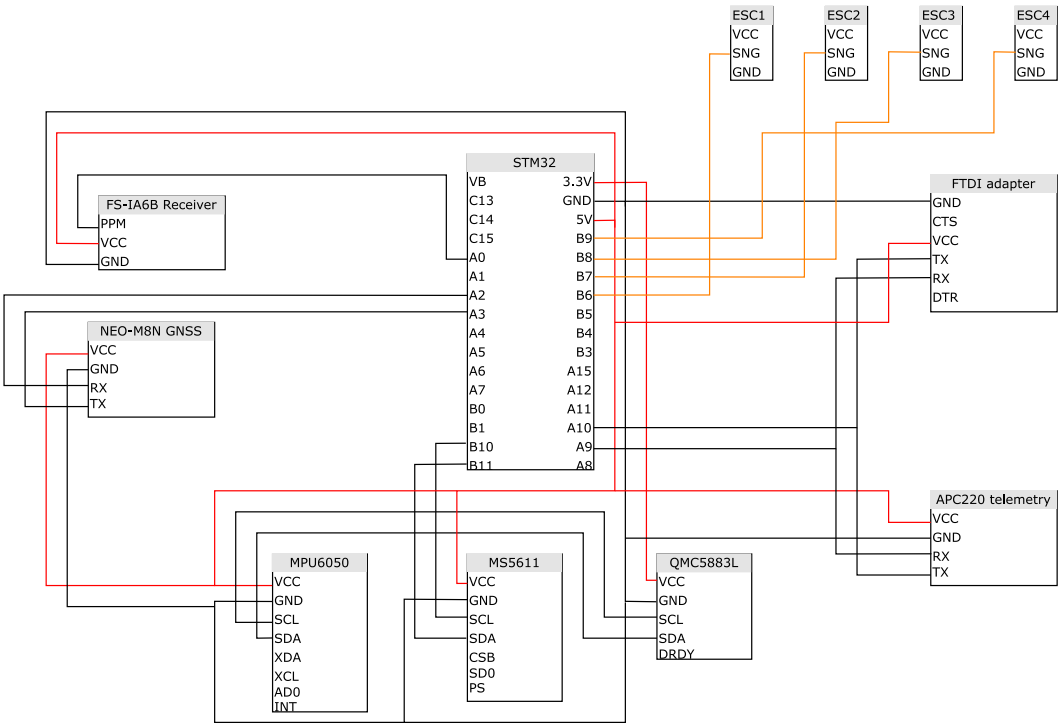


Figure 5.12 Schematic of the flight controller

5.3 Flight controller program architecture

In this section, the program architecture of the designed flight controller is introduced. The program is based on Arduino STM 32 and written with Arduino IDE.

The flowchart of the flight controller program is shown in Figure 5.13 below. Like most Arduino programs, this flight control program has three parts: the pre-setup part, the setup part, and the main loop part. The pre-setup part contains the libraries used by the program and all the variables and arrays are defined and initiated here. The setup part does some preparatory work before the main program loop runs. In this flight controller program, the pin mode configuration, the STM32 timer setup, all the sensors' setup, sensors' calibration, and storage device initialization are implemented in the setup part. The main loop part is executed after the setup part and it will be executed over and over again during the flight. The sensor data accessing, the quadcopter position and attitude estimation, the flight controller, and the data storage are all implemented in the main loop part.

Below, the tasks of all the functions used in this program are explained:

1. *timer_setup*. The STM32 timer2 is configured as input capture mode to receive signals from the radio controller, and the STM32 timer4 is configured to generate the PWM signal, which is used to control the speed of the motor.
2. *gps_setup*. Configure the GNSS receiver and set the refresh rate from 1Hz (default) to 5Hz and the baud rate from 9600 (default) to 57600.
3. *gyro_setup*. Set the full-scale range and digital low pass filter of the gyroscope and accelerometer.
4. *calibrate_gyro*. Calculate the bias of the gyroscope.
5. *gyro_signalen*. Access raw data from the gyroscope and the accelerometer.
6. *angle_calc*. Calculate attitude angles from the gyroscope and accelerometer data and perform a complementary filter to avoid noise and drift.
7. *setup_compass*. Set the full-scale range, data output rate, oversample rate, and operation mode of the electronic compass.
8. *read_compass*. Access raw data from the compass.
9. *Azimuth_calc*. Calculate the heading angle of the quadcopter relative to the North according to the compass measurements.
10. *read_barometer*. Access data from the barometer and calculate the altitude of the quadcopter from the barometer measurement. When the quadcopter is in autonomous flight mode, the attitude controller branch will be executed.
11. *read_gps*. Extract geographic coordinates (latitude and longitude) from the GNSS receiver output. When the quadcopter is in autonomous flight mode, the GNSS position controller branch will be executed.
12. *LQR_attitude*. Take roll and pitch angle reference as input and calculate control effort according to the LQR control strategy.
13. *start_stop_takeoff*. Detect the quadcopter's status.
14. *receiver_handle*. Parse PPM Signals from the radio controller.
15. *print_on*. Collect data (GNSS position, attitude, altitude) from sensors during flight to an SD card.

