UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ETSI Aeroespacial y Diseño Industrial

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

School of Aerospace Engineering and Industrial Design

Design and Implementation of multivariable control strategies for autopilots based on the PX4 control stack.

End of Degree Project

Bachelor's Degree in Aerospace Engineering

AUTHOR: Iborra Lérida, Álex

Tutor: García-Nieto Rodríguez, Sergio

ACADEMIC YEAR: 2024/2025

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ETSI Aeroespacial y Diseño Industrial

Design and implementation of multivariable control strategies for autopilots based on the PX4 control stack

MEMORY

AUTHOR: Iborra Lérida, Alex

TUTOR: García-Nieto Rodríguez, Sergio

Universitat Politècnica de València

Escuela Técnica Superior de Ingeniería Aeroespacial y de Diseño Industrial

Grado en Ingeniería Aeroespacial

2024-2025 Course

Abstract

This work lays the groundwork for experimental control algorithm implementations in the PX4 stack, focusing on the design and integration of an LQR (Linear Quadratic Regulator) control algorithm tailored for the H200 aircraft dynamic model, developed by the HORUS UPV project.

The LQR algorithm is selected due to its lower implementation complexity with the objective to compare it with the default PID controller. LQR demonstrated smoother trajectory rejection and better error-based indices for pitch control, although it requires further refinement in roll error rejection. Regarding trajectory error rejection, the performance is similar to PID implementation. The PX4 stack is modified using C++ to integrate the experimental control, enabling a calibration process via empirical methods to achieve optimal performance.

Software-in-the-loop (SIL) simulations are conducted, combining the dynamic model with a ground control station software, to evaluate the algorithm's effectiveness. For autonomous flight, the control mode allows switching between the standard PID controller of the commercial flight control unit and the experimental LQR implementation.

Resumen

Este trabajo sienta las bases para la implementación experimental de algoritmos de control en la plataforma PX4, centrándose en el diseño e integración de un algoritmo de control LQR (Regulador Cuadrático Lineal) adaptado al modelo dinámico del avión H200, desarrollado por el proyecto HORUS UPV.

El algoritmo LQR se selecciona debido a su menor complejidad de implementación con el objetivo de comparar el rendimiento con el controlador PID predeterminado. El LQR demostró un rechazo de trayectorias más suave y mejores índices basados en el error para el control de pitch, aunque requiere más refinamiento en el rechazo de error de alabeo. En cuanto al seguimiento de trayectorias, el rendimiento es similar a la implementación de PID. La plataforma PX4 se modifica en lenguaje C++ para integrar el control experimental, lo que permite un proceso de calibración mediante métodos empíricos para lograr un rendimiento óptimo.

Se realizan simulaciones software-in-the-loop (SIL) que combinan el modelo dinámico con un software de estación de control en tierra para evaluar la eficacia del algoritmo. Para el vuelo autónomo, el modo de control permite alternar entre el controlador PID estándar de la unidad de control comercial y la implementación experimental de LQR.

Resum

Aquest treball estableix les bases per a la implementació experimental d'algorismes de control en la plataforma PX4, centrant-se en el disseny i integració d'un algorisme de control LQR (Regulador Quadràtic Lineal) adaptat al model dinàmic de l'avió H200, desenvolupat pel projecte HORUS UPV.

L'algorisme LQR es selecciona per la seua menor complexitat d'implementació amb l'objectiu de comparar-lo amb el controlador PID predeterminat. El LQR va demostrar una rejeció de trajectòries més suau i millors índexs basats en l'error per al control de l'elevació, tot i que requereix més refinament en la rejeció de l'error de gir. En quant a l'error del seguiment de trajectòries, el rendiment es similar a la implementació de PID. La plataforma PX4 es modifica en llenguatge C++ per a integrar el control experimental, cosa que permet un procés de calibratge mitjançant mètodes empírics per a aconseguir un rendiment òptim.

Es realitzen simulacions software-in-the-loop (SIL) que combinen el model dinàmic amb un programari d'estació de control en terra per a avaluar l'eficàcia de l'algorisme. Per al vol autònom, el mode de control permet alternar entre el controlador PID estàndard de la unitat de control comercial i la implementació experimental de LQR.

Contents

List of Figures

List of Tables

Listings

List of Symbols

X_B	Body Axes reference system X axis
X_W	Wind Axes reference system X axis
X_E	Earth Axes reference system X axis
Y_B	Body Axes reference system Y axis
Y_W	Wind Axes reference system Y axis
Y_E	Earth Axes reference system Y axis
Z_B	Body Axes reference system Z axis
Z_W	Wind Axes reference system Z axis
Z_E	Earth Axes reference system Z axis
ϕ	Roll angle
θ	Pitch angle
ψ	Yaw angle
I_{xx}	Inertia moment of X axis
I_{yy}	Inertia moment of Y axis
I_{zz}	Inertia moment of Z axis
I_{xy}	Product of inertia XY
I_{yx}	Product of inertia YX
I_{xz}	Product of inertia XZ
I_{zx}	Product of inertia ZX
I_{yz}	Product of inertia YZ
I_{zy}	Product of inertia ZY
u	Linear longitudinal velocity in Body Axes reference system
v	Linear lateral velocity in Body Axes reference system
w	Linear vertical velocity in Body Axes reference system
p	Roll angular velocity
q	Pitch angular velocity
r	Yaw angular velocity
X	Resultant force in the X axis
Y	Resultant force in the Y axis
Z	Resultant force in the Z axis
L	Roll moment
M	Pitch moment
N	Yaw moment
L_A	Roll moment due to aerodynamics

M_A	Pitch moment due to aerodynamics
N_A	Yaw moment due to aerodynamics
x	Coordinate x of Earth Axes reference system
y	Coordinate y of Earth Axes reference system
z	Coordinate z of Earth Axes reference system
$\vec{F}$	Force vector
m	Aircraft's mass
t	Time
$\vec{V}_{ABS}$	Absolute velocity vector
$\vec{i}$	Director vector of X axis
$\vec{j}$	Director vector of Y axis
$\vec{k}$	Director vector of Z axis
F_{TX}, T_X	Thrust force in X axis
F_{TY}, T_Y	Thrust force in Y axis
F_{TZ}, T_Z	Thrust force in Z axis
F_{AX}, X_A	Aerodynamic force in X axis
F_{AY}, Y_A	Aerodynamic force in Y axis
F_{AZ}, X_Z	Aerodynamic force in Z axis
F_{GX}	Gravitational force in X axis
F_{GY}	Gravitational force in Y axis
F_{GZ}	Gravitational force in Z axis
ρ	Density of the atmosphere
g	Earth's gravity constant
V	Aerodynamic velocity
S_W	Aircraft's wing surface
$\vec{M}$	Moment vector
$\vec{H}_{ABS}$	Absolute angular momentum vector
x	State vector
u	Input vector
y	Output vector
A	State matrix
B	Input matrix
C	Output matrix
D	Feedthrough matrix
K, K_x	State feedback gain matrix
$\mathcal{C}$	Controlability matrix
J	LQR cost function
Q	State deviation weighting matrix
R	Control effort weighting matrix
P	Solution to Riccati equation
e	Error between reference and output
r	Reference
z	Integral of error over time
x_a	Augmented state vector
Q_a	Augmented state deviation weighting matrix
Q_z	Integral error weighting matrix

K_a	Augmented state feedback gain matrix
K_i	Integral state feedback gain matrix
α	Angle of attack
β	Drift angle
δ_P	Thrust lever position
δ_A	Aileron deflection
δ_E	Elevator deflection
δ_R	Rudder deflection
h	Height
X_u	Derivative of X axis aerodynamic force respect to X axis linear velocity
X_w	Derivative of X axis aerodynamic force respect to Z axis linear velocity
X_q	Derivative of X axis aerodynamic force respect to pitch angular velocity
$T_{X,\delta P}$	Derivative of X axis thrust force respect to thrust lever position
Y_v	Derivative of Y axis aerodynamic force respect to Y axis linear velocity
Y_p	Derivative of Y axis aerodynamic force respect to roll angular velocity
Y_r	Derivative of Y axis aerodynamic force respect to yaw angular velocity
$Y_{\delta A}$	Derivative of Y axis aerodynamic force respect to aileron deflection
$Y_{\delta R}$	Derivative of Y axis aerodynamic force respect to rudder deflection
Z_u	Derivative of Z axis aerodynamic force respect to X axis linear velocity
Z_w	Derivative of Z axis aerodynamic force respect to Z axis linear velocity
Z_q	Derivative of Z axis aerodynamic force respect to pitch angular velocity
$Z_{\dot{w}}$	Derivative of Z axis aerodynamic force respect to Z axis linear acceleration
$Z_{\delta E}$	Derivative of Z axis aerodynamic force respect to elevator deflection
L_v	Derivative of roll moment respect to Y axis linear velocity
L_p	Derivative of roll moment respect to roll angular velocity
L_r	Derivative of roll moment respect to yaw angular velocity
$L_{\delta A}$	Derivative of roll moment respect to aileron deflection
$L_{\delta R}$	Derivative of roll moment respect to rudder deflection

M_u	Derivative of pitch moment respect to X axis linear velocity
M_w	Derivative of pitch moment respect to Z axis linear velocity
M_q	Derivative of pitch moment respect to pitch angular velocity
$M_{\dot{w}}$	Derivative of pitch moment respect to Z axis linear acceleration
$M_{\delta E}$	Derivative of pitch moment respect to elevator deflection
N_v	Derivative of yaw moment respect to Y axis linear velocity
N_p	Derivative of yaw moment respect to roll angular velocity
N_r	Derivative of yaw moment respect to yaw angular velocity
$N_{\delta A}$	Derivative of yaw moment respect to aileron deflection
$N_{\delta R}$	Derivative of yaw moment respect to rudder deflection
M	Mach number
a	Speed of sound in air at determined conditions
L	Lift force
CL	Lift coefficient
D	Aerodynamic resistance force
CD	Aerodynamic resistance coefficient
c_W	Mean aerodynamic chord
b_W	Wingspan
C_X	Coefficient of aerodynamic force in X axis
C_Y	Coefficient of aerodynamic force in Y axis
C_Z	Coefficient of aerodynamic force in Y axis
C_l	Coefficient of aerodynamic roll moment
C_M	Coefficient of aerodynamic pitch moment
C_N	Coefficient of aerodynamic yaw moment
C_{Y_β}	Derivative of aerodynamic lateral force coefficient with respect to angle of drift
$C_{Y_{\delta_A}}$	Derivative of aerodynamic lateral force coefficient with respect to aileron deflection
$C_{Y_{\delta_R}}$	Derivative of aerodynamic lateral force coefficient with respect to rudder deflection
$C_{Y_{\dot{\beta}}}$	Derivative of aerodynamic lateral force coefficient with respect to the derivative of angle of drift over time
C_{Y_p}	Derivative of aerodynamic lateral force coefficient with respect to roll angular velocity
C_{Y_r}	Derivative of aerodynamic lateral force coefficient with respect to yaw angular velocity

C_{l_β}	Derivative of aerodynamic roll moment coefficient with respect to angle of drift
$C_{l_{\delta_A}}$	Derivative of aerodynamic roll moment coefficient with respect to aileron deflection
$C_{l_{\delta_R}}$	Derivative of aerodynamic roll moment coefficient with respect to rudder deflection
$C_{l_{\dot{\beta}}}$	Derivative of aerodynamic roll moment coefficient with respect to the derivative of angle of drift over time
C_{l_p}	Derivative of aerodynamic roll moment coefficient with respect to roll angular velocity
C_{l_r}	Derivative of aerodynamic roll moment coefficient with respect to yaw angular velocity
C_{M_0}	Coefficient of aerodynamic pitch moment with no angle of attack
C_{M_α}	Derivative of aerodynamic pitch moment coefficient with respect to angle of attack
$C_{M_{\delta_E}}$	Derivative of aerodynamic pitch moment coefficient with respect to elevator deflection
$C_{M_{\dot{\alpha}}}$	Derivative of aerodynamic pitch moment coefficient with respect to the derivative of angle of attack over time
C_{M_q}	Derivative of aerodynamic pitch moment coefficient with respect to pitch angular velocity
C_{N_β}	Derivative of aerodynamic yaw moment coefficient with respect to angle of drift
$C_{N_{\delta_A}}$	Derivative of aerodynamic yaw moment coefficient with respect to aileron deflection
$C_{N_{\delta_R}}$	Derivative of aerodynamic yaw moment coefficient with respect to rudder deflection
$C_{N_{\dot{\beta}}}$	Derivative of aerodynamic yaw moment coefficient with respect to the derivative of angle of drift over time
C_{N_p}	Derivative of aerodynamic yaw moment coefficient with respect to roll angular velocity
C_{N_r}	Derivative of aerodynamic yaw moment coefficient with respect to yaw angular velocity
C_{L_0}	Coefficient of lift force with no angle of attack
C_{L_α}	Derivative of lift coefficient with respect to angle of attack
$C_{L_{\delta_E}}$	Derivative of lift coefficient with respect to elevator deflection
$C_{L_{\dot{\alpha}}}$	Derivative of lift coefficient with respect to the derivative of angle of attack over time
C_{L_q}	Derivative of lift coefficient with respect to pitch angular velocity

C_{D_0}	Coefficient of drag force with no angle of attack or parasitic drag
C_T	Thrust coefficient
n	Blade angular velocity
D	Blade diameter
P	Power
C_P	Power coefficient
J	Non-dimensional induced velocity
L_elev	Elevator arm
L_prop	Propeller arm
IAE	Integral of Absolute Error
ISE	Integral of Squared Error
$ITAE$	Integral of Time-weighted Absolute Error
$RMSE$	Root Mean Square Error

Chapter 1

Introduction

1.1 Project definition

This project is based on H200 prototype, developed by HORUS UPV from Universitat Politècnica de València (UPV). This is a university project group that involves students and professors from different disciplines and is related to designing and manufacturing unmanned aircaft prototypes. In addition, the Instituto Universitario de Automática e Informática Industrial (ai2), and Clean Mobility and Transport (CMT), both also from UPV, are responsible for the project.

The motivation of it is to create a test bench for the research of hybrid propulsion systems. In this case, by using LiPo batteries and hydrogen propulsion. The objective is to exceed the current endurance of alternative propulsed aircraft in typical operations, which is around 45 minutes.

1.1.1 H200 platform

Conceptually, [1][2] this aircraft is based on a semimonocoque structure with CFRP frames and foam nucleus, with CFRP skin. Its main wing is suited with SD7062 profile, with a wingspan of 2.95 meters and a wing surface of around 1 m^2. For the tail wing, a V style wing is selected to increase surface. The maneuvers are possible with two ailerons and two flaps in the main wing, and two elevators-rudders in the tail wing. All of this yields a mass of 15 kg, thanks to the use of CFRP. A picture of the aircraft can be seen in Figure 1.1

Figure 1.1: H200 aircraft. Source: [3]

It incorporates a commercial module that includes the PX4 stack to control it. The predetermined control technique is based on PID control. There is the need to experiment with more advanced control techniques inside this stack. The project is oriented to address this necessity.

1.2 Unmanned Aircraft

In this section a description of UA (Unmanned Aircraft) is given. In addition, its history is revised to justify the different applications that emerged thanks to the historic context of each era. Finally, an enumeration of the different applications of this technology is presented.

1.2.1 Description

The *Global Air Traffic Management Operational Concept* from ICAO (International Civil Aviation Organization) states: "An unmanned aerial vehicle is a pilotless aircraft, in the sense of Article 8 of the Convention on International Civil Aviation, which is flown without a pilot-in-command on-board and is either remotely and fully controlled from another place (ground, another aircraft, space) or programmed and fully autonomous"[4]. Moreover, FAA (Federal Aviation Administration) defines it as the following: "It is defined by statute as an aircraft that is operated without the possibility of direct human intervention from within or on the aircraft."[5]. ICAO also defines in its regulations some terms regarding unmanned aircraft:

- Remotely-piloted aircraft: An aircraft where the flying pilot is not on board the aircraft. This is a subcategory of unmanned aircraft.
- Remotely-piloted aircraft system: A set of configurable elements consisting of a remotely-piloted aircraft, its associated remote pilot station(s), the required command and control links and any other system elements as may be required, at any point during flight operation.
- Unmanned aircraft system: An aircraft and its associated elements which are operated with no pilot on board. It is the term that substitutes the obsolete Unmanned Aerial Vehicles (UAV).

1.2.2 History

UAS (Unmanned Aircraft Systems), commonly known as drones, have a history that spans over a century. They evolved from simple, rudimentary designs to sophisticated, high-tech machines capable of various tasks. These range from military operations to commercial applications. The concept of UASs dates back to the early 20th century.

The first recorded instance of an unmanned vehicle was the Kettering Bug as seen in Figure 1.2, developed during World War I (1918) by Charles Kettering. It was an aerial torpedo meant to target enemy positions by using pre-set coordinates. It had a wingspan of 15 feet and a range of 75 miles. Powered by a simple four-cylinder engine, it could carry 180 pounds of explosives. Launched from a track, it used an on board guidance system to reach its target. Once the engine stopped, the wings detached, causing the Bug to fall onto the target.

Figure 1.2: Kettering Bug. Source: [6]

Later, between the World Wars, UASs saw significant advancements. However, it was during World War II that the development of this technology accelerated. As seen in Figure 1.3, the OQ-2 Radioplane was developed by Reginald Denny in 1939 to be mass-produced. It served as a tool for training anti-aircraft gunners. With 12 foot wingspan, it had a two-stroke engine capable of propelling the aircraft up to 85 mph.

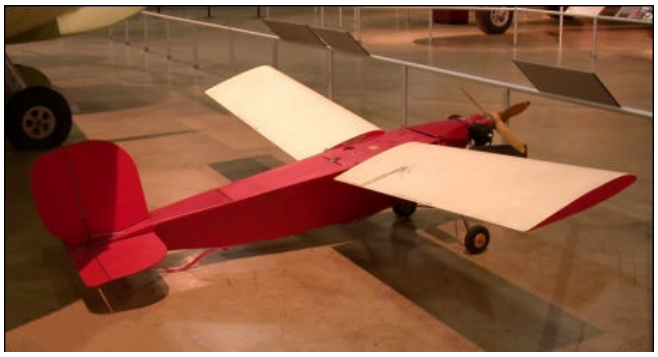


Figure 1.3: OQ-2A Radiocontrol. Source: [7]

During the Second World War, specifically in 1944, the Nazi Germany developed the V-1 Flying Bomb, seen in Figure 1.4. It was a jet-powered cruise missile with a range of 150 miles that can carry a 1,870 pound warhead. Once launched from a ground-based ramp or aircraft, a simple autopilot system was the responsible for guidance. It targeted London and other strategic locations, becoming one of the first UASs used for offensive purposes.

Figure 1.4: V-1 Flying Bomb. Source: [8]

Fast-forwarding to the Cold War era, the development of more sophisticated UASs was driven by the need for reconnaissance and surveillance without risking human lives. Ryan Aeronautical Company developed in 1951 the Ryan Firebee, seen in Figure 1.5. This was a drone with 1,400 miles range and speed of 700 mph. It was launched from ground or air and recovered via parachute and inspired the reconnaissance drones during this era.

Figure 1.5: Ryan Firebee. Source: [9]

During the 60s, CIA developed Project AQUILINE, a strategic reconnaissance UAS that resembled to a bird for covert operations. It can be seen in Figure 1.6 [10]. Designed to be stealthy, they were equipped with advanced guidance systems and provided long endurance. Although this project was eventually cancelled, it proved the potential of UASs for intelligence missions.

Figure 1.6: McDonnell Douglas Aquiline. Source: [11]

The latter part of the 20th century and the early 21st century saw a rapid expansion in UAV technology, driven by advancements in electronics, GPS, and materials science. An example of these advancements is the Northrop Grumman RQ-4 Global Hawk, seen in Figure 1.7. This is a high-altitude, long-endurance reconnaissance UAS, capable of flying at altitudes of 18,000 m for over 30 hours. Due to its advanced radar and imaging equipment it was extensively used in conflict areas to provide invaluable intelligence and battlefield awareness.

Figure 1.7: Northrop Grumman RQ-4 Global Hawk. Source: [12]

Finally, UASs have found applications beyond the military, including commercial, scientific, and recreational use. With the latter purpose, it can be seen in Figure 1.8 [13] the DJI Phantom, a consumer-grade aerial photography and videography drone. It uses technology such as GPS stabilization to provide intuitive controls. Its beginner friendly design philosophy sprang its popularity among hobbyists and professionals, and contributed with other commercial drones to make the UAS concept widespread among society. It proved useful for applications different from their initial intention, such as search and rescue.

Figure 1.8: DJI Phantom series. Source: [13]

The history of UASs is a testament to the relentless pursuit of technological advancement. From the early prototypes like the Kettering Bug to the sophisticated drones of today, they have become essential tools in modern society. As technology continues to evolve, the future of UASs promises greater capabilities and applications.

1.2.3 Applications

As seen in the previous section, this technology has many applications derived from the context when they were developed. In this section some examples are presented.

- Military and Defense: They can be used for Reconnaissance and Surveillance, monitoring enemy movements without risking human lives. In addition,armed ones are deployed for precise attacks on enemy targets, reducing collateral damage and minimizing risks to human soldiers. They also are utilized to transport supplies, ammunition, and medical equipment to troops in remote or hostile areas.

- Agriculture: Equipped with sensors and cameras they can monitor crop health, detect diseases, assess soil conditions. In summary they collect useful data to make informed decisions. They can also take action by applying fertilizers, pesticides or planting seeds.

- Rescue Operations: They provide real-time information during natural disasters such as floods, hurricanes, and earthquakes. They also help coordinate rescue operations and deliver supplies to affected areas.

- Law Enforcement and Public Safety: Responsible for the aerial surveillance for large public events, protests, or emergency situations. They help law enforcement monitor crowds, detect threats, and ensure public safety.

- Media and Entertainment: Used for film production, news coverage, and commercial photography due to their unique image perspectives. Also, drone light shows are becoming popular.

- Commercial and Industrial Applications: They are attractive for delivery industry, especially in urban areas. They can also be used for detailed map generation.

Chapter 2

Objectives and project scope

2.1 Project objective

The main goal of this project is to develop an experimental control implementation that can coexist with the regular autopilot implementation of the PX4 open-source software. This implementation should allow for switching between both control modes. In this case, the experimental controller will be based on an LQR (Linear Quadratic Regulator) algorithm. This setup should function during Software in the Loop (SITL) simulations using the H200 aircraft model.

2.2 Project scope

Different milestones will be set to assess the accomplishment of the project's main objective:

1. Adaptation of PX4 code to be suited for control mode switchability.
2. Comprobation of correct behaviour of the regular implementation and of control mode switchability.
3. Development of experimental LQR code.
4. Linearization of Bryan equations with aircraft dynamic model around an operating point.
5. Q and R matrices assignment.
6. Riccati's equation calculation.
7. LQR parameters insertion and simulation with H200 model.
8. Validation of experimental controller performance.
9. Comparison with original PID implementation.

Once the performance of the LQR controller implementation is assessed and compared with the original implementation, the project will be considered to be complete.

Chapter 3

Process description

HORUS project confronts different topics around the flight of an autonomous vehicle propelled by hydrogen. A general scheme of the system elements and examples of the knowledge areas they involve can be seen in Figure 3.1.

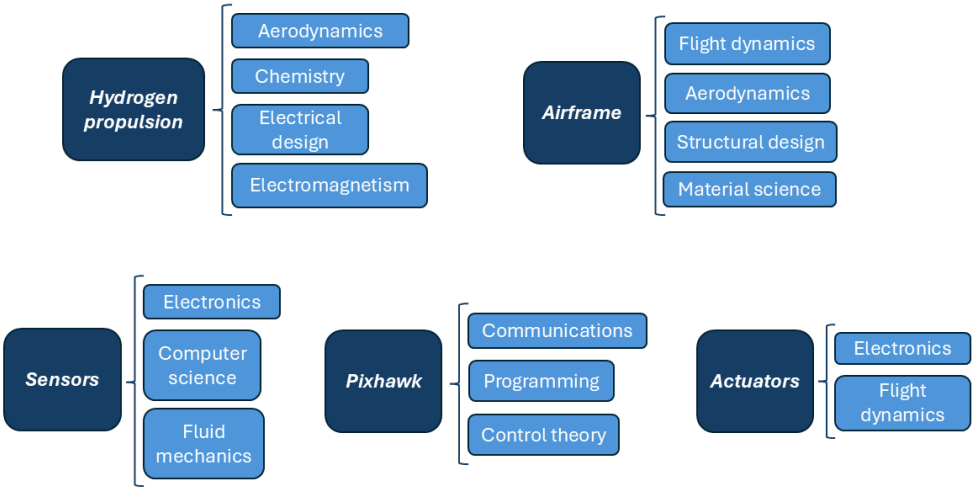


Figure 3.1: Scheme of the HORUS project. Source: own

Many of these lie outside project's scope. It will focus on the equation modelling of the airframe and the modification of the PX4 software to include different experimental control versions. These will be introduced along this chapter.

3.1 Fixed-Wing aircraft model

Designing the controller for a fixed-wing aircraft requires to mathematically model it. The model used for the representation in this case is the one created by George Hartley Bryan in 1911, known as Bryan's equations.

To understand these equations, an introduction is given into the different concepts that are needed.

3.1.1 Reference system

A reference system is needed to define aircraft position, velocities and accelerations. It consists of an orthogonal trihedral that has its origin set differently depending on the application. In the case of this work, by considering the assumption of little perturbations, the equations will be described by using Body Axes. However, as it will be seen in the following section, other reference systems are needed to fully represent an aircraft. It is assumed that the aircraft has a symmetry plane that goes from the nose to the back of the aircraft.

- Body Axes reference system: The origin of this system is in the center of gravity of the aircraft, and moves and rotates as the aircraft does.
  - X_B axis: Represents the longitudinal axis of the aircraft. It is directed towards the nose and is included in the symmetry plane.
  - Z_B axis: Perpendicular to the X_B axis and included in the symmetry plane. It points towards the bottom.
  - Y_B axis: Perpendicular to the symmetry plane of the aircraft and directed towards the right of the pilot. It is perpendicular to the aforementioned axes.
- Wind Axes reference system: This system is related to the aerodynamic velocity, with its origin at the center of gravity.
  - X_W axis: Identical to the aerodynamic velocity vector.
  - Z_W axis: Included in the symmetry plane and is perpendicular to X_W, pointed at the bottom of the aircraft.
  - Y_W axis: Perpendicular to the aforementioned axes.
- Earth Axes reference system: It is an inertial reference system respect to which the center of gravity will be represented. Its origin is settled in the Earth surface.
  - X_E axis: Directed to the North.
  - Z_E axis: Directed to the Earth's center.
  - Y_E axis: Directed to the East.
- Local Horizon Axes reference system: It is parallel to the Earth Axes reference system, but with the origin at the aircraft's center of gravity.

3.1.2 Aircraft attitude

To represent the aircraft the position of its center of gravity and its attitude is needed. There are different methods of obtaining the attitude, but in this case the Euler angles will be used.

Euler angles define the attitude of a non-inertial reference system with respect to an inertial one via rotations around the different axes. In this case, the inertial one is Earth Axes reference system, and the non-inertial the Body Axes reference system.

- Roll angle ϕ: Rotation angle around X_B axis.
- Pitch angle θ: Rotation angle around Y_B axis.
- Yaw angle (heading) ψ: Rotation angle around Z_B axis.

The transformation from one system to another has an specific sequence which is typically known as 3-2-1, which means that first the system is rotated around Z_B, then Y_B and finally X_B, represented as following: $(\psi \rightarrow \theta \rightarrow \phi)$. This yields the rotation matrix that if is multiplied by the Earth reference system, the Body reference system is obtained.

