UPPSALA UNIVERSITET

UPTEC F 24029

Examensarbete 30 hp

Juni 2024

Quadcopter Modeling and Linear Quadratic Regulator Design Using Simulink

Heja Cengiz

Civilingenjörsprogrammet i teknisk fysik

Quadcopter Modeling and Linear Quadratic Regulator Design Using Simulink

Heja Cengiz

Abstract

This thesis project aims to model a quadcopter and design a linear quadratic regulator (LQR) by means of Matlab/Simulink. To this purpose, the LQR-based optimal control theory for controlling a quadcopter is first studied which includes state-space representation (SSR) of a dynamic process or system, cost function, LQR, quadcopter flight dynamics and system linearization. A quadcopter model is developed in Matlab/Simulink, followed by the implementation of a LQR-based control system. The LQR parameters are tuned and the system is tested under various flight conditions (wind disturbance, in the simulation, specific/simplified model, etc.).

The simulation results show that the LQR is an effective controller for maintaining stable hover at a height straight up and compensating for wind disturbances. However, when the quadcopter moves to a new position, oscillations occur, highlighting the limitations of the LQR due to its reliance on a simplified and linearized model. Additionally, modifications to the model parameters, such as mass and inertia, impact the system performance, indicating potential robustness issues with the controller. It can be concluded that Matlab/Simulink is an effective tool for quadcopter modeling, LQR designing and LQR performance analyzing.

In this thesis project, only the LQR method is used for controlling a quadcopter and the LQR tuning process is not efficient. In future work other techniques such as regional linearization and alternative non-linear controllers, like model predictive control (MPC) or sliding mode control (SMC), can be explored. Development of optimization algorithms for LQR tuning in the LQR method is highly recommended.

Teknisk-naturvetenskapliga fakulteten

Uppsala universitet, Utgivningsort Uppsala/Visby

Handledare: Tomas Melin Ämnesgranskare: Ping Wu

Examinator: Tomas Nyberg

Populärvetenskaplig sammanfattning

Kvadkoptern är en mångsidig obemannad luftfarkost (drönare) med ett brett användningsområde, inklusive fotografering, navigering, säkerhet och jordbruk. Eftersom kvadkoptern har många tillämpningsområden har den blivit populär bland både hobbyister och professionella användare. För att kvadkoptern ska kunna utföra sina uppgifter med hög noggrannhet och hantera olika flygscenarier samt miljöförhållanden behöver den ett effektivt styrsystem. Denna avhandling utforskar modelleringen av en kvadkopter och tillämpningen av styrsystemet LQR (linear quadratic regulator). Arbetet har utförts i Matlab/Simulink, vilka är kraftfulla verktyg för modellering och simulering av dynamiska system.

En kvadkopter består av fyra rotorer monterade på ett plusformat ramverk. Genom att variera varje rotors hastighet kan man kontrollera kvadkopterns rörelse i sex dimensioner: längdriktningen (framåt och bakåt), vertikalt (uppåt och nedåt), sidled (höger och vänster) och rotation kring varje axel (roll-, stig- och kursvinklar). Dynamiken hos en kvadkopter är icke-linjär och den är dessutom känslig för yttre störningar, vilket gör styrningen utmanande. För att modellera kvadkopterns dynamik med hjälp av systemekvationer har bland annat Eulers ekvation och Newtons lagar använts.

Metoden LQR används för att designa ett styrsystem som löser ett optimeringsproblem genom att minimera en kostnadsfunktion som är en kombination av avvikelser från önskade tillstånd och kontrollansträngningar. Det innebär att målet är att minimera avvikelser från en önskad position och orientering samtidigt som energin som används av rotorerna begränsas. Eftersom modellen av en kvadkopter inte är linjär behöver den först linjäriseras kring en stationär (även kallad jämvikts-) punkt innan LQR kan implementeras. Linjäriseringen sker med hjälp av bland annat Taylorserieutveckling.

När en färdig modell för kvadkoptern har skapats i Simulink och styrsystemet har implementerats så går det att genomföra simuleringar för att utvärdera systemets prestanda. Resultaten visar att justering av LQR är en komplex och iterativ process. Vid den linjäriserade hovringspunkten presterar LQR-regulatorn väl och kan anpassa sig till olika typer av störningar som representerar vindbyar, vilket gör att kvadkoptern kan hovra stabilt på en given höjd rakt upp. När den försöker att ändra position uppstår istället oscillationer. Detta beror till stor del på att LQR är baserad på en förenklad och linjäriserad modell, samt justeringen av viktmatriserna för LQR. Utmaningar pekar på behovet av framtida förbättringar vilket kan göras genom att linjärisera ett område istället för en punkt, använda icke-linjära metoder (model predictive control (MPC) eller sliding mode control (SMC)) som styrsystem och använda optimeringsteknik vid justering av linjära metoder.

Sammanfattningsvis har detta projekt demonstrerat att LQR kan stabilisera en kvadkopter i hovring och hantera mindre störningar effektivt. De identifierade begränsningarna vid större avvikelser öppnar upp för vidare forskning och utveckling av mer avancerade styrsystem. Med fortsatta förbättringar kan kvadkoptrar bli stabilare, vilket breddar deras användningsområde ytterligare och ökar deras pålitlighet i praktiska tillämpningar.

Contents

List of abbreviations

CCW	Counter clockwise
CW	Clockwise
DOF	Degree(s) of freedom
LQR	Linear quadratic regulator
ODE	Ordinary differential equations
PC	Programmable controller
PCB	Printed circuit board
PID	Proportional-integral-derivative
PLC	Programmable logic controller
RPM	Revolutions per minute
SSR	State space representation
UAV	Unmanned aerial vehicle
VTOL	Vertical take-off and landing

1 Introduction

In step with humans' quest for efficiency and automation, control systems have played a pivotal role. From the simple mechanical contraptions of ancient civilizations to today's advanced computer-controlled devices, the evolution of control systems has revolutionized the way various systems are controlled and regulated. Understanding the historical development, principles, and applications of control systems is essential for engineers and researchers working in diverse fields. The devices have become smaller, more flexible and more powerful which means that they are easier to integrate while being able to carry out complex tasks. However, the small size of devices and their accompanying PCBs makes them increasingly susceptible to external influences. This vulnerability is evident in miniature drones, where their reduced size makes them more susceptible to environmental factors such as wind disturbances. To weaken the disturbances, control technology is used to develop control systems that effectively dampen various form of disturbances.

1.1 Background

To understand the research conducted in this thesis, it is beneficial to first explore the background of its main subjects: quadcopter and control system. This section offers an introductory overview and highlights some historical advancements that underpin these technologies.

1.1.1 Quadcopter

The quadcopter is a type of unmanned aerial vehicle (UAV) characterized by its four vertically oriented propellers. It is a versatile drone with a wide range of uses, including photography, agriculture, security and navigation. Its design, inspired by the principles of helicopter flight, enables vertical take-off and landing (VTOL) capabilities along with agile maneuvering in diverse environments. The history of the quadcopter traces back to its early conceptualization in the mid-20th century, with advancements in both civilian and military applications occurring in the late 20th and early 21st centuries.

Despite the progress in quadcopter technology, there are still some challenges, especially when it comes to making them more stable, easier to maneuver, optimizing their performance and improving their resilience to external disturbances. The motivation for research in quadcopter modeling and control design stems from the necessity to improve their reliability and safety. Addressing these challenges means that one needs to have an understanding of the complex dynamics involved in quadcopter flight and the development of robust control algorithms to stabilize and control their motion.

1.1.2 Control system

Background of control system
Biological control systems took place in early inhabitants of Earth and human-designed control system is a concept with a rich history dating back to ancient civilization. There are both simple and complex control systems and they share a common goal, which is to simplify/automate a process. A control system can be regulated by fluid pressure, mechanical means, electricity, etc. An early type of control system was when Roman engineers built a system with floating valves to maintain appropriate water levels. This is a type of feedback control, where the resulting information from the process is used to correct an operation. There is also feedforward where no information is used to correct an operation.[1]

There have been various types of control systems throughout history. Feedback systems were used by Greeks at around 300 BC. Ktesibios invented a water clock by letting water trickle into a container at a constant rate, then the level of the measuring container was used to tell the time. A float valve was used to keep the supply tank at a constant level. An oil lamp could also use the same idea of control by liquid. Around 1681 the invention of the safety valve took place, which was used to regulate steam pressure. By using a weight on the valve one could set the internal pressure of the boiler, since the valve would only open if the boiler exceeded the weight. The 20th century witnessed significant advancements in control systems. In the early 1900s the automatic steering of ships was achieved and in 1922 an automatic steering was installed, which improved the performance by using elements of compensation and adaptive control. The origin of the PID-controller comes from the theoretical development applied to the automatic steering of ships done by Nicholas Minorsky. A lot of the control theory that is still used till this day has been greatly contributed by Minorsky. Another important development was done by Bode and Nyquist at Bell Telephone Laboratories, where they made the analysis of feedback amplifiers.[2]

Modern control system
The control systems used today have a widespread application, such as navigation, spacecraft, industrial robots, heating systems and so on. Modern control systems have a computer involved in the circuit and it operates the systems electrically. The exponential development that has taken place in electronics has resulted in smaller circuits, which means that more advanced tasks can be accommodated in even smaller crafts. Today's control systems use complex mathematical models and advanced algorithms to be able to complete challenging tasks and contribute to better accuracy.

There are different names for control systems, the more modern ones are called programmable logic controller (PLC) or programmable controller (PC). They are defined as control systems with a programmable memory that can store instructions which control and regulate the process. The control system is built by using logic circuits and microprocessors. The device is often made to deal with noise from electric and magnetic fields, but also vibrations and high temperatures. The central unit in the control system gets information via the inputs and then the right instruction from the memory is chosen. These are used to calculate the state of the outputs and when the program is completed, it starts over and once again reads the inputs. Often the control system is modular so that it can be adapted to different types of projects.[3]

Flight control system
All aircrafts have some type of flight control system and it usually uses automation in some form. A unmanned aerial vehicle (UAV) is an unpiloted machine and this concept has been around for a long time. One of the first contributions to unpiloted mechanisms was created in 425 BC and it was a mechanical bird which was able to fly by using the mechanism inside of it, see Figure 1.1. One of the first use of a UAV as a combat air vehicle was in 1849 when Austrians attacked Italy (Venice) by using 200 balloons loaded with explosive devices that detonated using timers. The possibility of using radio to control aircrafts was intriguing. In 1916 target planes were remotely controlled by automatic Hewitt-Sperry, a system integrated with a gyroscope.[4] Modern UAV control system consists of a sophisticated network of sensors, actuators, and algorithms designed to stabilize, navigate, and control the aircraft autonomously or remotely.

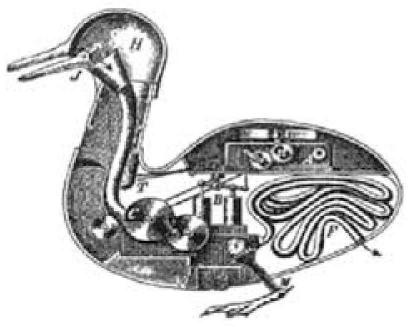


Figure 1.1: Archytas din Tarantas, the first UAV according to Prisacariu[4].

1.2 Purpose and goals

The project's objective is to formulate a model for a quadcopter that accurately represents its physical dynamics, followed by the integration of a linear quadratic regulator (LQR) as the controller in the automatic control system for the quadcopter. The software used is Simulink/Matlab. To incorporate the LQR effectively, the system needs initial simplification and linearization. Subsequently, analyzing and tuning of the LQR parameters are crucial steps to enhance its performance and optimize the quadcopter's overall functionality.

1.3 Tasks and scope

The project can be divided into tasks, but the sequence of tasks does not have to match the list below. Throughout the process of the project it is common to revisit certain tasks and make adjustments as necessary. This happens often, for instance when debugging. The tasks involved in this project are:

- Formulating a model of the quadcopter (get the system equations) by using Euler's equation, Newton's laws, dynamics of the quadcopter, etc.
- Study state space representation (SSR) and linear quadratic regulator (LQR).
- Simplify and linearize the system model, essential for implementation of LQR.
- Create a *Quadcopter* block in Simulink/Matlab. Verify the block with tests.
- Create a *Controller* block in Simulink/Matlab. Verify the block with tests.
- Tune LQR to achieve good performance.
- Test hovering under various conditions.
- Modify model parameters.
- Analyze and discuss the results.
- Determine conclusions and further work to improve the project.

The project has several boundaries, limitations and specific conditions. The modeling of the quadcopter is done using specific physical laws and simplifications. It does not delve into other modeling techniques. The controller is a LQR and no other control strategies are used. All simulations are conducted within Simulink/Matlab, without using other software tools or platforms. This also means that there has not been any testing done outdoors on a real quadcopter. The project is conducted within a specific time frame, which limits the extent of testing and refinement that can be done.

2 Theory

The project covers several different theoretical frameworks, each playing an important role in comprehending the remaining sections and getting a good foundation. Control systems form the cornerstone of modern engineering, enabling precise regulation and manipulation of dynamic processes across a multitude of domains. Control engineering is a specialized discipline within engineering which encompasses the theory and application of control systems to achieve desired system behavior. To create a model and implement a controller in a control system for a quadcopter, one needs to have an understanding of the dynamics.

2.1 Optimal control

Control theory addresses the behavior of a system, focusing on controlling and regulating it to achieve a desired outcome. Systems encompass various domains such as technical, economical and biological, among others. Each system has its unique desired behavior, for instance, ensuring stability and safety for a car or promoting robust economic growth with minimal unemployment. Control theory offers methods applicable to mechanisms that generate throttle or control signals to attain the desired system behavior. The specific mechanism varies depending on the system; for instance, in cars, it may involve electronic systems, while in economics, it could relate to policy rates set by central banks.[5]

The fundamental concept revolves around influencing a system, also referred to as a process, to exhibit specific behavior. Information about the system is gathered through measured output signals. Control signals are applied to affect the system, while external noise or interference signals, which are beyond our control, may also impact the system. The desired behavior aims to minimize the difference between the output signal and a reference signal, a problem called the servo problem. Alternatively, maintaining a constant output signal is known as the regulator problem. Two crucial concepts are dynamic and stability. A dynamic system is such that its output depends on its current input as well as the past inputs. Stability concerns the behavior of the output signal; if the output signal grows indefinitely despite limited input signals, the system is deemed unstable.[5] Two other important concepts are observability and controllability. Observability involves using the system's output to deduce its state variables. If a state variable has no effect on the output, it cannot be evaluated through observation alone. A system is considered observable if its initial state vector can be determined from measured input and output over a finite time interval, otherwise, it is deemed unobservable. Controllability relates to the system's ability to control its state variables through the input signal. If any state variable cannot be influenced by the input, the system is uncontrollable.[2]

Optimal control theory is a branch of control theory in which one defines a specific cost function to be minimized. When a system dynamics are described with a set of linear differential equations and the cost function is represented by a quadratic function, the problem with minimizing such a cost function is referred to as the linear quadratic (LQ) control problem. A process or system can be described with a block diagram, as shown in Figure 2.1 where u is the input signal and y is the output signal.

Figure 2.1: A process with an input u and an output y.

There are different ways to describe a system, e.g., using a linear differential equation and a transfer function in the case of a linear system. Two important concepts in association with the LQ control problem are state space representation (SSR) and linear quadratic regulator (LQR) which are addressed in detail in the following sections.