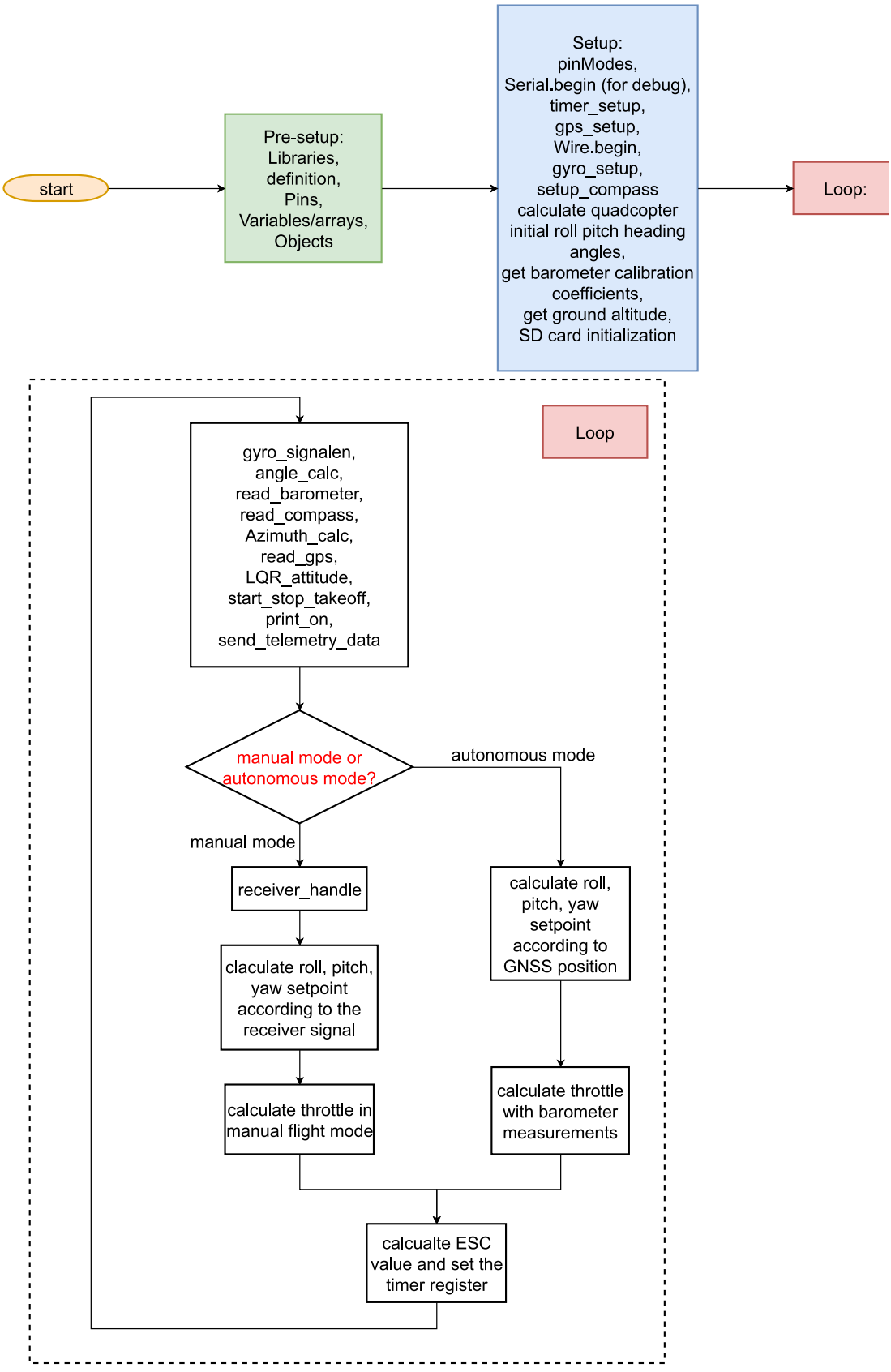


Figure 5.13 Flowchart of the program

Chapter 6: Experiment result

In this chapter, the real quadcopter platform is presented and experiments are implemented to validate the performance of the quadcopter when flying in manual control mode. The flight controller program is deployed on this quadcopter and flight tests are implemented to validate its control performance.

6.1 Quadcopter platform

The real quadcopter platform based on the STM32 development board is shown in Figure 6.1 below. This quadcopter platform consists of several different modules such as the sensor module, the flight controller, the actuator module, the radio controller module, the radio communication module, and the data storage module.

1. *Sensor module*. The sensor module consists of an IMU, an electronic compass, a GNSS receiver, and a barometer that are utilized to measure the crucial information of the quadcopter during flight. The IMU is used to measure the roll and pitch angles of the quadcopter. The electronic compass is used to measure the heading angle of the quadcopter relative to the North, with which advanced features like GNSS position hold and waypoint fly can be added to the quadcopter. The barometer measures the flight altitude based on changes in air pressure. As for the GNSS receiver, it is used to position the quadcopter in the horizontal plane.
2. *Flight controller*. The flight controller is the core of the quadcopter platform. In manual flight mode, the flight controller takes the sensor measurements and the pilot command signals as input to compute and generate the desired control signals. In autonomous flight mode, the flight controller works in conjugation with the sensor module to complete the tasks scheduled by the program.
3. *Actuator module*. The actuator module consists of four DC brushless motors and their corresponding electronic speed controllers. Each electronic speed controller takes the control signals from the flight controller as input to control and regulate the rotational speed of the motor. The four motors work together to generate control actions for the quadcopter.
4. *Radio controller module*. The radio controller module consists of a radio controller and a receiver. The radio controller is used to send the pilot's control commands to the quadcopter wirelessly. The receiver accepts the control commands and transmits these control commands to the flight controller.
5. *Radio communication module*. The radio communication module consists of two radio communication devices, one is mounted to the flight controller and the other is connected to the PC. The radio communication module is used to monitor the status of the quadcopter wirelessly while flying.
6. *Data storage module*. The data storage module is used to save the quadcopter's data and other

auxiliary information during flight, which can be used for subsequent analysis and flight tests. It is important to note that the barometer sensor is very susceptible to sunlight and being exposed to sunlight will seriously affect its measurement accuracy. Exposure to sunlight can cause a significant drift in barometer measurements. To overcome this issue, the barometer is packed in a small black box and placed below the PCB of the quadcopter. In this way, the effect of sunlight on the barometer can be counteracted.

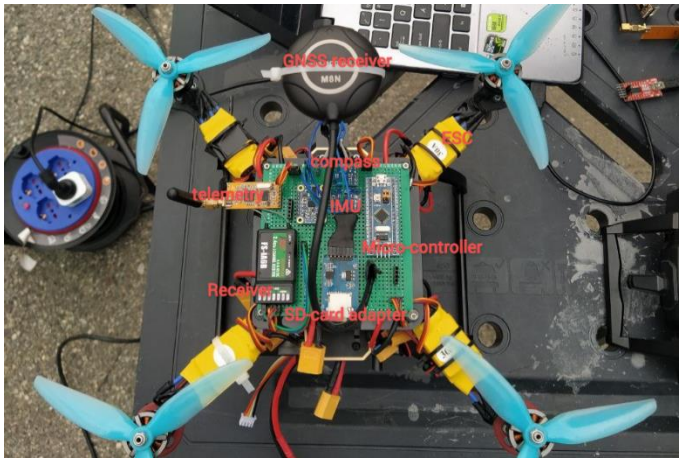


Figure 6.1 Quadcopter developed for implementing flight test (Top view)

6.2 Manual flight test

In this section, the results of the manual flight tests are presented. The main purpose of the manual flight test is to validate the performance of the designed quadcopter when flying in manual control mode. Three manual flight tests are performed and three different LQR designs are implemented on the attitude controller.