3.1.3 Bryan equations

Bryan equations is a set of twelve non-lineal coupled differential equations. They describe the behaviour of an aircraft considering it has six degrees of freedom: three rotations and three translations.

There is a list of assumptions that are considered to reduce the complexity of the model:

1. Aircraft is modelled as a rigid solid. This means it cannot be deformed, or two points of it always maintain their relative distance.
2. The aircraft has a symmetry plane that makes I_{xy} and I_{yx} become zero.
3. The rotation of the aircraft engine or propeller has no effect.
4. Mass change during a manoeuvre is neglected.
5. Translation equations are dependant on rotational equations.

The nomenclature used for the different variables of Bryan equations is represented in Figure 3.2 and collected in Table 3.1.

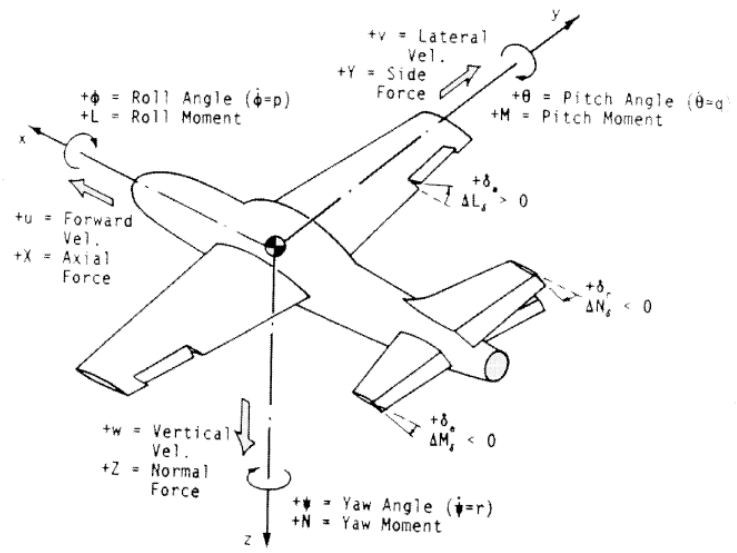


Figure 3.2: Representation of axes with used nomenclature. Source: [14]

	Angular and linear velocities	Forces and moments	Angles and distances
Forward (x axis)	u	X	x
Side (y axis)	v	Y	y
Vertical (z axis)	w	Z	z
Roll	p	L	ϕ
Pitch	q	M	θ
Yaw	r	N	ψ

Table 3.1: Nomenclature used in Bryan equations

Translational dynamics

From Newton's Second Law:

$$\sum \vec{F} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \frac{d(m \cdot \vec{V}_{ABS})}{dt} = m \cdot \dot{\vec{V}}_{ABS} \tag{3.1}$$

Absolute acceleration has contributions from translational velocities and rotational velocities.

$$\dot{\vec{V}}_{ABS} = \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} + \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ p & q & r \\ u & v & w \end{vmatrix} = \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} + \begin{bmatrix} q \cdot w - r \cdot v \\ r \cdot u - p \cdot w \\ p \cdot v - q \cdot u \end{bmatrix} \tag{3.2}$$

Then, dividing the forces of each axis into thrust, aerodynamic and gravitational forces the following set of equations is obtained. To obtain gravitational forces a transformation from Local Horizon Axes reference system to Body Axes reference system is performed.

$$m \cdot (\dot{u} + q \cdot w - r \cdot v) = F_{TX} + F_{AX} - F_{GX} \tag{3.3}$$
$$m \cdot (\dot{v} + r \cdot u - p \cdot w) = F_{TY} + F_{AY} - F_{GY} \tag{3.4}$$
$$m \cdot (\dot{w} + p \cdot v - q \cdot u) = F_{TZ} + F_{AZ} - F_{GZ} \tag{3.5}$$

Now, decomposing forces into their different terms the three first Bryan equations are obtained.

$$m \cdot (\dot{u} + q \cdot w - r \cdot v) = T_X + \frac{1}{2} \cdot \rho \cdot V^2 \cdot S_W \cdot C_X - m \cdot g \cdot sin(\theta) \tag{3.6}$$
$$m \cdot (\dot{v} + r \cdot u - p \cdot w) = T_Y + \frac{1}{2} \cdot \rho \cdot V^2 \cdot S_W \cdot C_Y - m \cdot g \cdot cos(\theta) \cdot sin(\phi) \tag{3.7}$$
$$m \cdot (\dot{w} + p \cdot v - q \cdot u) = T_Z + \frac{1}{2} \cdot \rho \cdot V^2 \cdot S_W \cdot C_Z - m \cdot g \cdot cos(\theta) \cdot cos(\phi) \tag{3.8}$$

Rotational dynamics

For the moments:

$$\sum \vec{M} = \begin{bmatrix} L \\ M \\ N \end{bmatrix} = \dot{\vec{H}}_{ABS} \tag{3.9}$$

Where H is the angular momentum.

$$\begin{bmatrix} L \\ M \\ N \end{bmatrix} = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix} \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} + \begin{bmatrix} 0 & -r & 0 \\ r & 0 & -p \\ 0 & p & 0 \end{bmatrix} \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{3.10}$$

This gives the following set of equations.

$$L = I_{xx} \cdot \dot{p} - I_{xz} \cdot \dot{r} - I_{xz} \cdot p \cdot q + (I_{zz} - I_{yy}) \cdot q \cdot r - I_{yz} \cdot (q^2 - r^2) - I_{xy} \cdot (\dot{q} - r \cdot p) \tag{3.11}$$

$$M = I_{yy} \cdot \dot{q} - I_{xz} \cdot (p^2 - r^2) + (I_{xx} - I_{zz}) \cdot p \cdot r + I_{xz} \cdot (\dot{p} + q \cdot r) - I_{yz} \cdot (\dot{r} - p \cdot q) \tag{3.12}$$

$$N = I_{zz} \cdot \dot{r} - I_{xz} \cdot \dot{p} + I_{xz} \cdot r \cdot q + (I_{yy} - I_{xx}) \cdot p \cdot q - I_{xy} \cdot (p^2 - q^2) - I_{yz} \cdot (\dot{q} + r \cdot p) \tag{3.13}$$

Due to the symmetry assumption mentioned above, $I_{xy} = I_{zy} = 0$. Rearranging the equation considering $A = I_{xx} \cdot I_{zz} - I_{xz}^2$ the next three Bryan equations are obtained.

$$\dot{p} = \frac{I_{xx}}{A} \cdot L + \frac{I_{xz}}{A} \cdot N + \left(\frac{I_{xz} \cdot (I_{xx} - I_{yy} + I_{zz})}{A} \right) \cdot p \cdot q + \left(\frac{I_{zz} \cdot (I_{yy} - I_{zz}) - I_{xz}^2}{A} \right) \cdot r \cdot q \tag{3.14}$$

$$\dot{q} = \frac{M}{I_{yy}} + \frac{I_{zz} - I_{xx}}{I_{yy}} \cdot p \cdot r + \frac{I_{xz}}{I_{yy}} \cdot (r^2 - p^2) \tag{3.15}$$

$$\dot{r} = \frac{I_{xx}}{A} \cdot N + \frac{I_{xz}}{A} \cdot L + \left(\frac{I_{xx} \cdot (I_{xx} - I_{yy}) + I_{xz}^2}{A} \right) \cdot p \cdot q + \left(\frac{I_{xz} \cdot (I_{yy} - I_{xx} - I_{zz})}{A} \right) \cdot r \cdot q \tag{3.16}$$

Euler relations

The Euler relations are given as following.

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & sin(\phi) \\ 0 & -sin(\phi) & cos(\phi) \end{bmatrix} \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & sin(\phi) \\ 0 & -sin(\phi) & cos(\phi) \end{bmatrix} \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} \tag{3.17}$$

The resulting equations are the next three Bryan Equations.

$$p = \dot{\phi} - \dot{\psi} \cdot sin(\theta) \tag{3.18}$$

$$q = \dot{\theta} \cdot cos(\phi) + \dot{\psi} \cdot cos(\theta) \cdot sin(\theta) \tag{3.19}$$

$$r = \dot{\psi} \cdot cos(\theta) \cdot cos(\phi) - \dot{\theta} \cdot sin(\phi) \tag{3.20}$$

Kinematics

Kinematic relations are obtained transforming the velocities in Body Axes reference system to Local Horizon Axes reference system, being the last three Bryan Equations.

$$\begin{aligned} \dot{x} = {} & u \cdot cos(\psi) \cdot cos(\theta) + v \cdot (cos(\psi) \cdot sin(\theta) \cdot sin(\phi) - cos(\phi) \cdot sin(\psi)) - \\ & - w \cdot (sin(\theta) \cdot cos(\phi) \cdot cos(\psi) + sin(\phi) \cdot sin(\psi)) \end{aligned} \tag{3.21}$$

$$\begin{aligned} \dot{y} = {} & u \cdot cos(\theta) \cdot sin(\psi) + v \cdot (cos(\phi) \cdot cos(\psi) + sin(\theta) \cdot sin(\phi) \cdot sin(\psi)) + \\ & + w \cdot (-cos(\psi) \cdot sin(\phi) + cos(\phi) \cdot sin(\theta) \cdot sin(\psi)) \end{aligned} \tag{3.22}$$

$$\dot{z} = -u \cdot sin(\theta) + v \cdot cos(\theta) \cdot sin(\phi) + w \cdot cos(\theta) \cdot cos(\phi) \tag{3.23}$$

Summary

The twelve equations being used for the model are the following.

$$m \cdot (\dot{u} + q \cdot w - r \cdot v) = T_X + \frac{1}{2} \cdot \rho \cdot V^2 \cdot S_W \cdot C_X - m \cdot g \cdot sin(\theta) \tag{3.6}$$

$$m \cdot (\dot{v} + r \cdot u - p \cdot w) = T_Y + \frac{1}{2} \cdot \rho \cdot V^2 \cdot S_W \cdot C_Y - m \cdot g \cdot cos(\theta) \cdot sin(\phi) \tag{3.7}$$

$$m \cdot (\dot{w} + p \cdot v - q \cdot u) = T_Z + \frac{1}{2} \cdot \rho \cdot V^2 \cdot S_W \cdot C_Z - m \cdot g \cdot cos(\theta) \cdot cos(\phi) \tag{3.8}$$

$$\dot{p} = \frac{I_{xx}}{A} \cdot L + \frac{I_{xz}}{A} \cdot N + \left(\frac{I_{xz} \cdot (I_{xx} - I_{yy} + I_{zz})}{A} \right) \cdot p \cdot q + \left(\frac{I_{zz} \cdot (I_{yy} - I_{zz}) - I_{xz}^2}{A} \right) \cdot r \cdot q \tag{3.14}$$

$$\dot{q} = \frac{M}{I_{yy}} + \frac{I_{zz} - I_{xx}}{I_{yy}} \cdot p \cdot r + \frac{I_{xz}}{I_{yy}} \cdot (r^2 - p^2) \tag{3.15}$$

$$\dot{r} = \frac{I_{xx}}{A} \cdot N + \frac{I_{xz}}{A} \cdot L + \left(\frac{I_{xx} \cdot (I_{xx} - I_{yy}) + I_{xz}^2}{A} \right) \cdot p \cdot q + \left(\frac{I_{xz} \cdot (I_{yy} - I_{xx} - I_{zz})}{A} \right) \cdot r \cdot q \tag{3.16}$$

$$p = \dot{\phi} - \dot{\psi} \cdot sin(\theta) \tag{3.18}$$

$$q = \dot{\theta} \cdot cos(\phi) + \dot{\psi} \cdot cos(\theta) \cdot sin(\theta) \tag{3.19}$$

$$r = \dot{\psi} \cdot cos(\theta) \cdot cos(\phi) - \dot{\theta} \cdot sin(\phi) \tag{3.20}$$

$$\begin{aligned} \dot{x} = {} & u \cdot cos(\psi) \cdot cos(\theta) + v \cdot (cos(\psi) \cdot sin(\theta) \cdot sin(\phi) - cos(\phi) \cdot sin(\psi)) - \\ & - w \cdot (sin(\theta) \cdot cos(\phi) \cdot cos(\psi) + sin(\phi) \cdot sin(\psi)) \end{aligned} \tag{3.21}$$

$$\begin{aligned} \dot{y} = {} & u \cdot cos(\theta) \cdot sin(\psi) + v \cdot (cos(\phi) \cdot cos(\psi) + sin(\theta) \cdot sin(\phi) \cdot sin(\psi)) + \\ & + w \cdot (-cos(\psi) \cdot sin(\phi) + cos(\phi) \cdot sin(\theta) \cdot sin(\psi)) \end{aligned} \tag{3.22}$$

$$\dot{z} = -u \cdot sin(\theta) + v \cdot cos(\theta) \cdot sin(\phi) + w \cdot cos(\theta) \cdot cos(\phi) \tag{3.23}$$

3.2 PX4 stack

The PX4 stack is an open-source flight control software platform designed primarily for Unmanned Aircraft Systems (UAS), including drones, multicopters, and fixed-wing aircraft. It provides the necessary software components and tools for controlling and managing various aspects of UAS flight, from basic stabilization to complex mission planning and execution. Its different sections will be explained to comprehend what will be modified in this project.

3.2.1 System overview

As seen in the PX4 user guide [15], the general structure of the PX4 system architecture is shown in Figure 3.3.

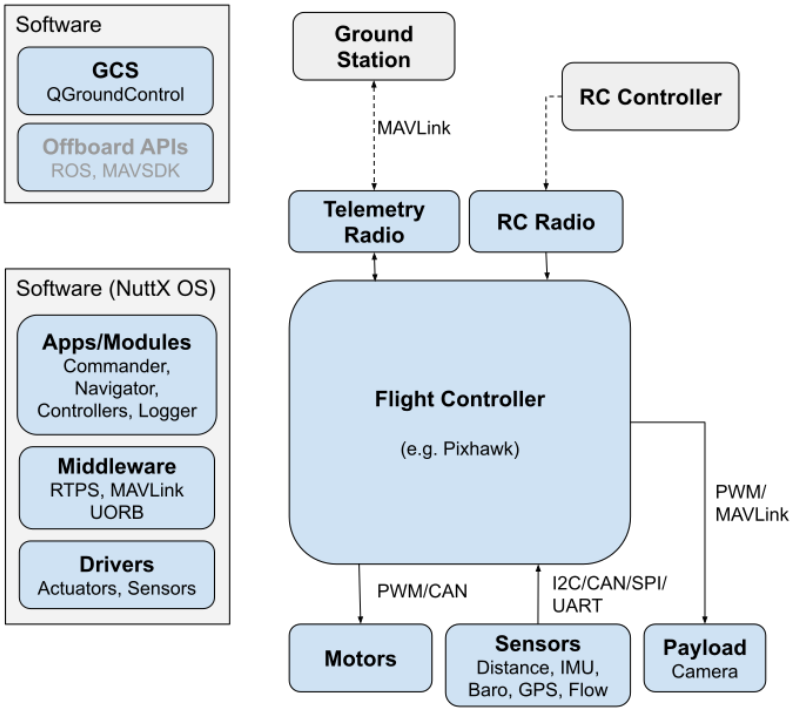


Figure 3.3: PX4 system architecture general scheme. Source: [15]

- Ground Control Station (GCS): It is software based on communication between a ground station and an aircraft. It allows users to design and monitor missions, visualize real-time telemetry data and send commands to the aircraft during flight. The communication between the ground station and the flight controller is handled via MAVLink, a lightweight communication protocol widely used in UAs.

- Flight Controller: The aircraft's flight controller, running the PX4 firmware, receives commands via telemetry and RC radios. It processes sensor data and computes the necessary control outputs to achieve the desired flight behaviour. This involves translating the mission goals or manual inputs into specific actions sent to the aircraft's actuators. Moreover, there is the possibility of manually controlling the aircraft through the RC Controller.

- uORB Middleware: A crucial part of the PX4 architecture is uORB (micro Object Request Broker). It is a middleware that handles communication between different software components. Its main feature is to act as a publish-subscribe messaging system, allowing various modules (e.g., sensors, estimators, controllers) to exchange data asynchronously. Therefore, modules subscribe to relevant topics (e.g., sensor data) and publish updated outputs (e.g., position estimates, control setpoints) that other components can use. This modular approach allows for better scalability, flexibility, and separation in the system.

- Sensors and Actuators: The sensors (IMU, GPS, barometer) provide real-time data about the aircraft's position, speed, and orientation. The actuators (motors, servos) execute the physical commands calculated by the flight controller based on the sensor inputs.

3.2.2 Flight Stack

The PX4 flight stack is a group of algorithms responsible for the guidance, navigation, and control of UAS, including estimators for attitude and position determination.

The diagram shown in Figure 3.4 illustrates the components and flow of the PX4 flight stack. beginning with inputs from the UAS's sensors, RC controller, and the Navigator. These inputs are processed to control the aircraft's actuators (e.g., motors or servos) through a series of computational blocks.

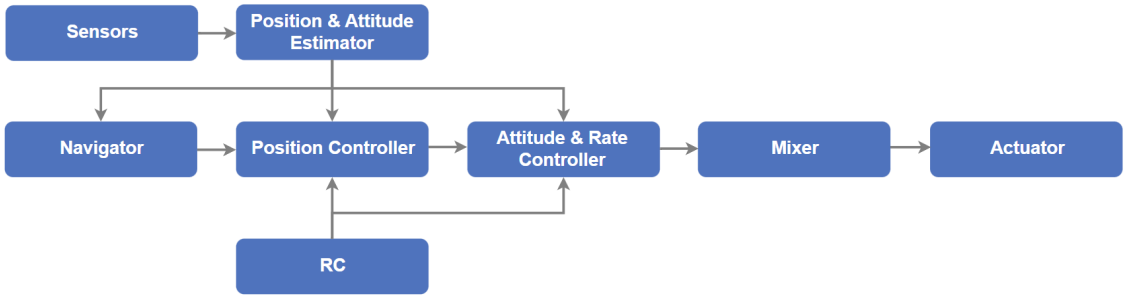


Figure 3.4: PX4 flight stack scheme. Source: [15]

- Position and Attitude Estimator: This block collects data from various sensors to determine the drone's current state, such as its attitude and position. For example, it uses IMU (Inertial Measurement Unit) data to estimate attitude.
- Controllers: After that, there are different controllers in the stack, each responsible for ensuring the aircraft's stability and trajectory:
  - Position Controller: This controller adjusts the drone's position by comparing the desired position (setpoint) with the current estimated position. It outputs commands to reach the target location.
  - Attitude and Rate Controller: It manages the drone's orientation and angular velocity by adjusting its pitch, roll, and yaw, based on the position controller's output.
- Mixer: The outputs of the controllers is delivered to the mixer and translates them into individual motor outputs, ensuring that the aircraft's physical constraints, such as torque and force limits, are respected. This is dependent on the aircraft's configuration, including its motor layout.
- Actuators: The final component of the control loop, the actuators (typically motors or servos), respond to the mixer's commands to control the UAS as intended, performing the actual flight maneuvers.

3.2.3 Fixed-Wing Controllers

The controllers that have been previously stated will be explained for a Fixed-Wing configuration, focusing more on the Attitude Controller as it will be the one modified in the project.

Position Controller

The Fixed-Wing Position Controller focuses on controlling the aircraft's position by managing its airspeed and altitude, primarily through the Total Energy Control System (TECS), which optimizes energy management during flight.

The diagram seen in Figure 3.5 highlights the two main control elements in this system.

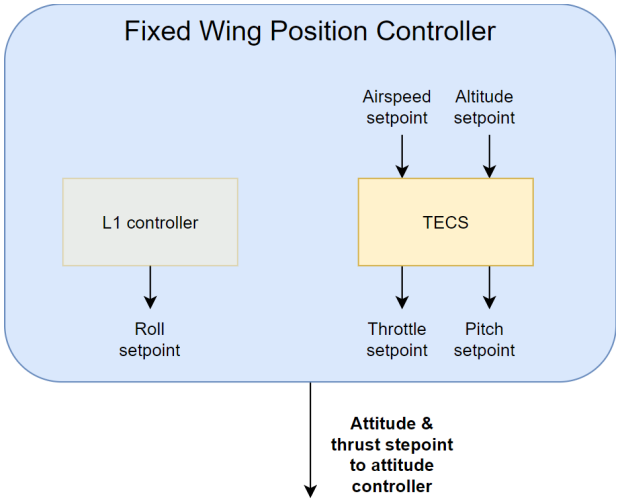


Figure 3.5: Total Energy Control System scheme. Source: [15]

- L1 Controller: This component generates roll setpoints by managing the aircraft's lateral navigation. It ensures that the aircraft follows the desired flight path by controlling its horizontal trajectory.
- TECS: The TECS algorithm regulates both airspeed and altitude simultaneously by converting these parameters into control outputs for throttle and pitch. The output of TECS feeds into the attitude controller, which then executes precise adjustments in the aircraft's pitch and thrust.

Regarding TECS, it uses an energy-based approach to manage aircraft's position. Thrust is responsible for managing the total energy of the aircraft, whereas pitch controls the distribution of energy between kinetic energy (airspeed) and potential energy (altitude). A higher pitch angle transfers kinetic energy into altitude, while a lower pitch reduces altitude and increases airspeed.

The simultaneous control of airspeed and altitude can be complex because both the pitch angle and throttle affect these variables. Increasing the throttle generally increases airspeed and altitude, while increasing the pitch will raise altitude but reduce airspeed. TECS decouples this problem by converting the original airspeed and altitude setpoints into energy terms.

Attitude Controller

The PX4 attitude controller uses a cascaded loop method to manage the aircraft's orientation shown in Figure 3.6.

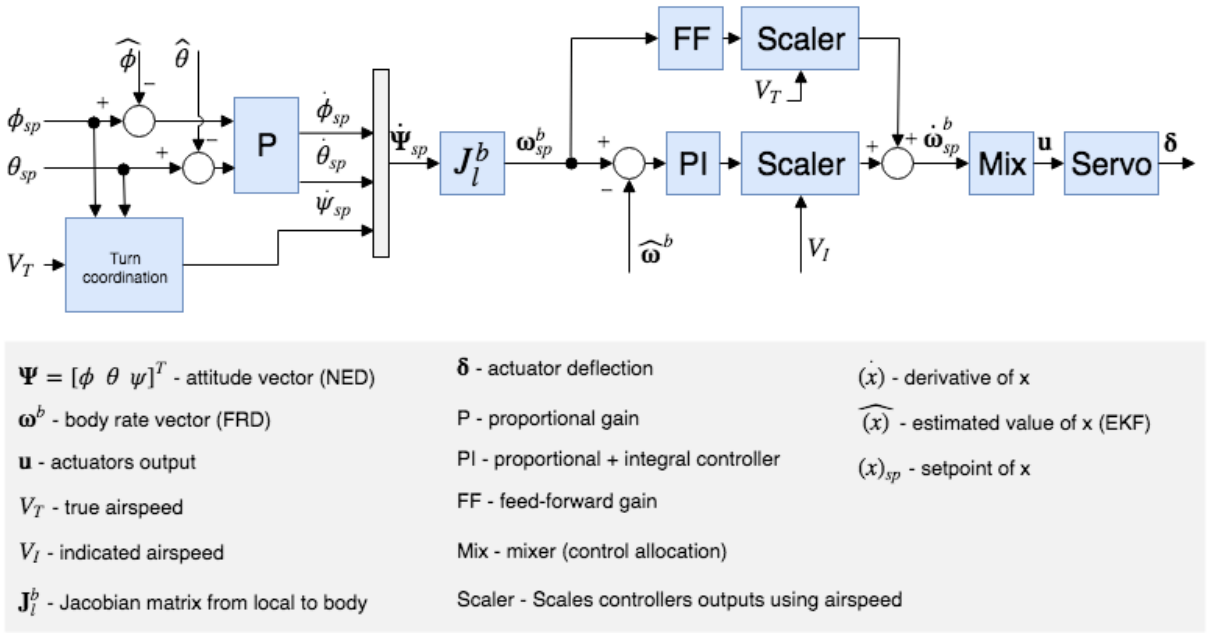


Figure 3.6: Attitude Controller scheme. Source: [15]

- Outer Loop (Attitude Control): This section first computes the error between the attitude setpoint (desired orientation) and the estimated attitude (current orientation). Afterwards, a proportional controller (P) multiplies the error by a gain to generate a rate setpoint (desired rate of change in orientation).
- Inner Loop (Rate Control): Similarly, the error between the rate setpoint and the current rates is computed. Then, a Proportional-Integral (PI) controller generates the desired angular acceleration.
- Mixer (Control Allocation): Computes the angular position of control effectors (ailerons, elevators, rudders) using the desired angular acceleration. After that, the output is scaled for adjusting control surface effectiveness based on airspeed measurements. If no airspeed sensor is used, gain scheduling is disabled.
- Feedforward Gain: Compensates for aerodynamic damping to maintain a constant rate by using feedforward in the rate loop.
- Turn Coordination: This algorithm is used to obtain yaw rate setpoint knowing the velocity of the aircraft. It aims to minimize lateral acceleration during slipping turns. Then, adds a yaw rate controller that helps counteract adverse yaw effects and dampen the Dutch roll mode by providing extra directional damping.

Attitude Controller Folder Structure

This section explains the code structure of the attitude controller as it is the one that will be modified to suit an experimental version.

As explained in the previous section, there are two main control loops. These are reflected in two folders, fw_att_control and fw_rate_control, seen in Figure 3.7. The fw refers to fixed-wing, and attitude and rate to the different loops. They are encompassed in the src folder, which contains all the source code folders, and the different modules folder.

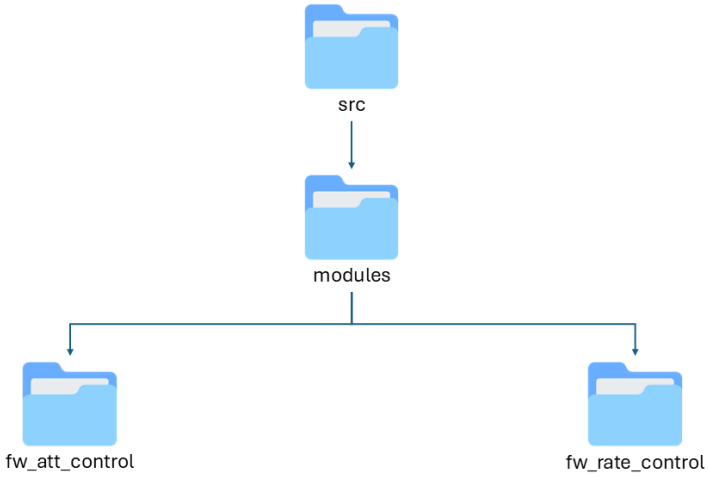


Figure 3.7: Attitude Controller general folder scheme. Source: own

Regarding the Attitude loop, the folder contains a file that includes all the main code of the loop. Inside it, different functions are called, which are defined in the remaining files. These are controllers for roll, pitch, yaw and the wheel. Finally, a file contains all the constant parameters needed for the algorithm. They are all represented in Figure 3.8.