2.1.1 State space representation (SSR)

In 1960 Moscow was the place for the First Congress of the International Federation of automatic Control. It was there that scientists and engineers learned about approaching control theory with time domain methods, rather than frequency domain ones. The dynamics of the system was described with the help of differential equations, instead of transfer functions. Lyapunov theory was used for stability instead of methods of Bode and Nyquist. The new path was the state space approach, which has had an enormous impact on control theory. Both the frequency domain methods and the state space methods are useful and complement each other.[6]

SSR is a representation that describes a system expressed as a function of input, output and state variables. As mentioned before a dynamic system's output does not only depend on the input at that moment, it also depends on all past inputs. This makes it hard to determine the effect of the input. The state of the system at time t is the information (at time t) that can be used to predict the effect of the applied input signal $u(\tau)$, where τ is greater than t. Since knowing the states of the system means that one knows how the input will affect the output, it is self-evident to use the states to determine the input. If a system is described by differential equations then one can determine the states by rewriting the equations. To be able to solve differential equations one needs the initial values and it is easier to treat high-order linear differential equations by rewriting them as a system of first-order equations. This results in the state being written as a vector, called state vector, and the components of the vector are called state variables. An example of a system given in differential equation is[5]

$$\dddot{y}(t) + a_1\ddot{y}(t) + a_2\dot{y}(t) + a_3y(t) = bu(t)\,. \tag{2.1}$$

Introducing

$$x_1(t) = y(t) \quad , \quad x_2(t) = \dot{y}(t) \quad , \quad x_3(t) = \ddot{y}(t) \tag{2.2}$$

and differentiating these x-variables gives

$$\dot{x}_1(t) = \dot{y}(t) = x_2(t)\,, \tag{2.3}$$

$$\dot{x}_2(t) = \ddot{y}(t) = x_3(t)\,,$$

$$\dot{x}_3(t) = \dddot{y}(t) = -a_1\ddot{y}(t) - a_2\dot{y}(t) - a_3y(t) + bu(t) =$$

$$= -a_1x_3(t) - a_2x_2(t) - a_3x_1 + bu(t)\,.$$

Denoting $\dot{x}(t) = [\dot{x}_1(t)\ \dot{x}_2(t)\ \dot{x}_3(t)]^T$ and $x(t) = [x_1(t)\ x_2(t)\ x_3(t)]^T$ means that the equations can be written in matrix form

$$\dot{x}(t) = \begin{bmatrix}\dot{x}_1(t)\\ \dot{x}_2(t)\\ \dot{x}_3(t)\end{bmatrix} = \begin{bmatrix}x_2(t)\\ x_3(t)\\ -a_1x_3(t) - a_2x_2(t) - a_3x_1 + bu(t)\end{bmatrix} = \begin{bmatrix}0 & 1 & 0\\ 0 & 0 & 1\\ -a_3 & -a_2 & -a_1\end{bmatrix}\begin{bmatrix}x_1(t)\\ x_2(t)\\ x_3(t)\end{bmatrix} + \begin{bmatrix}0\\ 0\\ b\end{bmatrix}u(t)\,,$$

$$y(t) = \begin{bmatrix}1 & 0 & 0\end{bmatrix}\begin{bmatrix}x_1(t)\\ x_2(t)\\ x_3(t)\end{bmatrix} + [0]u(t)\,. \tag{2.4}$$

Further, denoting matrices A, B, C and D as

$$A = \begin{bmatrix}0 & 1 & 0\\ 0 & 0 & 1\\ -a_3 & -a_2 & -a_1\end{bmatrix} \quad , \quad B = \begin{bmatrix}0\\ 0\\ b\end{bmatrix} \quad , \quad C = \begin{bmatrix}1 & 0 & 0\end{bmatrix} \quad , \quad D = [0]\,. \tag{2.5}$$

Equation (2.4) can be written as

$$\begin{cases}\dot{x}(t) = Ax(t) + Bu(t)\,,\\ y(t) = Cx(t) + Du(t)\,,\end{cases} \tag{2.6}$$

where $x(t)$, $y(t)$ and $u(t)$ are vectors, respectively and A, B, C and D are matrices, respectively. This represents the state of the system as a state vector, as mentioned earlier. This is for a dynamical system that is linear, time-invariant and finite-dimensional. One can use different methods to describe the system; another way is to use the transfer function. SSR is a compact way of describing the behavior of the system and another advantage is that one can use it to treat a multivariable (several inputs and outputs) system.

2.1.2 Linear quadratic regulator (LQR)

The LQR is a type of optimal control based on SSR. It employs full-state feedback, since it uses a feedback loop of the state vector and multiplies it with a gain matrix K. This enables the placement of closed-loop poles anywhere, under the assumption of system controllability and observability. Subsequently, the difference between the reference term and the feedback is computed. Figure 2.2 shows a block system of LQR and Figure 2.3 is similar but more detailed.

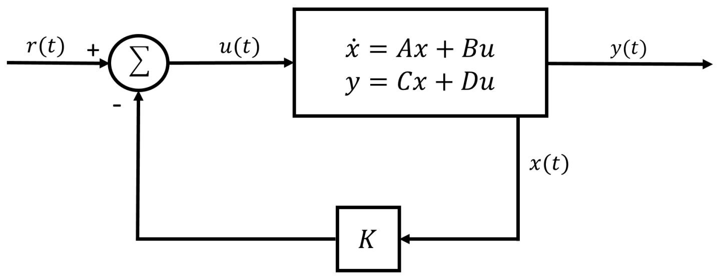


Figure 2.2: A LQR controller with the SSR in one block.

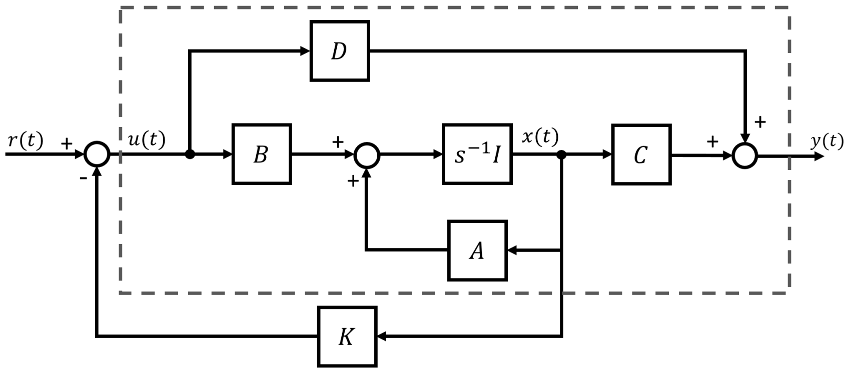


Figure 2.3: A LQR controller in a pure block diagram form.

The optimal gain matrix K is found by selecting important closed-loop characteristics, which are performance and effort. The weighing assigned to system performance and effort varies depending on the specific system requirements and user preferences. LQR finds the optimal gain matrix K by setting up a cost function in the following manner,

$$J = \frac{1}{2}\int_0^\infty \left(x^T Q x + u^T R u\right) dt = \tag{2.7}$$

$$= \frac{1}{2}\int_0^\infty \left(\begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix} \begin{bmatrix} Q_1 & & & \\ & Q_2 & & \\ & & \ddots & \\ & & & Q_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} u_1 & u_2 & \cdots & u_n \end{bmatrix} \begin{bmatrix} R_1 & & & \\ & R_2 & & \\ & & \ddots & \\ & & & R_n \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} \right) dt\,.$$

which is a sum of weighted performance and weighted effort over time. Q and R are weighing matrices, where adjustments to Q penalizes bad performance, while those to R penalize actuator effort. In this cost function, the evaluation of performance (which Q penalizes) is based on the state vector (x) and effort (penalized by R) is assessed according to the input (u). Q is a positive semi-definite (symmetric with non-negative eigenvalues) square matrix with the same number of rows as states. Often, Q is configured as a diagonal matrix, individually targeting each state. Similarly, R is positive definite (symmetric with positive eigenvalues) and acts on the inputs. The integral can represent the area under the curve and less area means that it spends more time closer to the curve. To avoid subtraction caused by negative eigenvalues, squaring the value is a feasible solution. A drawback is that larger errors are punished disproportionately more than smaller ones. However, the cost function becomes quadratic ($a^2 + b^2$) and a quadratic function is convex and has a definite minimum. When subjected to linear dynamics, quadratic functions remain quadratic, ensuring a definite minimum value. Expanding $x^T Qx + u^T Ru$ into a larger matrix form reveals that R and Q are part of a larger weighting matrix and instead of zeros in the off-diagonal matrices N and N^T are introduced. They penalize the cross product of x and u.

$$\begin{bmatrix} x^T & u^T \end{bmatrix} \begin{bmatrix} Q & N \\ N^T & R \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} . \tag{2.8}$$

The feedback control that minimizes the quadratic cost function is $u = -Kx$ and solving the optimization problem produces the optimal gain matrix K. Equation (2.9) shows the equation for K and Equation (2.10) shows the Riccati equation that needs to be solved in order to get matrix P which in turn is used to get optimal gain matrix K. When designing a control system one does not need to solve the problem by hand, instead a linear model is developed, weighing matrices are adjusted and with the help of a software (Matlab) the optimization problem is solved.[7] [8]

$$K = R^{-1}(B^T P + N^T) . \tag{2.9}$$

$$A^T P + PA - (PB + N)R^{-1}(B^T P + N^T) + Q = 0 . \tag{2.10}$$

The cost function J and the optimal gain matrix K are intrinsically related through the solution of the Riccati equation, denoted as matrix P. It has been shown above how K and P are related, with K being derived using matrix P. Additionally, the optimal cost J can be expressed in terms of the initial state x_0 and P as $J = \frac{1}{2}x_0^T P x_0$.

2.2 Quadcopter

The UAV used in this project is a quadcopter and to be able to create a model and implement LQR one needs to have an understanding of the dynamics, derive its system equations and linearize the system (essential for LQR).

2.2.1 Dynamics

A quadcopter (also called quadrotor) is a multirotor UAV equipped with four rotors, typically arranged in a plus- or cross-configuration. In this project the plus-configuration is used which refers to it nominal flight direction be such that the axes of roll and pitch angles coincides with the principal axes of the fuselage for engine pairs with the same direction of rotation. As shown in Figure 2.4.This configuration is easier to model and should thereby also be more intuitive for the reader. One diagonal pair of rotors rotate clockwise (CW) while the other pair of rotors rotate counter clockwise (CCW). This configuration is done to counteract the net torque (when maintaining a constant yaw movement), as the forces from the two pairs will cancel each other out. The quadcopter has nonlinear dynamics with three translational (movement of an object from one point to another) and three rotational (movement in which an object rotates around a fixed point) movements, resulting in a total of six degrees of freedom (6DOF).

Steering the quadcopter involves adjusting the revolutions per minute (RPM) of the four rotors, thereby altering thrust and torque forces, which in turn induces rotation and movement in different directions. Having 6DOF and four control inputs makes the system of the quadcopter underactuated. Figure 2.4 below shows, among other things, the thrust (F_i) and torque (M_i) forces for each rotor.

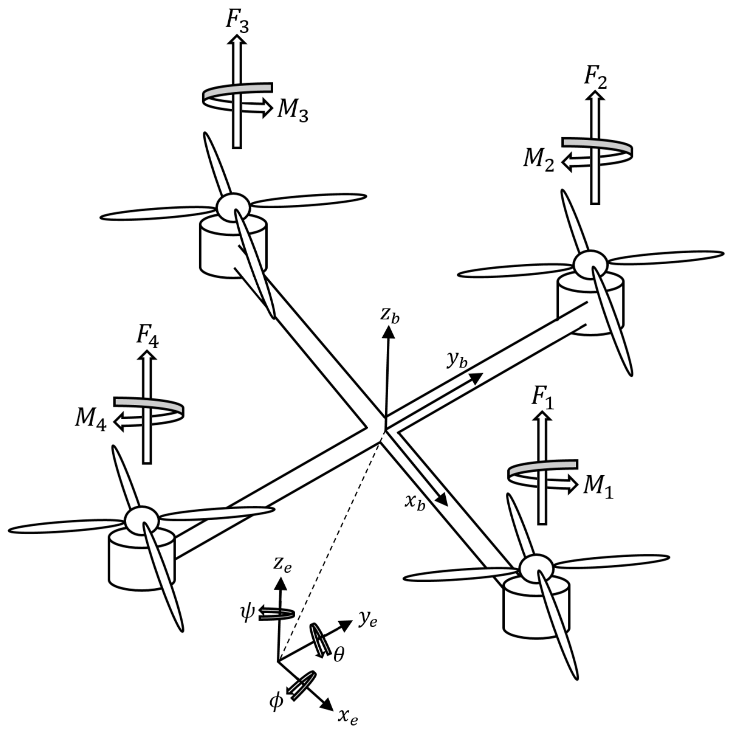


Figure 2.4: Axes and forces of a quadcopter.

In Earth frame the rotation about the z-axis is called yaw (denoted by the symbol ψ), the one about the x-axis is called roll (denoted by ϕ) and the rotation about the y-axis is called pitch (denoted by θ). These rotations are measured as angles around the center of the quadcopter. To adjust pitch and roll the net center of thrust is modified and to control yaw one varies the net torque. This can be described as:

- Move the quadcopter up (increase throttle): Increase F_1 , F_2 , F_3 , F_4
- Make the quadcopter roll: Keep F_1 , F_3 the same and increase (or decrease) F_2 and decrease (or increase) F_4
- Make the quadcopter pitch: Keep F_2 , F_4 the same and increase (or decrease) F_1 and decrease (or increase) F_3
- Rotate the quadcopter (make it yaw): Increase (or decrease) M_1 , M_3 and decrease (or increase) M_2 , M_4

A rotating propeller generates two main forces, as mentioned above, these are thrust and torque. The equations are

$$|F_i| = k_F \omega_i^2 \,, \tag{2.11}$$

$$|M_i| = k_M \omega_i^2 \,, \tag{2.12}$$

where ω is the rotation rate and k_F is a constant influenced by surrounding factors, such as back-EMF and the density of air. The propellers are designed to create a pressure differential when they spin, which results in thrust. Similarly, torque (also called moment) depends on the rotation rate and a constant k_M , typically found empirically. Torque acts in opposition to the rotation of the propeller, which means that a propeller spinning CW generates a CCW torque on the body of the quadcopter. This is in accordance with Newton's third law, which states that every action has an equal and opposite reaction. The dynamics of the rate of change for the rotation rate (ω) depends on several factors, but the dynamics of the powertrain is usually not taken into account in the design of control system.

When constructing a model based on the dynamics of an object (such as a quadcopter), one can consider numerous different forces that affect the object. However, to construct a manageable model, certain factors must be disregarded, assumed, or simplified. Essentially, there are four forces that affect a quadcopter: gravity, air resistance, thrust and torque. Since thrust and torque can be controlled by adjusting the speed of the rotors, they serve as the control signals for the system. The quadcopter can be described by looking at altitude, roll, pitch and yaw. The four control signals are defined as

$$\begin{cases} U_1 = F_1 + F_2 + F_3 + F_4 = k_F \left(\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2\right) & \text{(Altitude)}, \\ U_2 = L\left(F_2 - F_4\right) = Lk_F \left(\omega_2^2 - \omega_4^2\right) & \text{(Roll)}, \\ U_3 = L\left(F_3 - F_1\right) = Lk_F \left(\omega_3^2 - \omega_1^2\right) & \text{(Pitch)}, \\ U_4 = M_1 - M_2 + M_3 - M_4 = k_M \left(\omega_1^2 - \omega_2^2 + \omega_3^2 - \omega_4^2\right) & \text{(Yaw)}. \end{cases} \tag{2.13}$$

Assuming that the quadcopter is a perfectly stiff object in space allows one to use Euler's equation for the motion of a rigid body. With the help of Euler's equation and Newton's law of motion a more detailed description of the quadcopter's behavior can be enabled. Newton's second law (describes the change a force can produce on the motion of a body, Equation (2.14)) and Euler's equation (first order ODE describing the rotation of a rigid body, Equation (2.15)) are expressed as follows

$$F = ma\,, \tag{2.14}$$

$$M = I\dot{\omega} + \omega \times (I\omega)\,, \tag{2.15}$$

where F is force, m is mass, a is acceleration, M is torque, I is inertia matrix (not to be confused with identity matrix), ω is the rotation rate and $\dot{\omega}$ is the angular acceleration.

Coordinate systems are essential for accurately describing the quadcopter and understanding the position, orientation and movement. By using two frames, Earth and body, it is easier to control, navigate, model and simulate the quadcopter. With a consistent reference frame it would be challenging to communicate precise information about the quadcopter's location and orientation relative to other objects or reference points. A quadcopter relies on devices, such as accelerometer, GPS and gyroscope, to function and the data needs to be accurately interpreted. In aerospace applications, Tait-Bryan angles are frequently used to describe orientation. They describe the orientation of the aircraft relative to the Earth frame. While Euler angles have that the vertical orientation represents zero degrees, the Tait-Bryan angles have that zero degrees is represented by the horizontal position. Tait-Bryan converts the mathematical description from body frame to Earth frame. Various rotation sequences exist; the one used here is called ZYX, which means that the rotations occur successively around the z-axis, y-axis, and finally the x-axis. The rotation sequence is then applied to a point, which is done by first applying the rotation of the x-axis, followed by the y-axis and lastly the z-axis. The equations for the rotation matrices are

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c(\phi) & s(\phi) \\ 0 & -s(\phi) & c(\phi) \end{bmatrix}, \tag{2.16}$$

$$R_y(\theta) = \begin{bmatrix} c(\theta) & 0 & -s(\theta) \\ 0 & 1 & 0 \\ s(\theta) & 0 & c(\theta) \end{bmatrix}, \tag{2.17}$$

$$R_z(\psi) = \begin{bmatrix} c(\psi) & s(\psi) & 0 \\ -s(\psi) & c(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.18}$$

$$R_b^e = R_x(\phi)R_y(\theta)R_z(\psi) = \begin{bmatrix} c(\psi)c(\theta) & c(\psi)s(\theta)s(\phi) - c(\phi)s(\psi) & c(\psi)c(\phi)s(\theta) + s(\psi)s(\phi) \\ c(\theta)s(\psi) & s(\psi)s(\theta)s(\phi) + c(\psi)c(\phi) & c(\phi)s(\psi)s(\theta) - c(\psi)s(\phi) \\ -s(\theta) & c(\theta)s(\phi) & c(\theta)c(\phi) \end{bmatrix}, \tag{2.19}$$

where R_b^e is the rotation matrix, R_x, R_y and R_z are the elementary rotation matrices, c is cos and s is sin. The rotation matrix (also called transformation matrix) takes a vector (v_e) with coordinates expressed in the inertial frame and converts it to a vector (v_b) with coordinates expressed in the body frame, that is $v_b = R_b^e v_e$.