The parameters of the quadcopter (mass and moment of inertia) should be modified since the quadcopter in manual flight is slightly heavier than the quadcopter referenced when doing the simulation. Electronic components such as the compass, the barometer, the GNSS receiver, and the radio communication module are added to the quadcopter, leading to the change in its parameters. Among these components, the GNSS receiver has the greatest influence on the changes in the quadcopter's parameters since it's heavier than the other components, and is supported by a rod and placed away from the center of gravity of the quadcopter. Instead of implementing a pendulum test again to measure the mass and inertia moment of the quadcopter, the quadcopter parameters used in the simulation are multiplied by a scale factor as the new parameters. During the experiment, this scale can be tuned until good performance is achieved.

For manual flight test 1, The weighting matrices Q and R are the same as the weighting matrices used during the simulation (see Table 6.1). The results of the manual flight test 1 are shown in Figure 6.2 below. The experimental result shows both the roll and pitch angle can achieve acceptable and satisfactory tracking performance to the command. The roll angle can track the reference but there is some tracking error and overshooting. In the vast majority of cases, the roll angle tracking error is less than 5 degrees. The pitch angle's tracking performance is better than that of the roll angle.

To overcome the roll angle tracking error, the first entry of the weighting matrix Q (this entry is related to the roll angle control) is slightly increased to increase the response speed of the roll angle controller. This can make the quadcopter more reactive to the roll command. The modified weighting matrix is shown in Table 6.1 below. Manual flight test 2 is implemented to validate the control performance. The results of the manual flight test 2 are shown in Figure 6.3 below. Manual flight test 3 further increases the first component of the Q matrix based on manual flight test 2. The modified weighting matrix is shown in Table 6.1 and the test results are shown in Figure 6.4 below. The results of flight test 2 and flight test 3 show that the roll angle tracking performance is slightly improved by increasing the first entry of the Q matrix. Figure 6.5 demonstrates the roll angle tracking error during flight test 2, it can be seen that in the vast majority of cases, the roll angle tracking error is less than 5 degrees. Those particularly large errors are due to the slow response of the flight controller.

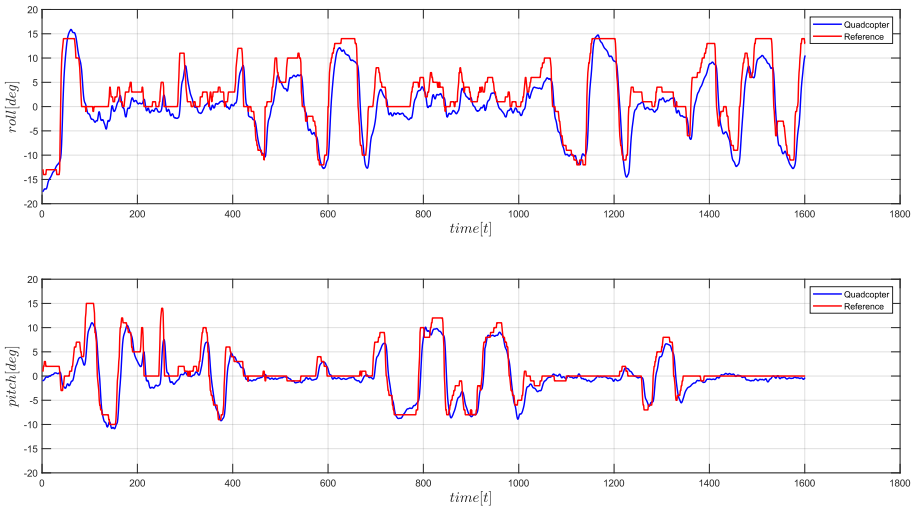


Figure 6.2 Manual flight test 1, weighting matrices are the same as the one used in the simulation

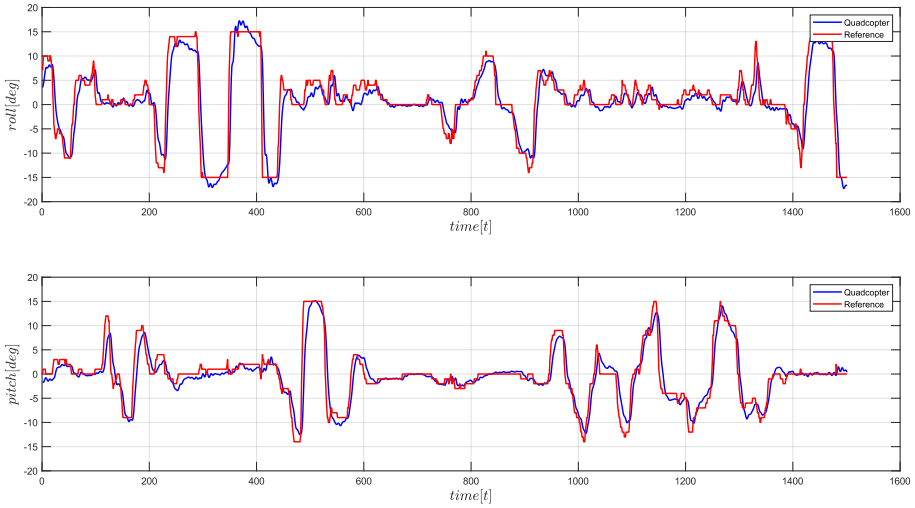


Figure 6.3 Manual flight test 2, the first component of the weighting matrix Q is slightly increased