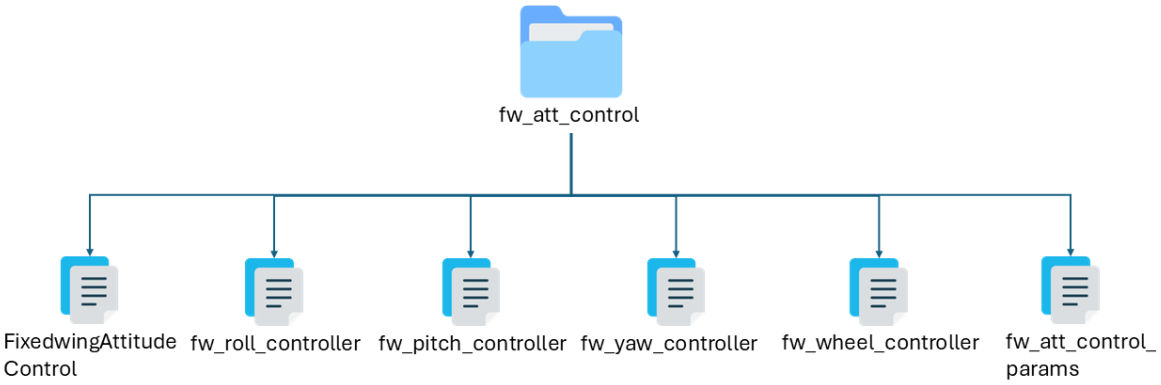


Figure 3.8: Attitude Controller loop folder scheme. Source: own

The Rates loop only includes the general script and the parameters one, represented in Figure 3.9.

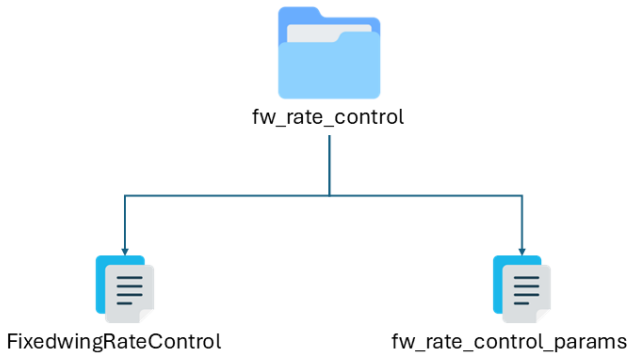


Figure 3.9: Rates Controller loop folder scheme. Source: own

Chapter 4

Requirements study and factors to be considered: limitations and conditioning aspects

4.1 Regulations

Due to the project's nature, regulations regarding autonomous aircraft flights do not affect it. However, as the project will evolve to real flights, European regulations will be considered. Precisely, "Reglamento de Ejecución (UE) 2019/947" and its modification "Reglamento de Ejecución (UE) 2024/1110"; and "Reglamento Delegado (UE) 2019/945" and its modification "Reglamento Delegado (UE) 2024/1108". These regulate the procedures for UAS.

4.2 Design and operational requirements

The aircraft type is fixed wing, so PX4 algorithm will run the required scripts of this type of behaviour. This project will modify these scripts to accommodate the experimental control.

Considering the aircraft design and its operational limits, the desired operation point is selected for cruise phase. The aircraft model will be linearized around this point, conditioning the operations that the experimental control can handle. This operational point is at a velocity of 18 meters per second and a height of 100 meters.

4.3 Miscellaneous requirements

The project involves modifying an already implemented PX4 solution to suit a different desired behaviour while maintaining previous capabilities. Therefore, a gradual approach has been preferred, in which the focus is the accommodation of two working solutions instead of the development of a highly complex control algorithm. This also contributes to comply to hardware limitations. The adopted strategy is to start with a simpler algorithm and expand to more complex ones and check if hardware is proving to be a limitation.

Chapter 5

Alternative solution proposal and adopted solution justification

Different solutions for the project elements will be proposed and their feasibility will be assessed considering different factors such as performance, computational cost or implementation complexity.

5.1 Ground control station

Ground control station is the software of the computer that stays on the ground, which enables visualization of the mission and communication with the UA to send commands, design a mission or change the current mission. Different solutions are proposed:

1. Mission Planner: An open-source ground station software designed for use with ArduPilot-based vehicles. It offers comprehensive mission planning, configuration, real-time telemetry, data logging and analysis.

2. APM Planner 2: Again, an open-source ground control software designed for ArduPilot-based systems. It offers a user-friendly interface for mission planning, parameter tuning, real-time telemetry, and data analysis.

3. UGCS (Universal Ground Control Software): A commercial ground control software that supports a wide range of drones and autopilots. It offers advanced mission planning and control features for professional and enterprise use, including 3D mission planning, terrain following, photogrammetry tools, real-time telemetry, and video streaming. It is compatible with many different options, including project's hardware.

4. FlytBase: Another commercial drone management platform. It offers cloud-based ground control capabilities, fleet management, real-time telemetry, and video streaming. It supports multiple drone platforms and provides extensive automation features.

5. QGroundControl: An open-source ground control station software that provides mission planning, flight monitoring, video streaming, parameter configuration,

and real-time telemetry for autonomous vehicles. It supports multiple vehicle types, including multicopters, fixed-wing aircraft, and VTOL, as well as it is compatible with different hardware such as PX4, ArduPilot or MAVLink protocol.

The first solutions are incompatible with project's hardware as they are developed for ArduPilot-based systems. The next ones are commercial software and exceed project's budget. The chosen solution will be QGroundControl, as it is open-source and recommended by PX4, which provides installation and connection tutorials to their hardware. Moreover, it is compatible with MAVLink protocol, which is used in PX4 stack to communicate with the aircraft during a mission.

5.2 Dynamic model simulation

No alternatives are proposed for the dynamic model simulation of H200 aircraft as it has been granted by HORUS project. It is implemented in MATLAB and Simulink due to its connectivity with QGroundControl. Therefore it offers the required simulation environment which enables visualization and manipulation through the chosen ground control station software.

5.3 Control strategies

Different control strategies are considered to be the first experimental control added to the project. The selection will be determined by the algorithm performance and its implementation complexity into the PX4 environment.

5.3.1 PID

PID control adjusts the control inputs based on the error between a desired setpoint and the current state of a linearised model. The Proportional (P) term adjusts the control output proportionally to the current error, the Integral (I) term accounts for past errors to eliminate steady-state offsets, and the Derivative term (D) predicts future errors to dampen oscillations and improve response times. These can be seen in the schematics of Figure 5.1

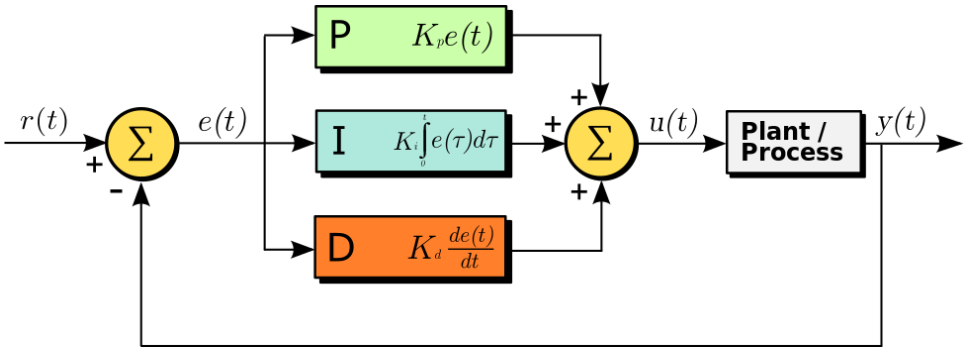


Figure 5.1: PID scheme. Source: [16]

5.3.2 Pole Placement

It is a control strategy based on full state feedback, where the desired closed-loop pole locations are specified, and the feedback gain is designed to achieve these poles. It works with a linearised model. In full state feedback, all the states of the system are fed back into the controller to determine the control input. Its schematics are shown in Figure 5.2.

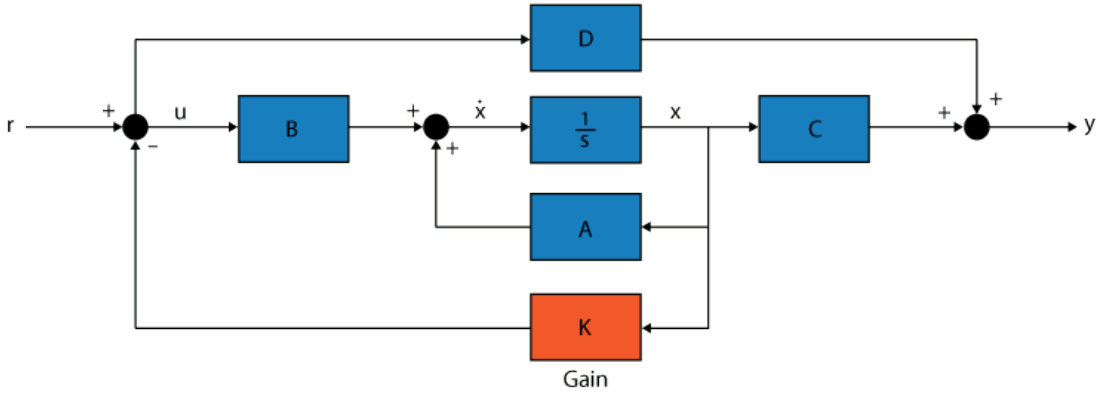


Figure 5.2: Pole Placement scheme. Source: [17]

5.3.3 Linear Quadratic Regulator (LQR)

LQR is an optimal control strategy based on full state feedback. It minimizes a cost function representing the trade-off between state deviations and control effort. It uses a linearized model of the aircraft around an operating point. LQR determines the optimal control inputs to achieve desired performance while balancing the costs of control actions. Without the integral part, LQR will not achieve zero steady state error. Its schematics are the same as Pole Placement strategy, as the only difference is the pole location selection.

5.3.4 LQR with Integral Action

LQRI enhances the basic LQR by including integral action to eliminate steady-state errors. This involves augmenting the system state with the integral of the error, allowing the controller to account for accumulated discrepancies between the desired and actual states. It is represented in Figure 5.3

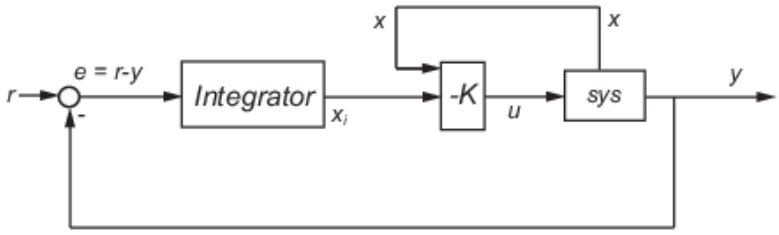


Figure 5.3: Linear Quadratic Regulator with integral action scheme. Source: [18]

5.3.5 Model Predictive Control (MPC)

MPC uses a linearised model of the aircraft to predict its future states over a finite time horizon and optimizes control inputs by solving a constrained optimization problem at each time step. It can handle multi-variable control tasks, accommodate constraints (such as actuator limits and collision avoidance), and optimize performance by anticipating future disturbances and reference trajectory changes. Its schematics can be seen in Figure 5.4.

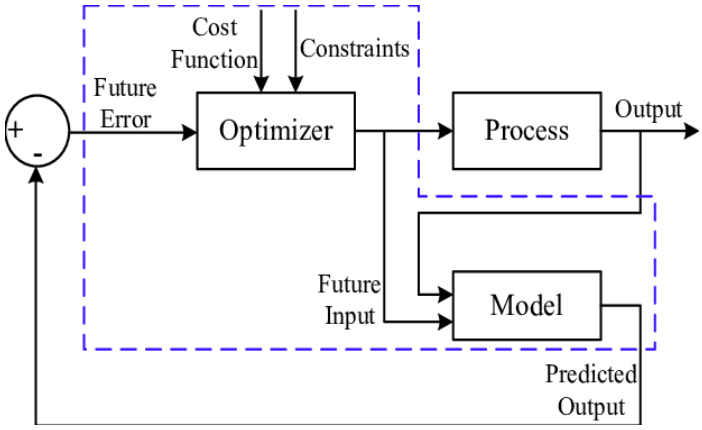


Figure 5.4: Model Predictive Control scheme. Source: [19]

5.3.6 Fuzzy based

Uses fuzzy logic to handle uncertainty and imprecision, making it especially useful for complex or nonlinear systems where traditional control methods struggle. Unlike classical logic that works with binary values (true or false, 0 or 1), fuzzy logic allows for varying degrees of truth, enabling the controller to process inputs that are vague or approximate, as human understanding of problems is. Inputs are mapped to fuzzy sets using membership functions, and decisions are made based on fuzzy "IF-THEN" rules that mimic human reasoning. The controller then converts the fuzzy outputs back into precise control actions through defuzzification. A general scheme is shown in Figure 5.5.

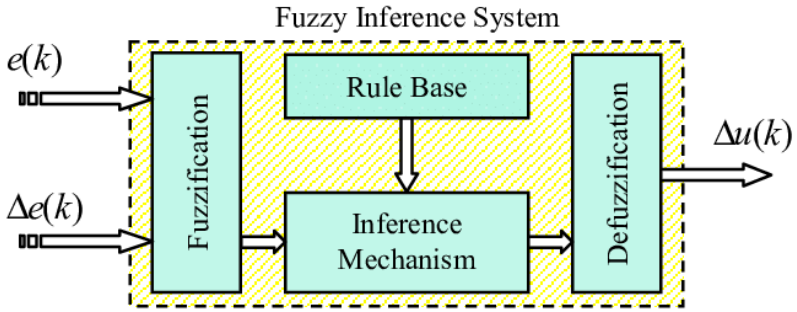


Figure 5.5: Model Predictive Control scheme. Source: [20]

5.3.7 Neural Network-Based

This method uses neural networks to model and control the dynamics of the aircraft. Neural networks can learn complex, nonlinear relationships from data, making them suitable for adaptive and robust control in highly dynamic and uncertain environments. Moreover, it can improve performance over time through learning from data. They can be used standalone or in combination with traditional control methods like MPC or LQR to enhance performance. Figure 5.6 shows an example of its structure.

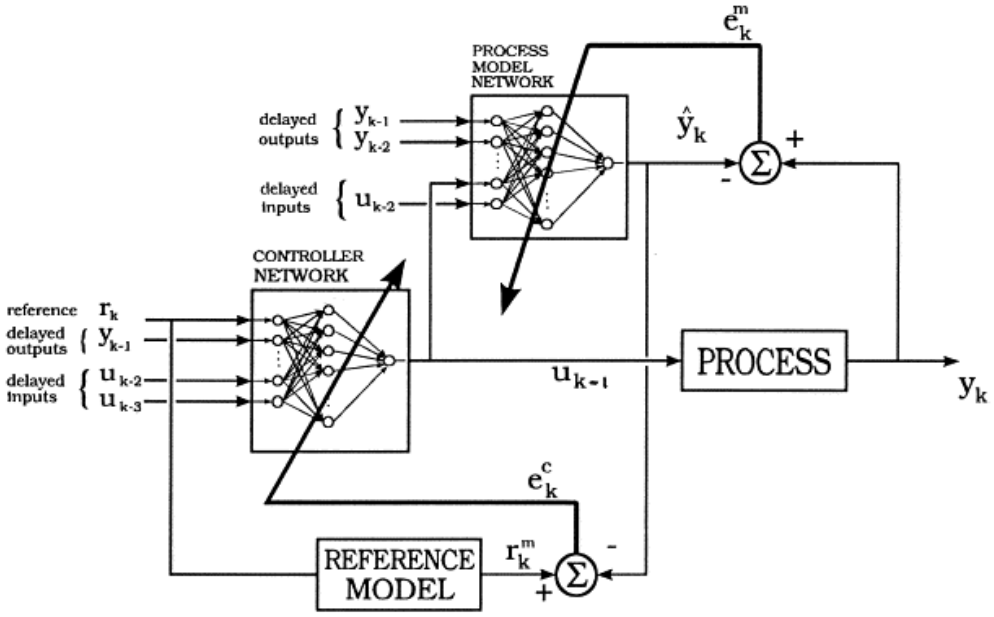


Figure 5.6: Neural Network-Based scheme. Source: [21]

5.3.8 Controller decision

Between the different controllers, some are discarded because of their implementation complexity. These are MPC, Fuzzy Based and Neural Network-Based. Although they have a lot of potential to obtain performance, the project needs a controller that offers good performance with little implementation complexity.

On the other hand, PID control is already implemented in PX4 with a solution that includes a cascade control. Any PID implementation would be simpler and have worse capabilities, and therefore is not considered.

The remaining solution is between Pole Placement, LQR and LQR with Integral Action. They are all based on state space representation of the dynamics, the difference lies on pole location policy. **LQR with Integral Action** is preferred due to being an optimal control, and having capability to mitigate steady state error, which enables maintaining desired altitude, heading or other critical parameters. This is accomplished with little added implementation effort.

5.4 PX4 implementation

5.4.1 Module Policy

As the folder structure has been presented in the previous chapter, two main implementation philosophies are proposed.

1. Use existing module: By modifying an existing code of a control module, the required experimental control code is included. An if-else statement is employed to access the different codes, and the selection is triggered by a parameter that can be modified at run-time via MAVLink communication protocol.

2. Create a new module: This solution involves creating a module that is compatible with uORB hierarchy. The module to use is defined through the mission launch script or via a parameter provided to the pre-flight configuration. This selection is made before the drone takes off, and no dynamic switching (such as an if-else structure) is needed within the code during flight.

The preferred solution is to use an existing module due to little experience with modifying PX4 architecture.

5.4.2 Substituted Fixed-Wing Controllers

Two possibilities regarding the selection of substituted controllers are proposed:

1. Attitude Controller: This option would imply that the Position controller is the TECS algorithm (3.2.3), which delivers attitude setpoints to the experimental Attitude Controller.

2. Position and Attitude Controller: Both controllers are developed with the LQR with integral action.

Although the second option is potentially more powerful, the decision is that simplicity at the first experimental implementation is preferred. Therefore, the option selected is number one, where Attitude Controller is the substituted one.

Chapter 6

Detailed description of adopted solution

The exact solution theoretical background and its implementation, as well as the extra elements considered will be explained along this chapter.

6.1 Theoretical background

6.1.1 State Space

State-space representation is a mathematical model used to describe the dynamic behavior of systems. It encapsulates a system's dynamics using state variables, providing a compact way to represent complex systems, especially the ones with multiple inputs and outputs (MIMO).

States

The state of a system is defined by a set of variables, known as state variables, that capture all the necessary information to describe the system's behavior over time. These variables form the state vector and allow the prediction of future states based on the current state and input.

In a system with n state variables, the state vector $x(t)$ is expressed as the following:

$$x(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_n(t) \end{bmatrix} \tag{6.1}$$

where each element represents a state variable of the system at time t.

Representation

A linear time-invariant (LTI) system can be represented in state-space form as:

$$\dot{x}(t) = A \cdot x(t) + B \cdot u(t) \tag{6.2}$$
$$y(t) = C \cdot x(t) + D \cdot u(t) \tag{6.3}$$

- $x(t)$ is the state vector with n elements. Contains all the state variables.
- $u(t)$ is the input vector with m elements. Represents external influences or controls applied to the system.
- $y(t)$ is the output vector containing p elements and representing the measurable response of the system.
- A is the state matrix, with size $(n \times n)$. It describes the internal dynamics between state variables.
- B is the input matrix, with size $(n \times m)$. Maps the different inputs to state variables.
- C is the output matrix, whose size is $(p \times n)$. Relates state variables to outputs.
- D is the feedthrough matrix, whose size is $(p \times m)$. It maps inputs directly to outputs (often zero in many practical cases).

It can be divided into two parts:

1. State Equation: This equation (6.2) describes how the state evolves over time based on the current state and input. The matrix A encapsulates how each state variable affects the others, while B represents how the input $u(t)$ influences the states.
2. Output Equation: On the other hand, this equation (6.3) shows how the states map to the system's outputs. Here, C determines the relationship between the states and the outputs, and D describes any direct influence of the input on the output.

Figure 5.2 gives a scheme of a state space based feedback controller.

Properties

- Controllability: Defines the possibility to drive the state $x(t)$ from any initial state to any desired state in finite time using an appropriate input $u(t)$.
- Observability: A system is observable if the state vector $x(t)$ can be reconstructed based on measurements of the output $y(t)$ over time.

6.1.2 Full State Feedback

Full-state feedback [22] is a control method where the entire state vector of a system is used to compute the control input. By using all state variables, the controller has a complete picture of the system, allowing precise adjustments.

Formulation

Going back to the LTI system representation (6.2)(6.3):

$$\dot{x}(t) = A \cdot x(t) + B \cdot u(t) \tag{6.2}$$
$$y(t) = C \cdot x(t) + D \cdot u(t) \tag{6.3}$$

In this solution output matrix is typically $C = I$, and feedthrough matrix is $D = 0$. This implies that the output $y(t)$ directly reflects the state variables with no direct influence from the input $u(t)$, which simplifies the control design by focusing only on state dynamics.

Moreover, the control input $u(t)$ is designed as a linear combination of all the state variables. The control law is the following:

$$u(t) = -K \cdot x(t) \tag{6.4}$$

Here, K is the state feedback gain matrix. The negative sign indicates feedback to counteract the states, so the controller will drive the state equation (6.2) to 0.

Substituting the control law (6.4) into the state equation (6.2), the closed-loop system becomes:

$$\dot{x}(t) = (A - B \cdot K) \cdot x(t) \tag{6.5}$$

Here, $(A - B \cdot K)$ is the closed-loop system matrix, which governs the dynamics under state feedback. By appropriately selecting K, the position of the system eigenvalues -or poles- is manipulated, thus ensuring stability and the desired dynamic behaviour.

Required properties

This method implies that all states are controllable and can be measured. In practice, this may require state estimators like observers (and therefore observability) if certain states are not directly measurable.

For a system to be controllable, the controllability matrix $\mathcal{C}$ is defined as:

$$\mathcal{C} = \begin{bmatrix} B & AB & A^2B & \dots & A^{n-1}B \end{bmatrix} \tag{6.6}$$

This must have full rank (rank n). If this condition is met, any pole configuration can be achieved by appropriately selecting K.

Limitations

Some limitations of this method are found:

– Although theoretically the poles can be placed anywhere with K, in reality the dynamic behaviour is influenced by the capabilities of the system, so certain responses may not be achieved.

- The more states a system has, the larger K becomes, potentially increasing implementation complexity.
- While this technique stabilizes the system, it may not handle steady-state errors well. In such cases, integral action can be added, as it will be the case.

6.1.3 LQR

As mentioned before, LQR [23] is an optimal control based on full state feedback which minimizes a defined cost function.

Cost function

This function penalizes deviations of the states from desired values and penalizes high control efforts, thus balancing performance and energy usage. The standard cost function for LQR in continuous-time is:

$$J = \int_0^\infty \left(x^T \cdot Q \cdot x + u^T \cdot R \cdot u\right) \cdot dt$$

- x with size $(n \times 1)$ is the state vector of the system.
- u with size $(m \times 1)$ is the control input.
- Q with size $(n \times n)$ is the weighting matrix that penalizes the state deviations.
- R with size $(m \times m)$ is the weighting matrix that penalizes the control effort.

The product $\left(x^T \cdot Q \cdot x\right) \geq 0$, which means is positive semidefinite. On the other hand, $\left(u^T \cdot R \cdot u\right) > 0$ is positive definite. This means that as the cost function follows a quadratic trend, it always has a minimum, as depicted in Figure 6.1

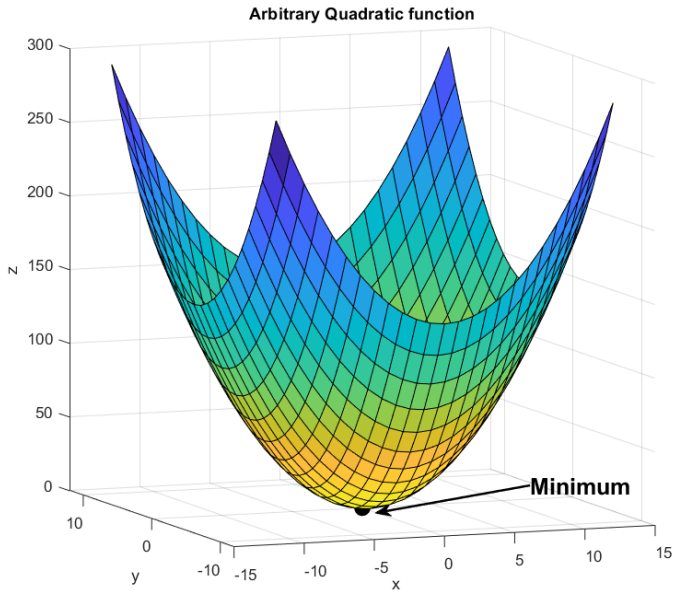


Figure 6.1: Arbitrary quadratic function plot. Source: own

This yields the optimal control, as it can find the best candidate for state feedback gain matrix given the weighting matrices.

Riccati equation

The objective of LQR is to determine the feedback gain matrix K that minimizes the cost function J, resulting in the control law (6.4) defined in Subsection 6.1.2. To minimize J, the Algebraic Riccati Equation (ARE) is solved:

$$A^T \cdot P + P \cdot A - P \cdot B \cdot R^{-1} \cdot B^T \cdot P + Q = 0 \tag{6.7}$$

Here P is the symmetric, positive definite solution to the Riccati equation.

Finding P generally involves solving this nonlinear matrix equation, typically using numerical algorithms (such as the 'care' solver in MATLAB, which stands for Continuous-time Algebraic Riccati Equation). This will be the choice to calculate it for the project.

With P found, the optimal feedback gain matrix K is given by the following equation:

$$K = R^{-1} \cdot B^T \cdot P \tag{6.8}$$

6.1.4 Integral Action addition

LQR solution lacks reference tracking and steady state error rejection. Adding integral action addresses this by accumulating the error over time and incorporating it into the control law. What is more, the system can better handle constant disturbances and model inaccuracies, as the integral term forces the system to adapt and compensate for them.

Augmented state vector

The state space model is augmented by including an additional integral state. Therefore it includes both the original states and the integral of the output error.

The error terms are defined in the following equations:

$$e(t) = r(t) - y(t) \tag{6.9}$$

$$z(t) = \int e(t) \cdot dt \tag{6.10}$$

Where $e(t)$ is the error between output and reference, and $z(t)$ is the integral of the error over time.

The augmented state vector is then:

$$x_a = \begin{bmatrix} x \\ z \end{bmatrix} \tag{6.11}$$

The augmented system dynamics can be represented as:

$$\begin{bmatrix} \dot{x} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} A & 0 \\ -C & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ z \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} \cdot u + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \cdot r \tag{6.12}$$

Augmented cost function

The cost function is modified to include the augmented states:

$$J = \int_0^\infty \left(x_a^T \cdot Q_a \cdot x_a + u^T \cdot R \cdot u \right) \cdot dt \tag{6.13}$$

Where a new weighting matrix is defined:

$$Q_a = \begin{bmatrix} Q & 0 \\ 0 & Q_z \end{bmatrix} \tag{6.14}$$

Here, Q_z represents the weight on the integral of the error z.

Control law

With the augmented system and cost function, the LQR procedure is applied to find the optimal feedback gain K_a that minimizes J. This gain is used in the new control law:

$$u = -K_a \cdot x_a = -\begin{bmatrix} K_x & K_i \end{bmatrix} \cdot \begin{bmatrix} x \\ z \end{bmatrix} \tag{6.15}$$

Here, K_x is the gain for the original states and K_i is the gain for the integral state. K_a is derived from the solution of the Riccati equation for the augmented system.

Now, choosing the matrices Q_a and R enables balancing the trade-off between tracking accuracy, transient performance, and control effort, so a new third effect is to be taken into account.

6.1.5 Antiwindup

This technique is essential for systems that rely on integral action in their controllers, especially when the actuators have physical limitations that can lead to saturation. In such cases, the integral term can "wind up" or accumulate a large error that continues to drive the control signal even when the actuator is already saturated. This phenomenon, known as integral windup, can result in significant overshoot, instability, and prolonged settling times once the system desaturates, as the controller struggles to counteract the effects of the accumulated integral error. Anti-windup mechanisms are therefore designed to prevent or correct for this integral accumulation.