If $[u \ \ v \ \ w]^T$ is representing the linear velocity vector in the body reference frame, then the transformation matrix R_b^e can be used to obtain the linear velocity in the Earth reference frame

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = R_b^e \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} c(\psi)c(\theta) & c(\psi)s(\theta)s(\phi) - c(\phi)s(\psi) & c(\psi)c(\phi)s(\theta) + s(\psi)s(\phi) \\ c(\theta)s(\psi) & s(\psi)s(\theta)s(\phi) + c(\psi)c(\phi) & c(\phi)s(\psi)s(\theta) - c(\psi)s(\phi) \\ -s(\theta) & c(\theta)s(\phi) & c(\theta)c(\phi) \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix}. \tag{2.20}$$

Which leads to

$$\begin{cases} \dot{x} = u(c(\psi)c(\theta)) + v(c(\psi)s(\theta)s(\phi) - c(\phi)s(\psi)) + w(c(\psi)c(\phi)s(\theta) + s(\psi)s(\phi)), \\ \dot{y} = u(c(\theta)s(\psi)) + v(s(\psi)s(\theta)s(\phi) + c(\psi)c(\phi)) + w(c(\phi)s(\psi)s(\theta) - c(\psi)s(\phi)), \\ \dot{z} = u(-s(\theta)) + v(c(\theta)s(\phi)) + w(c(\theta)c(\phi)). \end{cases} \tag{2.21}$$

To derive the expression for linear velocity in the body frame, Newton's law can be applied to the quadcopter in this frame. Given that $[u \ \ v \ \ w]^T$ is the linear velocity, $[\dot{u} \ \ \dot{v} \ \ \dot{w}]^T$ denotes linear acceleration in the body frame. As mentioned before, Newton's second law is $F = ma$ and for the body frame this can be expressed as $f_b = m \cdot a_b$. The acceleration can be defined as the sum of the time derivative of the linear velocity ($\dot{v}_b$) and the effect of the angular momentum ($\omega_b \times v_b$). This gives

$$f_b = m(\omega_b \times v_b + \dot{v}_b) \,. \tag{2.22}$$

Rewriting this yields

$$\dot{v}_b = \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \frac{f_b}{m} - \omega_b \times v_b = \frac{f_b}{m} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \frac{f_b}{m} - \begin{bmatrix} qw - rv \\ ru - pw \\ pv - qu \end{bmatrix} \,. \tag{2.23}$$

The total force can be divided into components that describe each force along the three different axes, denoted as $f_b = [f_{bx} \;\; f_{by} \;\; f_{bz}]$. This gives the following expressions

$$\begin{cases} \dot{u} = \dfrac{f_{bx}}{m} - qw + rv \,, \\ \dot{v} = \dfrac{f_{by}}{m} - ru + pw \,, \\ \dot{w} = \dfrac{f_{bz}}{m} - pv + qu \,. \end{cases} \tag{2.24}$$

The forces acting on the quadcopter in different directions can be deconstructed into the gravitational force components acting along the axes. In the z-direction the thrust force is also taken into account. The gravitational force components in the x-, y- and z-directions are influenced by the angles ϕ and θ, as ψ does not alter the orientation of the quadcopter. The gravitational force component in the x-direction is expressed as $mg(s(\theta))$, in the y-direction it becomes $-mg(c(\theta)s(\phi))$ and in the z-direction it is equal to $-mg(c(\theta)c(\phi)) + U_1$. That is

$$f_{bx} = mg(s(\theta)) \quad , \quad f_{by} = -mg(c(\theta)s(\phi)) \quad , \quad f_{bz} = -mg(c(\theta)c(\phi)) + U_1 \,. \tag{2.25}$$

The expressions for the linear acceleration in the body frame are then

$$\begin{cases} \dot{u} = g(s(\theta)) - qw + rv \,, \\ \dot{v} = -g(c(\theta)s(\phi)) - ru + pw \,, \\ \dot{w} = -g(c(\theta)c(\phi)) + \dfrac{U_1}{m} - pv + qu \,. \end{cases} \tag{2.26}$$

Denoting the angular velocity vector as $\dot{\omega} = [\dot{p} \;\; \dot{q} \;\; \dot{r}]^T$ in the body reference frame and using Euler's equation (2.15) gives

$$I \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} L(F_2 - F_4) \\ L(F_3 - F_1) \\ M_1 - M_2 + M_3 - M_4 \end{bmatrix} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times I \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{2.27}$$

$$(\, compare \;\; with \;\; I\dot{\omega} = M_{TorqueMatrix} - \omega \times (I\omega) \,)\,,$$

where I represents the inertia matrix of the quadcopter and L is the distance from the center of mass of the quadcopter to the center of mass of the rotor. This distance remains consistent for all levers/arms since the quadcopter is symmetric. The inertia matrix is a diagonal matrix consisting of the elements I_x, I_y and I_z. The torque matrix ($M_{TorqueMatrix}$) in Equation (2.27) is derived from the fact that the torque around the z-axis is the sum of all torques generated by the individual rotors. For the x-axis, which is defined along the arms of the quadcopter where rotors 1 and 3 are located (see Figure 2.4), it results in the thrust forces from these rotors not contributing to rotation around the x-axis. It is instead rotors 2 and 4 that determine rotation around the x-axis, which is why the torque matrix has element $L(F_2 - F_4)$ in the x-direction. The signs in the element are determined solely by the direction in which torque for the x-axis is defined as positive, which means that it could just as easily be $L(F_4 - F_2)$, which is torque in the opposite x-direction. The same principle applies to the y-axis, but where rotors 1 and 3 contribute to torque instead.

The relationship between angular velocity (denoted $[p \;\; q \;\; r]^T$) in the body reference frame and the rate of the angles ϕ (roll), θ (pitch) and ψ (yaw), is given by

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & s(\phi)t(\theta) & c(\phi)t(\theta) \\ 0 & c(\phi) & -s(\phi) \\ 0 & s(\phi)sec(\theta) & c(\phi)sec(\theta) \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}\,. \tag{2.28}$$

This arises from using intermediate steps, where the initial step considers a rotation between different reference frames before proceeding to the next rotation. In other words, the outcome is derived from the following equation

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = R_x(\phi)R_y(\theta) \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} + R_x(\phi) \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix}\,. \tag{2.29}$$

2.2.2 System equations

The previous subsection provided a description and derivation of the equations belonging to the dynamics of a quadcopter. With this the expressions utilized for the system can be formulated. If one needs the expressions for $[\ddot{x}\; \ddot{y}\; \ddot{z}]$ in the system equations, see section Appendix A.1.

To compute the time derivative (rate) of the roll, pitch, and yaw angles, one can simplify Equation (2.28), which yields

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & s(\phi)t(\theta) & c(\phi)t(\theta) \\ 0 & c(\phi) & -s(\phi) \\ 0 & s(\phi)sec(\theta) & c(\phi)sec(\theta) \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} p + qs(\phi)t(\theta) + rc(\phi)t(\theta) \\ qc(\phi) - rs(\phi) \\ qs(\phi)sec(\theta) + rc(\phi)sec(\theta) \end{bmatrix}\,. \tag{2.30}$$

To get the expression for the rate of angular velocity, one utilizes Equation (2.27) and multiplies both sides with the inverse of the inertia matrix. This results in

$$\begin{bmatrix}\dot{p}\\\dot{q}\\\dot{r}\end{bmatrix}=\left(\begin{bmatrix}L(F_2-F_4)\\L(F_3-F_1)\\M_1-M_2+M_3-M_4\end{bmatrix}-\begin{bmatrix}p\\q\\r\end{bmatrix}\times I\begin{bmatrix}p\\q\\r\end{bmatrix}\right)I^{-1}=\left(\begin{bmatrix}L(F_2-F_4)\\L(F_3-F_1)\\M_1-M_2+M_3-M_4\end{bmatrix}-\begin{bmatrix}p\\q\\r\end{bmatrix}\times\begin{bmatrix}I_x&0&0\\0&I_y&0\\0&0&I_z\end{bmatrix}\begin{bmatrix}p\\q\\r\end{bmatrix}\right)I^{-1}=$$

$$=\left(\begin{bmatrix}L(F_2-F_4)\\L(F_3-F_1)\\M_1-M_2+M_3-M_4\end{bmatrix}-\begin{bmatrix}p\\q\\r\end{bmatrix}\times\begin{bmatrix}pI_x\\qI_y\\rI_z\end{bmatrix}\right)I^{-1}=\left(\begin{bmatrix}L(F_2-F_4)\\L(F_3-F_1)\\M_1-M_2+M_3-M_4\end{bmatrix}-\begin{bmatrix}qrI_z-rqI_y\\rpI_x-prI_z\\pqI_y-qpI_x\end{bmatrix}\right)I^{-1}=$$

$$=\left(\begin{bmatrix}L(F_2-F_4)+qr(I_y-I_z)\\L(F_3-F_1)+rp(I_z-I_x)\\M_1-M_2+M_3-M_4+pq(I_x-I_y)\end{bmatrix}\right)I^{-1}=\begin{bmatrix}\frac{L}{I_x}(F_2-F_4)+qr\frac{I_y-I_z}{I_x}\\\frac{L}{I_y}(F_3-F_1)+rp\frac{I_z-I_x}{I_y}\\\frac{1}{I_z}(M_1-M_2+M_3-M_4)+pq\frac{I_x-I_y}{I_z}\end{bmatrix}. \tag{2.31}$$

Using the equations for F_i (Equation (2.11)) and M_i (Equation (2.12)) in the final expression for the rate of angular velocity leads to

$$\begin{bmatrix}\dot{p}\\\dot{q}\\\dot{r}\end{bmatrix}=\begin{bmatrix}\frac{L}{I_x}k_F(\omega_2^2-\omega_4^2)+qr\frac{I_y-I_z}{I_x}\\\frac{L}{I_y}k_F(\omega_3^2-\omega_1^2)+rp\frac{I_z-I_x}{I_y}\\\frac{1}{I_z}k_M(\omega_1^2-\omega_2^2+\omega_3^2-\omega_4^2)+pq\frac{I_x-I_y}{I_z}\end{bmatrix}. \tag{2.32}$$

Finally, this results in the following system equations (expressed in terms of control signals U_i (2.13))

$$\begin{cases}\dot{x}=u(c(\psi)c(\theta))+v(c(\psi)s(\theta)s(\phi)-c(\phi)s(\psi))+w(c(\psi)c(\phi)s(\theta)+s(\psi)s(\phi))\,,\\\dot{y}=u(c(\theta)s(\psi))+v(s(\psi)s(\theta)s(\phi)+c(\psi)c(\phi))+w(c(\phi)s(\psi)s(\theta)-c(\psi)s(\phi))\,,\\\dot{z}=u(-s(\theta))+v(c(\theta)s(\phi))+w(c(\theta)c(\phi))\,,\\\dot{\phi}=p+q\cdot s(\phi)t(\theta)+r\cdot c(\phi)t(\theta)\,,\\\dot{\theta}=q\cdot c(\phi)-r\cdot s(\phi)\,,\\\dot{\psi}=q\cdot s(\phi)sec(\theta)+r\cdot c(\phi)sec(\theta)\,,\\\dot{u}=g(s(\theta))-qw+rv\,,\\\dot{v}=-g(c(\theta)s(\phi))-ru+pw\,,\\\dot{w}=-g(c(\theta)c(\phi))+\frac{U_1}{m}-pv+qu\,,\\\dot{p}=\frac{U_2}{I_x}+qr\frac{I_y-I_z}{I_x}\,,\\\dot{q}=\frac{U_3}{I_y}+rp\frac{I_z-I_x}{I_y}\,,\\\dot{r}=\frac{U_4}{I_z}+pq\frac{I_x-I_y}{I_z}\,.\end{cases} \tag{2.33}$$

A reminder of the symbols:

$[x\ y\ z]$	Linear position in Earth reference frame
$[\phi\ \theta\ \psi]$	Angular position in Earth reference frame
$[p\ q\ r]$	Linear velocity in body reference frame
$[u\ v\ w]$	Angular velocity in body reference frame

The values of the parameters are given by Table 2.1 below. A test bed can be developed and used for conducting the values, but in this case the data is obtained from[9]. The biggest reason for not doing an own test bed is time constraints.

Description	Parameter	Value [SI unit]
Quadcopter mass	m	0.547 $[kg]$
Acceleration of gravity	g	9.81 $[m/s^2]$
Arm length	L	0.17 $[m]$
Thrust coefficient	k_F	$1.5 \cdot 10^{-7}$ $[N \cdot RPM^{-2}]$
Torque coefficient	k_M	$3.75 \cdot 10^{-7}$ $[Nm \cdot RPM^{-2}]$
Air resistance coefficient	C_d	1.0 [-]
Inertia for x_b	I_x	$3.3 \cdot 10^{-3}$ $[kg \cdot m^2]$
Inertia for y_b	I_y	$3.3 \cdot 10^{-3}$ $[kg \cdot m^2]$
Inertia for z_b	I_z	$5.8 \cdot 10^{-3}$ $[kg \cdot m^2]$
Cross-sectional area for x_b	A_x	0.011 $[m^2]$
Cross-sectional area for y_b	A_y	0.011 $[m^2]$
Cross-sectional area for z_b	A_z	0.022 $[m^2]$

Table 2.1: The values of the parameters used in the model.

2.2.3 Linearization and SSR

Now the aim is to express the system equations in linear state space representation (SSR) form to be able to implement a LQR controller. Therefore, necessary steps involve determining the state vector and linearizing the system equations around a stationary point.[8] As mentioned in the section about SSR, the state vector is given by χ (the Greek letter chi, not to be confused with state x) and the SSR is then represented as $\dot{\chi} = A\chi + Bu$. Considering the system equations (2.33) it is reasonable to define the state vector as

$$\chi = \begin{bmatrix} x \\ y \\ z \\ \phi \\ \theta \\ \psi \\ u \\ v \\ w \\ p \\ q \\ r \end{bmatrix} , \qquad \dot{\chi} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ \dot{u} \\ \dot{v} \\ \dot{w} \\ \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = A\chi + Bu\,. \tag{2.34}$$

The linearization of a nonlinear system can be done by approximating (linearizing) the system around a stationary (equilibrium) point (χ_s, u_s) (sometimes denoted (χ_e, u_e) or (χ_0, u_0)), where the system behaves linear in a small vicinity around this point. The LQR controller can adjust the input to maintain stability of the quadcopter model around the stationary point, even when disturbances are present. Before the linearization, simplifying the model is needed since the current system equations involve numerous interdependent trigonometric functions. Choosing the stationary point as an arbitrary height in space where the quadcopter hovers straight up results in changes only in altitude, not in roll and pitch for attitude control. This enables the following simplifications (small angle approximation)

$$\begin{cases} \cos(\text{arg}) \approx 1\,, \\ \sin(\text{arg}) \approx \text{arg}\,, \\ \tan(\text{arg}) \approx \text{arg}\,. \end{cases} \tag{2.35}$$

Using this simplification on the system equations gives

$$\begin{cases} \dot{x} = u + v(\theta \cdot \phi - \psi) + w(\theta + \psi \cdot \phi)\,, \\ \dot{y} = u \cdot \psi + v(\psi \cdot \theta \cdot \phi + 1) + w(\psi \cdot \theta - \phi)\,, \\ \dot{z} = u \cdot (-\theta) + v \cdot \phi + w\,, \\ \dot{\phi} = p + q \cdot \phi \cdot \theta + r \cdot \theta\,, \\ \dot{\theta} = q - r \cdot \phi\,, \\ \dot{\psi} = q \cdot \phi + r\,, \\ \dot{u} = g \cdot \theta - qw + rv\,, \\ \dot{v} = -g \cdot \phi - ru + pw\,, \\ \dot{w} = -g + \dfrac{U_1}{m} - pv + qu\,, \\ \dot{p} = \dfrac{U_2}{I_x} + qr\dfrac{I_y - I_z}{I_x}\,, \\ \dot{q} = \dfrac{U_3}{I_y} + rp\dfrac{I_z - I_x}{I_y}\,, \\ \dot{r} = \dfrac{U_4}{I_z} + pq\dfrac{I_x - I_y}{I_z}\,. \end{cases} \tag{2.36}$$

Then $\dot{\chi}$ becomes

$$\dot{\chi} = f(\chi, u) = \begin{bmatrix} u + v(\theta \cdot \phi - \psi) + w(\theta + \psi \cdot \phi) \\ u \cdot \psi + v(\psi \cdot \theta \cdot \phi + 1) + w(\psi \cdot \theta - \phi) \\ u \cdot (-\theta) + v \cdot \phi + w \\ p + q \cdot \phi \cdot \theta + r \cdot \theta \\ q - r \cdot \phi \\ q \cdot \phi + r \\ g \cdot \theta - qw + rv \\ -g \cdot \phi - ru + pw \\ -g + \frac{U_1}{m} - pv + qu \\ \frac{U_2}{I_x} + qr\frac{I_y - I_z}{I_x} \\ \frac{U_3}{I_y} + rp\frac{I_z - I_x}{I_y} \\ \frac{U_4}{I_z} + pq\frac{I_x - I_y}{I_z} \end{bmatrix} . \tag{2.37}$$