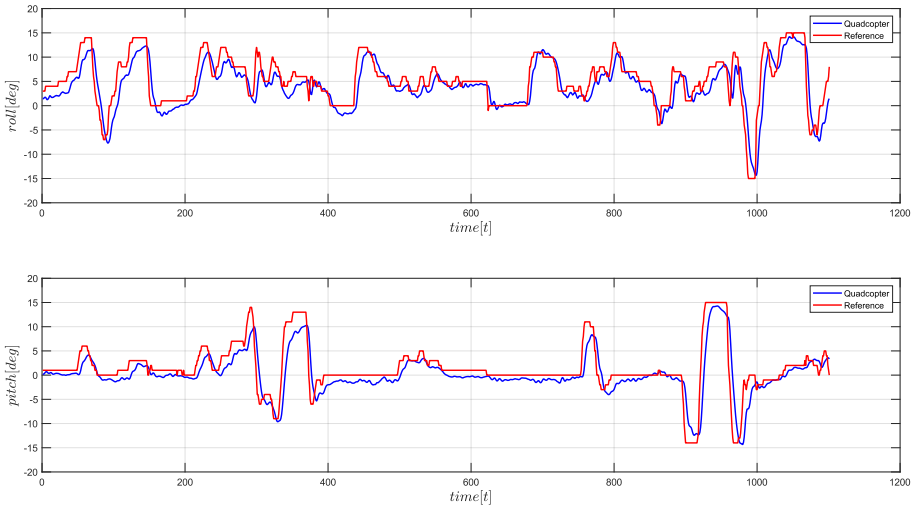


Figure 6.4 Manual flight test 3 result

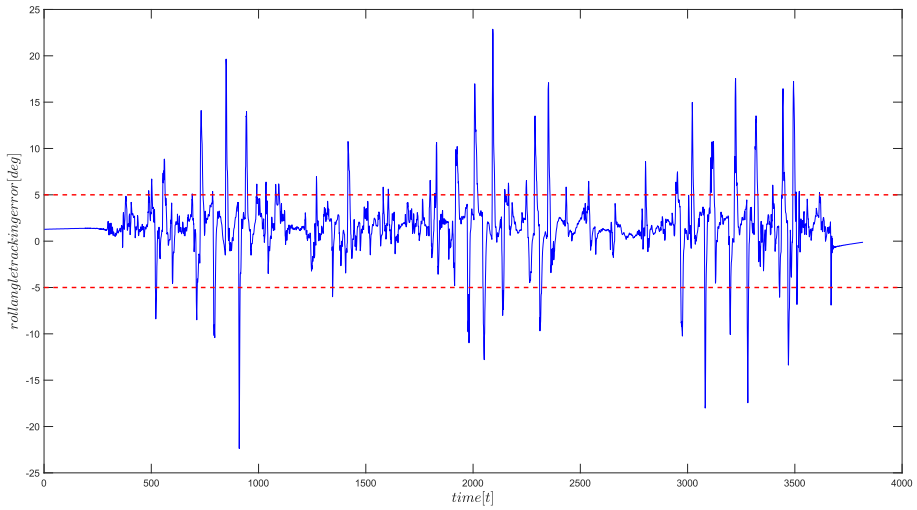


Figure 6.5 Manual flight test 2, roll angle tracking error

Test	Q	R
Simulation	$diag(0.05,0.07,0.1,0.001,0.001,0.1)$	$diag(1,1,1) * 10^{-6}$
# test 1	$diag(0.05,0.07,0.1,0.001,0.001,0.1)$	$diag(1,1,1) * 10^{-6}$
# test 2	$diag(0.08,0.07,0.1,0.001,0.001,0.1)$	$diag(1,1,1) * 10^{-6}$
# test 3	$diag(0.09,0.07,0.1,0.001,0.001,0.1)$	$diag(1,1,1) * 10^{-6}$

Table 6.1 Weighting matrices used during flight test

Chapter 7: Conclusion and future works

In this thesis, a quadcopter autonomous flight controller based on the Linear Quadratic Regulator (LQR) control strategy is designed and presented. Considering the under-actuated characteristics of the quadcopter, the flight controller adopts an inner-outer loop structure. The inner loop corresponds to the fast dynamics of the quadcopter and is related to the attitude and altitude control. The outer loop corresponds to the slow dynamics of the quadcopter and is related to the horizontal position control. This inner-outer loop structure has been proved to be more suitable for quadcopter control. The flight controller is designed and tuned in a simulation environment considering the quadcopter's dynamic model. The performance of the designed flight controller is validated and evaluated in simulation, taking into account the step response performance and trajectory tracking performance. Then, an open-source quadcopter based on STM32, being able to interact with various sensors such as IMU, compass, barometer, and GNSS receiver, with remote communication and data storage functions, is designed. The designed flight controller is deployed on this quadcopter to achieve manual control flight, autonomous flight, GNSS position hold, waypoint fly, autonomous return, and other functions. To deploy the designed flight controller to the quadcopter, the flight control program based on Arduino STM32 is developed. The programming language is C++. Finally, manual flight tests are carried out on the quadcopter platform to verify and evaluate its manual flight performance and achieved good results.

During the simulation, the designed controller shows a good step response performance with a fast response speed, small oscillation, small overshoot, and very small steady-state error. Due to the property of the LQR control strategy, the steady-state error always presents in step response. A solution to this problem could be using the LQI control. The LQI control is a control strategy adding an integration action to the LQR control. The trajectory tracking simulation is performed assuming the yaw angle is always zero, and the trapezoidal speed profile is used to generate a trajectory between two waypoints. The simulation result demonstrates the designed controller achieves good trajectory tracking performance, especially with altitude tracking. But the horizontal position tracking has two drawbacks: (1) velocity tracking shows a large rising/settling time, leading to a lag in position tracking. and (2) the flight path can only be very close to each waypoint, but cannot reach and pass through these waypoints. The possible solution to these problems could be adding integration action to the LQR controller or designing a controller using nonlinear control strategies such as SMC. In general, the simulation results show that despite the above flaws, the designed LQR controller can achieve satisfactory performance in step signal response and trajectory tracking.

The designed open-source quadcopter comprises various sensors and other auxiliary electronic components. The IMU is used to measure the quadcopter's attitude. The attitude calculation method based on Euler angle & rotation matrix is utilized to estimate the roll pitch angles of the quadcopter from IMU and a complementary sensor fusion algorithm is implemented to avoid noise and drift during IMU measurements. Experimental tests demonstrate that with this method, the attitude measurement is very accurate, reliable, and reactive. A tilt-compensated compass is utilized to measure the heading angles of the quadcopter during flight to achieve features like GNSS/GPS

position hold and waypoint fly. The calibration method based on scale biases is implemented to lighten and remove the influence of soft iron distortion and hard iron distortion on the compass measurement. Results show that this tilt-compensated compass can perform accurate heading angle estimation. The GNSS receiver interacts with the quadcopter flight control through the UART port and its data output rate and baud rate are configurated from default values to 5Hz and 57600 bits/s, respectively. By doing this, the quadcopter flight controller can get more position information from the GNSS receiver and achieve more smooth position control. The barometer measures the absolute air pressure, and the flight altitude of the quadcopter can be computed according to the relationship between the air pressure and altitude. A digital low pass filter is implemented on the barometer measurements to avoid the influence of the noise. Other auxiliary components such as telemetry and SD card adapter are integrated into the quadcopter to realize functionality like radio communication and data collection. Overall, this quadcopter integrates various electronic components and is a good example of system integration.

To deploy the designed autonomous flight controller to the quadcopter, the flight control program based on Arduino STM32 is developed. The quadcopter is programmed to fly in manual flight mode or autonomous flight mode. Experiments have been carried out to evaluate manual flight performance. During the experiment, the weighting matrix Q is fine-tuned to achieve good flight performance. Results demonstrate that the designed controller can achieve very good control performance when the quadcopter is in manual flight mode. Although experimental data show that there is a tracking error of 1 degree to 5 degrees in the roll angle reference tracking, during the experiment, even in the presence of gusts, the aircraft can still achieve a very stable flight. The excellent performance of manual flight also lays a foundation for the later autonomous flight test and tuning.