Techniques

Common anti-windup techniques include:

1. Clamping: Limits the growth of the integral term during saturation.

2. Integral Reset: Resets or adjusts the integral term when saturation is detected.

3. Back Calculation: Introduces a corrective feedback term to reduce the accumulated integral error when the actuator is saturated.

Integral State resets the integral term entirely so it can lead to sudden control changes when the saturation condition is resolved. On the other hand, Back Calculation is more complex to implement and requires a gain to be tuned, for which poor tuning can lead to oscillations or instability. Therefore, Clamping is chosen.

Clamping

Saturation limits are defined:

$$u(t) = \max(u_{\min}, \min(u(t), u_{\max})) \tag{6.16}$$

As well as for integral accumulation:

– If $u(t) = u_{\max}$ or $u(t) = u_{\min}$, stop updating the integral term.

6.1.6 Linearization of equations

As LQR is a technique that needs a linearized model, the Bryan equations seen in Subsection 3.1.3 are linearized.

Sources of non-linerarity

The Bryan equations are expressed as a set of nonlinear ordinary differential equations (ODEs) due to the presence of trigonometric functions of the orientation angles and product terms involving velocities and angular velocities.

Objective and limitations

Linearizing Bryan equations simplifies them into a linear form around a chosen operating point, typically defined by a steady flight condition (such as level flight at a constant velocity).

The linearized model is only accurate for small perturbations around the operating point. As the aircraft deviates further from this point (e.g., in aggressive maneuvers), the nonlinear effects become more significant, and the linear approximation loses accuracy. For scenarios with large deviations, more advanced techniques may be required to maintain control.

Figure 6.2 shows an example in which an arbitrary quadratic function is linearized around an operating point. Only the terms inside a deviation bound (in this case the black rectangle) would be considered as properly approximated.

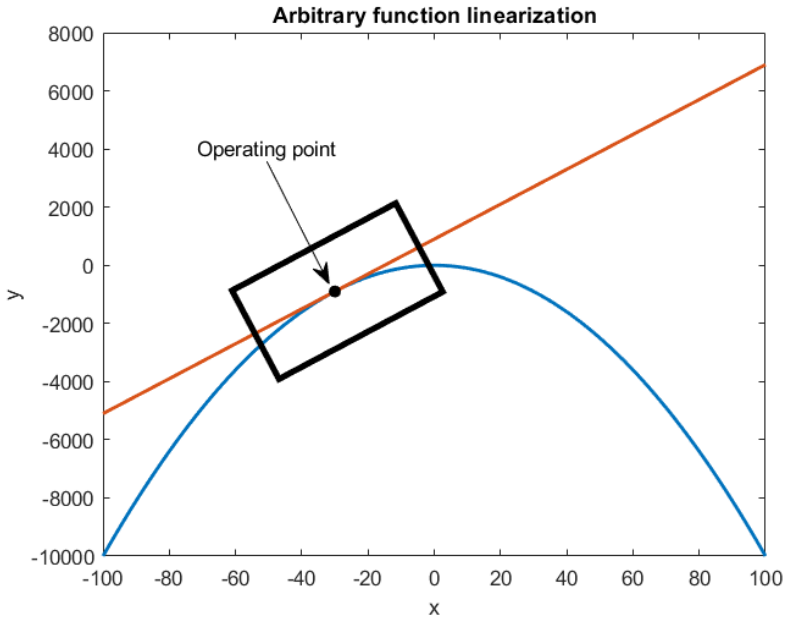


Figure 6.2: Linearized arbitrary quadratic function. Source: own

Procedure

The linealization process is based on a Taylor series expansion around a fixed operating point in which only the first-order terms are retained.

The process [24] [25] is divided into these steps, which will be exemplified for the first Bryan equation (3.6):

- **Step 1:** Trim Condition definition.

  A steady-state flight condition is chosen as the operating point. In this case, cruise condition is considered, which implies steady and horizontal levelled flight conditions. Moreover, mass is assumed to be constant.

  These conditions are reflected in the following variables:

$$\text{Euler angles:} \begin{cases} \theta_0 & \neq 0 \\ \phi_0 & = 0 \\ \psi_0 & = 0 \end{cases}$$

$$\text{Linear velocities:} \begin{cases} u_0 & = V_0 \\ v_0 & = 0 \\ w_0 & = 0 \\ \alpha_0 & = 0 \\ \beta_0 & = 0 \end{cases}$$

$$\text{Angular velocities:} \begin{cases} p_0 & = 0 \\ q_0 & = 0 \\ r_0 & = 0 \end{cases}$$

Moreover, steady flight conditions imply that all derivatives are null.

Therefore, first Bryan equation (3.3):

$$m \cdot (\dot{u} + q \cdot w - r \cdot v) = F_{TX} + F_{AX} - F_{GX} \tag{3.3}$$

turns into the first Bryan equation in Trim conditions:

$$0 = T_{x,0} + X_{A,0} - m \cdot g \cdot sin(\theta_0) \tag{6.17}$$

– **Step 2:** Perturbation around Trim Condition.

Small perturbations are defined around each state variable to represent deviations from the steady state.

$$\text{Perturbation examples:} \begin{cases} u & = u_0 + \Delta u \\ v & = v_0 + \Delta v \\ w & = w_0 + \Delta w \end{cases}$$

The perturbation terms (Δu, Δv, etc.) represent small deviations from the operating point and are assumed to be linear in this approach.

Applying perturbations to the first Bryan equation (3.6), the perturbed version is obtained:

$$\begin{aligned} m \cdot \left[\frac{d\,(u_0 + \Delta u)}{dt} + (q_0 + \Delta q) \cdot (w_0 + \Delta w) - (r_0 + \Delta r) \cdot (v_0 + \Delta v) \right] = \\ = (T_{x,0} + \Delta T_x) + (X_{A,0} + \Delta X_A) - m \cdot g \cdot sin(\theta_0 + \Delta\theta) \end{aligned} \tag{6.18}$$

– **Step 3:** Expand terms.

Each term in the Bryan equation is expanded around the operating point and only the first-order (linear) terms are retained. Some properties for trigonometric relations are taken:

$$
\begin{aligned}
&sin(\Delta\theta) \approx \Delta\theta \\
&cos(\Delta\theta) \approx 1 \\
&sin(\theta_0+\Delta\theta) = sin\theta_0 \cdot cos\Delta\theta + cos\theta_0 \cdot sin\Delta\theta \\
&sin(\theta_0+\Delta\theta) = sin\theta_0 + cos\theta_0 \cdot \Delta\theta_0 \\
&cos(\theta_0+\Delta\theta) = cos\theta_0 \cdot cos\Delta\theta - sin\theta_0 \cdot sin\Delta\theta \\
&cos(\theta_0+\Delta\theta) = cos\theta_0 - sin\theta_0 \cdot \Delta\theta
\end{aligned}
$$

Expanding the perturbed equation (6.18):

$$
\begin{aligned}
m \cdot [\dot{u}_0 + \Delta\dot{u} + (q_0 + \Delta q) \cdot (w_0 + \Delta w) - (r_0 + \Delta r) \cdot (v_0 + \Delta v)] = \\
= (T_{x,0} + \Delta T_x) + (X_{A,0} + \Delta X_A) - m \cdot g \cdot (sin\theta_0 + cos\theta_0 \cdot \Delta\theta)
\end{aligned}
$$

Considerations from Step 1 are included.

$$
\begin{aligned}
m \cdot (\Delta\dot{u} + \Delta q \cdot \Delta w - \Delta r \cdot \Delta v) = \\
= (T_{x,0} + \Delta T_x) + (X_{A,0} + \Delta X_A) - m \cdot g \cdot sin\theta_0 - m \cdot g \cdot cos\theta_0 \cdot \Delta\theta
\end{aligned}
$$

The product terms given by Δx^2 are discarded as they are considered to have little magnitude:

$$
m \cdot \Delta\dot{u} = (T_{x,0} + \Delta T_x) + (X_{A,0} + \Delta X_A) - m \cdot g \cdot sin\theta_0 - m \cdot g \cdot cos\theta_0 \cdot \Delta\theta
$$

Considerations from Trim equation (6.17) are included, so the linearized first Bryan equation is obtained:

$$
m \cdot \Delta\dot{u} = -m \cdot g \cdot cos\theta_0 \cdot \Delta\theta + \Delta T_x + \Delta X_A \tag{6.19}
$$

The twelve linearized Bryan Equations are the following:

$$m \cdot \Delta\dot{u} = -m \cdot g \cdot cos\theta_0 \cdot \Delta\theta + \Delta T_x + \Delta X_A \tag{6.19}$$

$$m \cdot \Delta\dot{v} = -m \cdot V_0 \cdot \Delta r + m \cdot g \cdot \cos\theta_0 \cdot \Delta\phi \cdot \Delta Y_A \tag{6.20}$$

$$m \cdot \Delta\dot{w} = m \cdot V_0 \cdot \Delta q - m \cdot g \cdot \sin\theta_0 \cdot \Delta\theta + \Delta Z_A \tag{6.21}$$

$$\Delta\dot{p} = \frac{I_{zz}}{A} \cdot \Delta L + \frac{I_{xz}}{A} \cdot \Delta N \tag{6.22}$$

$$\Delta\dot{q} = \frac{\Delta M}{I_{yy}} \tag{6.23}$$

$$\Delta\dot{r} = \frac{I_{xz}}{A} \cdot \Delta L + \frac{I_{xx}}{A} \cdot \Delta N \tag{6.24}$$

$$\Delta\dot{\phi} \approx \Delta p + \tan\theta_0 \cdot \Delta r \tag{6.25}$$

$$\Delta\dot{\theta} \approx \Delta q \tag{6.26}$$

$$\Delta\dot{\psi} \approx \frac{\Delta r}{\cos\theta_0} \tag{6.27}$$

$$\dot{x} = V_0 \cdot \left(\cos\theta_0 + \cos\theta_0 \cdot \frac{\Delta u}{V_0} + \sin\theta_0 \cdot \Delta\alpha - \sin\theta_0 \cdot \Delta\theta\right) \tag{6.28}$$

$$\dot{y} = V_0 \cdot (\Delta\beta + \Delta\psi \cdot \cos\theta_0) \tag{6.29}$$

$$\dot{z} = -V_0 \cdot \left(\sin\theta_0 + \sin\theta_0 \cdot \frac{\Delta u}{V_0} + \cos\theta_0 \cdot \Delta\theta - \cos\theta_0 \cdot \Delta\alpha\right) \tag{6.30}$$

Next, a Taylor series expansion is applied to forces and moment equations to obtain a model for the forces and moments that will be included to the linearized equations.

Considering as state variables the linear and angular velocities and as control variables thrust lever position (δ_P) and aileron (δ_A), rudder (δ_R) and elevator (δ_E) deflection:

$$\text{State variables} \begin{cases} u \approx V \\ v \approx \alpha \cdot V \\ w \approx \beta \cdot V \\ p \\ q \\ r \end{cases}$$

$$\text{Controls:} \begin{cases} \delta_A \\ \delta_E \\ \delta_R \\ \delta_P \end{cases}$$

This is the Taylor expansion for one and two variable functions:

$$\text{One variable:} \quad f(x_0+\Delta x) = f(x_0) + \frac{1}{1!}\left.\frac{df}{dx}\right|_{x_0}\cdot\Delta x + \frac{1}{2!}\left.\frac{d^2f}{dx^2}\right|_{x_0}\cdot(\Delta x)^2 + \frac{1}{3!}\left.\frac{d^3f}{dx^3}\right|_{x_0}\cdot(\Delta x)^3 + \cdots + \frac{1}{n!}\left.\frac{d^nf}{dx^n}\right|_{x_0}\cdot(\Delta x)^n$$

$$\text{Two variables:} \quad f(x_0+\Delta x, y_0+\Delta y) = f(x_0,y_0) + \frac{1}{1!}\left(\left.\frac{\partial f}{\partial x}\right|_{x_0,y_0}\cdot\Delta x + \left.\frac{\partial f}{\partial y}\right|_{x_0,y_0}\cdot\Delta y\right) + \frac{1}{2!}\left[\left.\frac{\partial^2 f}{\partial x^2}\right|_{x_0,y_0}\cdot(\Delta x)^2 + \left.\frac{\partial^2 f}{\partial x\partial y}\right|_{x_0,y_0}\cdot\Delta x\cdot\Delta y + \left.\frac{\partial^2 f}{\partial y^2}\right|_{x_0,y_0}\cdot(\Delta y)^2\right] + \cdots$$

By using this nomenclature:

$$X_u = \left(\frac{\partial X}{\partial u}\right)_0$$

Finally, after applying simplifications based on geometry, symmetry and physical sense, and adding to Z and M equations effects related to acceleration in Z axis $\dot{w}$, a model of forces and moments is obtained:

$$\Delta X + \Delta T_X = X_u\cdot\Delta u + X_w\cdot\Delta w + X_q\cdot\Delta q + T_{X,\delta_P}\cdot\Delta\delta_P \tag{6.31}$$
$$\Delta Y = Y_v\cdot\Delta v + Y_p\cdot\Delta p + Y_r\cdot\Delta r + Y_{\delta_A}\cdot\Delta\delta_A + Y_{\delta_R}\cdot\Delta\delta_R \tag{6.32}$$
$$\Delta Z = Z_u\cdot\Delta u + Z_w\cdot\Delta w + Z_q\cdot\Delta q + Z_{\dot{w}}\cdot\Delta\dot{w} + Z_{\delta_E}\cdot\Delta\delta_E \tag{6.33}$$

$$\Delta L = L_v\cdot\Delta v + L_p\cdot\Delta p + L_r\cdot\Delta r + L_{\delta_A}\cdot\Delta\delta_A + L_{\delta_R}\cdot\Delta\delta_R \tag{6.34}$$
$$\Delta M = M_u\cdot\Delta u + M_w\cdot\Delta w + M_q\cdot\Delta q + M_{\dot{w}}\cdot\Delta\dot{w} + M_{\delta_E}\cdot\Delta\delta_E \tag{6.35}$$
$$\Delta N = N_v\cdot\Delta v + N_p\cdot\Delta p + N_r\cdot\Delta r + N_{\delta_A}\cdot\Delta\delta_A + N_{\delta_R}\cdot\Delta\delta_R \tag{6.36}$$

Note that the subscript of aerodynamic forces has been removed for convenience $X_A \to X$.

From this set it can be derived that longitudinal and lateral-directional dynamics are uncoupled.

Linearized equations (6.19 to 6.30) are manipulated to include the model previously defined (6.31 to 6.36). Thus, they can be divided into the longitudinal system and the lateral-directional system.

Longitudinal system (variables Δu, Δw, Δq, $\Delta\theta$):

$$\Delta\dot{u} = \frac{X_u}{m}\cdot\Delta u + \frac{X_w}{m}\cdot\Delta w + \frac{X_q}{m}\cdot\Delta q - g\cdot\cos\theta_0\cdot\Delta\theta + \frac{T_{X,\delta_P}}{m}\cdot\Delta\delta_P \tag{6.37}$$

$$\Delta\dot{w} = \frac{Z_u}{m-Z_{\dot{w}}}\cdot\Delta u + \frac{Z_w}{m-Z_{\dot{w}}}\cdot\Delta w + \frac{Z_q + m\cdot V_0}{m-Z_{\dot{w}}}\cdot\Delta q - \frac{m\cdot g\cdot\sin\theta_0}{m-Z_{\dot{w}}}\cdot\Delta\theta + \frac{Z_{\delta_E}}{m-Z_{\dot{w}}}\cdot\Delta\delta_E \tag{6.38}$$

$$\Delta\dot{q} - \frac{M_{\dot{w}}}{I_{yy}}\cdot\Delta\dot{w} = \frac{M_u}{I_{yy}}\cdot\Delta u + \frac{M_w}{I_{yy}}\cdot\Delta w + \frac{M_q}{I_{yy}}\cdot\Delta q + \frac{M_{\delta_E}}{I_{yy}}\cdot\Delta\delta_E \tag{6.39}$$

$$\Delta\dot{\theta} \approx \Delta q \tag{6.40}$$

Lateral-directional system (variables Δv, Δp, Δr, $\Delta\phi$):

$$\Delta\dot{v} = g\cdot\cos\theta_0\cdot\Delta\phi + \frac{Y_v}{m}\cdot\Delta v + \frac{Y_p}{m}\cdot\Delta p \tag{6.41}$$

$$+ (\frac{Y_r}{m} - V_0)\cdot\Delta r + \frac{Y_{\delta_A}}{m}\cdot\Delta\delta_A + \frac{Y_{\delta_R}}{m}\cdot\Delta\delta_R \tag{6.42}$$

$$\Delta\dot{p} = \frac{I_{zz}}{A}\left(L_v\cdot\Delta v + L_p\cdot\Delta p + L_r\cdot\Delta r + L_{\delta A}\cdot\Delta\delta_A + L_{\delta R}\cdot\Delta\delta_R\right) \tag{6.43}$$

$$+ \frac{I_{xz}}{A}\left(N_v\cdot\Delta v + N_p\cdot\Delta p + N_r\cdot\Delta r + N_{\delta A}\cdot\Delta\delta_A + N_{\delta R}\cdot\Delta\delta_R\right) \tag{6.44}$$

$$\Delta\dot{r} = \frac{I_{xz}}{A}\left(L_v\cdot\Delta v + L_p\cdot\Delta p + L_r\cdot\Delta r + L_{\delta A}\cdot\Delta\delta_A + L_{\delta R}\cdot\Delta\delta_R\right) \tag{6.45}$$

$$+ \frac{I_{xx}}{A}\left(N_v\cdot\Delta v + N_p\cdot\Delta p + N_r\cdot\Delta r + N_{\delta A}\cdot\Delta\delta_A + N_{\delta R}\cdot\Delta\delta_R\right) \tag{6.46}$$

$$\Delta\dot{\phi} \approx \Delta p + \tan\theta_0\cdot\Delta r \tag{6.47}$$

State space matrices

These equations can be written in the form of state space representation $\dot{x} = A\cdot x + B\cdot u$:

$$\begin{bmatrix}\Delta\dot{u}\\ \Delta\dot{w}\\ \Delta\dot{q}\\ \Delta\dot{\theta}\end{bmatrix} = \begin{bmatrix} \frac{X_u}{m} & \frac{X_w}{m} & \frac{X_q}{m} & -g\cos\theta_0 \\ \frac{Z_u}{m-Z_{\dot{w}}} & \frac{Z_w}{m-Z_{\dot{w}}} & \frac{Z_q+mV_0}{m-Z_{\dot{w}}} & -\frac{mg\sin\theta_0}{m-Z_{\dot{w}}} \\ \frac{M_u}{I_{yy}} + \frac{M_{\dot{w}}Z_u}{I_{yy}(m-Z_{\dot{w}})} & \frac{M_w}{I_{yy}} + \frac{M_{\dot{w}}Z_w}{I_{yy}(m-Z_{\dot{w}})} & \frac{M_q}{I_{yy}} + \frac{M_{\dot{w}}(mV_0+Z_q)}{I_{yy}(m-Z_{\dot{w}})} & -\frac{mg\sin\theta_0 M_{\dot{w}}}{I_{yy}(m-Z_{\dot{w}})} \\ 0 & 0 & 1 & 0 \end{bmatrix}\begin{bmatrix}\Delta u\\ \Delta w\\ \Delta q\\ \Delta\theta\end{bmatrix} + \begin{bmatrix} \frac{T_{X,\delta_P}}{m} & 0 \\ 0 & \frac{Z_{\delta_E}}{m-Z_{\dot{w}}} \\ 0 & \frac{M_{\delta_E}}{I_{yy}} + \frac{M_{\dot{w}}Z_{\delta_E}}{I_{yy}(m-Z_{\dot{w}})} \\ 0 & 0 \end{bmatrix}\begin{bmatrix}\Delta\delta_P\\ \Delta\delta_E\end{bmatrix} \tag{6.48}$$

$$\begin{bmatrix} \Delta\dot{v} \\ \Delta\dot{p} \\ \Delta\dot{r} \\ \Delta\dot{\phi} \end{bmatrix} = \begin{bmatrix} \frac{Y_v}{m} & \frac{Y_p}{m} & \frac{Y_r}{m} - U_0 & g\cos\theta_0 \\ \frac{L_v I_{zz} + N_v I_{xz}}{A} & \frac{L_p I_{zz} + N_p I_{xz}}{A} & \frac{L_r I_{zz} + N_r I_{xz}}{A} & 0 \\ \frac{N_v I_{xx} + L_v I_{xz}}{A} & \frac{N_p I_{xx} + L_p I_{xz}}{A} & \frac{N_r I_{xx} + L_r I_{xz}}{A} & 0 \\ 0 & 1 & \tan(\theta_0) & 0 \end{bmatrix} \begin{bmatrix} \Delta v \\ \Delta p \\ \Delta r \\ \Delta\phi \end{bmatrix}$$

$$+ \begin{bmatrix} 0 & \frac{Y_{\delta_R}}{m} \\ \frac{L_{\delta_A} I_{zz} + N_{\delta_A} I_{xz}}{A} & \frac{L_{\delta_R} I_{zz} + N_{\delta_R} I_{xz}}{A} \\ \frac{N_{\delta_A} I_{xx} + L_{\delta_A} I_{xz}}{A} & \frac{N_{\delta_R} I_{xx} + L_{\delta_R} I_{xz}}{A} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta\delta_A \\ \Delta\delta_R \end{bmatrix} \tag{6.49}$$

$$\text{Note that} : A = (I_{xx} \cdot I_{zz} - I_{xz}^2) \approx \text{constant}$$

The eigenvalues of matrix A indicate the stability of the system at the operating point.

Aerodynamic derivatives

The components of the matrices will be calculated once the aerodynamic derivatives inside them are computed.

The forces and moments can be written as [26]:

$$X_A = \frac{1}{2} \cdot \rho(z) \cdot S_W \cdot [(V_0 + \Delta u)^2 + \Delta v^2 + \Delta w^2] \cdot C_X \tag{6.50}$$

$$Y_A = \frac{1}{2} \cdot \rho(z) \cdot S_W \cdot [(V_0 + \Delta u)^2 + \Delta v^2 + \Delta w^2] \cdot C_Y \tag{6.51}$$

$$Z_A = \frac{1}{2} \cdot \rho(z) \cdot S_W \cdot [(V_0 + \Delta u)^2 + \Delta v^2 + \Delta w^2] \cdot C_Z \tag{6.52}$$

$$L_A = \frac{1}{2} \cdot \rho(z) \cdot S_W \cdot b_W \cdot [(V_0 + \Delta u)^2 + \Delta v^2 + \Delta w^2] \cdot C_l \tag{6.53}$$

$$M_A = \frac{1}{2} \cdot \rho(z) \cdot S_W \cdot c_W \cdot [(V_0 + \Delta u)^2 + \Delta v^2 + \Delta w^2] \cdot C_M \tag{6.54}$$

$$N_A = \frac{1}{2} \cdot \rho(z) \cdot S_W \cdot b_W \cdot [(V_0 + \Delta u)^2 + \Delta v^2 + \Delta w^2] \cdot C_N \tag{6.55}$$

The coefficients of the previous equations are given by:

$$C_X = \frac{\cos\alpha}{\cos\beta} \cdot C_D - \cos\alpha \cdot \tan\beta \cdot C_Y + \sin\alpha \cdot C_L \tag{6.56}$$

$$C_Y = C_{Y_\beta} \cdot \beta + C_{Y_{\delta_A}} \cdot \delta_A + C_{Y_{\delta_R}} \cdot \delta_R + \frac{b_w}{2 \cdot V} \cdot (C_{Y_{\dot\beta}} \cdot \dot\beta + C_{Y_p} \cdot p + C_{Y_r} \cdot r) \tag{6.57}$$

$$C_Z = \frac{\sin\alpha}{\cos\beta} \cdot C_D - \sin\alpha \cdot \tan\beta \cdot C_Y - \cos\alpha \cdot C_L \tag{6.58}$$

$$C_l = C_{l_\beta} \cdot \beta + C_{l_{\delta_A}} \cdot \delta_A + C_{l_{\delta_R}} \cdot \delta_R + \frac{b_w}{2 \cdot V} \cdot (C_{l_{\dot\beta}} \cdot \dot\beta + C_{l_p} \cdot p + C_{l_r} \cdot r) \tag{6.59}$$

$$C_M = C_{M_0} + C_{M_\alpha} \cdot \alpha + C_{M_{\delta_E}} \cdot \delta_E + \frac{c_w}{2 \cdot V} \cdot (C_{M_{\dot\alpha}} \cdot \dot\alpha + C_{M_q} \cdot q) \tag{6.60}$$

$$C_N = C_{N_\beta} \cdot \beta + C_{N_{\delta_A}} \cdot \delta_A + C_{N_{\delta_R}} \cdot \delta_R + \frac{b_w}{2 \cdot V} \cdot (C_{N_{\dot\beta}} \cdot \dot\beta + C_{N_p} \cdot p + C_{N_r} \cdot r) \tag{6.61}$$

$$C_L = C_{L_0} + C_{L_\alpha} \cdot \alpha + C_{L_{\delta_E}} \cdot \delta_E + \frac{c_w}{2 \cdot V} \cdot (C_{L_{\dot\alpha}} \cdot \dot\alpha + C_{L_q} \cdot q) \tag{6.62}$$

$$C_D = C_{D_0} + C_{D_1} \cdot \alpha + C_{D_2} \cdot \alpha^2 \tag{6.63}$$

These coefficients are used to describe the behaviour of an aircraft independently of the flight conditions.

Once the forces and moment equations include these coefficients, they are derived to obtain the components of A and B matrices, but for one component in the latter matrix related to propulsion.

Propulsive model

A propulsive model is needed for each aircraft engine type. Generally, turbojet and turbofan engines have different models that accomplish good results, but for propeller based engines it is not as straightforward. In this case, propellers are used combined with hydrogen based propulsion.

The model is given by HORUS [2] [27], and is obtained by fitting equations to experimental data.

Thrust and power are defined as well with dimensionless coefficients:

$$T = C_T \cdot \left(\rho \cdot n^2 \cdot D^4\right)$$
$$P = C_P \cdot \left(\rho \cdot n^3 \cdot D^5\right)$$

Here, T and P are thrust and power respectively, n is the blade angular velocity expressed in Hz, and D is the blade diameter.

Blade angular velocity is defined as:

$$n = p_{n_1} \cdot \delta_P$$

The coefficients are given by:

$$
\begin{aligned}
C_T &= p_{C_{T_3}} \cdot J^3 + p_{C_{T_2}} \cdot J^2 + p_{C_{T_1}} \cdot J + p_{C_{T_0}} \\
C_P &= p_{C_{P_6}} \cdot J^6 + p_{C_{P_5}} \cdot J^5 + p_{C_{P_4}} \cdot J^4 \\
&\quad + p_{C_{P_3}} \cdot J^3 + p_{C_{P_2}} \cdot J^2 + p_{C_{P_1}} \cdot J + p_{C_{P_0}}
\end{aligned}
$$

A new term known as non-dimensional induced velocity (J) is introduced, defined as:

$$
J = \frac{V}{D \cdot n} = \frac{V}{D \cdot p_{n_1} \cdot \delta_P}
$$

Summarizing, this is the propulsive model for H200 aircraft, which depends on thrust lever position (δ_P):

$$
T(\delta_P) = C_T(\delta_P) \cdot \left(\rho \cdot D^4 \cdot n^2(\delta_P)\right) \tag{6.64}
$$

$$
P(\delta_P) = C_P(\delta_P) \cdot \left(\rho \cdot D^5 \cdot n^3(\delta_P)\right) \tag{6.65}
$$

$$
C_T(\delta_P) = p_{C_{T_3}} \cdot J^3(\delta_P) + p_{C_{T_2}} \cdot J^2(\delta_P) + p_{C_{T_1}} \cdot J(\delta_P) + p_{C_{T_0}} \tag{6.66}
$$

$$
C_P(\delta_P) = p_{C_{P_6}} \cdot J^6(\delta_P) + p_{C_{P_5}} \cdot J^5(\delta_P) + p_{C_{P_4}} \cdot J^4(\delta_P) \tag{6.67}
$$

$$
+ p_{C_{P_3}} \cdot J^3(\delta_P) + p_{C_{P_2}} \cdot J^2(\delta_P) + p_{C_{P_1}} \cdot J(\delta_P) + p_{C_{P_0}} \tag{6.68}
$$

$$
J(\delta_P) = \frac{V}{D \cdot n(\delta_P)} \tag{6.69}
$$

$$
n(\delta_P) = p_{n_1} \cdot \delta_P \tag{6.70}
$$

6.2 Implementation

This section describes the theoretical background implementation into the project, and additional considerations that arise during this process. Moreover, concepts needed for the understanding of the implementation are introduced.