Defining $\delta\chi = \chi - \chi_s$ and $\delta u = u - u_s$ as small perturbations, a Taylor series expansion around the stationary point (while only keeping the linear terms, thus neglecting the term $f(\chi_s, u_s)$ and all high order terms) yields

$$\delta\dot{\chi} = \frac{\partial f(\chi_s, u_s)}{\partial \chi}\delta\chi + \frac{\partial f(\chi_s, u_s)}{\partial u}\delta u . \tag{2.38}$$

The partial derivative (∂) represents the Jacobian matrix. If χ_s and u_s are stationary (equilibrium) solutions, then $\chi = f(\chi_s, u_s)$ becomes zero. This implies that for equilibrium point χ_s, the input u_s leads to the solution of the algebraic system $f(\chi_s, u_s) = 0$. As mentioned earlier, the stationary point is an arbitrary point in space where the quadcopter hovers, which means that the state vector can be written as

$$\chi_s = \begin{bmatrix} x_s & y_s & z_s & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} . \tag{2.39}$$

To achieve hovering straight up only the thrust force for each rotor needs to change equally, which means that $U_1 = mg$ (to counteract gravity) and $U_2 = U_3 = U_4 = 0$. Hence, u_s is

$$u_s = \begin{bmatrix} mg & 0 & 0 & 0 \end{bmatrix}^T . \tag{2.40}$$

Examining Equation (2.38) (which is in the form of a linear time-invariant state equation $\dot{\chi} = A\chi + Bu$), one can observe that for the linear approximation system, the matrix A (the Jacobian of $f(\chi, u)$ with respect to χ) and B (the Jacobian of $f(\chi, u)$ with respect to u) are expressed as

$$A = \left.\frac{\partial f(\chi, u)}{\partial \chi}\right|_{\chi=\chi_s, u=u_s} = \left.\begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} & \frac{\partial f_1}{\partial z} & \frac{\partial f_1}{\partial \phi} & \cdots & \frac{\partial f_1}{\partial r} \\ \frac{\partial f_2}{\partial \dot{x}} & \frac{\partial f_2}{\partial \dot{y}} & \frac{\partial f_2}{\partial \dot{z}} & \frac{\partial f_2}{\partial \phi} & \cdots & \frac{\partial f_2}{\partial r} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_{12}}{\partial \dot{x}} & \frac{\partial f_{12}}{\partial \dot{y}} & \frac{\partial f_{12}}{\partial \dot{z}} & \frac{\partial f_{12}}{\partial \phi} & \cdots & \frac{\partial f_{12}}{\partial r} \end{bmatrix}\right|_{\chi=\chi_s, u=u_s} , \tag{2.41}$$

$$B = \left.\frac{\partial f(\chi,u)}{\partial u}\right|_{\chi=\chi_s,u=u_s} = \left.\begin{bmatrix} \frac{\partial f_1}{\partial U_1} & \frac{\partial f_1}{\partial U_2} & \frac{\partial f_1}{\partial U_3} & \frac{\partial f_1}{\partial U_4} \\ \frac{\partial f_2}{\partial U_1} & \frac{\partial f_2}{\partial U_2} & \frac{\partial f_2}{\partial U_3} & \frac{\partial f_2}{\partial U_4} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial f_{12}}{\partial U_1} & \frac{\partial f_{12}}{\partial U_2} & \frac{\partial f_{12}}{\partial U_3} & \frac{\partial f_{12}}{\partial U_4} \end{bmatrix}\right|_{\chi=\chi_s,u=u_s} . \tag{2.42}$$

Solving for A and B matrices results in

$$A = \left.\begin{bmatrix} 0 & 0 & 0 & v\theta+w\psi & v\phi+w & -v+w\phi & 1 & \theta\phi-\psi & \theta+\psi\phi & 0 & 0 & 0 \\ 0 & 0 & 0 & v\psi\theta-w & v\psi\phi+w\psi & u+v\theta\phi+w\theta & \psi & \psi\theta\phi+1 & \psi\theta-\phi & 0 & 0 & 0 \\ 0 & 0 & 0 & v & -u & 0 & -\theta & \phi & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & q\theta & q\phi+r & 0 & 0 & 0 & 0 & 1 & \phi\theta & \theta \\ 0 & 0 & 0 & -r & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -\phi \\ 0 & 0 & 0 & q & 0 & 0 & 0 & 0 & 0 & 0 & \phi & 1 \\ 0 & 0 & 0 & 0 & g & 0 & 0 & r & -q & 0 & -w & v \\ 0 & 0 & 0 & -g & 0 & 0 & -r & 0 & p & w & 0 & -u \\ 0 & 0 & 0 & 0 & 0 & 0 & q & -p & 0 & -v & u & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & r\frac{I_y-I_z}{I_x} & q\frac{I_y-I_z}{I_x} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & r\frac{I_z-I_x}{I_y} & 0 & p\frac{I_z-I_x}{I_y} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & q\frac{I_x-I_y}{I_z} & p\frac{I_x-I_y}{I_z} & 0 \end{bmatrix}\right|_{\chi=\chi_s,u=u_s} =$$

$$= \begin{bmatrix} 0&0&0&0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&g&0&0&0&0&0&0&0 \\ 0&0&0&-g&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0 \end{bmatrix} , \tag{2.43}$$

$$B = \left.\begin{bmatrix} 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ \frac{1}{m}&0&0&0 \\ 0&\frac{1}{I_x}&0&0 \\ 0&0&\frac{1}{I_y}&0 \\ 0&0&0&\frac{1}{I_z} \end{bmatrix}\right|_{\chi=\chi_s,u=u_s} = \begin{bmatrix} 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ \frac{1}{m}&0&0&0 \\ 0&\frac{1}{I_x}&0&0 \\ 0&0&\frac{1}{I_y}&0 \\ 0&0&0&\frac{1}{I_z} \end{bmatrix} . \tag{2.44}$$

Now the SSR ($\dot{\chi} = A\chi + Bu$) of the linear system is obtained, where matrices A and B are defined by Equations (2.43) and (2.44), respectively. The expression for $\dot{\chi}$ is as follows

$$\begin{cases} \dot{x} = u \\ \dot{y} = v \\ \dot{z} = w \\ \dot{\phi} = p \\ \dot{\theta} = q \\ \dot{\psi} = r \\ \dot{u} = g\theta \\ \dot{v} = g\phi \\ \dot{w} = -\frac{1}{m} \\ \dot{p} = \frac{U_2}{I_x} \\ \dot{q} = \frac{U_3}{I_y} \\ \dot{r} = \frac{U_4}{I_z} \end{cases} , \quad \dot{\chi} = \begin{bmatrix} u \\ v \\ w \\ p \\ q \\ r \\ g\theta \\ g\phi \\ -\frac{1}{m} \\ \frac{U_2}{I_x} \\ \frac{U_3}{I_y} \\ \frac{U_4}{I_z} \end{bmatrix} . \tag{2.45}$$

With this established, one can proceed to implement the LQR controller. This can be accomplished using a software, such as Matlab. With the help of Matlab commands *ctrb* (controllability), *obsv* (observability) and *rank*, one can also verify that the system is both controllable and observable. This is done by observing that both the controllability matrix and the observability matrix have full rank (which corresponds to the number of states, 12).

3 Method

The model is created in Matlab and Simulink. This project does not use any UAV hardwares, it relies solely on a laptop and software. However, it is possible to convert the Matlab models into C/C++ code and integrate them into the flight controller of a real UAV.

3.1 Hardware

The hardware used to conduct the work is a *Dell* laptop. It serves as the computing platform for all programming, simulations, and data analysis tasks. The device and windows specifications are

- Processor: 13th Gen Intel(R) Core(TM) i7-13850HX 2.10 GHz
- Installed RAM: 32,0 GB (31,7 GB usable)
- System type: 64-bit operating system, x64-based processor

- Edition: Windows 10 Enterprise LTSC
- Version: 21H2
- OS build: 19044.4291 .

3.2 Software

Matlab and Simulink were developed by MathWorks, which was founded in 1984 by Jack Little and Cleve Moler. They recognized the need for a more powerful computation environment among engineers and scientists. Initially, they introduced Matlab, followed by the development of Simulink and other add-on products for specialized application areas.[10]

Simulink is a versatile tool designed for creating models, conducting simulations, and analyzing systems. It uses a block system, which enhances the comprehensibility of system dynamics and signal flow, making it more intuitive. The graphical block diagramming feature is beneficial for solving problems in automatic control and signal processing. The block diagram environment also provides the creation of multidomain models, allowing for simulation prior to hardware implementation, and deploy without writing code. This project uses the latest available version of Matlab, which at the beginning of the project was MATLAB $R2023b$ and later transitioning to $R2024a$ (released 2024 Mar 20).

3.3 Implementation

An overview of the *Simulink* model is shown in Figure 3.1. The system consists of two blocks: *Controller* and *Quadcopter*. The *Controller* is a LQR controller responsible for receiving error values (the difference between the desired states and the current states) and generating the input values (U) to the *Quadcopter*. The *Quadcopter* block mainly contains the control signals and system equations necessary for calculating the dynamics of the quadcopter.

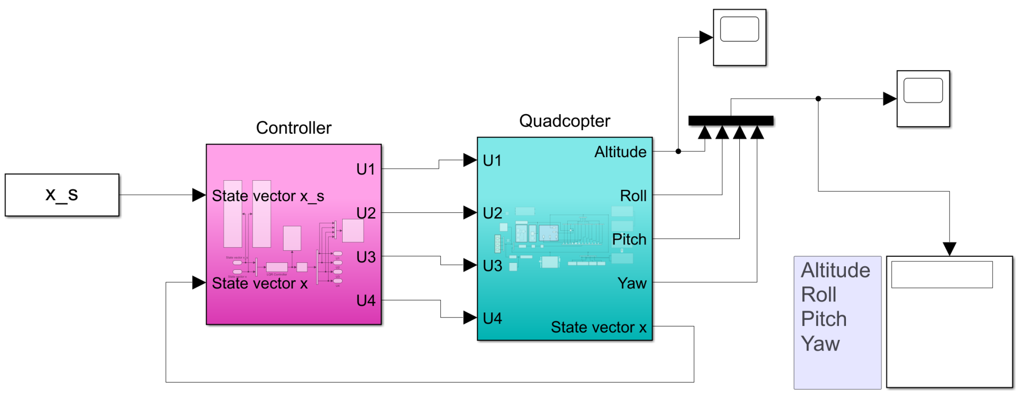


Figure 3.1: The Simulink model.

3.3.1 Overview of files

The project largely consists of eight files in Matlab, where one is a Simulink file (.slx) and the rest are script files (.m). In order to more easily understand the structure of the project, what the main purpose for each file is and how the files interact, there is a list with short descriptions below.

- *InitilazerSystem.m* - This file initializes the system. It sets parameters that multiple files use as global variables to make the code more effective. It starts four other files, first *ReferenceValues* to ask the user for the desired position (x, y, z), then it initiates the Simulink system. Additionally, it starts *Plot_states* and *Animation_Quad*, which can be commented out if plots or animations are not required.
- *MainQuad.m* - This file consists of the quadcopter's system equations. It computes (the time derivative of) all states to obtain the dynamic of the quadcopter.
- *LQR_controller.m* - As the name suggests, this file is the LQR controller. It takes the difference between the desired state vector (from the user, *ReferenceValues*) and the current state vector (from *MainQuad*). Utilizing matrices A, B, Q, and R, it computes the gain matrix K using the LQR command *lqr(A,B,Q,R)*. The output u can then be calculated.
- *ReferenceValues.m* - This file prompts the user for desired x, y, and z value, creating the global state vector x_s.
- *U_to_w.m* - This file takes in a vector consisting of the control signals U_i and converts these to the rotational speed (ω_i) of each rotor. With the help of the *fsolve* command, it determines the rotor speeds by solving four equations (control signals U_i) with four unknowns (rotor speeds ω_i).
- *Plot_states.m* - Just as the name suggests, this file plots the different states and the altitude with wind disturbances. It does this by using output data matrices from the Simulink model.
- *Animation_Quad.m* - This file uses the output data from Simulink to create an animation of the quadcopter's movement. Figure 3.2 is a still image of the animation.

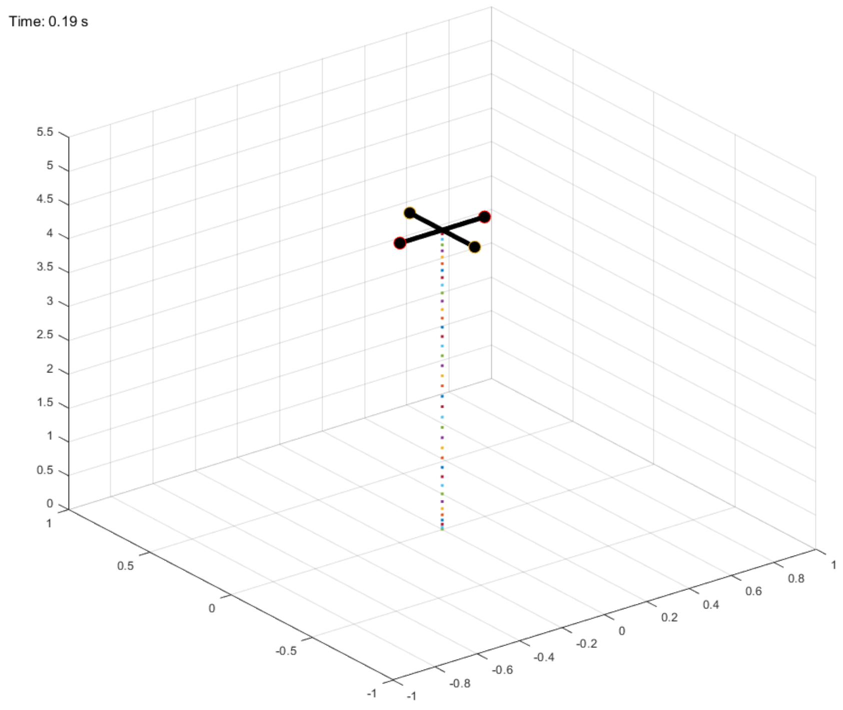


Figure 3.2: A still image of the animation of a quadcopter.

3.3.2 Quadcopter block

The inside of the *Quadcopter* block is shown in Figure 3.3. It uses two functions (Matlab scripts) called *MainQuad* and *U_to_w*. There are six output data matrices to workspace, denoted as *out.'name'*. Additionally, three displays are used to verify the reasonability of the values and for debugging purposes. When the *Quadcopter* is used alongside the *Controller* block, the inputs are U_i, but when the *Quadcopter* is tested the inputs are ω_i. The user adjusts the *MainQuad* file according to the situation. Also, during block testing, the output *state vector x* is commented out. The wind disturbance is also adjusted (commented out) according to the user's desires. There were problems in the form of oscillations with the *Quadcopter* block, see the section Appendix A.2.

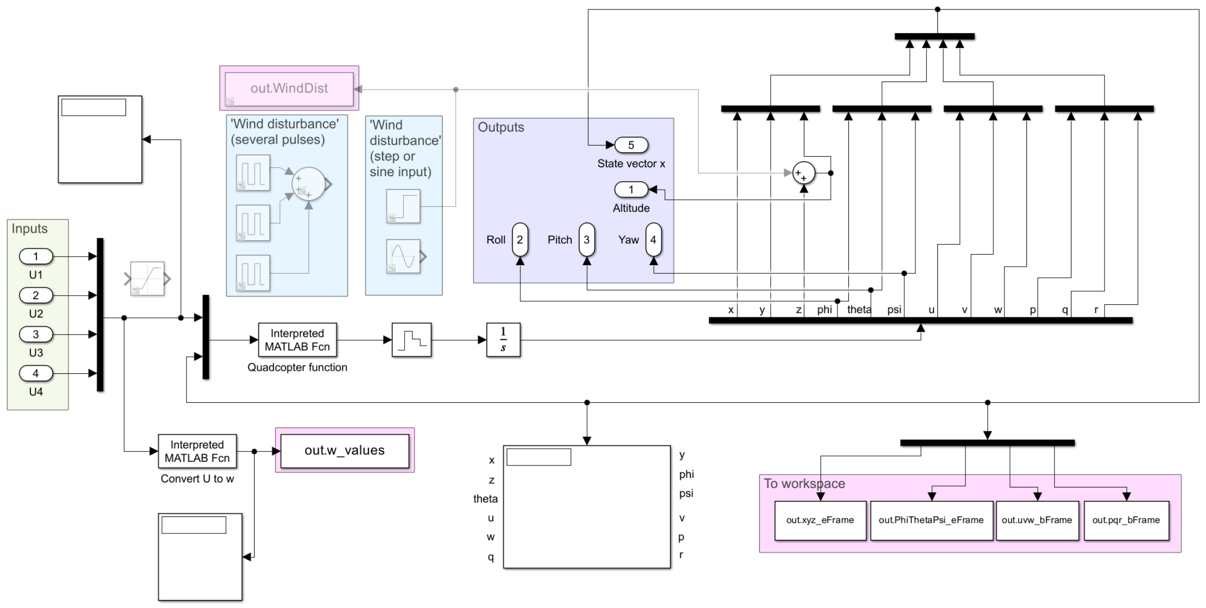


Figure 3.3: The inside of the *Quadcopter* block.