There is still much room for improving this project and a path for the continuous work of this project could be:

1. *Improve current simulation model*. The simulation model can be improved and closer to reality by adding the state estimation/observer block and the environment simulation block. The sensors' mathematical model and the sensor fusion algorithms that are used to estimate states can be included in the current model to achieve more realistic state feedback. Also, modeling the state estimation block can help deal with the sensor noise and different sample ratios of sensors. The environment simulation block can model the effects of the environment on the quadcopter such as the effects of aerodynamics, which can improve the robustness of the designed flight controller.
2. *Test and tune the autonomous fly*. Manual flight tests have shown the excellent performance of the attitude controller. Experiments should be carried out to test the altitude and position controller on the real quadcopter and fine-tune the controller parameters.
3. *Add GNSS + IMU sensor fusion and barometer + IMU sensor fusion*. The current stage of the work uses the position and altitude measurements directly from the GNSS module and the barometer. The GNSS module has the problems of low measurement accuracy, low measurement frequency, and weak signals or even signal loss due to the influence of surrounding buildings. Implementing the GNSS and IMU sensor fusion with a Kalman filter or a complementary filter can help deal with the problem of low measurement frequency and achieve more accurate horizontal position and speed measurement. As for the barometer, it's

susceptible to sunlight and disturbances in the surrounding airflow, which will cause the altitude measurement to drift. Implementing the barometer and IMU sensor fusion can handle the problem of barometer interference and achieve more altitude and vertical speed measurements.

4. *Implement a nonlinear control strategy*. The linear model of the quadcopter is a simplification of its mathematical model. The linear controller based on the linear model can't handle all the dynamic behavior of the quadcopter. A solution to this problem can be implementing the nonlinear controller to the quadcopter.

Appendix A Propeller's thrust and torque measurements

To find the mathematical relationship between the PWM signal provided to the ESC from the flight controller and the thrust and torque generated by the motor, the propeller's thrust and torque tests are mandatory. This allows the conversion between the commands generated by the flight controller expressed as the PWM signals and the physical input vector u defined in chapter 2.

$$u = \begin{bmatrix} F_t \\ M_x \\ M_y \\ M_z \end{bmatrix} \tag{A-1}$$

The measurements of the torque and thrust generated by the motor are implemented on the RCBenchmark Series 1580 test stand (Figure A.1), on which it is possible to measure up to 5kgf of thrust and 2 Nm of torque as well as voltage, current, power, motor rotation speed, vibration, and efficiency. The test stand has three measurement sensors, one of which is for measuring the thrust and the left two are for measuring the torque. The board on the test stand provides the PWM signal to the motor ESC and allows connecting the stand to a PC with a USB cable. Using the support software of this test stand, it's possible to manually control the pulse length of the PWM signal and collect the test data. To collect a sufficient number of samples, the software is set to continuous sample mode and the PWM pulse length is manually increased from 1000us to 1650us.

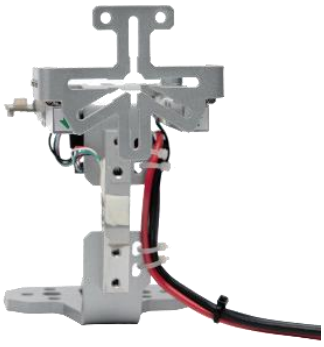
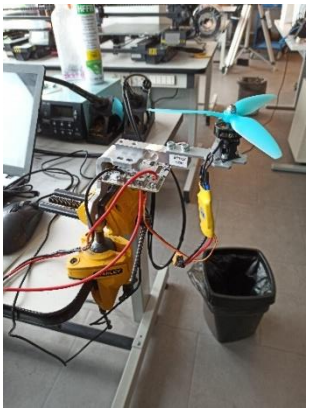


Figure A.1 RCBenchmark Series 1580 test stand

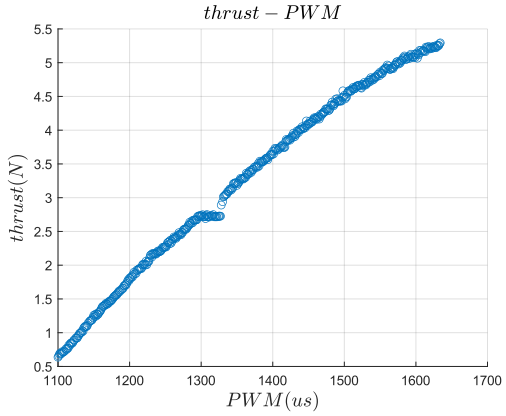


(a)

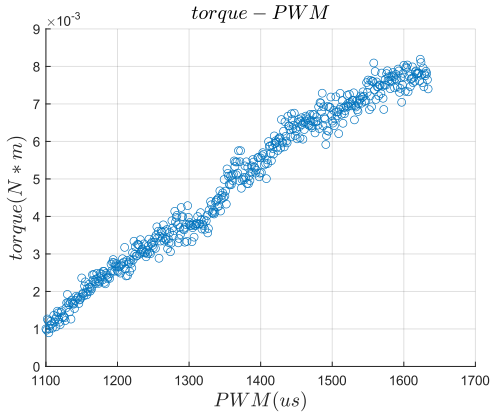
(b)

Figure A.2 Pictures during the experiment

The behavior of one motor is shown in Figure A.3 below, and the experimental data shows that the torque-PWM and thrust-PWM characteristics of the motor show a nearly linear behavior.



(a) Thrust behavior



(b) Torque behavior

Figure A.3 Behavior of single motor

Linear approximation of the torque-PWM and thrust-PWM characteristics can be derived using the Least Square method. The obtained data can be fitted using first-order functions, Take the measurement error e into consideration, the model equation can be suitably written in the following format:

$$F = n_F \cdot PWM + q_F + e_F$$
$$T = n_T \cdot PWM + q_T + e_T \quad \text{(A-2)}$$

Where the gain n_F, n_T and the offset q_F, q_T are unknown parameters to be estimated. By considering the N measurements of the thrust collected in the experiment, the following system of linear equations is derived:

$$\begin{aligned} F_1 &= PWM_1 \cdot n_F + q_F + e_{F1} \\ F_2 &= PWM_2 \cdot n_F + q_F + e_{F2} \\ &\vdots \\ F_N &= PWM_N \cdot n_F + q_F + e_{FN} \end{aligned} \quad \text{(A-3)}$$