6.2.1 Considerations

Some considerations are to be first taken into account to understand the followed implementation.

State space matrices

There is no reading of airspeed provided by the sensors, which means that angle of attack α and drift angle β cannot be provided to the model seen in Section 6.1.6. Moreover, aerodynamic speed (V) is managed by TECS (3.2.3) algorithm. Therefore, longitudinal system will not include u and w, and lateral system v.

Recalling the longitudinal state space matrices (Equation 6.48), the first and second rows and columns are cancelled for state matrix, and first and second rows and first column are cancelled for input matrix:

$$\begin{bmatrix} \cancel{\Delta\dot{u}} \\ \cancel{\Delta\dot{w}} \\ \Delta\dot{q} \\ \Delta\dot{\theta} \end{bmatrix} = \begin{bmatrix} \cancel{\frac{X_u}{m}} & \cancel{\frac{X_w}{m}} & \cancel{\frac{X_q}{m}} & \cancel{-g\cos\theta_0} \\ \cancel{\frac{Z_u}{m-Z_{\dot{w}}}} & \cancel{\frac{Z_w}{m-Z_{\dot{w}}}} & \cancel{\frac{Z_q+mV_0}{m-Z_{\dot{w}}}} & \cancel{-\frac{mg\sin\theta_0}{m-Z_{\dot{w}}}} \\ \cancel{\frac{M_u}{I_{yy}} + \frac{M_{\dot{w}}Z_u}{I_{yy}(m-Z_{\dot{w}})}} & \cancel{\frac{M_w}{I_{yy}} + \frac{M_{\dot{w}}Z_w}{I_{yy}(m-Z_{\dot{w}})}} & \frac{M_q}{I_{yy}} + \frac{M_{\dot{w}}(mV_0+Z_q)}{I_{yy}(m-Z_{\dot{w}})} & -\frac{mg\sin\theta_0 M_{\dot{w}}}{I_{yy}(m-Z_{\dot{w}})} \\ \emptyset & \emptyset & 1 & 0 \end{bmatrix} \begin{bmatrix} \cancel{\Delta u} \\ \cancel{\Delta w} \\ \Delta q \\ \Delta\theta \end{bmatrix}$$

$$+ \begin{bmatrix} \cancel{\frac{T_{X,\delta_P}}{m}} & \emptyset \\ \emptyset & \cancel{\frac{Z_{\delta_E}}{m-Z_{\dot{w}}}} \\ \emptyset & \frac{M_{\delta_E}}{I_{yy}} + \frac{M_{\dot{w}}Z_{\delta_E}}{I_{yy}(m-Z_{\dot{w}})} \\ \emptyset & 0 \end{bmatrix} \begin{bmatrix} \cancel{\Delta\delta_P} \\ \Delta\delta_E \end{bmatrix}$$

For the lateral-directional case (Equation 6.49), first row and column is cancelled for the state matrix, and first row is cancelled for the input matrix:

$$\begin{bmatrix} \cancel{\Delta\dot{v}} \\ \Delta\dot{p} \\ \Delta\dot{r} \\ \Delta\dot{\phi} \end{bmatrix} = \begin{bmatrix} \cancel{\frac{Y_v}{m}} & \cancel{\frac{Y_p}{m}} & \cancel{\frac{Y_r}{m} - U_0} & \cancel{g\cos\theta_0} \\ \cancel{\frac{L_vI_{zz}+N_vI_{xz}}{A}} & \frac{L_pI_{zz}+N_pI_{xz}}{A} & \frac{L_rI_{zz}+N_rI_{xz}}{A} & 0 \\ \cancel{\frac{N_vI_{xx}+L_vI_{xz}}{A}} & \frac{N_pI_{xx}+L_pI_{xz}}{A} & \frac{N_rI_{xx}+L_rI_{xz}}{A} & 0 \\ \emptyset & 1 & \tan(\theta_0) & 0 \end{bmatrix} \begin{bmatrix} \cancel{\Delta v} \\ \Delta p \\ \Delta r \\ \Delta\phi \end{bmatrix}$$

$$+ \begin{bmatrix} \emptyset & \cancel{\frac{Y_{\delta_R}}{m}} \\ \frac{L_{\delta_A}I_{zz}+N_{\delta_A}I_{xz}}{A} & \frac{L_{\delta_R}I_{zz}+N_{\delta_R}I_{xz}}{A} \\ \frac{N_{\delta_A}I_{xx}+L_{\delta_A}I_{xz}}{A} & \frac{N_{\delta_R}I_{xx}+L_{\delta_R}I_{xz}}{A} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta\delta_A \\ \Delta\delta_R \end{bmatrix}$$

Therefore, the resulting matrices are the following:

$$\begin{bmatrix} \Delta\dot{q} \\ \Delta\dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{M_q}{I_{yy}} + \frac{M_{\dot{w}}(mV_0+Z_q)}{I_{yy}(m-Z_{\dot{w}})} & -\frac{mg\sin\theta_0 M_{\dot{w}}}{I_{yy}(m-Z_{\dot{w}})} \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \Delta q \\ \Delta\theta \end{bmatrix}$$

$$+ \begin{bmatrix} \frac{M_{\delta_E}}{I_{yy}} + \frac{M_{\dot{w}}Z_{\delta_E}}{I_{yy}(m-Z_{\dot{w}})} \\ 0 \end{bmatrix} \begin{bmatrix} \Delta\delta_E \end{bmatrix} \tag{6.71}$$

$$\begin{bmatrix} \Delta\dot{p} \\ \Delta\dot{r} \\ \Delta\dot{\phi} \end{bmatrix} = \begin{bmatrix} \frac{L_pI_{zz}+N_pI_{xz}}{A} & \frac{L_rI_{zz}+N_rI_{xz}}{A} & 0 \\ \frac{N_pI_{xx}+L_pI_{xz}}{A} & \frac{N_rI_{xx}+L_rI_{xz}}{A} & 0 \\ 1 & \tan(\theta_0) & 0 \end{bmatrix} \begin{bmatrix} \Delta p \\ \Delta r \\ \Delta\phi \end{bmatrix}$$

$$+ \begin{bmatrix} \frac{L_{\delta_A}I_{zz}+N_{\delta_A}I_{xz}}{A} & \frac{L_{\delta_R}I_{zz}+N_{\delta_R}I_{xz}}{A} \\ \frac{N_{\delta_A}I_{xx}+L_{\delta_A}I_{xz}}{A} & \frac{N_{\delta_R}I_{xx}+L_{\delta_R}I_{xz}}{A} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta\delta_A \\ \Delta\delta_R \end{bmatrix} \tag{6.72}$$

Integral Action addition

The longitudinal system includes the pitch error state which follows the pitch reference given by TECS algorithm (3.2.3):

$$\begin{bmatrix} \Delta\dot{q} \\ \Delta\dot{\theta} \\ \dot{z}_\theta \end{bmatrix} = \begin{bmatrix} \frac{M_q}{I_{yy}} + \frac{M_{\dot{w}}(mV_0+Z_q)}{I_{yy}(m-Z_{\dot{w}})} & -\frac{mg\sin\theta_0 M_{\dot{w}}}{I_{yy}(m-Z_{\dot{w}})} & 0 \\ 1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} \Delta q \\ \Delta\theta \\ z_\theta \end{bmatrix} + \begin{bmatrix} \frac{M_{\delta_E}}{I_{yy}} + \frac{M_{\dot{w}} Z_{\delta_E}}{I_{yy}(m-Z_{\dot{w}})} \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} \Delta\delta_E \end{bmatrix} \tag{6.73}$$

For the lateral-directional system, only the roll error is needed as changing the heading is accomplished by rolling the aircraft. The setpoint comes from FixedWing-Position script, which forms part of the Position Controller (3.2.3).

$$\begin{bmatrix} \Delta\dot{p} \\ \Delta\dot{r} \\ \Delta\dot{\phi} \\ \dot{z}_\phi \end{bmatrix} = \begin{bmatrix} \frac{L_p I_{zz}+N_p I_{xz}}{A} & \frac{L_r I_{zz}+N_r I_{xz}}{A} & 0 & 0 \\ \frac{N_p I_{xx}+L_p I_{xz}}{A} & \frac{N_r I_{xx}+L_r I_{xz}}{A} & 0 & 0 \\ 1 & \tan(\theta_0) & 0 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} \Delta p \\ \Delta r \\ \Delta\phi \\ z_\phi \end{bmatrix} + \begin{bmatrix} \frac{L_{\delta_A} I_{zz}+N_{\delta_A} I_{xz}}{A} & \frac{L_{\delta_R} I_{zz}+N_{\delta_R} I_{xz}}{A} \\ \frac{N_{\delta_A} I_{xx}+L_{\delta_A} I_{xz}}{A} & \frac{N_{\delta_R} I_{xx}+L_{\delta_R} I_{xz}}{A} \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta\delta_A \\ \Delta\delta_R \end{bmatrix} \tag{6.74}$$

PX4 communications

The software will be running in Linux connected to Windows via WSL2. This connection will enable applications like Visual Studio Code to modify the files stored in Linux, or communication with MATLAB and QGroundControl.

6.2.2 Scheme

With the previous information, Figure 6.3 shows the experimental implementation control scheme:

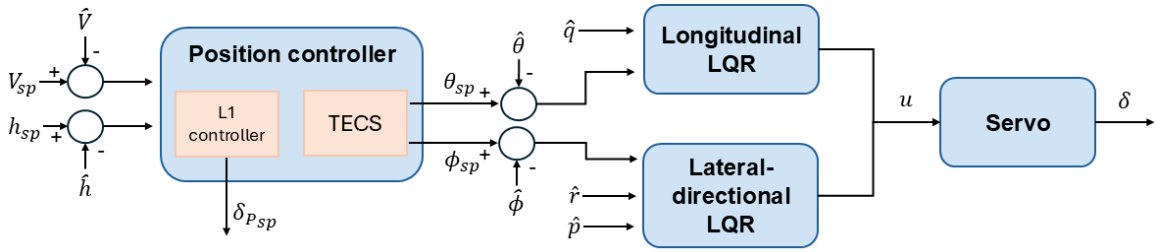


Figure 6.3: Control scheme of experimental implementation. Source: own

where u is representing actuators output and δ actuators deflection. Estimated variables ($\hat{x}$), are obtained from an Extended Kalman filter.

6.2.3 State space matrices calculation

To obtain the linearized model parameters of Bryan equations expressed in state space matrices, a MATLAB script developed by HORUS is used. The input for this function is the H200 model and the operating point. This operating point (cruise flight condition) is set at $V = 18\ m/s$ and $h = 100\ m$.

H200 model

Here are the different parameters that describe the H200 model, seen in Tables 6.1, 6.2, 6.3 and 6.4.

Geometric parameters			
Parameter	**Symbol**	**Value**	**Units**
Wingspan	b_w	2.95	m
Mean aerodynamic chord	c_w	0.3731	m
Wing surface	S_w	1.0162	m^2
Reference mass	m	15	kg
Elevator arm	L_{elev}	1.1	m
Propellers arm	L_{prop}	0	m
X-Axis inertia	I_{xx}	1.609	kg/m^3
Y-Axis inertia	I_{yy}	2.773	kg/m^3
Z-Axis inertia	I_{zz}	4.310	kg/m^3
X-Y axis cross inertia	I_{xy}	$-4.804 \cdot 10^{-3}$	kg/m^3
X-Z axis cross inertia	I_{xz}	$-8.232 \cdot 10^{-2}$	kg/m^3
Y-Z axis cross inertia	I_{yz}	$-9.468 \cdot 10^{-4}$	kg/m^3

Table 6.1: Geometric parameters of H200 aircraft model

Note that propeller arm is 0, so it does not generate pitch moment (M).

Aerodynamic forces coefficients					
C_D		C_Y		C_L	
C_{D_0}	0.039	C_{Y_β}	-0.206777	C_{L_0}	0.308
C_{D_1}	0.007	C_{Y_p}	0.016541	C_{L_α}	5.140879
C_{D_2}	0.057	C_{Y_r}	0.126227	$C_{L_{\dot{\alpha}}}$	0.667
		$C_{Y_{\delta_A}}$	-0.000410	C_{L_q}	7.326102
		$C_{Y_{\delta_R}}$	-0.003172	$C_{L_{\delta_E}}$	0.007585
				$C_{L_{\delta_F}}$	0.013771

Table 6.2: Aerodynamic forces parameters of H200 aircraft model

Aerodynamic moments coefficients					
C_l		C_M		C_N	
C_{l_β}	-0.105871	C_{M_0}	0.00835	C_{n_β}	0.031839
C_{l_p}	-0.476318	C_{m_α}	-0.507412	C_{n_p}	-0.046901
C_{l_r}	0.057	$C_{m_{\dot{\alpha}}}$	-2.1287	C_{n_r}	-0.06186
$C_{l_{\delta_A}}$	-0.003786	C_{m_q}	-7.275333	$C_{n_{\delta_A}}$	-0.000072
$C_{l_{\delta_R}}$	-0.000452	$C_{m_{\delta_E}}$	-0.0185	$C_{n_{\delta_R}}$	0.001064
		$C_{m_{\delta_F}}$	-0.001192		

Table 6.3: Aerodynamic moments parameters of H200 aircraft model

Propulsive model coefficients							
D	0.3302	p_{n_1}	179.9970	p_{CT_3}	0.07115	p_{CP_6}	0.1446
				p_{CT_2}	-0.1954	p_{CP_5}	0.0126
				p_{CT_1}	-0.02019	p_{CP_4}	-0.4078
				p_{CT_0}	0.1068	p_{CP_3}	0.2859
						p_{CP_2}	-0.1337
						p_{CP_1}	0.0424
						p_{CP_0}	0.03482

Table 6.4: Propulsive coefficients of H200 aircraft model

Longitudinal system

The results for the longitudinal system are:

$$A_{longitudinal} = \begin{bmatrix} -2.5830 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \quad B_{longitudinal} = \begin{bmatrix} 17.7474 \\ 0 \\ 0 \end{bmatrix} \tag{6.75}$$

Moreover, as the initial pitch angle and elevator deflection are not zero, their values will be taken into account for the calculations:

$$\theta_{Trim} = 4.759^\circ$$
$$\delta_{E,Trim} = 2.925^\circ$$

Lateral-directional system

The results for the lateral-directional system are:

$$A_{lateral} = \begin{bmatrix} -14.2934 & 4.4844 & 0 & 0 \\ -0.5254 & -0.6930 & 0 & 0 \\ 1 & 0.0832 & 0 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix} \quad B_{lateral} = \begin{bmatrix} 41.5931 & -1.4594 \\ 0.2953 & 1.2825 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{6.76}$$

6.2.4 Simulink LQR simulation

To check if the matrices are properly obtained, calculations are made with MATLAB and the LQR with integral action is simulated using Simulink.

Procedure

First, the state space model (A, B, C and D matrices) is defined. Then, by using *ctrb* function, the system is checked to be controllable. This is to check that the integral action is implemented correctly. The function has the controllability matrix (6.1.2) calculation integrated. Listing 6.1 shows this function being used.

```
Co = ctrb(A, B);
rank_Co = rank(Co);

if rank_Co < size(A, 1)
    disp('System is not controllable');
    return
end
```

Listing 6.1: Controllability check in MATLAB

After that, the weighting matrices Q and R are defined. To obtain the state feedback gain K, a built-in MATLAB function named *lqr* is employed. This takes as an input A, B, Q and R matrices (model and weighting factors), and outputs the state feedback gain. In the process, it minimizes the cost function seen in Subsection 5.3.3. It can be seen in Listing 6.2.

```
K = lqr(Ahat,Bhat,Qhat,Rhat);
```

Listing 6.2: Feedback gain calculation in MATLAB

where *Ahat* and *Bhat* are the state space matrices augmented with integral action and *Qhat*, and *Rhat* are the augmented weighting matrices.

The following step is to set initial conditions and required setpoints for the system that is simulated. Listing 6.3 gives an example of initial conditions and setpoints for the longitudinal system.

```
initialq = deg2rad(0);
initialtheta = deg2rad(0);
x0 = [initialq,initialtheta];

initialthetasetpoint = deg2rad(0);
finalthetasetpoint = deg2rad(12);
```

Listing 6.3: Example of initial conditions and setpoints for the longitudinal system

Finally, the model is simulated and plots are obtained.

Longitudinal system

In Figure 6.4 the schematics of the controller are shown for the longitudinal system, where integral action is applied through an integrator block to the pitch error.

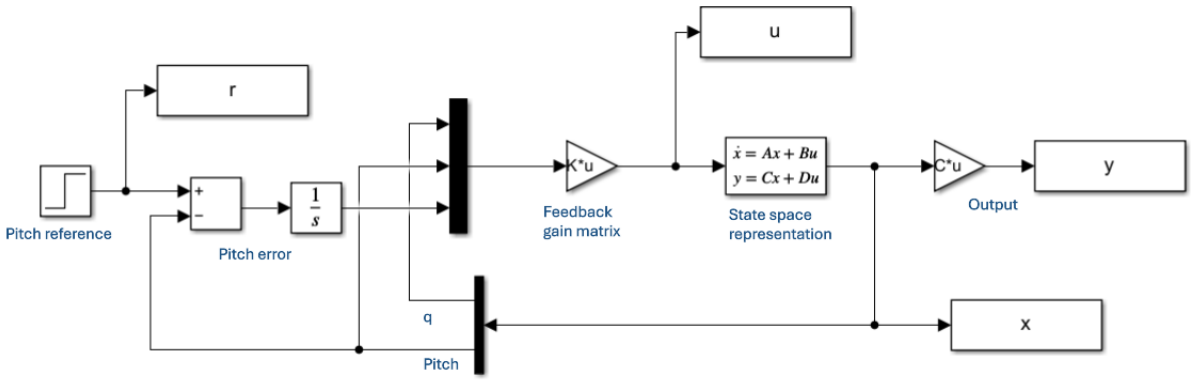
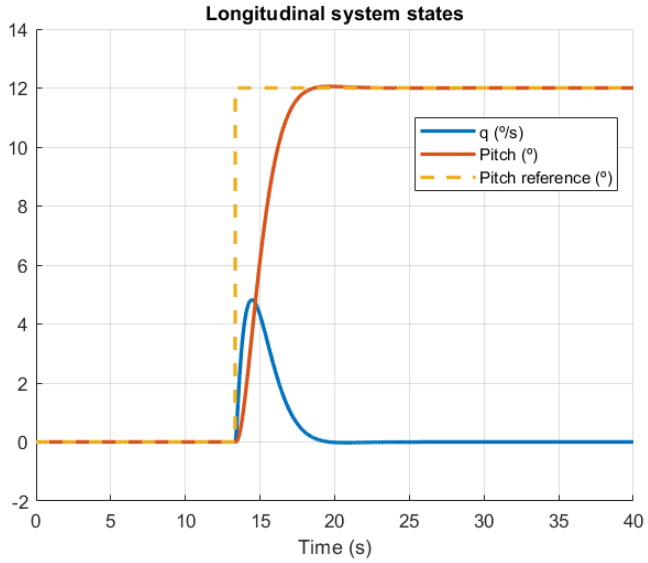


Figure 6.4: Simulink scheme of longitudinal system LQR with integral action. Source: own

For this first approach Q and R are set to identity matrices, the initial conditions to zero. With this, the gain matrix becomes:

$$K = \begin{bmatrix} 0.960 & 1.792 & -0.999 \end{bmatrix}$$

Pitch setpoint is initially 0° and changes to 12° mid-simulation. Figure 6.5 shows the results of the simulation.



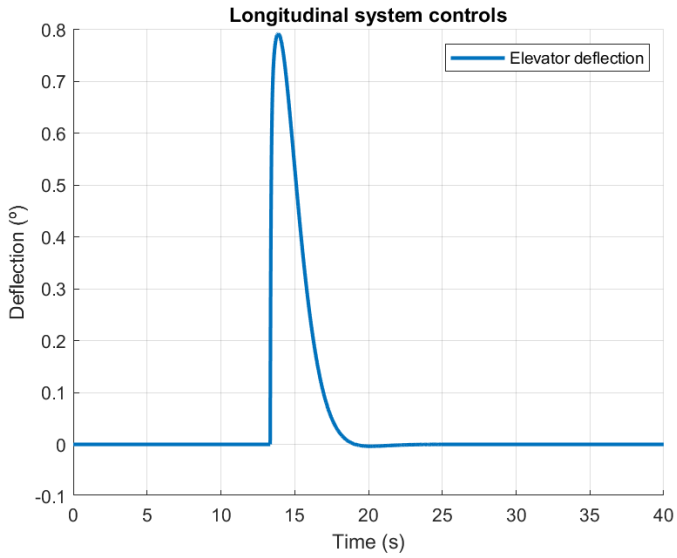


Figure 6.5: Longitudinal system simulation results. Source: own

It can be seen that, initially, the elevator is not deflected, and the states stay at zero. When the pitch setpoint is changed, the elevator is deflected and pitch angular velocity is generated until reaching the desired pitch angle.

Lateral-directional system

For the lateral-directional system, Figure 6.6 shows the Simulink diagram.

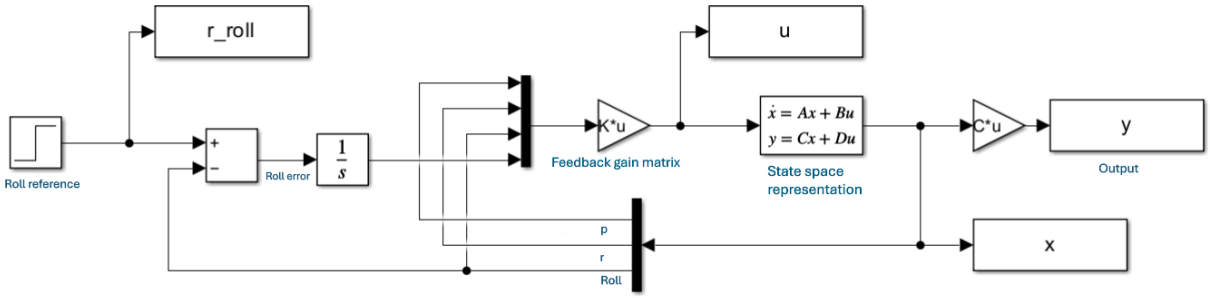


Figure 6.6: Simulink scheme of lateral system LQR with integral action. Source: own

With same weights as the longitudinal system, the gain matrix is the following:

$$K = \begin{bmatrix} 0.752 & 0.098 & 1.792 & -0.997 \\ -0.028 & 0.604 & 0.201 & -0.072 \end{bmatrix}$$

The simulation, yield and roll setpoint initially at 0° and finally at 25°, is shown in Figure 6.7.

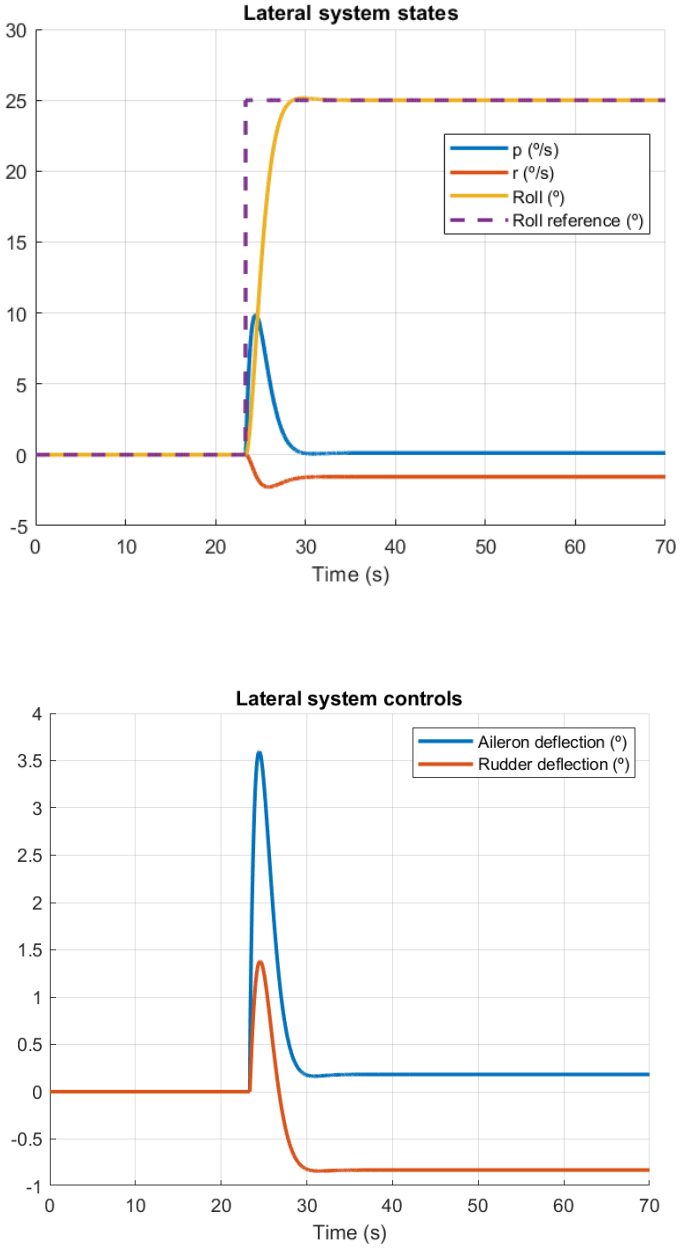


Figure 6.7: Lateral system simulation results. Source: own

From the results, similar conclusions are drawn to longitudinal system. Control action only appears when roll setpoint changes, and achieves the setpoint value without steady-state error. The result is that some yaw angular rate is generated, and thus the heading is changed throughout this process. Moreover, from the perspective of flight mechanics the results are reasonable, as the aircraft will roll and maintain a small aileron deflection to keep roll angle constant, and a small rudder deflection to generate some lift at the back of the aircraft.

Therefore, it can be concluded that the model and integral action matrices are correct.

6.2.5 SITL simulation

To perform a Software in The Loop (SITL) simulation of PX4 stack, the required steps are the following:

1. Compile PX4 in WSL2 with *make px4_sitl none_plane* command
2. Load H200 model variables
3. Run Simulink Dynamic H200 model
4. Open QGroundControl

The command compiles PX4 stack in SITL mode (*px4_sitl*), without any included simulator and in fixed-wing mode (*none_plane*). This enables using a custom simulator, in this case implemented in Simulink by HORUS. Its conceptual general diagram is shown in Figure 6.8.