To verify that the system equations of the quadcopter are correct and ensure proper functionality, several tests are conducted. The first test in Figure 3.4 shows the result of altitude, roll, pitch and yaw when all four rotors are set to the same speed. In order to counteract the gravitational force the rotors must generate a total force of $F = mg = 0.547 \cdot 9.81 = 5.36607N$. Each rotor is required to produce a force of approximately $1.3415N$. Converting the force into a rotation rate: $\omega_i = \sqrt{F_i/k_F = 1.3415/(1.5 \cdot 10^{-7})} \approx 2991$ RPM. As expected, the outcome in the figure indicates that only the altitude changes.

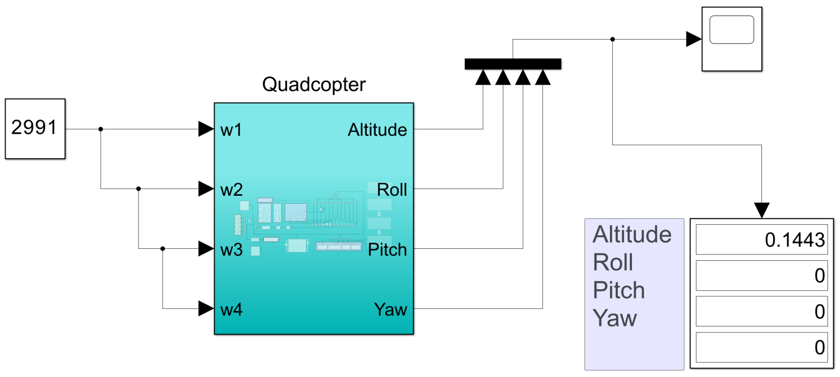


Figure 3.4: A test done on the *Quadcopter* block to verify the altitude.

The second test in Figure 3.5 demonstrates the yaw movement of the quadcopter. Rotor 1 and 3 are set to the same speed, while rotor 2 and 4 are set to a slightly lower speed. Due to the diagonal arrangement where rotor 1 and 3 rotate in the same direction, while rotor 2 and 4 rotate in the opposite direction, this configuration leads to a net torque making the quadcopter yaw. The result shows the quadcopter changing yaw angle, accompanied by change in altitude.

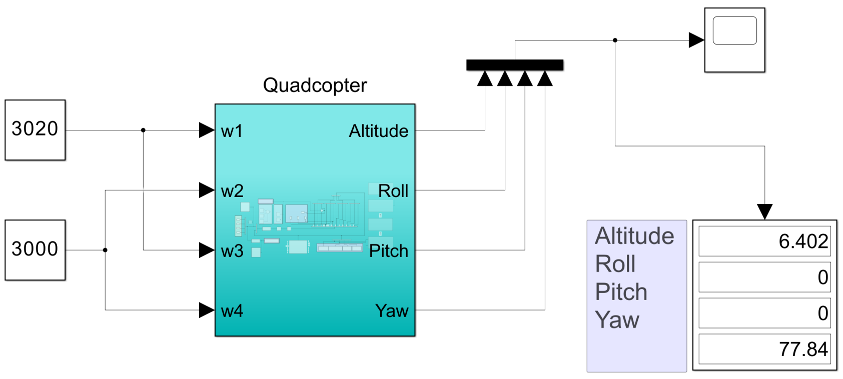


Figure 3.5: A test done on the *Quadcopter* block to verify yaw movement.

The final test aims to validate the functionality of the roll and pitch angles. While it is challenging to precisely verify the values displayed in Figure 3.6, they appear reasonable - neither too great nor too small.

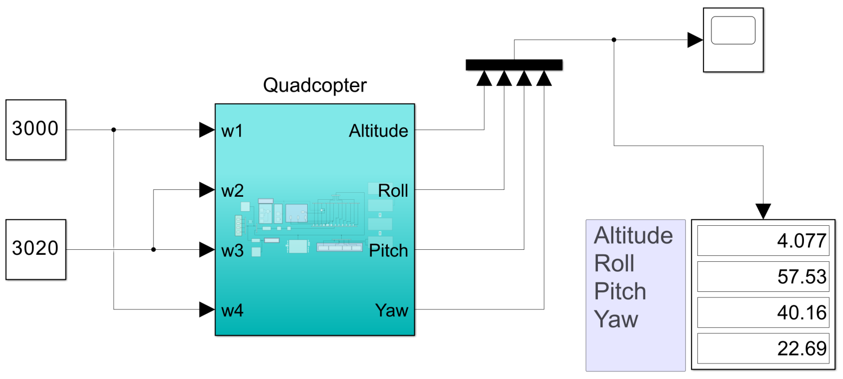


Figure 3.6: A test done on the *Quadcopter* block to verify roll and pitch movements.

3.3.3 Controller block

Figure 3.7 provides the inside of the *Controller* block. It almost only consists of the *LQR_controller* function (Matlab script). This function utilizes the A and B matrices obtained from linearization and calculates $\delta\chi$ by using the difference between the desired state vector χ_s and the calculated state vector χ from the *Quadcopter* block. Matlab command *lqr(A,B,Q,R)* yields the optimal matrix gain K, which is then utilized to determine the inputs to the quadcopter, $u = -K \cdot \chi$ (in reality $\delta u = -K \cdot \delta\chi$). Since the state vector χ is needed from the *Quadcopter* block it is hard to conduct tests on solely the *Controller* block, thus the controller is evaluated when connected to the *Quadcopter* block.

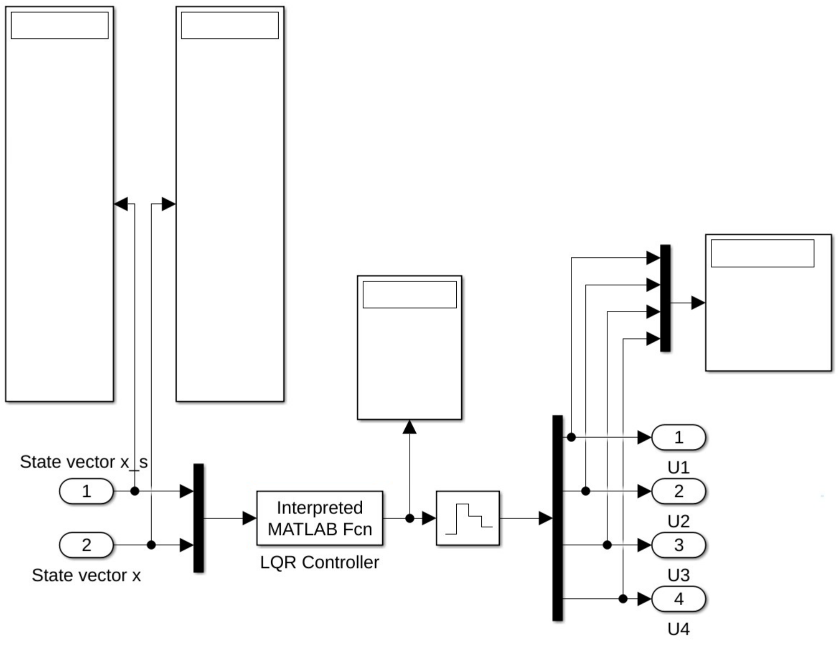


Figure 3.7: The inside of the *Controller* block

3.4 Components of a diagonal Q matrix

Selecting the elements for the matrices R and Q in the LQR controller is an iterative process, since there are no rules or methods for estimating their values. It requires comparing the system's results with the desired performance and adjusting the elements accordingly. This challenge underscores the necessity of an understanding of the system. As mentioned in the subsection about LQR, the cost function in a bigger matrix form results in

$$\begin{bmatrix} \chi^T \\ u^T \end{bmatrix} \cdot \begin{bmatrix} Q & N \\ N^T & R \end{bmatrix} \cdot \begin{bmatrix} \chi \\ u \end{bmatrix} = \begin{bmatrix} \chi^T Q \chi + \chi^T N u \\ u^T N \chi + u^T R u \end{bmatrix} . \tag{3.1}$$

Denoting matrix W_{tot} as

$$W_{tot} = \begin{bmatrix} Q & N \\ N^T & R \end{bmatrix} , \tag{3.2}$$

$$W_{tot} = \begin{bmatrix} \begin{bmatrix} eq_{1,1} & eq_{1,2} & \cdots & eq_{1,12} \\ eq_{2,1} & eq_{2,2} & \cdots & eq_{2,12} \\ \vdots & \vdots & \ddots & \vdots \\ eq_{12,1} & eq_{12,2} & \cdots & eq_{12,12} \end{bmatrix} & \begin{bmatrix} n_{1,1} & n_{2,1} & n_{3,1} & n_{4,1} \\ n_{1,2} & n_{2,2} & n_{3,2} & n_{4,2} \\ \vdots & \vdots & \vdots & \vdots \\ n_{1,12} & n_{2,12} & n_{3,12} & n_{4,12} \end{bmatrix} \\ \begin{bmatrix} n_{1,1} & n_{1,2} & \cdots & n_{1,12} \\ n_{2,1} & n_{2,2} & \cdots & n_{2,12} \\ n_{3,1} & n_{3,2} & \cdots & n_{3,12} \\ n_{4,1} & n_{4,2} & \cdots & n_{4,12} \end{bmatrix} & \begin{bmatrix} er_{1,1} & er_{1,2} & er_{1,3} & er_{1,4} \\ er_{2,1} & er_{2,2} & er_{2,3} & er_{2,4} \\ er_{3,1} & er_{3,2} & er_{3,3} & er_{3,4} \\ er_{4,1} & er_{4,2} & er_{4,3} & er_{4,4} \end{bmatrix} \end{bmatrix} , \tag{3.3}$$

where $eq_{i,j}$ (stands for *element q*, not to be confused with state q) denotes elements in matrix Q, $n_{i,j}$ represents elements in matrix N and $er_{i,j}$ denotes elements in matrix R. Matrices

W_{tot} and Q are obliged to be semi-definite (symmetric with non-negative eigenvalues) and R is required to be positive definite (symmetric with positive eigenvalues). With this quadcopter system the size of Q is 12×12, the size of R is 4×4 and the size of W_{tot} becomes 16×16. When matrix Q is non-diagonal, the introduction of matrix N (which can be zero when Q is diagonal) is necessary to ensure that W_{tot} is semi-definite. The tuning process typically involves diagonal matrices Q and R; this simplifies the consideration of weights between the states χ and inputs u.

When the quadcopter is in a hovering state, the primary concern lies with its altitude, with minimal angular deviations. Denoting Q_{tot} as $\chi^T Q \chi$ and evolving it with a diagonal Q matrix leads to

$$Q_{tot} = x^2 \cdot eq_{1,1} + y^2 \cdot eq_{2,2} + z^2 \cdot eq_{3,3} + \phi^2 \cdot eq_{4,4} + \ \cdots \ + r^2 \cdot eq_{12,12}\,. \tag{3.4}$$

The equation shows that each element of matrix Q becomes a direct weight to each state. In the results section, it is shown how well the quadcopter performs when different values on the elements $eq_{i,j}$ are tested.

4 Results

In this section, the results of tests and analyzes are done to evaluate the performance of the LQR controller and the dynamics of the quadcopter model. The section is divided into subsections: *Tune LQR*, where different tuning of the LQR is done; *Hovering (with wind disturbances)*, where the quadcopter's behavior under varying wind conditions is examined; and *Model Parameters*, where the impact of different model parameters on the system's performance is explored. In the section Appendix A.3 there is a beginning of an attempt to tune LQR for a non-diagonal matrix Q.

4.1 Tune LQR (diagonal Q matrix)

The initial test consists of a diagonal version of Q with various values for its elements, as outlined in Table 4.1. All remaining diagonal elements $eq_{i,j}$ not specified in the table are set to 1×10^{-6}, due to constraints prohibiting them from being zero. The reason behind the selection of elements for testing, begins with $eq_{3,3}$ corresponding to the weight controlling the z state (height), followed by the weights controlling states ϕ, θ, u, v, and w, given their direct influence on the state z as shown in the system equations. Matrix R is configured as a diagonal matrix with all elements set to one. The desired state vector $x_s = [0\ 0\ 5\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^T$, indicates a desired height of 5 meters. The simulation runtime is set to 10 seconds, using *ode4* solver and a step size of 0.001.

	$eq_{3,3}$	$eq_{4,4}$	$eq_{5,5}$	$eq_{7,7}$	$eq_{8,8}$	$eq_{9,9}$	Final z [m]	Angles (ϕ, θ, ψ) [°]
Run 1	10	1	1	1	1	1	3.303	$\approx 10^{-12}$
Run 2	1000	1	1	1	1	1	4.83	$\approx 10^{-6}$, 10^{-6} , 10^{-12}
Run 3	1000	0.001	0.001	0.001	0.001	0.001	4.83	$\approx 10^{-12}$, 10^{-11} , 10^{-13}
Run 4	10000	0.001	0.001	0.001	0.001	0.001	4.946	$\approx 10^{-11}$, 10^{-10} , 10^{-13}

Table 4.1: Tuning of LQR when the matrix Q is diagonal. Focusing on state z.

The optimal approach, based on the outcomes presented in Table 4.1, is to maintain a high value for $eq_{3,3}$ while keeping all other elements $eq_{i,j}$ low. This outcome is unsurprising, given that the primary concern is the vertical movement denoted by z, with $eq_{3,3}$ being directly linked to it. The second test evaluates the quadcopter's hovering performance at alternative positions by adjusting the elements of Q associated with the x and y states. Since altering the position requires adjustments in angles, which challenges the linearization assumption of small angles, this test aims to assess the quadcopter's behavior under such conditions, see Table 4.2. The desired position is set to $(x, y, z) = (2, 2, 5)$ meters and once again, all remaining diagonal elements are set to $1 \cdot 10^{-6}$.

	$eq_{3,3}$	$eq_{1,1}$	$eq_{2,2}$	Final position (x,y,z) [m]	Biggest (ϕ, θ, ψ) [°]
Run 1	10	0.01	0.01	≈ (-1.6 , -1.8 , 1.7)	≈ (-1.3 , 1.2 , 0.09)
Run 2	1000	0.01	0.01	≈ (6.6 , 6.7 , 4.8)	≈ (3.8 , 3.6 , -0.11)
Run 3	1000	1	1	≈ (-1.3 , -0.63 , 4.82)	≈ (-7.5 , 7.5 , 0.1)
Run 4	1000	100	100	≈ (4.6 , 3 , 4.7)	≈ (-15 , 20 , -2.4)

Table 4.2: Tuning of LQR when the matrix Q is diagonal. Focusing on states z, x, y.

Higher values of $eq_{1,1}$ and $eq_{3,3}$ lead to oscillations in the system states, more at the beginning of the run as shown in Figures 4.1 and 4.2 for the linear position and velocity of *Run 4* with a simulation time of 50 seconds instead of 10 seconds, since the oscillation seems to reduce after 10 seconds. Increasing the value of $eq_{1,1}$, $eq_{2,2}$, $eq_{3,3}$ crashes the system, causing all states to become very large (such as 10^{67} to 10^{180}) which results in invalid output.

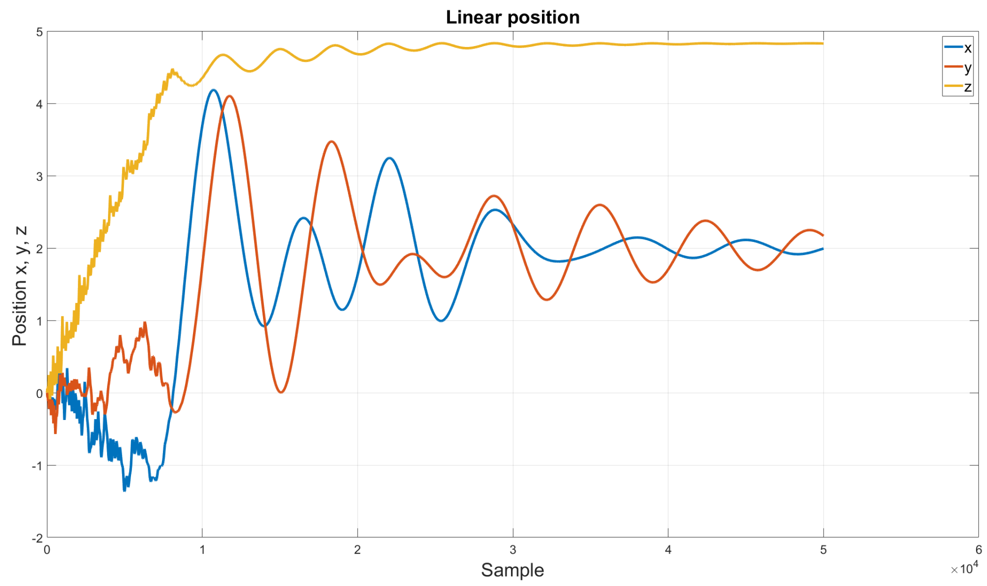


Figure 4.1: The position of *Run 4* from Table 4.2. Remember that the step size is 0.001 which means that sample $1 \cdot 10^4$ is 10 seconds.