The previous equations can be written in matrix form:

$$\begin{bmatrix} F_1 \\ F_2 \\ \vdots \\ F_N \end{bmatrix} = \begin{bmatrix} PWM_1 & 1 \\ PWM_2 & 1 \\ \vdots & \vdots \\ PWM_N & 1 \end{bmatrix} \cdot \begin{bmatrix} n_F \\ q_F \end{bmatrix} + \begin{bmatrix} e_{F1} \\ e_{F2} \\ \vdots \\ e_{FN} \end{bmatrix} \tag{A-4}$$

The estimation problem is recast in the standard Least Square format:

$$y = \phi \cdot \vartheta + e \tag{A-5}$$

Where $y \in \mathbb{R}^N$, $\phi \in \mathbb{R}^{N\times 2}$, $\vartheta \in \mathbb{R}^2$ and $e \in \mathbb{R}^N$. Since the unknown is the vector ϑ, the problem to be solved is overdetermined, because the number of unknowns is smaller than the number of measurement equations taken into account and then, in general, the system of equations does not admit any solution, since the matrix ϕ cannot be inverted. Using the least square algorithm as an estimation method, the solution could be:

$$\hat{\vartheta} = (\phi^T \cdot \phi)^T \phi^T \cdot y \tag{A-6}$$

Under MATLAB, the least-squares solution is provided by the operator "\" or the command mldivide, and the result is:

$$\hat{\vartheta} = \begin{bmatrix} \hat{n}_F \\ \hat{q}_F \end{bmatrix} = \begin{bmatrix} 0.0088 \\ -8.8085 \end{bmatrix} \tag{A-7}$$

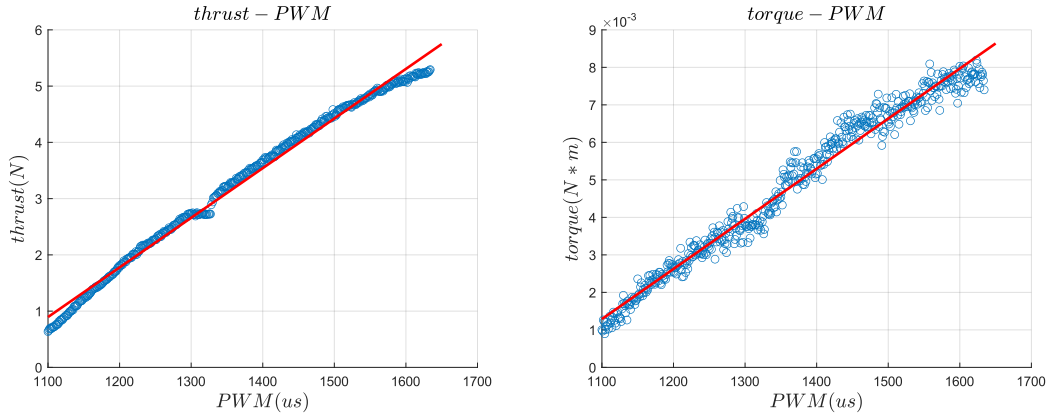
As a result, the thrust F in the function of the PWM signal is:

$$F_{(PWM)} = 0.0088 \cdot PWM - 8.8085 \tag{A-8}$$

Figure A.4 (a) shows the result of overlapping the equation to the experimental thrust curve, the equation can fit the linearity between thrust and PWM signal very nicely. Using the same method, the relationship between the torque T and the PWM signal is:

$$T_{(PWM)} = 0.0000136 \cdot PWM - 0.0134 \tag{A-9}$$

And Figure A.4 (b) shows the result of overlapping this equation to the experimental torque curve.



(a) Estimated thrust behavior (b) Estimated torque behavior

Figure A.4 Estimated single motor behavior

Lastly, the command input can be written in the function of the i^{th} motor PWM signal $PWM^{(i)}$:

$$u_{(PWM)} = \begin{bmatrix} -(F_{(PWM_1)} + F_{(PWM_2)} + F_{(PWM_3)} + F_{(PWM_4)}) \\ (F_{(PWM_1)} - F_{(PWM_2)} - F_{(PWM_3)} + F_{(PWM_4)})Lsin(\alpha) \\ (F_{(PWM_1)} + F_{(PWM_2)} - F_{(PWM_3)} - F_{(PWM_4)})Lcos(\alpha) \\ -T_{(PWM_1)} + T_{(PWM_2)} - T_{(PWM_3)} + T_{(PWM_4)} \end{bmatrix} \tag{A-10}$$

In this formula, PWM_i is the PMW signal to the i^{th} motor.

Appendix B PWM signal generation with STM32 timer

As mentioned in section 4.5, the electronic speed controller (ESC) takes the PWM signal generated by the flight controller to control and regulate the speed of the motor. This section describes how to generate the desired PWM signals using the STM32 microcontroller.

The STM32 timer can be used as a time-based generator to generate a PWM signal. The STM32 timer unit includes a counter register (CNT), a pre-scaler register (PSC), an auto-reload register (ARR), and a repetition counter register (RCR). The timer can generate a PWM signal in edge-aligned mode or center-aligned mode depending on the configuration of the timer register. When the STM32 timer module is set to the PWM generation mode, the counter register (CNT) gets the clock signal from the internal clock and counts up to the auto-reload register (ARR) value, then the output channel pin is driven high. And it remains until the counter register (CNT) value reaches the compare register (CCRx) value, when the counter register (CNT) value is greater than the compare register (CCRx) value, the output channel pin is driven low. And it remains until the counter register (CNT) counts up to the auto-reload register (ARR) value, and so on. This is the edge-aligned PWM generation mode in an up-counting configuration. The frequency of the resulting PWM signal is determined by the internal clock, the pre-scalar (PSC) value, and the auto-reload register (ARR) value. And the duty cycle is determined by the compare register (CCRx) value and the auto-reload register (ARR) value. The formula below can be used for calculating the PWM frequency for the output:

$$F_{PWM} = \frac{F_{CLK}}{(ARR+1)\times(PSC+1)} \tag{B-1}$$

Where F_{CLK} is the internal clock frequency. The PWM duty cycle percentage is controlled by changing the value of the compare register (CCPx), and the duty cycle can be expressed as:

$$DutyCycle_{PWM}[\%] = \frac{CCRx}{ARR}[\%] \tag{B-2}$$

Figure B.1 shows how the auto-reload register (ARR) value affects the frequency (period) of the PMW signal, how the compare register (CCPx) value affects the corresponding PWM signal's duty cycle and illustrates the whole process of PWM signal generation in the edge-aligned up-counting configuration.