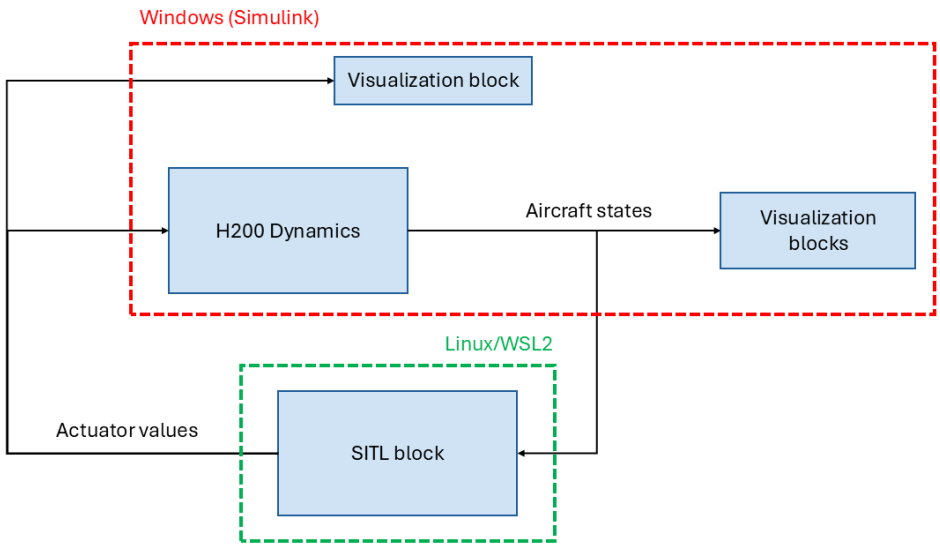


Figure 6.8: Conceptual diagram of SITL simulator. Source: own

Here, PX4 communicates with Simulink via TCP using port 4560, and receives the current aircraft states via the SITL block. PX4 uses this information to calculate actuator commands, which are then returned to the SITL block. Finally, the SITL block forwards these commands to the dynamic model, which computes the updated aircraft states. Moreover, SITL block communicates with QGroundControl via UDP using port 18570 to display the simulation. The real scheme can be seen in Figure 6.9.

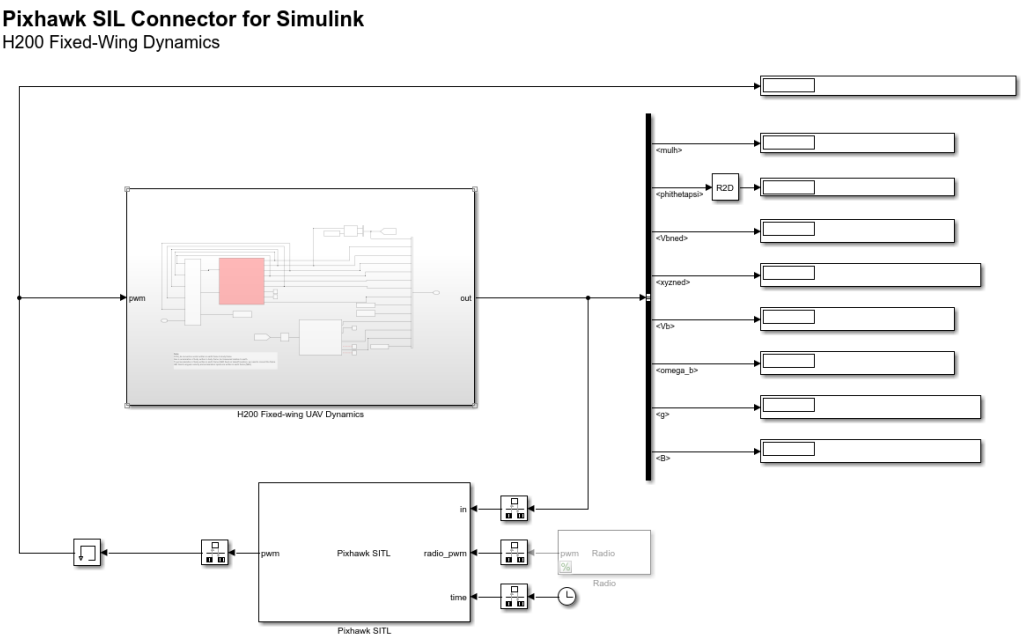


Figure 6.9: Simulator of H200 dynamic model implemented in Simulink [28]. Source: own

After loading the variables of the model and running the simulator it connects to PX4 and the simulation starts. To visualize, send commands and plan missions, QGroundControl is configured to communicate with PX4. Figure 6.10 shows QGroundControl interface connected to PX4:

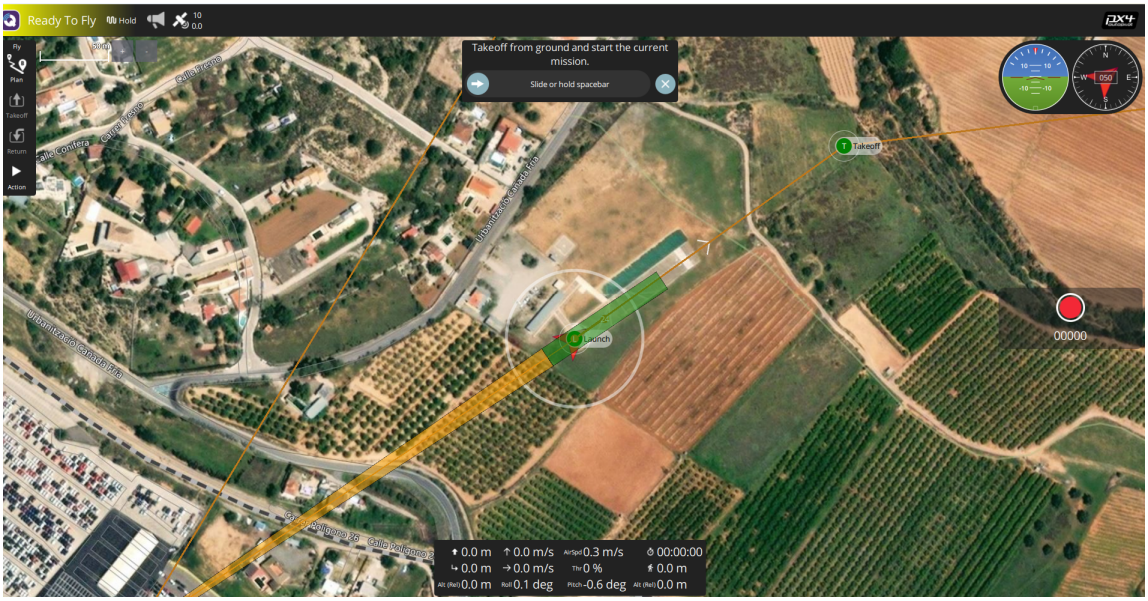


Figure 6.10: Simulation display in QGroundControl. Source: own

6.2.6 PX4 implementation

To understand how to modify the original files, basic concepts about how PX4 is implemented are explained.

File hierarchy

Each module contains at least a header file and a cpp file.

Header files structure is the following:

1. Include all libraries
2. Declare subscriptions to uORB topics
3. Declare publishers to uORB topics
4. Declare parameters
5. Declare functions

On the other hand cpp files have this structure:

1. Define constructor
2. Define destructor
3. Define auxiliary functions
4. Define Run function

In the constructor, param values are fetched initially. To do so, a function named *parameters_update()* is defined. Moreover, Run function is the main loop, which contains all the code that runs at a frequency set for each file. This frequency is useful for implementing the integral action.

Params

Params are variables that can be changed through the MAVLink console in mid-simulation and monitored using QGroundControl Graphical User Interface (GUI). Moreover, they are used in PX4 to store constant values of aircraft or algorithm behaviour.

Through the code implementation, they will be used to store values such as the State feedback gain matrix (K), or the control switch parameter. Listing 6.4 shows the general implementation for each module param file:

```
/**
 * Description
 *
 * @group group_name
 */
PARAM_DEFINE_DATATYPE(PARAM_NAME, INITIAL_VALUE);
```

Listing 6.4: General param implementation

where *group_name* is used to classify the params and *DATATYPE* is replaced with the required datatype (e.g, *INT_32* or *FLOAT*).

In addition, the header file contains a parameter to retrieve the param, as seen in Listing 6.5:

```
DEFINE_PARAMETERS(
  (ParamDatatype<px4::params::PARAM_NAME>) param_retrieval
)
```

Listing 6.5: Param header implementation

where *Datatype* is substituted by the previously defined datatype (for *INT_32* it would be *Int*).

The PX4 code uses class objects to store this params. However, new of them will be implemented with simple variables. To retrieve the param and store it to a variable, the Run function of cpp file will include the code seen in Listing 6.6:

```
param_retrieval.update();

datatype param_storage = param_retrieval.get();
```

Listing 6.6: Param retrieval implementation

Finally, to change a param mid-simulation through MAVLink, the command seen in Listing 6.7 is used:

```
param set PARAM_NAME VALUE
```

Listing 6.7: Param change command using MAVLink protocol

Alternatively, inside QGroundControl software settings, specifically in the Parameters tab of Vehicle Setup, the parameters can be changed in a more intuitively way. There, all parameters are classified into the groups explained above. Figure 6.11 shows this tab.

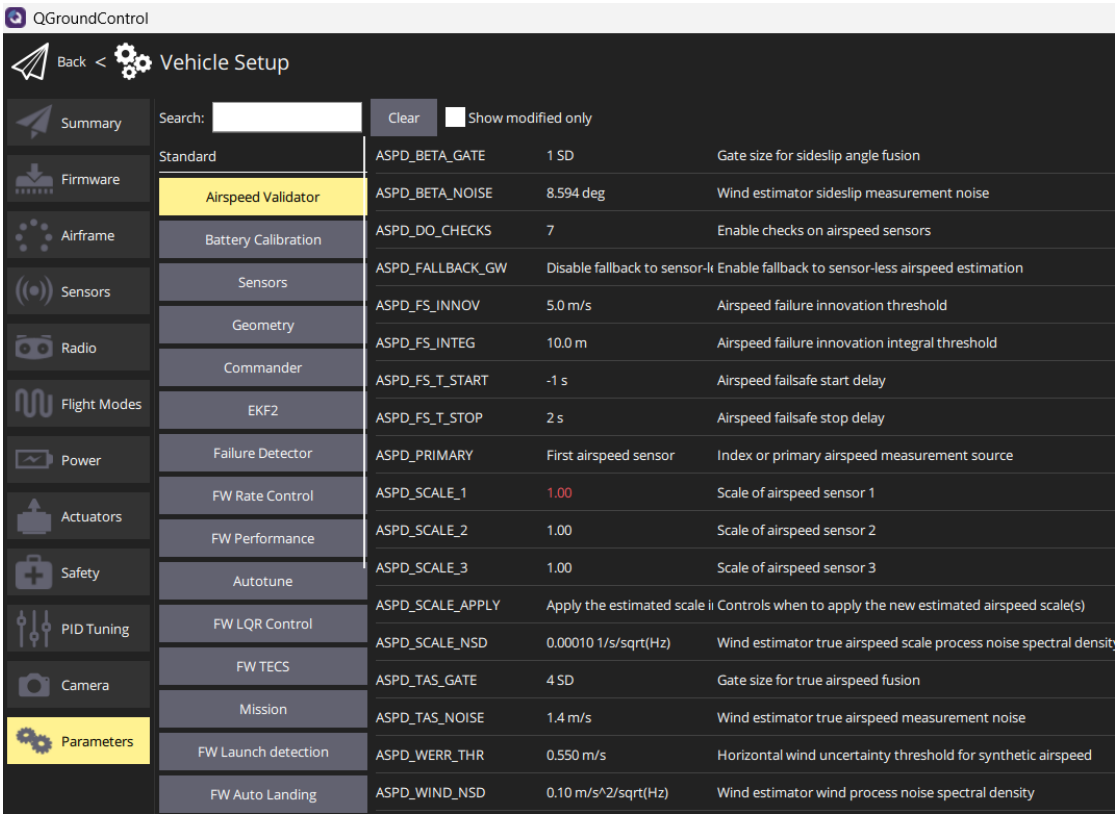


Figure 6.11: Parameters tab of QGroundControl. Source: own

Subscribers

These variables subscribe to uORB topics to retrieve messages that contain information from the different modules. List 6.8 shows the declaration of a subscriber in a header file:

```
#include <uORB/topics/topic_name.h>
uORB::Subscription

_message_sub{ORB_ID(topic_name)};

_message_s     _message{};
```

Listing 6.8: Subscriber header declaration

where *message* and *topic* words are changed by the corresponding ones. Note that the header file for the specific topic has also to be included.

Publishers

These are responsible for publishing the messages related to a topic. List 6.9 shows the declaration of a publisher in a header file:

```
uORB::Publication<message_s>   _message_pub{ORB_ID(topic)};
```

Listing 6.9: Publisher header declaration

Listing 6.10 shows how a variable is published in the Run main loop of a cpp file:

```
_message_pub.publish(variable);
```

Listing 6.10: Message publishing example

Extended Kalman Filter module

The state variables used in the algorithm are estimated variables published to uORB by the Extended Kalman Filter module. How this filter is implemented and its behaviour lies outside the scope of the project, but a brief introduction is given:

The Extended Kalman Filter (EKF) is a state estimation algorithm that estimates the vehicle's states (e.g., position, velocity and attitude) by combining sensor measurements (e.g., GPS, IMU, barometer, and magnetometer) with a mathematical model of the aircraft's dynamics.

It works with a nonlinear state-space model of the aircraft's dynamics:

$$\dot{x} = f(x, u, t) + w \tag{6.77}$$

$$z = h(x, t) + v \tag{6.78}$$

where x represents the states, u is the input vector, z represents measurements from onboard sensors, and finally w and v are process and measurement noise. The former are the uncertainty coming from the measurement of the sensors and from the model used to represent the system.

The EKF approximates the nonlinear model using a linearization process around the current state estimate. Then does a prediction step, in which uses aircraft dynamics to predict the next state and its uncertainty. Finally, it does a correct step in which sensor measurements are used to correct the predicted state, considering sensor noise.

Modified files

Recalling the file structure from Section 3.2.3, the modified files are *FixedwingAttitudeControl*, *FixedwingRateControl* and *fw_rate_control_params*, shown in Figures 6.12 and 6.13.

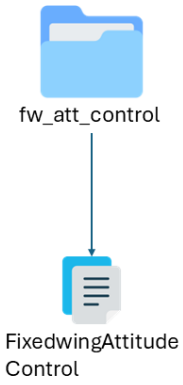


Figure 6.12: Modified Attitude Controller files. Source: own

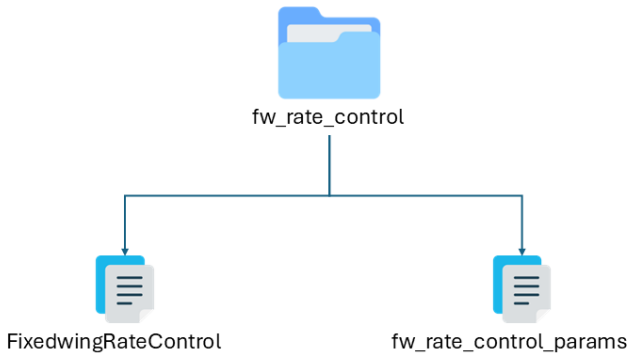


Figure 6.13: Modified Rates Controller files. Source: own

The modified part of *FixedwingAttitudeControl* will be in the main loop. It is the same for *FixedwingRateControl*, as well as an addition of a function to include LQR. Finally, *fw_rate_control_params* includes all the parameters needed for the LQR to work.

The following sections explain this in more detail.

Attitude Control file

The file FixedWingAttitudeControl contains the first loop of the cascade seen in Figure 3.6, which generates the angular velocity setpoints for FixedWingRateControl. To leave other functionalities working, this code will run as it normally would, so it will publish angular velocity setpoints that will not be used by FixedWingRateControl.

This script, however, will print to the console a warning when the experimental code is being used. To achieve this functionality, first, a param needs to be created using what is seen in this section, whose name will be *FW_CTRL*. After that, if and else statements are employed to check the param value and print it to the console. The result is observed in Figure 6.14:

```
pxh> param set FW_CTRL 0
+ FW_CTRL: curr: 1 -> new: 0
pxh> INFO  [fw_att_control] Running normal attitude control
pxh> param set FW_CTRL 1
  FW_CTRL: curr: 0 -> new: 1
pxh> WARN  [fw_att_control] Running experimental attitude control
```

Figure 6.14: Switch between controls using console. Source: own

The value 0 represents the default control, and 1 the LQR control. Alternatively, this parameter can be more intuitively selected into QGrounControl from a dropdown menu, shown in Figure 6.15.

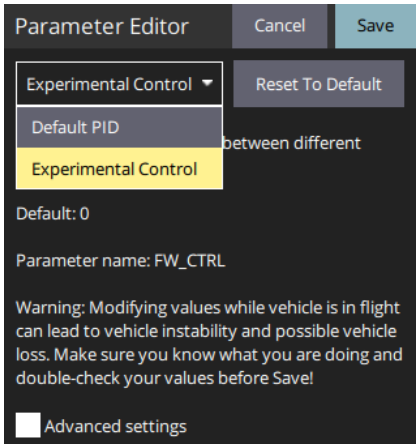


Figure 6.15: Switch between controls using console. Source: own

The control type can only be selected before flight, as the change between controls in the middle of it needs extra implementation.

Rate Control file

This file will include all the experimental code. Regarding the header file, a subscription to the attitude and attitude setpoint topics from uORB is required to obtain all data. Listing 6.11 illustrates the required code:

```
1
2 #include <uORB/topics/vehicle_attitude.h>
3 #include <uORB/topics/vehicle_attitude_setpoint.h>
4
5 uORB::Subscription _att_sub{ORB_ID(vehicle_attitude)};
6 uORB::Subscription _att_sp_sub{ORB_ID(vehicle_attitude_setpoint)};
7
8 vehicle_attitude_s      _att{};
9 vehicle_attitude_setpoint_s   _att_sp{};
```

Listing 6.11: Rate Control required topics subscription

The cpp file starts by fetching the params related to LQR implementation. Listing 6.12 shows the different params:

```
1   _lqr_lat_k11 = _param_fw_lqr_lat_k11.get();
2   _lqr_lat_k12 = _param_fw_lqr_lat_k12.get();
3   _lqr_lat_k13 = _param_fw_lqr_lat_k13.get();
4   _lqr_lat_k14 = _param_fw_lqr_lat_k14.get();
5   _lqr_lat_k21 = _param_fw_lqr_lat_k21.get();
6   _lqr_lat_k22 = _param_fw_lqr_lat_k22.get();
7   _lqr_lat_k23 = _param_fw_lqr_lat_k23.get();
8   _lqr_lat_k24 = _param_fw_lqr_lat_k24.get();
9
10  _lqr_lon_k1 = _param_fw_lqr_lon_k1.get();
11  _lqr_lon_k2 = _param_fw_lqr_lon_k2.get();
12  _lqr_lon_k3 = _param_fw_lqr_lon_k3.get();
13
14  _deltaE_trim = _param_fw_deltaE_trim.get();
15  _lqr_Ts = _param_fw_lqr_Ts.get();
```

```
  _rudder_max = _param_fw_rudder_max.get();
  _rudder_min = _param_fw_rudder_min.get();
  _theta_trim = _param_fw_theta_trim.get();
```

Listing 6.12: Rate Control params obtention

These are the variables of the state feedback gain matrices of longitudinal and lateral systems. Next one is the value of $\delta_{E,Trim}$ that is added to the elevator output as LQR calculates increments respect to trim values. After that, the time constant of the loop, which is the inverse of the frequency at which runs, in this case 250 Hz. It has been obtained with *work_queue status* command. It is followed by minimum and maximum rudder angles for antiwindup. Finally, the trim value of pitch angle is obtained to feed back the increment of theta (current pitch subtracted by trim value).

After that, in Run main loop an if-else statement is used to switch between codes. If experimental code is selected, a function named *LQR* will be computed, seen in Listing 6.13.

```
void FixedwingRateControl::LQR(){

  bool params_updated = _parameter_update_sub.updated();

  // check for parameter updates
  if (params_updated) {
    // clear update
    parameter_update_s pupdate;
    _parameter_update_sub.copy(&pupdate);

    // update parameters from storage
    updateParams();
    parameters_update();
  }

  //Update attitude values and setppoints
  _att_sp_sub.update(&_att_sp);
  _att_sub.update(&_att);

  //Obtain euler angles
  const matrix::Eulerf euler_angles(matrix::Quatf(_att.q));

  //Rates setpoint is updated to obtain thrust setpoint
  _rates_sp_sub.update(&_rates_sp);

  /* throttle passed through if it is finite */
  _vehicle_thrust_setpoint.xyz[0] = PX4_ISFINITE(_rates_sp.
   thrust_body[0]) ? _rates_sp.thrust_body[0] : 0.0f;

  //Calculate roll error and apply antiwindup by stopping the update
    of roll error
  float roll_error = _att_sp.roll_body - euler_angles.phi();
  float ei_roll = (_antiwindup_flag) ? _ei_roll_1 : _ei_roll_1 +
   roll_error * _lqr_Ts;
  _ei_roll_1 = ei_roll;

  //Calculate pitch error
```

```
35  float pitch_error = _att_sp.pitch_body - euler_angles.theta();
36  float ei_pitch = _ei_pitch_1 + pitch_error*_lqr_Ts;
37  _ei_pitch_1 = ei_pitch;
38
39  //Obtain angular velocities and calculate body rates
40  vehicle_angular_velocity_s angular_velocity{};
41  Vector3f rates(angular_velocity.xyz);
42  float p = rates(0) - rates(2)*sin(euler_angles.theta());
43  float q = rates(1)*cos(euler_angles.phi()) + rates(2)*sin(
     euler_angles.phi())*cos(euler_angles.theta());
44  float r = -rates(1)*sin(euler_angles.phi()) + rates(2)*cos(
     euler_angles.phi())*cos(euler_angles.theta());
45
46  //Aileron and rudder calculation (lateral LQR)
47  //They are defined as torque variables, but they are angles
48  _vehicle_torque_setpoint.xyz[0] = -1*(_lqr_lat_k11*p +
     _lqr_lat_k12*r + _lqr_lat_k13*euler_angles.phi() + _lqr_lat_k14*
     ei_roll);
49  _vehicle_torque_setpoint.xyz[2] = -1*(_lqr_lat_k21*p +
     _lqr_lat_k22*r + _lqr_lat_k23*euler_angles.phi() + _lqr_lat_k24*
     ei_roll);
50
51  //Antiwindup limits and flag
52  if(_vehicle_torque_setpoint.xyz[2] > _rudder_max ||
     _vehicle_torque_setpoint.xyz[2] < _rudder_min){
53    _antiwindup_flag = 1;
54    _vehicle_torque_setpoint.xyz[2] = math::constrain(
     _vehicle_torque_setpoint.xyz[2], _rudder_min, _rudder_max);
55  }
56
57  else
58    _antiwindup_flag = 0;
59
60  //Elevator calculation (longitudinal LQR)
61  //Initial elevator deflection is added at the end as LQR
     calculates the increment from trim value
62  //Initial pitch angle is subtracted to the current pitch angle to
     obtain the increment
63  _vehicle_torque_setpoint.xyz[1] = -1*(_lqr_lon_k1*q + _lqr_lon_k2
     * (euler_angles.theta()-_theta_trim) + _lqr_lon_k3*ei_pitch) +
     _deltaE_trim;
64
65
66  /* scale effort by battery status */
67  if (_param_fw_bat_scale_en.get() && _vehicle_thrust_setpoint.xyz
     [0] > 0.1f) {
68
69    if (_battery_status_sub.updated()) {
70      battery_status_s battery_status{};
71
72      if (_battery_status_sub.copy(&battery_status) &&
     battery_status.connected && battery_status.scale > 0.f) {
73        _battery_scale = battery_status.scale;
74      }
75    }
76
77    _vehicle_thrust_setpoint.xyz[0] *= _battery_scale;
```

```
78   }
79
80   //Publish thrust and deflection of control surfaces
81   _vehicle_thrust_setpoint.timestamp = hrt_absolute_time();
82   _vehicle_thrust_setpoint.timestamp_sample = angular_velocity.
      timestamp_sample;
83   _vehicle_thrust_setpoint_pub.publish(_vehicle_thrust_setpoint);
84
85   _vehicle_torque_setpoint.timestamp = hrt_absolute_time();
86   _vehicle_torque_setpoint.timestamp_sample = angular_velocity.
      timestamp_sample;
87   _vehicle_torque_setpoint_pub.publish(_vehicle_torque_setpoint);
88 }
```

Listing 6.13: LQR function

where the last part is omitted, which enables manual flaps and spoilers control. Note that the control output is defined as torque because of quadcopter drone tradition, but they represent deflection angle of the control surfaces.

The control law (6.4) is the published output, used by servos to deflect the control surfaces.

6.2.7 Weighting matrices calibration

After the algorithm implementation, its behaviour has to be adjusted by tuning Q and R. The objective of the project is not to define a calibration methodology for them, so an empirical approach is used for it.

Theoretical foundations

The followed approach has some foundations:

- Raising the value of Q associated to an state generates that the controller tries to minimize it quicker, so it has a stronger influence reflected in the values of the feedback gain matrix K. This means that it will try to reach the initial or trim value seen in Subsection 6.1.6.
- Lowering the value of R associated to a control has a similar effect as explained above. A low value will result in high deflections of the control surface to achieve the desired effects.

The following chapter, dedicated to experimental results, will show this process of simulations and its results.

Simulink simulations

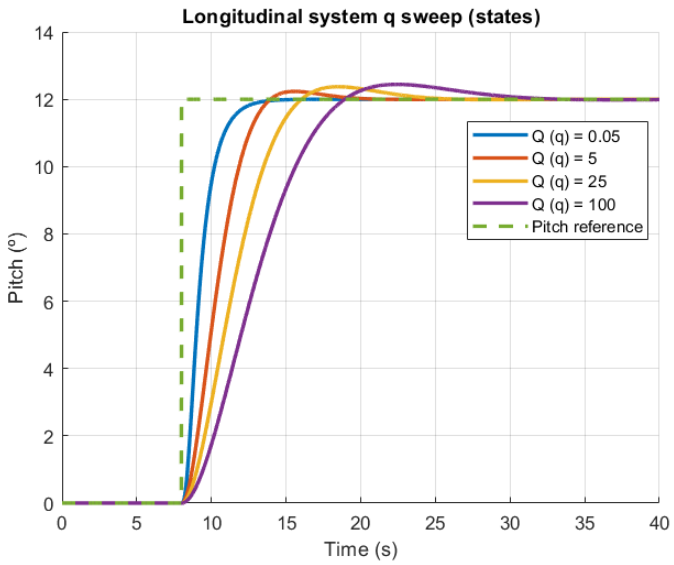
A series of Simulink simulations are conducted to demonstrate the impact of varying the weight assigned to each state and control. In these simulations, all weights are set to 1, except for the one being analyzed.

- **Longitudinal system**

  The state that will be analysed is pitch as it is the one that needs to follow a reference.

  - Sweep of q

    Figure 6.16 shows that increasing the weight associated to q a slower response with more overshoot is obtained. Moreover, the elevator response is also lower and slower. This is because by increasing the weight q tries to get to the trim value, which is 0, thus making difficult a fast pitch change. It is concluded that a low weight value is preferred to obtain a faster response.