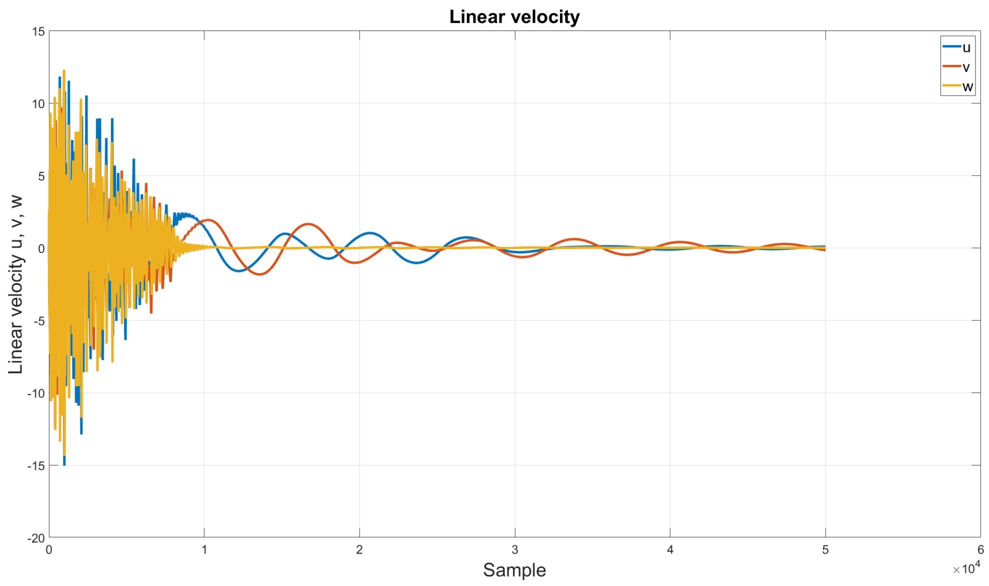


Figure 4.2: The angular position of *Run 4* from Table 4.2.

Better tuning of the LQR may lead to the oscillations decreasing and the development of a more robust controller. Given that the states x, y, and z depend on the other states, such as u, v, w, ϕ, etc., adjusting the corresponding $eq_{i,j}$ element can result in better accuracy. To observe the behavior of oscillations over time, the simulation stop time is extended to 50 seconds, see Table 4.3.

	Run 1	Run 2	Run 3	Run 4
$eq_{1,1}$	100	100	100	10000
$eq_{2,2}$	100	100	100	10000
$eq_{3,3}$	100	100	100	10000
$eq_{4,4}$	0.1	1	10	10
$eq_{5,5}$	0.1	1	10	10
$eq_{6,6}$	0.1	1	10	10
$eq_{7,7}$	0.1	1	10	10
$eq_{8,8}$	0.1	1	10	10
$eq_{9,9}$	0.1	1	10	10
$eq_{10,10}$	0.1	1	10	10
$eq_{11,11}$	0.1	1	10	10
$eq_{12,12}$	0.1	1	10	10
Final z [m]	$\approx$ (1.3, 3.3, 4.4)	$\approx$ (1.46, -1.26, 4.4)	$\approx$ (-5.56, 7.64, 4.36)	$\approx$ (1.79, 1.75, 4.94)
Biggest (ϕ, θ, ψ) [°]	$\approx$ (14, 20, -1.6)	$\approx$ (-10.6, 17, -0.75)	$\approx$ (-6.8, 12, -0.31)	$\approx$ (-16, 27, -3.5)

Table 4.3: Tuning of LQR when all diagonal elements of Q are non(-close-to)-zero.

Figures 4.3 and 4.4 show the linear and angular position of *Run 4* in Table 4.3. One can see that there are oscillations, reducing with time.

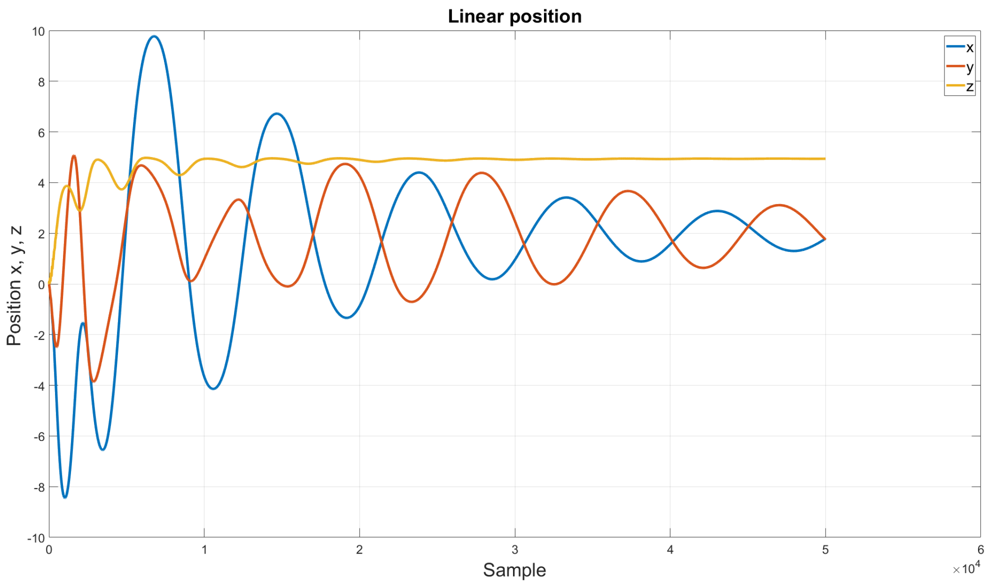


Figure 4.3: The position of *Run 4* from Table 4.3.

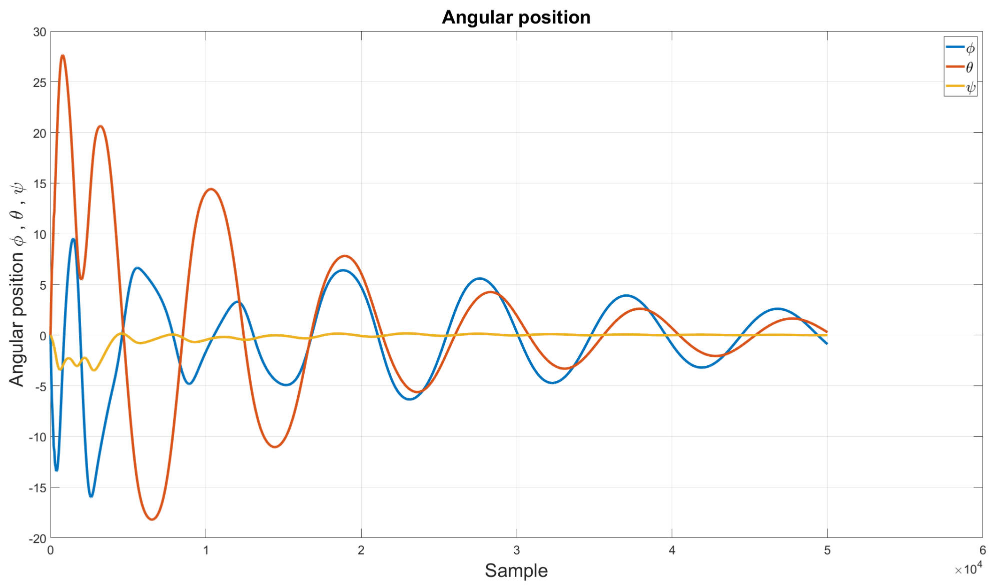


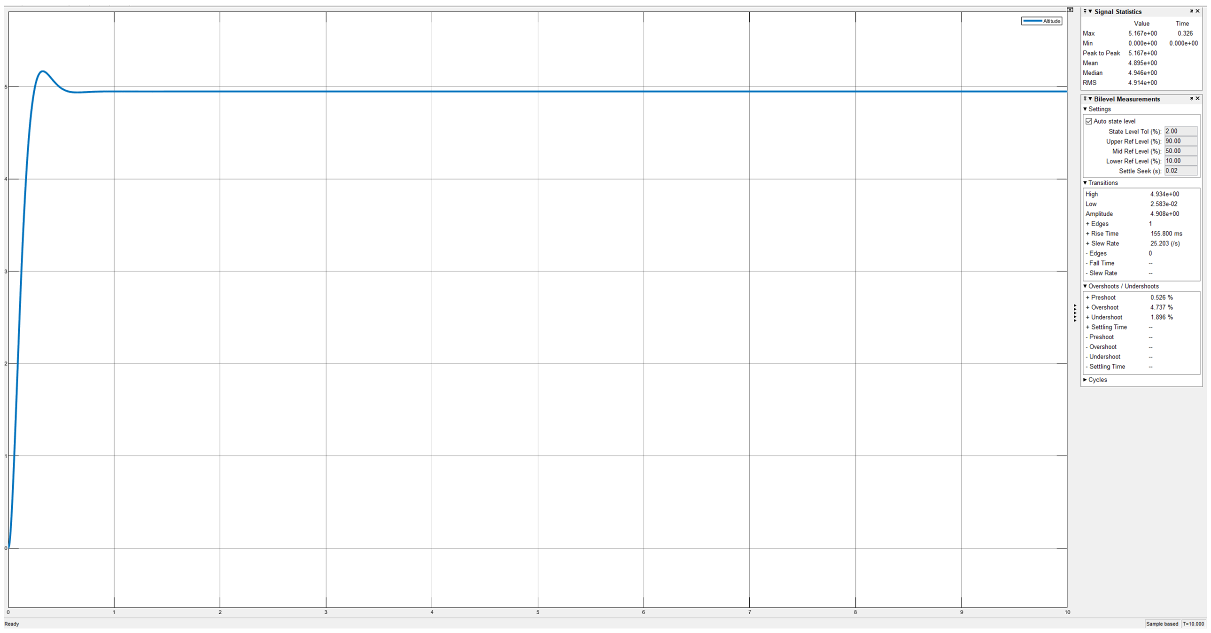
Figure 4.4: The angular position of *Run 4* from Table 4.3.

4.2 Hovering (with wind disturbance)

To assess which tuning values contribute to better accuracy and robustness, different characteristics are evaluated. The transient response is a step response when the desired behavior is for the quadcopter to hover. Since the quadcopter reaches the desired height as quickly as possible and then maintains the altitude. Therefore, reasonable characteristics for comparison include rise time, overshoot, undershoot and steady-state error. Table 4.4 shows the characteristics for the different runs done and written in Table 4.1. The values of the characteristics are obtained by using a scope in Simulink to view the altitude and using the *Tools* $\rightarrow$ *Measurements* feature. Figure 4.5 shows the scope for *Run 4*.

	Rise time [s]	Overshoot [%]	Undershoot [%]	Steady-state error
Run 1	1.023	1.531	1.998	1.697
Run 2	0.2858	3.646	1.959	0.17
Run 3	0.2779	4.737	1.987	0.17
Run 4	0.1558	4.737	1.896	0.054

Table 4.4: The results from the runs done and written in Table 4.1.



```
Signal Statistics
               Value        Time
Max            5.167e+00    0.326
Min            0.000e+00    0.000e+00
Peak to Peak   5.167e+00
Mean           4.895e+00
Median         4.946e+00
RMS            4.914e+00

Bilevel Measurements
Settings
[✓] Auto state level
    State Level Tol (%): 2.00
    Upper Ref Level (%): 90.00
      Mid Ref Level (%): 50.00
    Lower Ref Level (%): 10.00
        Settle Seek (s): 0.02
Transitions
High            4.934e+00
Low             2.583e-02
Amplitude       4.908e+00
+ Edges         1
+ Rise Time     155.800 ms
+ Slew Rate     25.203 (/s)
- Edges         0
- Fall Time     --
- Slew Rate     --
Overshoots / Undershoots
+ Preshoot      0.526 %
+ Overshoot     4.737 %
+ Undershoot    1.896 %
+ Settling Time --
- Preshoot      --
- Overshoot     --
- Undershoot    --
- Settling Time --
Cycles

Ready                                   Sample based  T=10.000
```

Figure 4.5: The view of a scope in Simulink, where parameter values are visible.

To see how well the quadcopter performs under the influence of wind disturbances, several tests are conducted using the tuning values specified in Table 4.1 for *Run 4*. Initially, a step input, representing a vertical wind gust, is applied. The step input begins at time 5 seconds and different values on the amplitude are tested. The outcomes are shown in Figures 4.6-4.8.

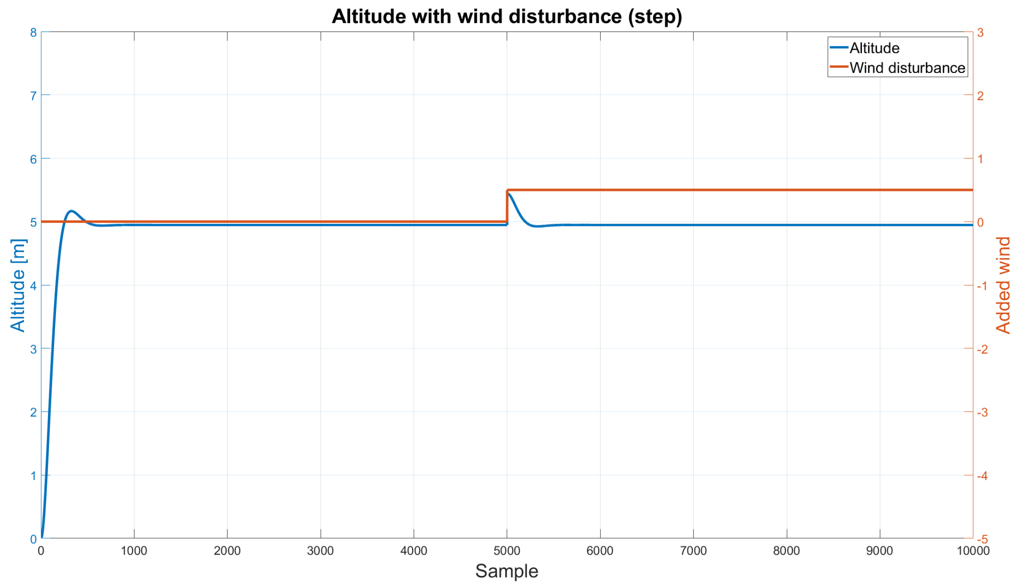


Figure 4.6: The altitude when it is disturbed by a step input with amplitude 0.5 m.

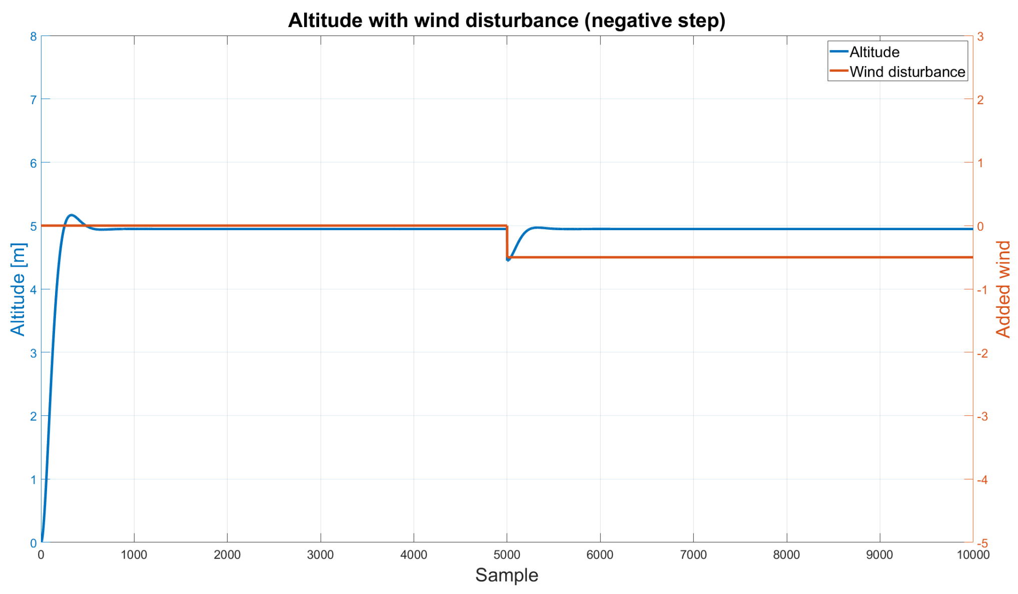


Figure 4.7: The altitude when it is disturbed by a step input with amplitude -0.5 m.