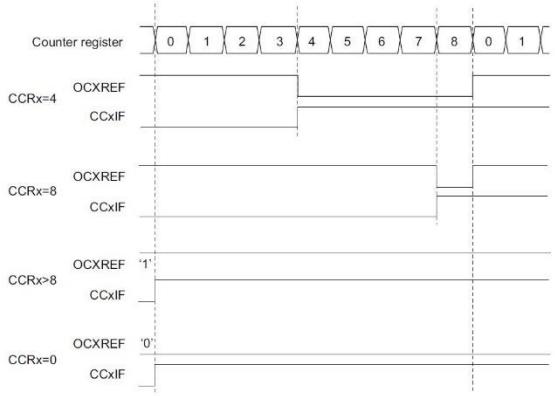


Figure B.1 Edge-aligned PWM waveforms (ARR = 8)

One of the most important features of a PWM signal is the resolution, which is the number of discrete duty cycle levels that the PWM signal can get. The resolution determines how many steps the duty cycle can take until it reaches the maximum value. Hence, the PWM signal resolution can determine how fine the duty cycle can be changed to obtain a certain percentage. This can be extremely important for controlling the quadcopter's motors. The STM32 PWM resolution formula can be expressed as:

$$Resolution_{PWM}[Bits] = \frac{\log\left(\frac{F_{CLK}}{F_{PWM}}\right)}{\log(2)}[Bits] \tag{B-3}$$

$$Resolution_{PWM}[Bits] = \frac{\log(ARR+1)}{\log(2)}[Bits] \tag{B-4}$$

Equation B-3 can be used to calculate the resolution of the PWM signal given a specific frequency or the opposite. Equation B-4 describes the relationship between the auto-reload register (ARR) value and the PWM signal resolution, it can be used to calculate the auto-reload register (ARR) value given a resolution.

The STM32F103C8T6 has four timers. In this project, timer 4 is used to generate the PWM signal, the four channels of which correspond to the PB6, PB7, PB8, and PB9 pins of the board. The pre-scaler register (PSC) value is set to 71, since the internal clock frequency is 72MHz, the counter register (CNT) will count up every 1 microsecond. The auto-reload register is set to 5000. The counter register (CNT) is initiated every 4000 microseconds. Hence, the frequency of the generated PWM signal is 250Hz, which satisfies the requirement that the refresh rate of the controller should be 250Hz. The compare register (CCPx) is set according to the PWM pulse width value calculated by the controller (section 3.2), which is limited to 1000 to 2000. With these configurations, the STM32 timer can generate PWM signals with 250Hz and pulse width limited between $1000us$ and $2000us$.

Bibliography

[1] "The Complete History Of Drones (1898-2021) - INFOGRAPHIC," [Online]. Available: https://dronesgator.com/the-history-of-drones/.

[2] "4 May 1924 - This Day in Aviation," [Online]. Available: https://www.thisdayinaviation.com/4-1924/.

[3] "History of Quadcopters and Multirotors — Krossblade Aerospace Systems," [Online]. Available: https://www.krossblade.com/history-of-quadcopters-and-multirotors/.

[4] "All the World's Rotorcraft - helicopters, autogyros, tilt-wing and tilt-rotor aircraft," [Online]. Available: http://www.aviastar.org/helicopters.html.

[5] G. Singhal, B. Bansod and L. Mathew, "Unmanned aerial vehicle classification, applications and challenges: A review.," *Preprint,* no. November, pp. 1-19, 2018.

[6] V. Praveen and A. Pillai, "Modeling and simulation of quadcopter using PID controller," *International Journal of Control Theory and Applications,* vol. 9, no. 15, pp. 7151-7158, 2016.

[7] N. Bao, X. Ran, Z. Wu, Y. Xue and K. Wang, "Research on attitude controller of quadcopter based on cascade PID control algorithm," in *2017 IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, 2017.

[8] P. Wang, Z. Man, Z. Cao, J. Zheng and Y. Zhao, "Dynamics modelling and linear control of quadcopter," in *2016 International Conference on Advanced Mechatronic Systems (ICAMechS)*, 2016.

[9] S. Abdelhay and A. Zakriti, "Modeling of a Quadcopter Trajectory Tracking System Using PID Controller," *Procedia Manufacturing,* vol. 32, pp. 564-571, 2019.

[10] M. Idres, O. Mustapha and M. Okasha, "Quadrotor trajectory tracking using PID cascade control," *IOP Conference Series: Materials Science and Engineering,* vol. 270, no. 1, 2017.

[11] E. Okyere, A. Bousbaine, G. T. Poyi, A. K. Joseph and J. M. Andrade, "LQR controller design for quad-rotor helicopters," *The Journal of Engineering,* vol. 2019, no. 17, pp. 4003-4007, 2019.

[12] L. Martins, C. Cardeira and P. Oliveira, "Linear quadratic regulator for trajectory tracking of a quadrotor," *IFAC-PapersOnLine,* vol. 52, no. 12, pp. 176-181, 2019.

[13] F. Ahmad, P. Kumar, A. Bhandari and P. P. Patil, "Simulation of the Quadcopter Dynamics with LQR based Control," *Materials Today: Proceedings,* vol. 24, pp. 326-332, 2020.

[14] A. L. Luna, I. C. Vega and J. M. Carranza, "Gain-Scheduling and PID Control for an Autonomous Aerial Vehicle with a Robotic Arm," in *2018 IEEE 2nd Colombian Conference on Robotics and Automation (CCRA)*, 2018.

[15] A. Ataka, H. Tnunay and R. Inovan, "Controllability and observability analysis of the gain scheduling based linearization for UAV quadrotor," in *2013 International Conference on*

Robotics, Biomimetics, Intelligent Computational Systems, 2013.

[16] I. Sadeghzadeh, M. Abdolhosseini and Y. Zhang, "Payload Drop Application of Unmanned Quadrotor Helicopter Using Gain-Scheduled PID and Model Predictive Control Techniques," in *Springer*, Berlin, Heidelberg, 2012.

[17] J. Shah, M. Okasha and W. Faris, "Gain scheduled integral linear quadratic control for quadcopter," *International Journal of Engineering and Technology(UAE),* vol. 7, no. 4, pp. 81-85, 2018.

[18] V. Alexandrov, I. Rezkov, D. Shatov and V. Chestnov, "Discrete-time H∞ Optimization for Quadcopter Altitude Control," 2021.

[19] C. MASSÉ, O. GOUGEON, D.-T. NGUYEN and D. SAUSSIÉ, "Modeling and Control of a Quadcopter Flying in a Wind Field: A Comparison Between LQR and Structured $\mathscr{H}\infty$ Control Techniques," in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2018.