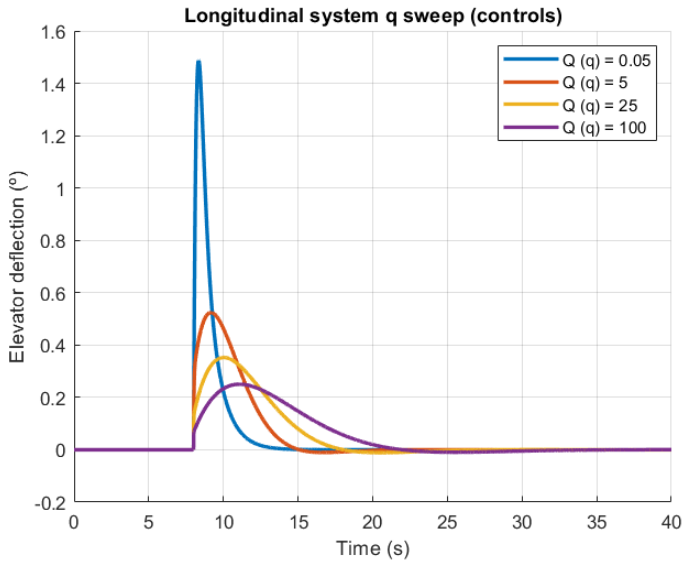
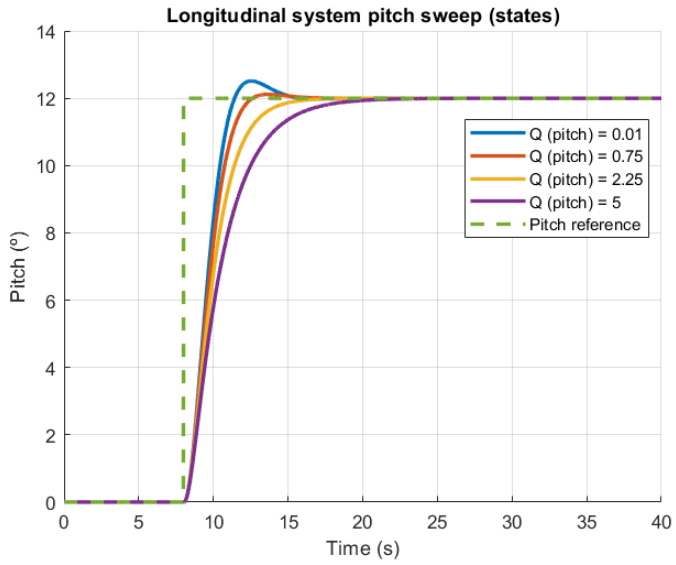


Figure 6.16: Sweep of simulations with different associated weight values to q. Source: own

– Sweep of pitch

In Figure 6.17 it can be seen that increasing the weight associated to pitch a slower response with less overshoot is obtained. The elevator response is also lower. This is because by increasing the weight pitch tries to get to the trim value (in this case 0°). It opposes to a faster change of pitch, but eliminates overshoot, so it will be increased until overshoot is overcome.



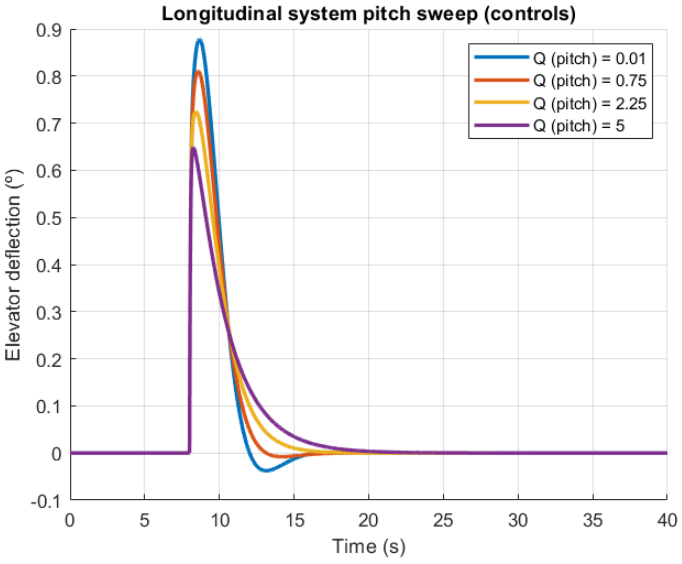
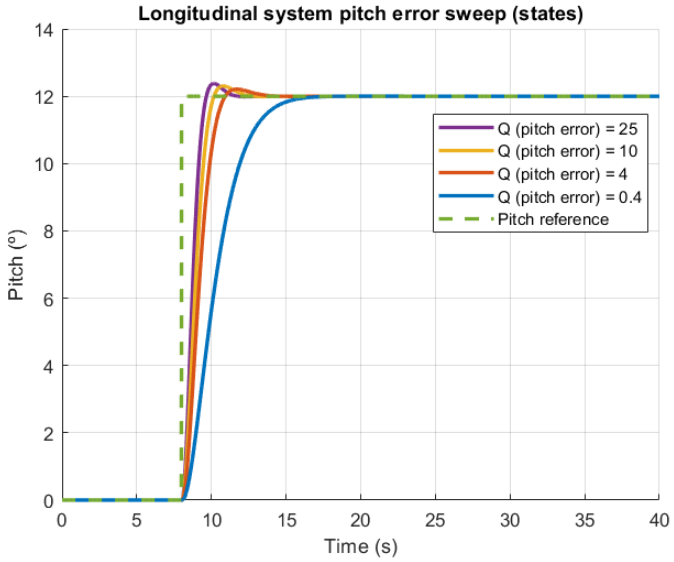


Figure 6.17: Sweep of simulations with different associated weight values to pitch. Source: own

– Sweep of pitch error

From Figure 6.18 it can be derived that increasing the weight associated to pitch error a faster response with more overshoot is obtained. The elevator response is also higher. This is because by increasing the weight pitch tries to get faster to the reference value. It is concluded that it will be the highest weight value.



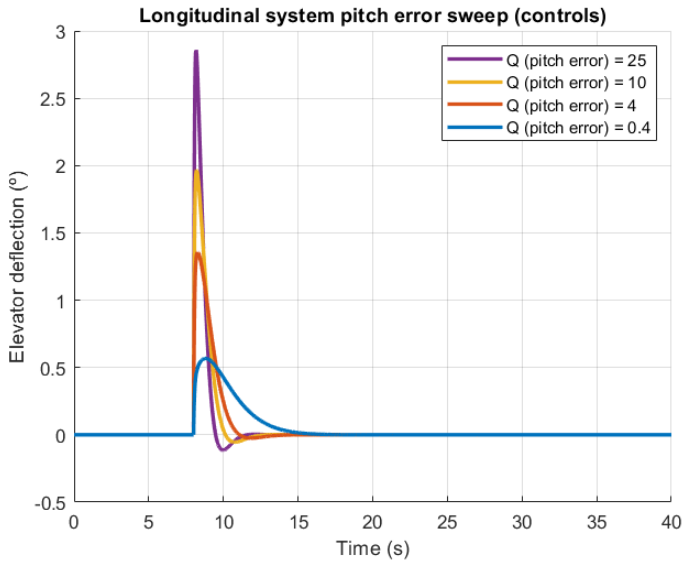
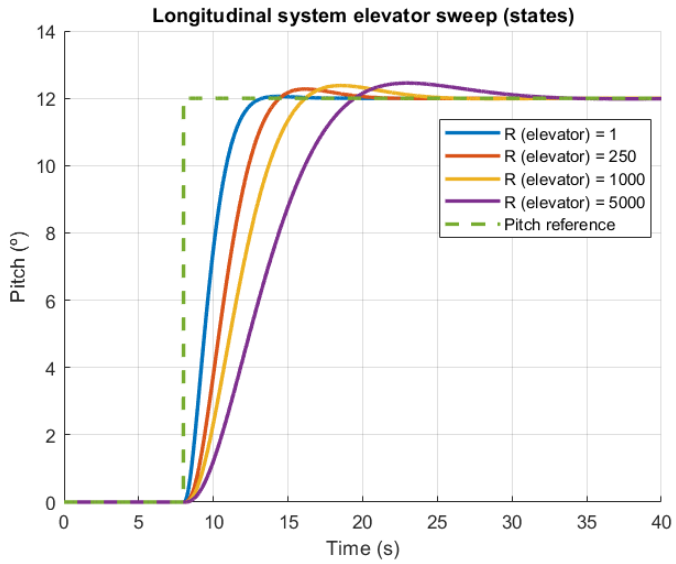


Figure 6.18: Sweep of simulations with different associated weight values to pitch error. Source: own

– Sweep of elevator

For the elevator sweep, Figure 6.19 shows that increasing the weight associated to elevator a slower response with more overshoot is obtained. The elevator response is also lower and slower. It can be explained as increasing the weight makes the cost of using the control higher. The conclusion is that this weight value will be as low as possible.



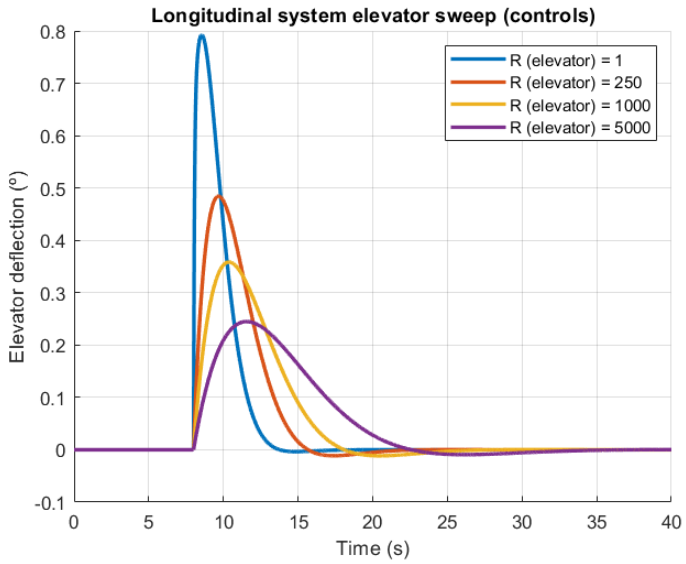
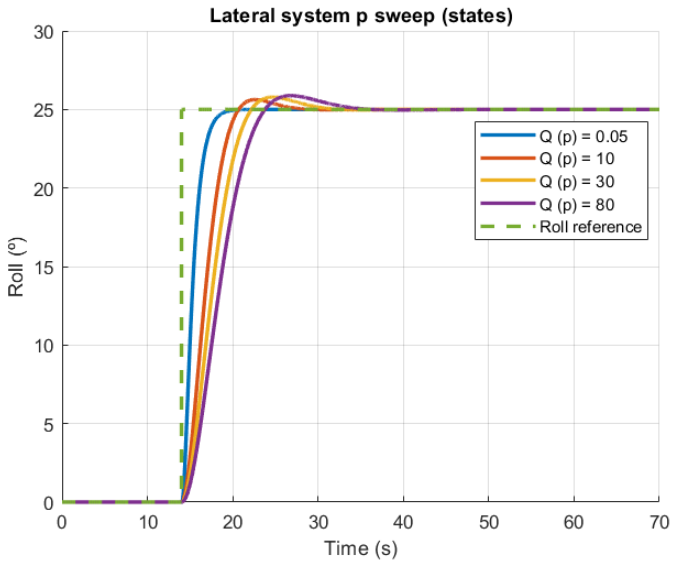


Figure 6.19: Sweep of simulations with different associated weight values to elevator. Source: own

- **Lateral system** The state that will be analysed is roll as it is the one that needs to follow a reference.

  – Sweep of p

  Figure 6.20 shows that increasing the weight associated to p a slower response with more overshoot is obtained. The aileron response is lower and slower, and the rudder is higher to compensate. Similarly to q, increasing p weight makes the effort to get it to trim value is higher, therefore opposing to changes in roll. It is concluded that this weight value will be as low as possible.



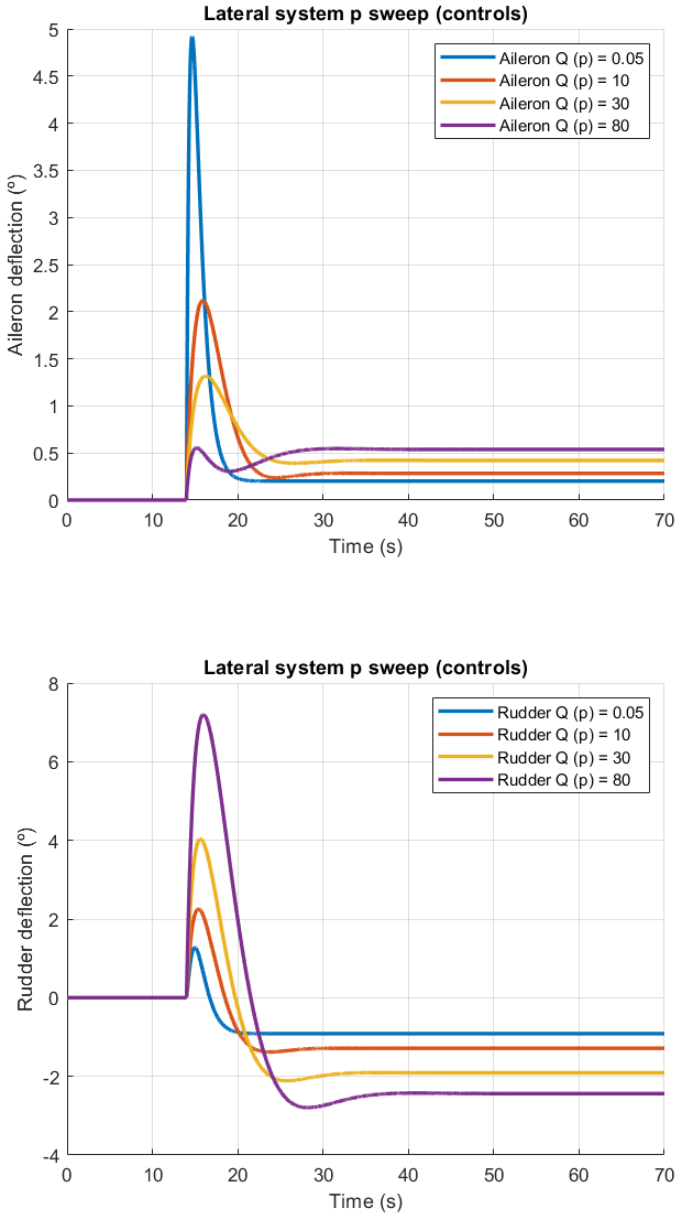
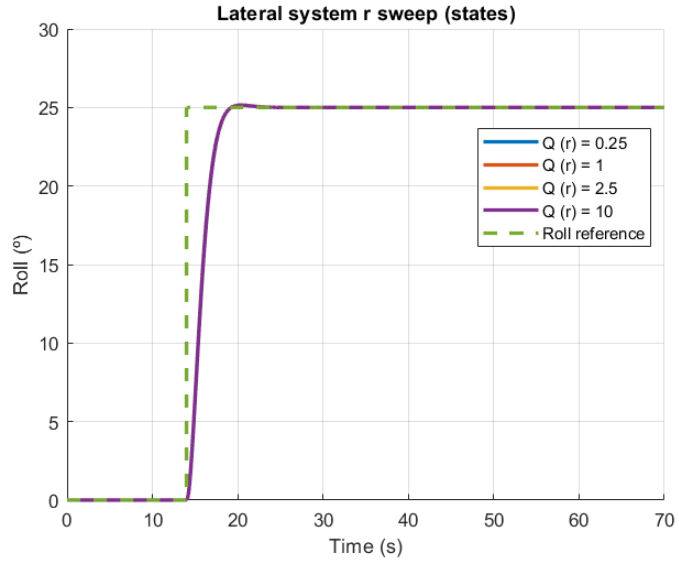
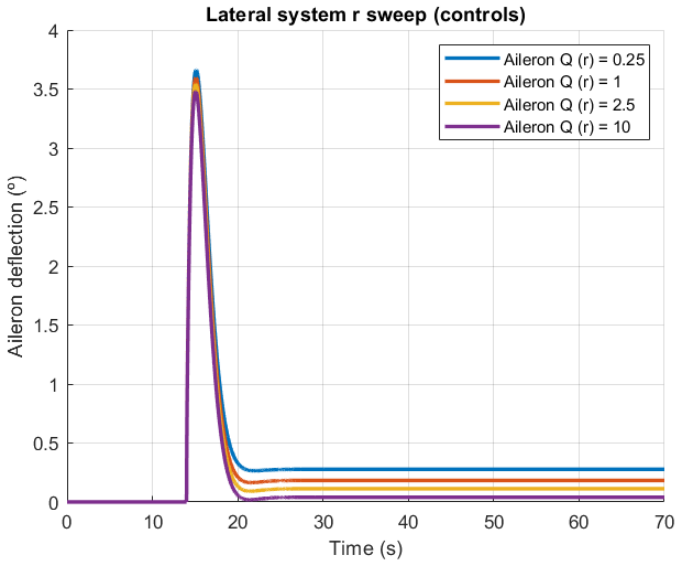


Figure 6.20: Sweep of simulations with different associated weight values to p. Source: own

– Sweep of r

Changing the weight associated to r does not have an impact in roll, as seen in Figure 6.21. The aileron response is not changed significantly. However, rudder response has a higher peak, and a lower steady value. It is explained as the final r value will be closer to 0 $°/s$ (trim value) thanks to rudder action. Thus less aileron is required in steady conditions to maintain a lower r. A low weight value will be preferred.


Lateral system r sweep (states)
Roll (°)
Time (s)
Q (r) = 0.25
Q (r) = 1
Q (r) = 2.5
Q (r) = 10
Roll reference


Lateral system r sweep (controls)
Aileron deflection (°)
Time (s)
Aileron Q (r) = 0.25
Aileron Q (r) = 1
Aileron Q (r) = 2.5
Aileron Q (r) = 10

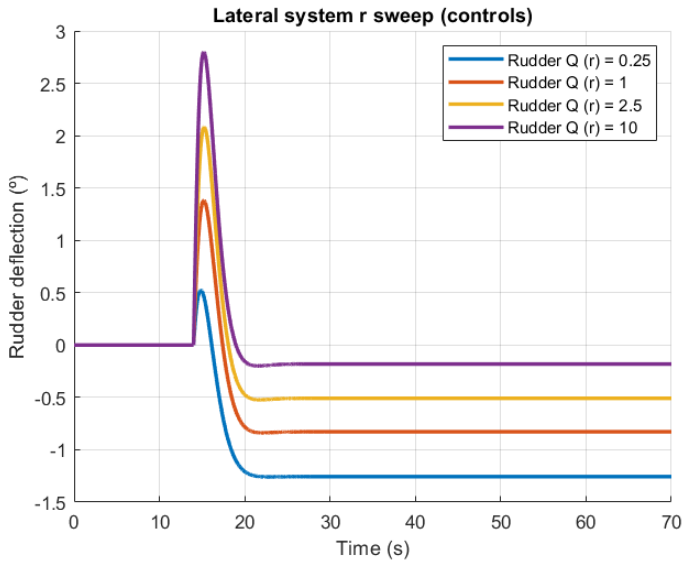
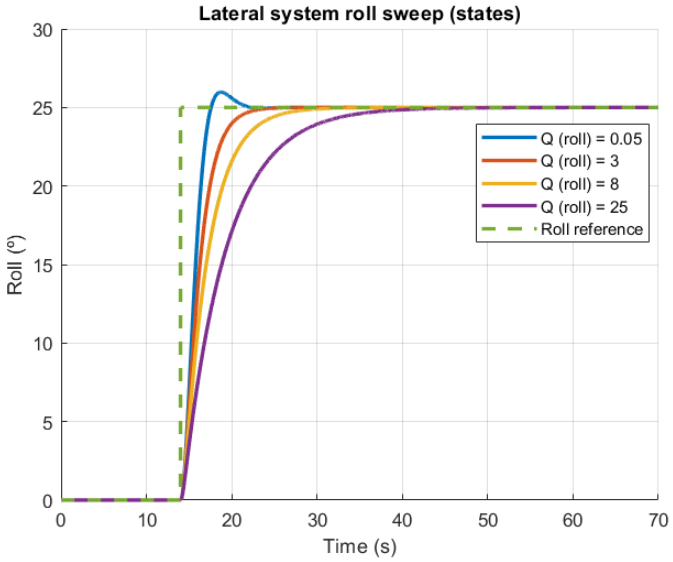


Figure 6.21: Sweep of simulations with different associated weight values to r. Source: own

– Sweep of roll

In Figure 6.22 it can be seen that increasing the weight associated to roll the response is lower and with less overshoot. The aileron and rudder responses are also lower. The explanation is the same as for pitch in the longitudinal system, it tries to get to 0°. It will be used to overcome overshoot.



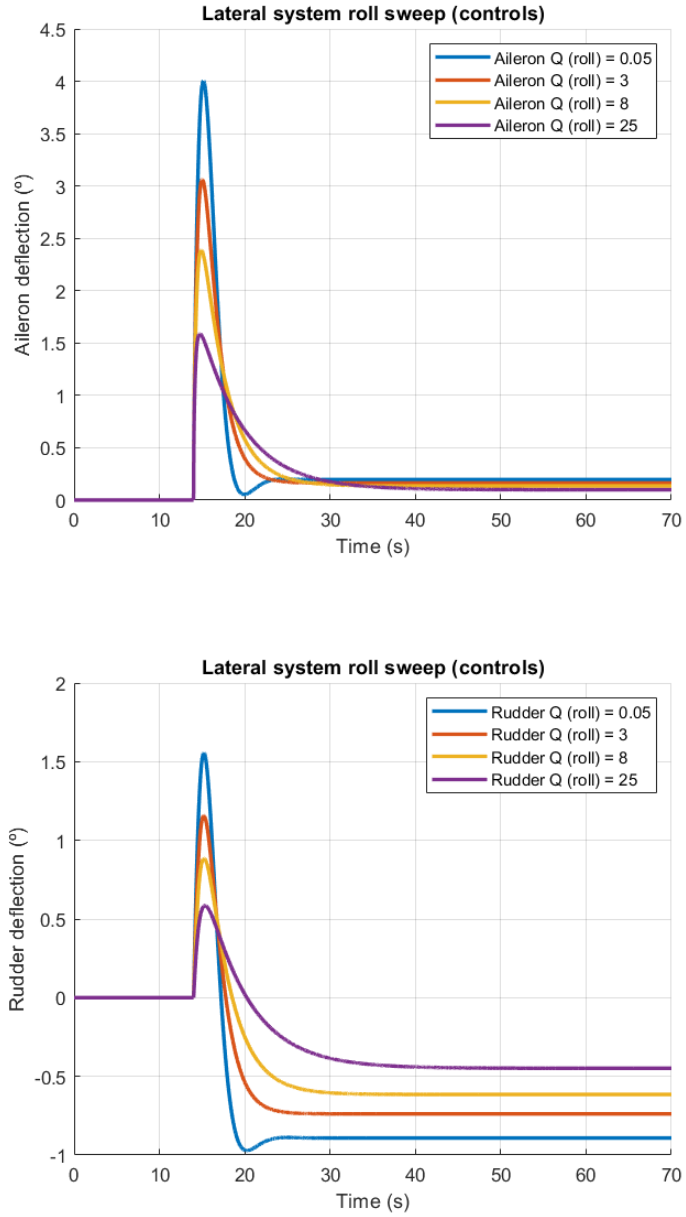
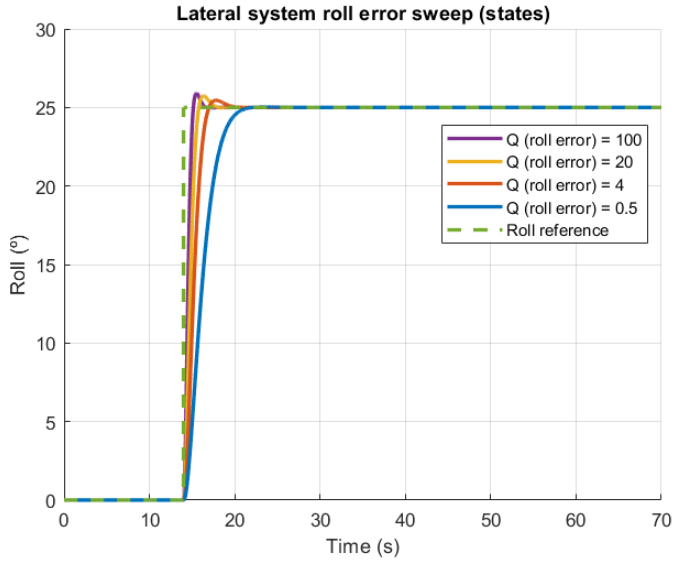
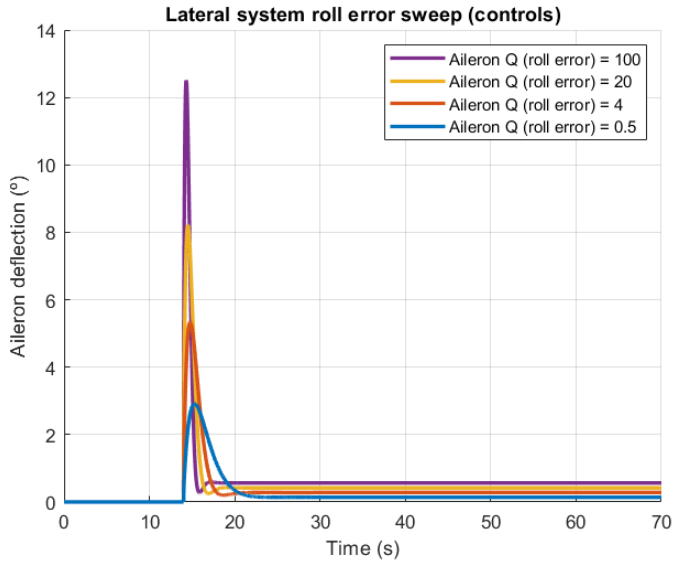


Figure 6.22: Sweep of simulations with different associated weight values to roll. Source: own

– Sweep of roll error

Increasing the weight associated to roll error, as seen in Figure 6.23, generates a faster response with more overshoot. The aileron and rudder responses are also higher. It is because roll gets faster to its reference value to reduce error. It will be set to maximum.


Lateral system roll error sweep (states)
Roll (°)
Time (s)
Q (roll error) = 100
Q (roll error) = 20
Q (roll error) = 4
Q (roll error) = 0.5
Roll reference


Lateral system roll error sweep (controls)
Aileron deflection (°)
Time (s)
Aileron Q (roll error) = 100
Aileron Q (roll error) = 20
Aileron Q (roll error) = 4
Aileron Q (roll error) = 0.5

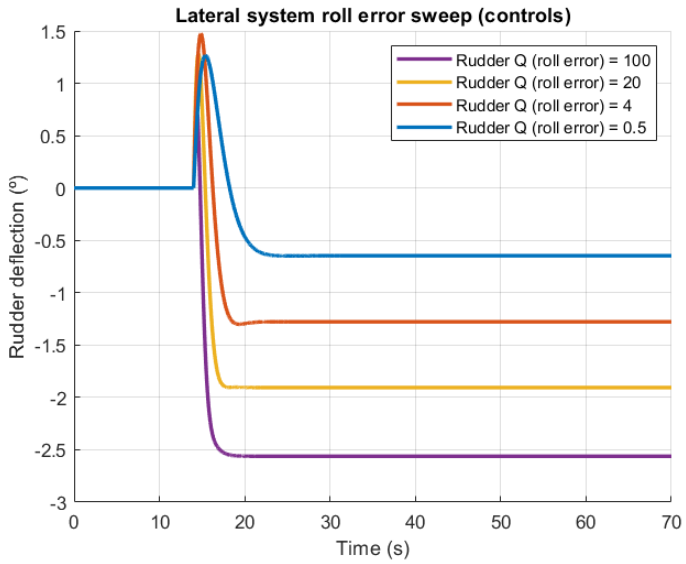
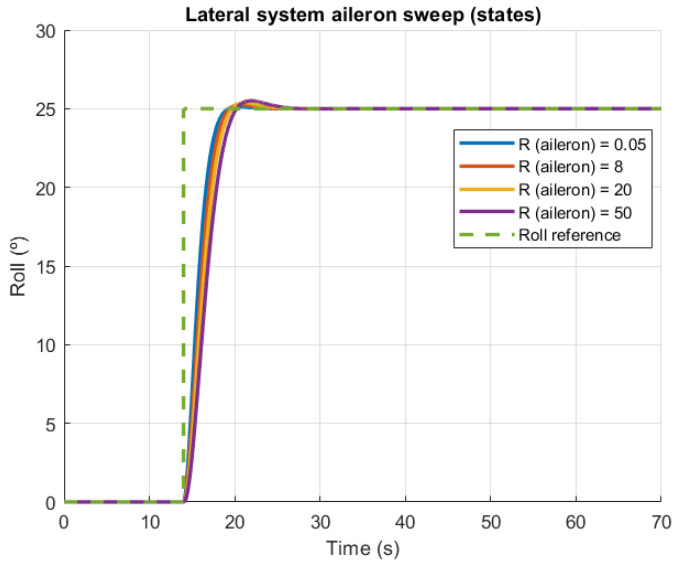


Figure 6.23: Sweep of simulations with different associated weight values to roll. Source: own

– Sweep of aileron

The effect of increasing the weight associated to aileron is a slower response with more overshoot ,as seen in Figure 6.24. The aileron response loses importance respect to rudder, so they are decreased and increased respectively. A low weight value is preferred to obtain a faster response.