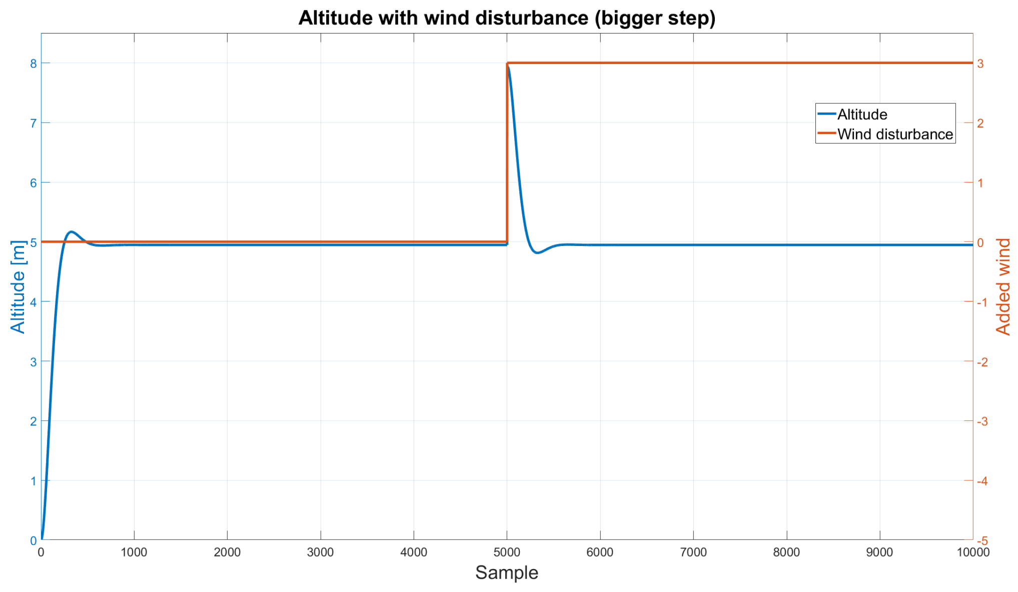


Figure 4.8: The altitude when it is disturbed by a step input with amplitude 3 m.

The result is promising; it indicates that the quadcopter recovers fast from the wind gust and returns to hovering at nearly the desired height, even when subjected to a bigger wind gust. To further test the quadcopter's robustness, a sum of several pulse generators is used as input, which represents a series of vertical wind gusts with varying amplitudes. Figure 4.9 shows the result. When the wind changes more gradually the controller is able to adjust more effectively, as shown in Figure 4.10, where the wind disturbance is represented by a sine wave with an amplitude of 1 m and a frequency of 1 rad/s.

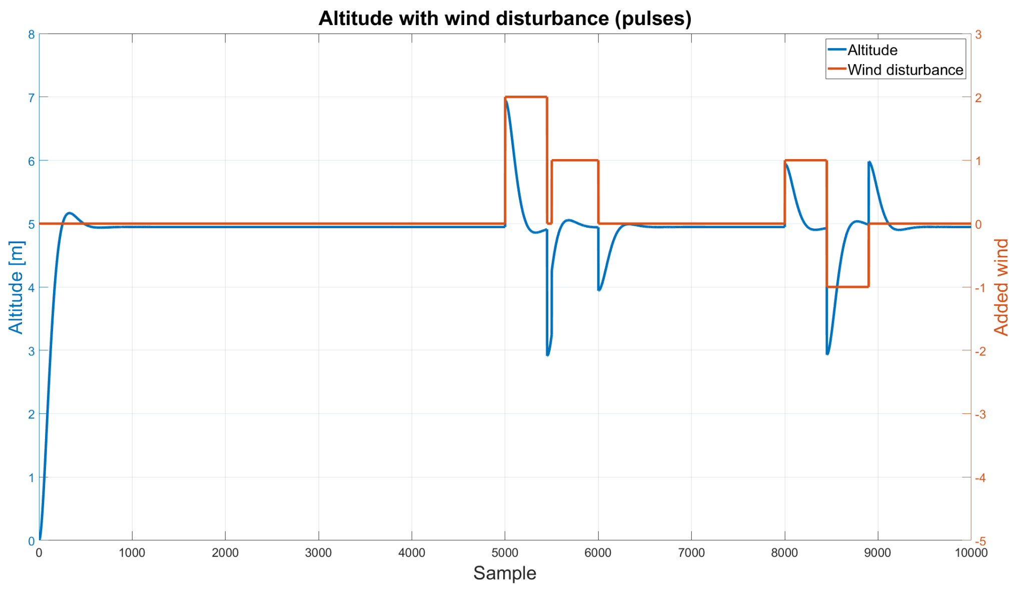


Figure 4.9: The altitude when it is disturbed by several pulses.

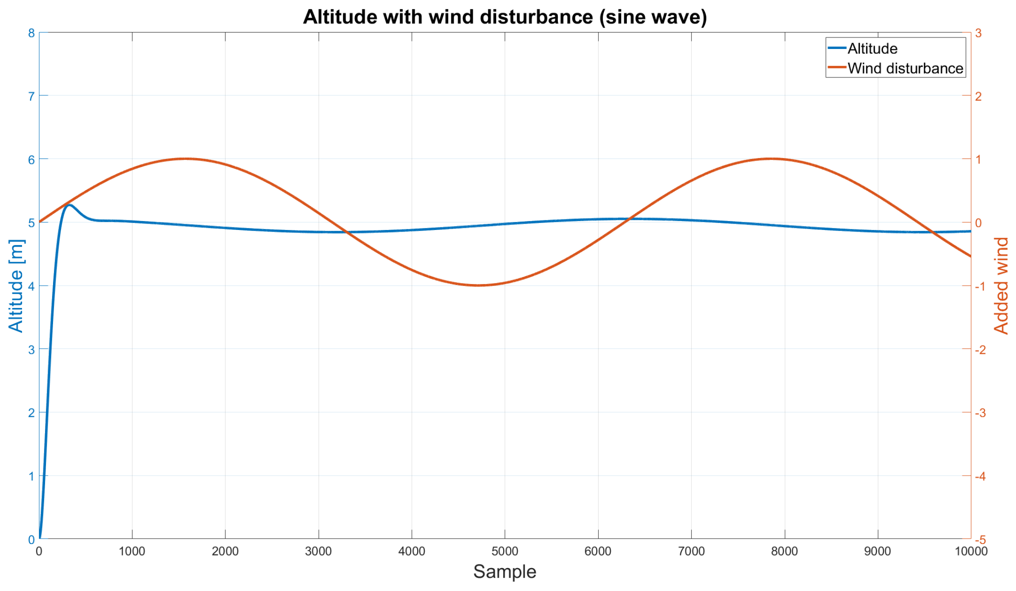


Figure 4.10: The altitude when it is disturbed by a sine wave.

4.3 Model parameters

Another method to analyze the system involves changing model parameters, such as mass and inertia, to observe their impact on the system's performance. Table 4.5 shows the system's performance as the mass varies, with the original mass being 0.547 kg. All other parameters remain constant, with the values for $eq_{i,j}$ being consistent with those presented in Table 4.1 for *Run 4*. Tables 4.6-4.7 present the system's performance when the inertia varies. The original inertia values are $I_x = 3.3 \cdot 10^{-3}$, $I_y = 3.3 \cdot 10^{-3}$ and $I_z = 5.8 \cdot 10^{-3}\ kg \cdot m^2$.

	m [kg]	Rise time [s]	Overshoot [%]	Undershoot [%]	Steady-state error
Run 1	0.2	0.09376	4.737	1.783	0.02
Run 2	0.4	0.13306	4.737	1.999	0.039
Run 3	0.6	0.16323	4.737	1.886	0.059
Run 4	0.8	0.18865	4.737	1.953	0.078
Run 5	1.5	0.25875	4.737	1.964	0.147

Table 4.5: Performance parameters when the quadcopters mass varies.

	Run 1	Run 2	Run 3
Inertia, I_x, I_y $[kg \cdot m^2]$	1.5 $\cdot 10^{-5}$	1.5 $\cdot 10^{-4}$	1.5 $\cdot 10^{-2}$
Inertia, I_z $[kg \cdot m^2]$	5.8 $\cdot 10^{-3}$	5.8 $\cdot 10^{-3}$	5.8 $\cdot 10^{-3}$
Rise time [s]	1.529	0.1558	0.1558
Overshoot [%]	0.505	4.737	4.737
Undershoot [%]	2.0	1.896	1.896
Steady-state error	0.053	0.054	0.054
Biggest (ϕ, θ, ψ) [°]	(7.7, -1.1, -0.07)	(9.7e-6, 5.6e-8, 1.2e-11)	(-1.1e-12, 8.8e-13, 2.8e-12)
Final (x, y, z) [m]	(-2, 5.6, 4.947)	(-6.4e-7, 1.1e-4 , 4.946)	(-5e-11, -6.1e-11, 4.946)

Table 4.6: Performance parameters when the quadcopters inertia (I_x, I_y) varies.

	Run 4	Run 5	Run 6
Inertia, I_x, I_y $[kg \cdot m^2]$	3.3 $\cdot 10^{-3}$	3.3 $\cdot 10^{-3}$	3.3 $\cdot 10^{-3}$
Inertia, I_z $[kg \cdot m^2]$	1.5 $\cdot 10^{-5}$	1.5 $\cdot 10^{-2}$	1.5
Rise time [s]	0.1558	0.1558	0.1558
Overshoot [%]	4.737	4.737	4.737
Undershoot [%]	1.896	1.896	1.896
Steady-state error	0.054	0.054	0.054
Biggest (ϕ, θ, ψ) [°]	(-6.3e-12, -5.8e-10, 3.2e-10)	(1e-11, 7.7e-11, -2e-12)	(3.9e-11, -3.8e-11, -6.4e-14)
Final (x, y, z) [m]	(1.9e-8, -2.1e-10, 4.946)	(-2.6e-9, 3.4e-10, 4.946)	(1.3e-9, 1.3e-9, 4.946)

Table 4.7: Performance parameters when the quadcopters inertia (I_z) varies.

5 Discussion

Tuning the LQR controller is challenging, even for just a diagonal Q matrix. For simple hovering, setting element $eq_{3,3}$ to a high value and other elements $eq_{i,j}$ to a low value (as shown in Table 4.1) is favorable. By further tuning diagonal elements, positional control can be achieved. Table 4.3 demonstrates that with elevated values for $eq_{1,1}$, $eq_{2,2}$, and $eq_{3,3}$ and 1000 times smaller values for the remaining diagonal Q weights ($eq_{i,j}$), a final position fairly close to the desired one is attained, even with significant initial oscillations. However, the simulation duration of 50 seconds is lengthy for achieving a position that deviates from the target. This underscores the challenge of achieving precise positioning within a reasonable time frame. While the LQR excels in its primary function of hovering straight up, robustness testing reveals limitations in positional adjustments. Assuming that the linearization's small-angle approximation contributes to this behavior, it emphasizes the controller's primary focus on maintaining vertical stability rather than facilitating lateral movement.

The results indicate that for vertical hovering, emphasizing the weight element corresponding to the z-axis while keeping other elements low (as shown in Table 4.4) is optimal. Increasing $eq_{3,3}$ and decreasing the remaining $eq_{i,j}$ values generally improves rise time, undershoot, and steady-state error, at the expense of increased overshoot (though 4.737% overshoot can be deemed acceptable, depending on the situation). Figures 4.6-4.10 illustrate the quadcopter's rapid adjustment to wind disturbances, swiftly modulating its rotors RPM to counteract wind effects. Continuous wind inputs prompt adjustments, while gradual changes (like a sine wave) are handled adeptly, maintaining stable hovering.

Analysis of model parameters (Table 4.5) reveals that a smaller mass yields improved rise time, undershoot, and steady-state error, with overshoot unaffected. Varying inertia values (Tables 4.6-4.7) show that reducing I_x and I_y worsens rise time and undershoot while improving overshoot. Changes to I_z has minimal impact on system performance.

6 Conclusion and further work

The project demonstrates, among other things, how to derive the system equations for a quadcopter and successfully implement both a quadcopter model and a LQR controller in Simulink. The results show that good hovering straight up can be obtained, but changing the position is not optimal. This is mainly due to the system simplification and linearization, which assumes that the angles are zero. The LQR controller recovers well from vertical wind disturbances and makes sure to return to hovering at the desired position. Modification in model parameters show that the controller's robustness might be limited, as the system's performance may degrade when the system deviates from the nominal model.

The oscillations when attempting to hover at a different position are probably due to the simplified linearization assumptions not accurately capturing the dynamics of the system when it deviates from its nominal hovering position. When linearizing a system around a stationary point, the resulting linear model is valid only within a small range around that point. Deviations beyond this range can lead to inaccuracies in the linear model, causing instability. Also, the quadcopter's position, attitude (orientation), and rotor speeds are coupled. Linearizing around a hovering point with zero attitude angles assumes that attitude and position dynamics are decoupled. To further improve the system, different measures can be taken. Instead of relying solely on linear control techniques like LQR, consider nonlinear control approaches (such as model predictive control (MPC) or sliding mode control (SMC)), which can better handle the system's nonlinear dynamics. One can also linearize the system around multiple operating points to create a linearized region.

As previously discussed, fine-tuning the LQR controller is a complex and time-consuming process. For greater control, one can adjust non-diagonal elements in Q and R matrices, effectively tuning the W_{tot} matrix. However, manually determining the values of all 256 elements (given W_{tot}'s 16×16 size) can be daunting. To streamline this task, developing a code that automates the iterative tuning process would be beneficial. One can use optimization techniques, such as particle swarm optimization (PSO) or genetic algorithm (GA).

It is also important to note that the LQR controller uses a simplification of the model, which may result in suboptimal performance in real-world scenarios. While LQR offers theoretical advantages in terms of control stability and optimality, its effectiveness relies on the accuracy of the underlying model assumptions. To evaluate the performance of the LQR controller, tests were done in the Simulink/Matlab environment. However, it's essential to acknowledge that these simulations do not provide a comprehensive evaluation of the LQR controller's performance. This is because the simplified model used in the Simulink environment may not accurately represent the complexities and dynamics of real-world quadcopter systems.

References

[1] Britannica. control system. https://www.britannica.com/technology/control-system, Accessed 2024-02-06.

[2] Norman S. Nise. *Control systems engineering (8th edition)*. John Wiley & Sons Inc, 2019.

[3] Nationalencyklopedin; Folke Larsson. Programmerbart styrsystem. https://www.ne.se/uppslagsverk/encyklopedi/l%C3%A5ng/programmerbart-styrsystem, Accessed 2024-02-06.

[4] Vasile Prisacariu. The history and the evolution of uavs from the beginning till the 70s. https://www.researchgate.net/publication/371439698_The_history_and_the_evolution_of_UAVs_from_the_beginning_till_the_70s, Accessed 2024-02-07, 2017.

[5] Torkel Glad; Lennart Ljung. *Reglerteknik Grundläggande teori*. Studentlitteratur, 2006.

[6] Bernard Friedland; Jourdain Jourdain. *Control System Design: An Introduction to State-Space Methods*. Dover Publications, 2005.

[7] MATLAB. What is linear quadratic regulator (lqr) optimal control? | state space, part 4c. https://www.youtube.com/watch?v=E_RDCFOlJx4, 2019.

[8] Brian L. Stevens; Frank L. Lewis; Eric N. Johnson. *Aircraft control and simulation: Dynamics, controls design, and autonomous systems, 3rd edition*. John Wiley & Sons Inc, 2016.

[9] Wei Dong; Guoying Gu; Xiangyang Zhu; Han Ding. Development of a quadrotor test bed - modelling, parameter identification, controller design and trajectory generation. https://www.researchgate.net/publication/272366812_Development_of_a_Quadrotor_Test_Bed_-_Modelling_Parameter_Identification_Controller_Design_and_Trajectory_Generation, Accessed 2024-03-27.

[10] MATLAB. Accelerating the pace of engineering and science. https://se.mathworks.com/company.html?s_tid=hp_ff_a_company, Accessed 2024-04-14.

A Appendix

A.1 Derivation of equations [$\ddot{x}\ \ddot{y}\ \ddot{z}$]

The expressions for [$\ddot{x}\ \ddot{y}\ \ddot{z}$] are not needed in this project when creating system equations for a model where LQR is implemented. However, it may be needed in the future, hence here is the derivation of it. Using Newton's motion equation (which is Equation (2.14)) and the rotation matrix (Equation (2.19)) gives

$$m\begin{bmatrix}\ddot{x}\\ \ddot{y}\\ \ddot{z}\end{bmatrix} = \begin{bmatrix}0\\ 0\\ -mg\end{bmatrix} + R_b^e\begin{bmatrix}0\\ 0\\ \sum F_i\end{bmatrix} + F_d \tag{A.1}$$

$$(\,compare\ \ with\ \ ma = F\,)\,,$$

where m is the mass of the quadcopter, g is standard acceleration of gravity, F_d is the force caused by air resistance (also called drag force) and the equation for it is given by

$$F_d = \frac{1}{2}\rho v^2 C_d A\,, \tag{A.2}$$

where ρ is the density of the fluid, v is the speed of the object relative to the fluid, A is the cross sectional area, C_d is the drag coefficient, which depends on the shape of the object and Reynolds number. The velocity and cross sectional area are vectors: $v = [\dot{x}\ \ \dot{y}\ \ \dot{z}]^T$ and $A = [A_x\ \ A_y\ \ A_z]^T$.