[20] E. Kuantama, I. Tarca and R. Tarca, "Feedback Linearization LQR Control for Quadcopter Position Tracking," *2018 5th International Conference on Control, Decision and Information Technologies, CoDIT 2018,* pp. 204-209, 2018.

[21] H. Suresh, A. Sulficar and V. Desai, "Hovering control of a quadcopter using linear and nonlinear techniques," *International Journal of Mechatronics and Automation,* vol. 6, no. 2-3, pp. 120-129, 2018.

[22] Y.-M. Kim and W.-B. Baek, "Adaptive Sliding Mode Control based on Feedback Linearization for Quadrotor with Ground Effect," *Journal of Advanced Information Technology and Convergence,* vol. 8, no. 2, pp. 101-110, 2018.

[23] L. Martins, C. Cardeira and P. Oliveira, "Feedback Linearization with Zero Dynamics Stabilization for Quadrotor Control," *Journal of Intelligent and Robotic Systems: Theory and Applications,* vol. 101, no. 1, 2021.

[24] V. Sumathy and D. Ghose, "Robust Adaptive Feedback Linearization Controller for an Aerial Robot Working in Narrow Corridor and In-door Environments," in *2021 International Symposium of Asian Control Association on Intelligent Robotics and Industrial Automation (IRIA)*, 2021.

[25] H. Abrougui, S. Hachicha, C. Zaoui, H. Dallagi and S. Nejim, "Flight Controller Design Based on Sliding Mode Control for Quadcopter Waypoints Tracking," *Proceedings of the International Conference on Advanced Systems and Emergent Technologies, IC_ASET 2020,* pp. 13-20, 2020.

[26] V. K. Tripathi, L. Behera and N. Verma, "Design of sliding mode and backstepping controllers for a quadcopter," in *2015 39th National Systems Conference (NSC)*, 2015.

[27] K. Runcharoon and V. Srichatrapimuk, "Sliding Mode Control of quadrotor," in *2013 The International Conference on Technological Advances in Electrical, Electronics and Computer Engineering (TAEECE)*, 2013.

[28] D. Matouk, F. Abdessemed, O. Gherouat and Y. Terchi, "Second-order sliding mode for position and attitude tracking control of quadcopter UAV: Super-twisting algorithm," *International Journal of Innovative Computing, Information and Control,* vol. 16, no. 1, pp.

29-43, 2020.

[29] A. M. Elhennawy and M. K. Habib, "Trajectory tracking of a quadcopter flying vehicle using sliding mode control," in *IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society*, 2017.

[30] M. Islam, M. Okasha and M. M. Idres, "Dynamics and control of quadcopter using linear model predictive control approach," *IOP Conference Series: Materials Science and Engineering,* vol. 270, no. 1, pp. 0-9, 2017.

[31] Z. Cai, S. Zhang and X. Jing, "Model Predictive Controller for Quadcopter Trajectory Tracking Based on Feedback Linearization," in *IEEE Access*, 2021.

[32] Y. Wang, A. Ramirez-Jaime, F. Xu and V. Puig, "Nonlinear Model Predictive Control with Constraint Satisfactions for a Quadcopter," *Journal of Physics: Conference Series,* vol. 783, no. 1, p. 012025, 2017.

[33] H. Merabti, I. Bouchachi and K. Belarbi, "Nonlinear model predictive control of quadcopter," in *2015 16th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA)*, 2015.

[34] R. Roy, M. Islam, N. Sadman, M. A. P. Mahmud, K. D. Gupta and M. M. Ahsan, "A Review on Comparative Remarks, Performance Evaluation and Improvement Strategies of Quadrotor Controllers," *Technologies,* vol. 9, no. 2, p. 37, 2021.

[35] 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 and Control,* vol. 12, no. 4, pp. 519-532, 2017.

[36] M. Rabah, A. Rohan, Y. J. Han and S. H. Kim, "Design of fuzzy-PID controller for quadcopter trajectory-tracking," *International Journal of Fuzzy Logic and Intelligent Systems,* vol. 18, no. 3, pp. 204-213, 2018.

[37] S. Bari, S. S. Z. Hamdani, H. U. Khan, M. u. Rehman and H. Khan, "Artificial Neural Network Based Self-Tuned PID Controller for Flight Control of Quadcopter," in *2019 International Conference on Engineering and Emerging Technologies (ICEET)*, 2019.

[38] S. Omidshafiei, "Reinforcement Learning-based Quadcopter Control," no. 1, 2013.

[39] "Euler angles," [Online]. Available: https://en.wikipedia.org/wiki/Euler_angles.

[40] E. Bryson and Y.-C. Ho, Applied optimal control optimization, estimation, and control, Washington Hemisphere Publishing Corporation New York Wiley, 1975.

[41] E. Capello, H. Park, B. Tavora, G. Guglieri and M. Romano, "Modeling and experimental parameter identification of a multicopter via a compound pendulum test rig," in *2015 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS)*, Cancun, Mexico, 2015.

[42] S. Joo, J. Lee and J. Seo, "Minimum-Jerk Trajectory Generation for Control Applications," *Transactions of the Korean Nuclear Society Autumn Meeting,* pp. 25-27, 2018.

[43] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE International Conference on Robotics and Automation*, Shanghai, China, 2011.

[44] B. Siciliano, Robotics modelling planning and control, London: Springer, 2010.

[45] I. Inc., "MPU-6000 and MPU-6050 product Specification Revision 3 - TDK," 2013.

[46] M. J. Caruso, "Applications of magnetoresistive sensors in navigation systems," *Sensors and Actuators,* 1997.

[47] "QMC5883L Datasheet | QST - Datasheetspdf.com," [Online]. Available: https://datasheetspdf.com/datasheet/QMC5883L.html.

[48] "GNSS," [Online]. Available:https://www.unoosa.org/oosa/en/ourwork/psa/gnss/gnss.html.

[49] "Ionospheric Remote Sensing with GNSS | Encyclopedia MDPI," [Online]. Available: https://encyclopedia.pub/item/revision/873f6e50138abed8f822c94f39b9aba6.

[50] "IBM Docs," [Online]. Available: https://www.ibm.com/docs/en/informix-servers/12.10?topic=data-geographic-coordinate-system.

[51] "What are geographic coordinate systems?—ArcMap | Documentation," [Online]. Available: https://desktop.arcgis.com/en/arcmap/latest/map/projections/about-geographic-coordinate-systems.htm.

[52] A. G. &. C. KG, "MS5611 - high resolution barometric sensor 10 - 1200 mbar," [Online]. Available: https://www.amsys-sensor.com/products/pressure-sensor/ms5611-high-resolution-barometric-sensor-10-1200-mbar/.