By incorporating the transformation matrix from Equation (2.19) into Equation (A.1), which describes the relationship between linear acceleration and forces, one obtains

$$m\begin{bmatrix}\ddot{x}\\ \ddot{y}\\ \ddot{z}\end{bmatrix} = \begin{bmatrix}0\\ 0\\ -mg\end{bmatrix} + \begin{bmatrix}c(\psi)c(\theta) & c(\psi)s(\theta)s(\phi) - c(\phi)s(\psi) & c(\psi)c(\phi)s(\theta) + s(\psi)s(\phi)\\ c(\theta)s(\psi) & s(\psi)s(\theta)s(\phi) + c(\psi)c(\phi) & c(\phi)s(\psi)s(\theta) - c(\psi)s(\phi)\\ -s(\theta) & c(\theta)s(\phi) & c(\theta)c(\phi)\end{bmatrix}\begin{bmatrix}0\\ 0\\ \sum F_i\end{bmatrix} + F_d\,. \tag{A.3}$$

Simplifying the equation leads to

$$\begin{bmatrix}\ddot{x}\\ \ddot{y}\\ \ddot{z}\end{bmatrix} = \begin{bmatrix}0\\ 0\\ -mg\end{bmatrix}\frac{1}{m} + \begin{bmatrix}(c(\psi)c(\phi)s(\theta) + s(\psi)s(\phi))\cdot k_F(\omega_1+\omega_2+\omega_3+\omega_4)\\ (c(\phi)s(\psi)s(\theta) - c(\psi)s(\phi))\cdot k_F(\omega_1+\omega_2+\omega_3+\omega_4)\\ (c(\theta)c(\phi))\cdot k_F(\omega_1+\omega_2+\omega_3+\omega_4)\end{bmatrix}\frac{1}{m} + \left(\tfrac{1}{2}\rho v^2 C_d A\right)\frac{1}{m}\,. \tag{A.4}$$

This results in the following equations (expressed in terms of control signals U (2.13) and with $c_i = \frac{1}{2m}\rho C_d A_i$)

$$\begin{cases} \ddot{x} = (c(\psi)c(\phi)s(\theta) + s(\psi)s(\phi))\,\dfrac{U_1}{m} - v_x^2 c_x\,, \\ \ddot{y} = (c(\phi)s(\psi)s(\theta) - c(\psi)s(\phi))\,\dfrac{U_1}{m} - v_y^2 c_y\,, \\ \ddot{z} = -g + (c(\theta)c(\phi))\,\dfrac{U_1}{m} - v_z^2 c_z\,. \end{cases} \tag{A.5}$$

A.2 Oscillations in Quadcopter block

One issue encountered with the *Quadcopter* block was the occurrence of excessively large positional values (x, y, z). (When testing the *Quadcopter* block with $\omega_1 = \omega_4 = 3000$ rad/s and $\omega_2 = \omega_3 = 3020$ rad/s). Upon debugging, the system equations were analyzed to identify potential causes. When plotting the state values, oscillations occurred in the linear velocity (u, v, w), which likely contributed to the enlargement of the positional values (since these depend on u, v, w). The equations governing the linear velocity were examined in search for errors in signs or trigonometric functions (cos, sin), but none were found. However, reducing the step size in Simulink alleviated the issue. While some oscillations persist, they are significantly diminished. I transitioned the ODE solver to *ode4* and set the step size to 0.001 in Simulink, whereas initially, it was set to auto (*ode3*) with a step size of 0.1. The figures below depict the remaining oscillations.

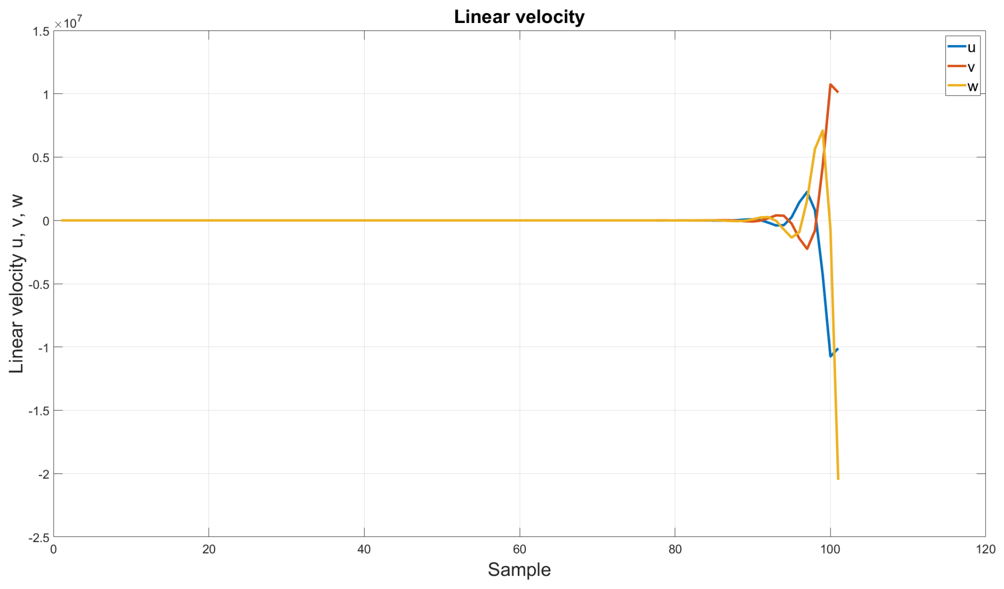


Figure A.1: The oscillations of the linear velocity when the step size is 0.1 (and ode3 is used).

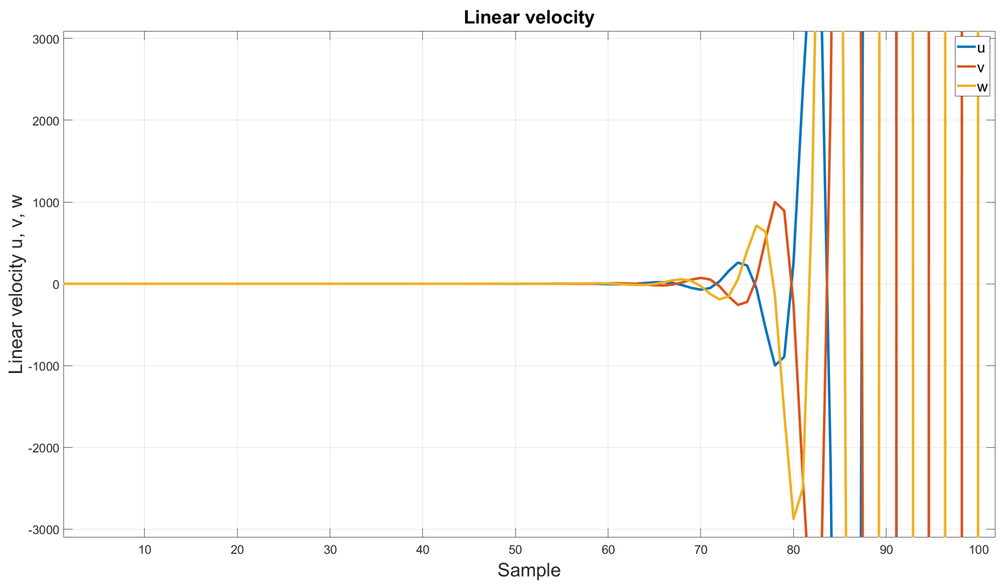


Figure A.2: The close up of oscillations of the linear velocity when the step size is 0.1 (and ode3 is used).

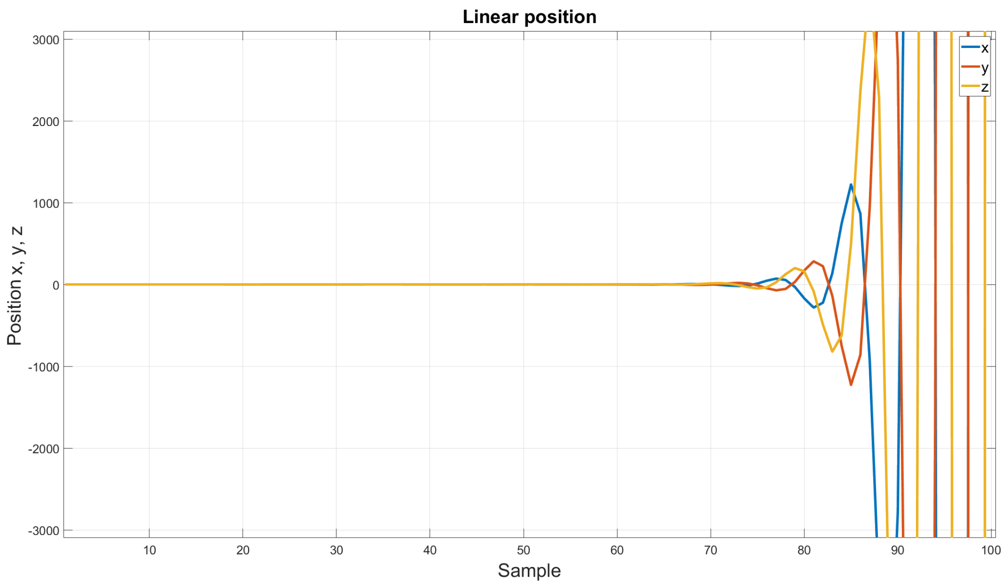


Figure A.3: The close up of oscillations of the position when the step size is 0.1 (and ode3 is used).

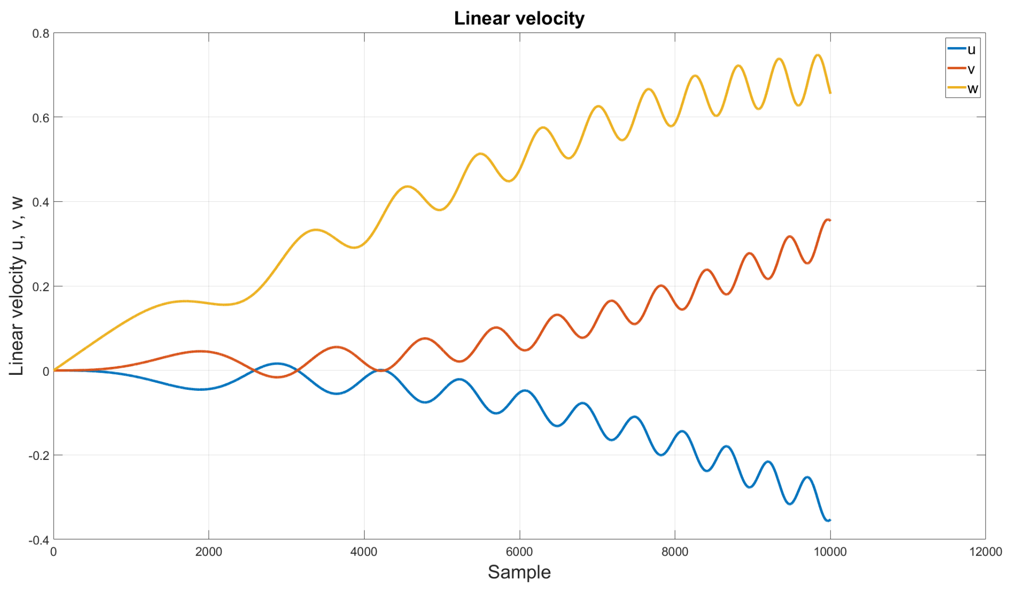


Figure A.4: The oscillations of the linear velocity when the step size is 0.001 (and ode4 is used).

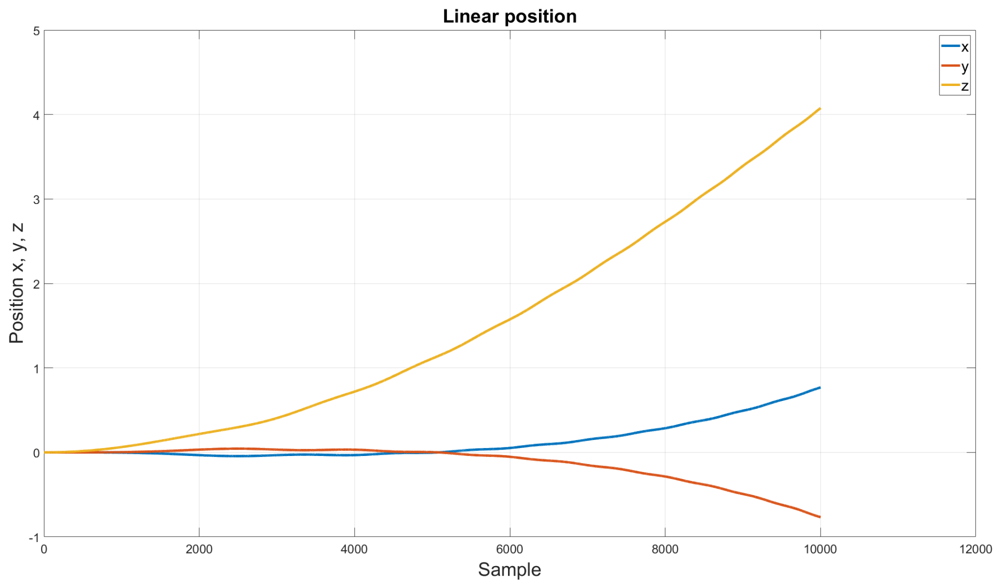


Figure A.5: The position when the step size is 0.001 (and ode4 is used).

The primary solution to the problem involved reducing the step size, which worked likely due to the highly nonlinear and dynamic nature of the system, necessitating fast changes in state values. A larger step size would fail to capture the system's behavior accurately, and an ODE solver must be able to accurately calculate the system's dynamics.

A.3 Attempt at tuning LQR for non-diagonal Q

Matrix Q_{tot} for a non-diagonal Q consists of a sum of *12 large terms*. Equation (A.6) shows the first three large terms and the last one.

$$
\begin{aligned}
Q_{tot} = & \left(x \cdot eq_{1,1} + y \cdot eq_{1,2} + z \cdot eq_{1,3} + \phi \cdot eq_{1,4} + \ \cdots \ + r \cdot eq_{1,12}\right) \cdot x \ + \\
& + \ \left(x \cdot eq_{2,1} + y \cdot eq_{2,2} + z \cdot eq_{2,3} + \phi \cdot eq_{2,4} + \ \cdots \ + r \cdot eq_{2,12}\right) \cdot y \ + \\
& + \ \left(x \cdot eq_{3,1} + y \cdot eq_{3,2} + z \cdot eq_{3,3} + \phi \cdot eq_{3,4} + \ \cdots \ + r \cdot eq_{3,12}\right) \cdot z \ + \\
& + \ \cdots\cdots \ + \\
& + \ \left(x \cdot eq_{12,1} + y \cdot eq_{12,2} + z \cdot eq_{12,3} + \phi \cdot eq_{12,4} + \ \cdots \ + r \cdot eq_{12,12}\right) \cdot r \, . \qquad \text{(A.6)}
\end{aligned}
$$

If one focuses solely on the state z, it can be observed that column three of matrix Q contains elements $eq_{i,j}$ ($eq_{1,3}$, $eq_{2,3}$, $eq_{3,3}$, ...), which are multiplied by the state z. Additionally, in the *12 large terms* (as shown in Equation (A.6)), these elements further interact with different states. Row three, on the other hand, multiplies every variable ($state \cdot eq_{i,j}$) with the state z. By assigning values to these elements $eq_{i,j}$, one determines the weight assigned to the correlations between state z and the other states. Rather than setting Q as diagonal, an alternative approach is to concentrate on column and row three of Q.

However, there are certain conditions when assigning values to the matrix elements. Both M_{tot} and Q must be positive semi-definite (symmetric with non-negative eigenvalues), while R must be positive definite (symmetric with positive eigenvalues). Thus, these conditions must be satisfied during the tuning process. A Matlab code is developed to verify compliance with these conditions. Symmetry can be assessed by comparing a matrix with its transpose, which should yield equality if the matrix is symmetric (for example $isQSymmetric = isequal(Q, Q');$). Additionally, the eigenvalues can be checked easily in Matlab (for example $isQPositiveSemiDefinite = all((eig(Q)) >= 0);$). While this verification is helpful, determining the matrix values remains a manual task and selecting elements for matrices of sizes 12×12, 12×4, and 4×4 is excessively time-consuming.

There are various optimization techniques that can be employed to tune the parameters of an LQR controller. One approach involves using biogeography-based optimization (BBO) to adjust the controller gains and another one called particle swarm optimization (PSO) can be used to optimize the weighing matrices of the controller through iterative search processes. Additional methods include genetic algorithm (GA), ant colony optimization (ACO), artificial bee colony optimization (ABC), simulated annealing (SA), and differential evolution (DE). The primary objective of these techniques is to enhance the dynamic performance of the LQR controller by effectively optimizing its parameters.