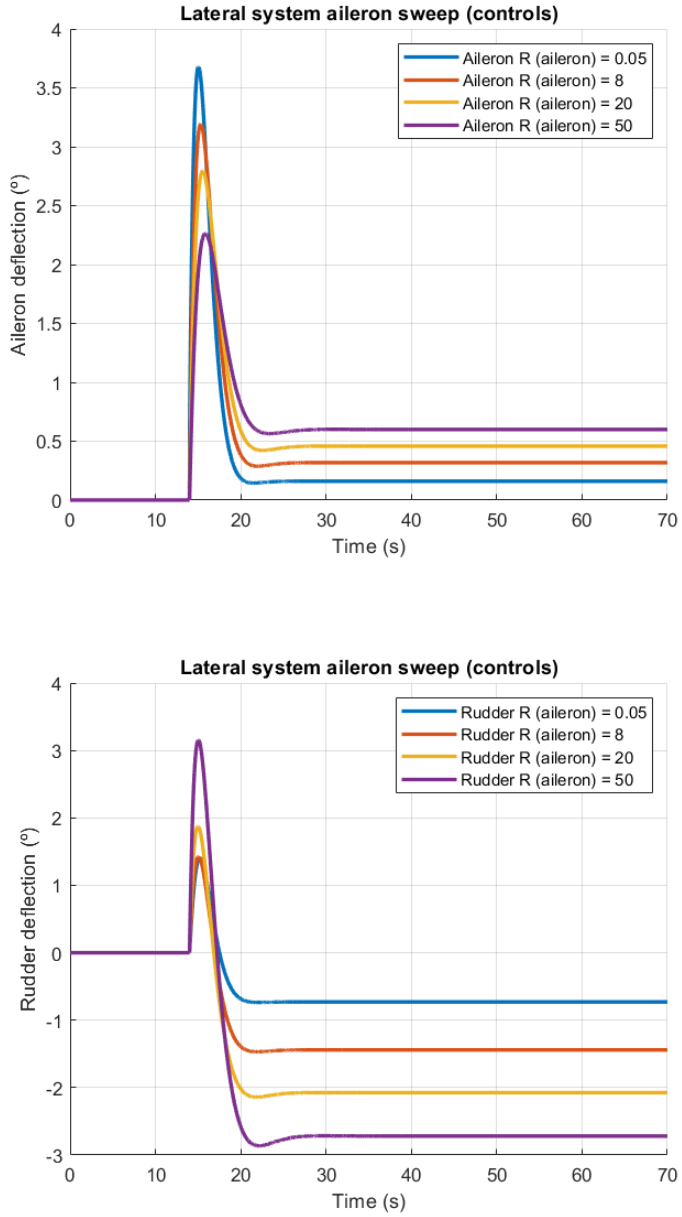
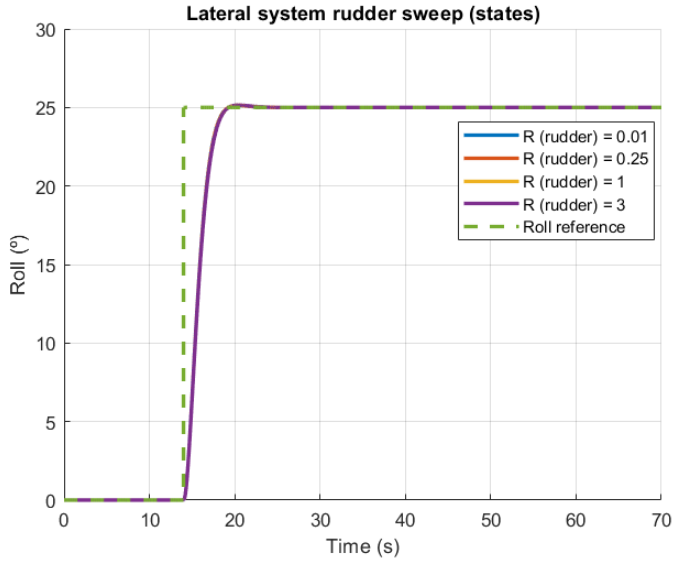
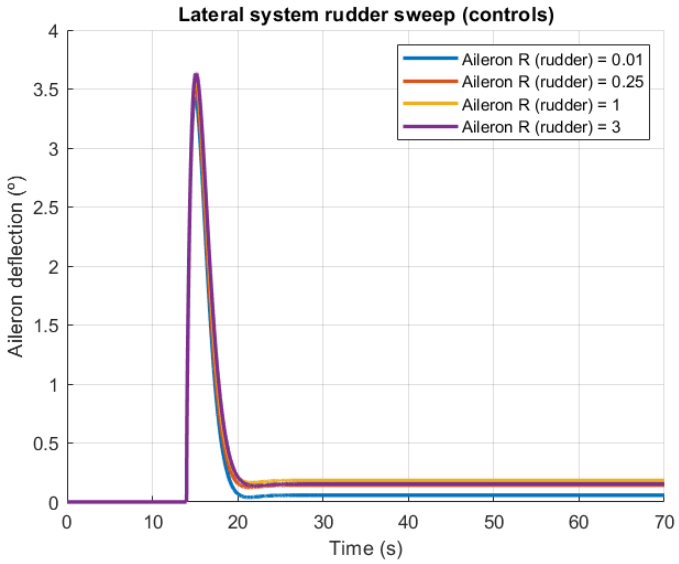


Figure 6.24: Sweep of simulations with different associated weight values to aileron. Source: own

– Sweep of rudder

No effect is observed when changing the weight associated to rudder in roll response, as seen in Figure 6.25. The rudder response, however, is decreased. No major changes are seen in aileron response. A low weight value is preferred.


Lateral system rudder sweep (states)
Roll (°)
Time (s)
R (rudder) = 0.01
R (rudder) = 0.25
R (rudder) = 1
R (rudder) = 3
Roll reference


Lateral system rudder sweep (controls)
Aileron deflection (°)
Time (s)
Aileron R (rudder) = 0.01
Aileron R (rudder) = 0.25
Aileron R (rudder) = 1
Aileron R (rudder) = 3

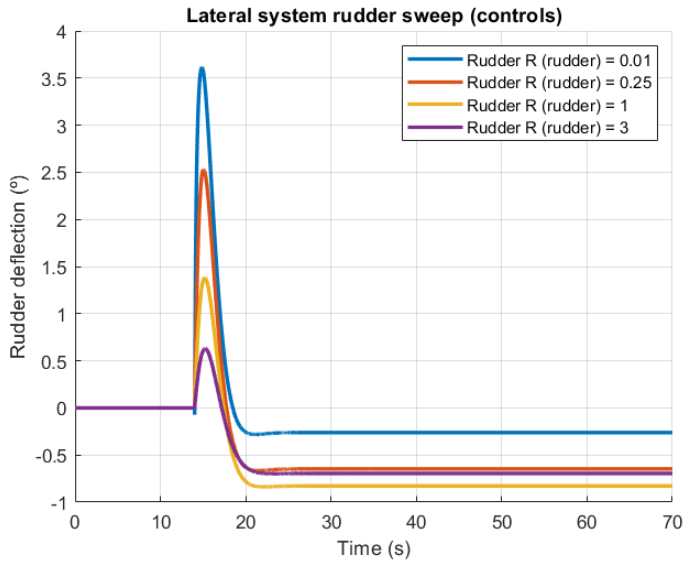


Figure 6.25: Sweep of simulations with different associated weight values to rudder. Source: own

Param change after calibration

All the params of the gain matrix K that require changes after calibration are grouped into FW LQR CONTROL inside Parameters tab of QGroundControl, as seen in Figure 6.26.

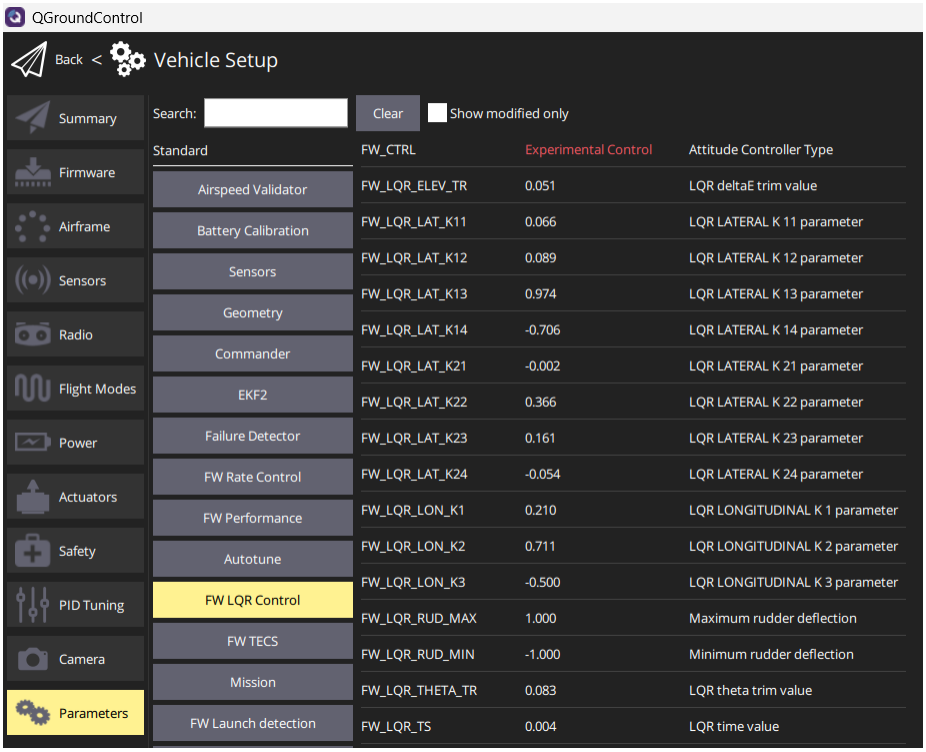


Figure 6.26: LQR parameters in QGroundControl. Source: own

Chapter 7

Experimental results

This chapter shows results of simulations to justify the followed calibration and to compare the performance against the regular implementation.

To obtain the information of the simulations, the log file is downloaded from QGrounControl and analysed in MATLAB.

7.1 Weighting matrices calibration

The conclusions drawn from Section 6.2.7 will be considered during the calibration in the SITL scenario.

7.1.1 Longitudinal System

This system is simpler to calibrate as it contains fewer states and control elements.

Control effort

In the calibration process, the control effort variable proves to be the most sensitive. Reducing its value to achieve greater surface deflection results in oscillatory behaviour, while high values lead to a slower response from the controller. However, there exists a range where oscillations are absent, and the controller performance is maximized.

To find a value within this range, simulations are performed with different R values, and keeping $Q = I$. They start from trim conditions, and make a descent to $h = 70\ m$, and an ascent to $h = 100\ m$. Figure 7.1 shows the results of them.

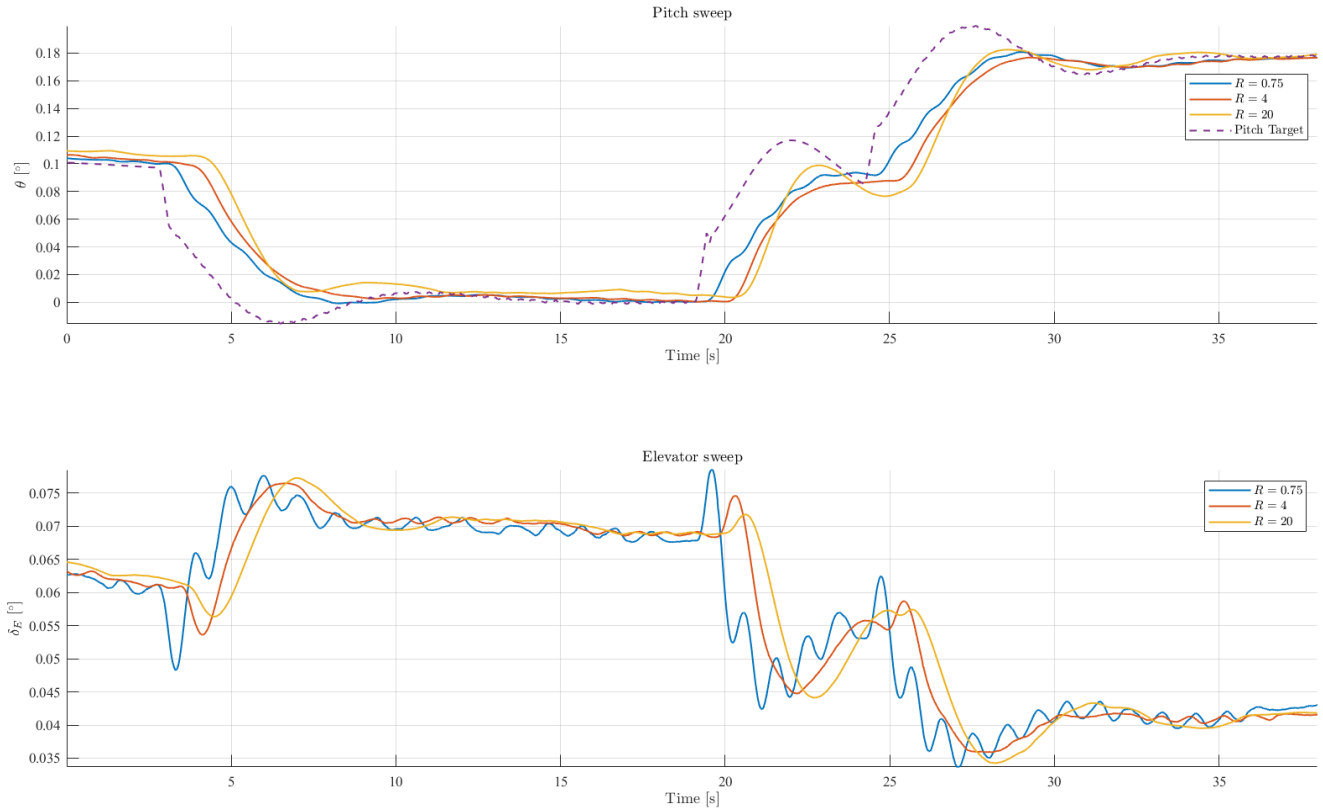


Figure 7.1: Pitch and elevator deflection for different R values. Source: own

It is concluded that the value in which oscillations disappear but performance is maximised is with $R = 4$.

State deviation

The states in the system are q, pitch and pitch error. Raising their associated value will generate a stronger response to bring them to trim values (0, θ_{Trim} and 0, respectively). Therefore, the value associated to pitch error should be maintained high, therefore 1. In the case of q, it needs a low value as high pitch angular velocity is needed to reduce pitch error, so it is set to 0.01.

However, as seen in Section 6.2.7, a high weight associated to pitch error generates overshoot. Thus, pitch associated weight will be used to overcome it. Figure 7.2 shows Simulink simulations where different weights to pitch are set. The other weights are:

$$Q = \begin{bmatrix} 0.01 & 0 & 0 \\ 0 & Q_\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R = 4$$

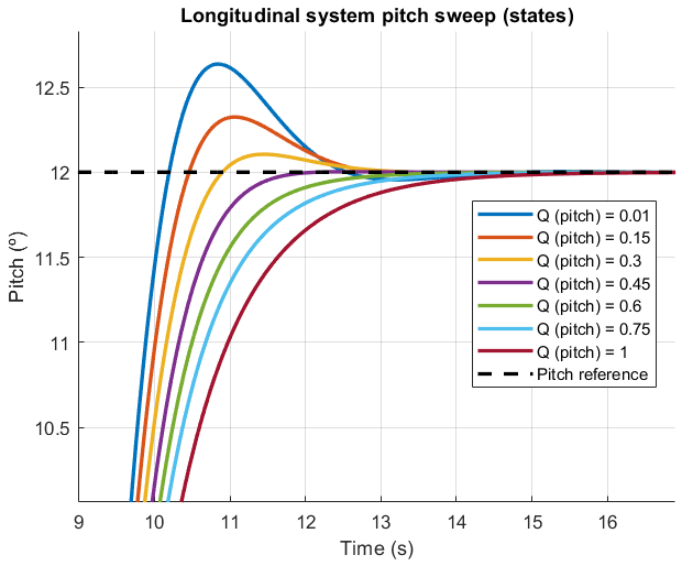


Figure 7.2: Sweep of simulations with different weights associated to pitch. Source: own

It can be seen that $Q_\theta = 0.45$ is the value with no overshoot with the fastest response, so it is the selected one.

Results

The final matrices are:

$$Q = \begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 0.45 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{7.1}$$

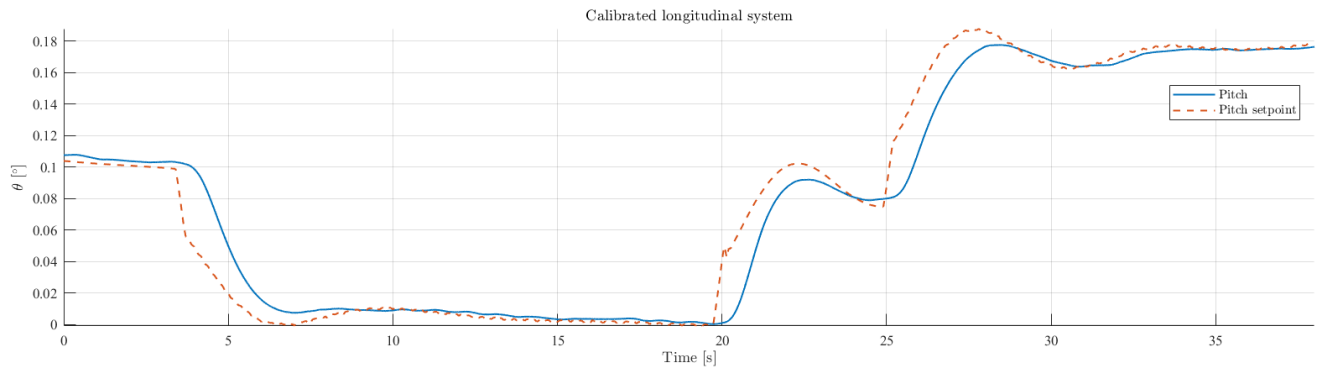
$$\tag{7.2}$$

$$R = 4 \tag{7.3}$$

Which after solving the Riccati equation, the gain matrix results in:

$$K = \begin{bmatrix} 0.166 & 0.651 & -0.500 \end{bmatrix} \tag{7.4}$$

Figure 7.3 shows the results of a simulation with the descent to $h = 70\ m$ and ascent to $h = 100\ m$.



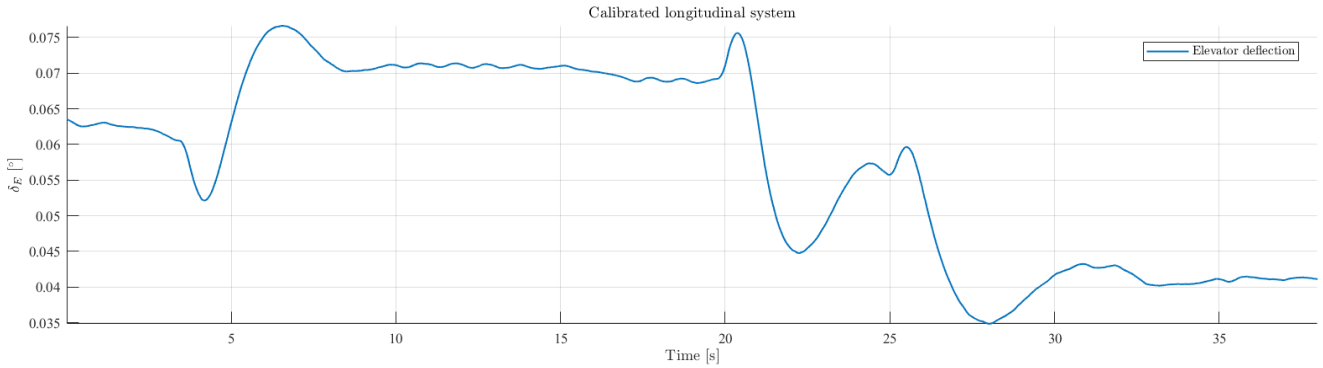


Figure 7.3: Pitch and elevator deflection for the calibrated longitudinal system. Source: own

7.1.2 Lateral-directional System

Contrary to the previous section, this system has more variables to tune. Still, the effect of changing these has analogies with the conclusions drawn from longitudinal system.

Control effort

In this case, low R values (high control effort) also lead to oscillations. Again, simulations are performed to compare the response by changing these values. The followed trajectory consists of two wide turns and two sharp turns, seen in Figure 7.4.

Figure 7.4: Target trajectory used for lateral controls calibration. Source: own

Figure 7.5 shows the results of the different simulations with $Q = I$.

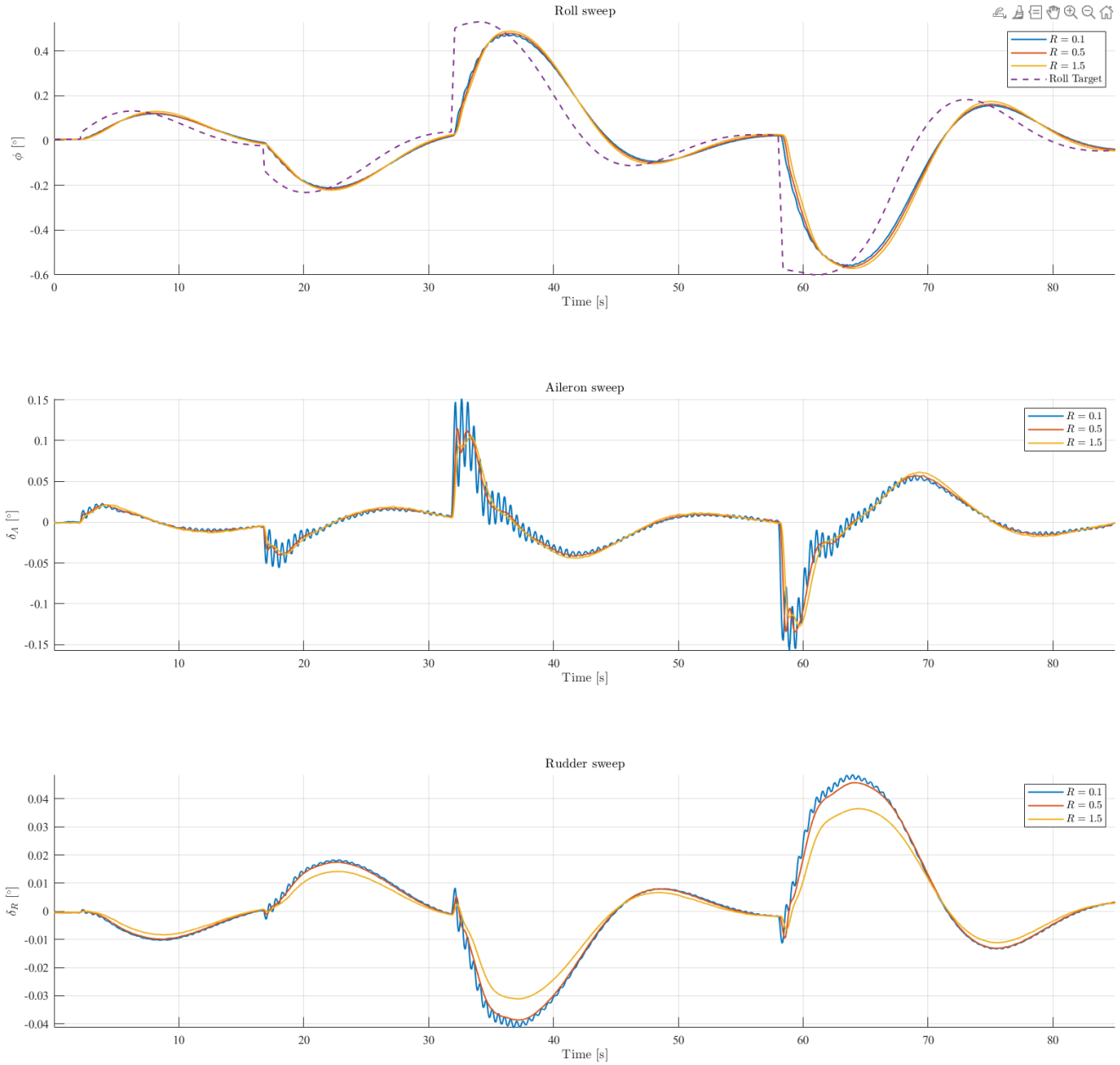


Figure 7.5: Roll and aileron and rudder deflection for different values of R. Source: own

From the oscillations seen in the simulations, $R = 0.1 \cdot I$ is discarded. Between the remaining values $R = 0.5 \cdot I$ offers better performance, so it is chosen.

State deviation

As in the longitudinal system, the value associated to roll error will be the highest, therefore 1. On the other hand, the associated to p will be the lowest to have a faster response that eliminates roll error. In addition, r will also take the lowest value to generate the most change in yaw. Both will be set to 0.01. For the weight associated to roll, it is chosen to eliminate overshoot with $R = 0.5 \cdot I$. Figure 7.6 shows these simulations.

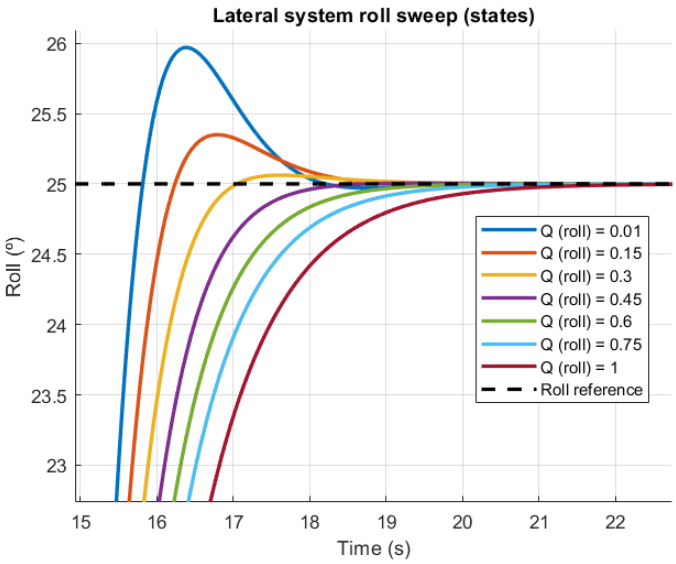


Figure 7.6: Simulink simulation with different weight values associated to roll. Source: own

It can be concluded that the best value is $Q_\phi = 0.45$.

Results

The final matrices are:

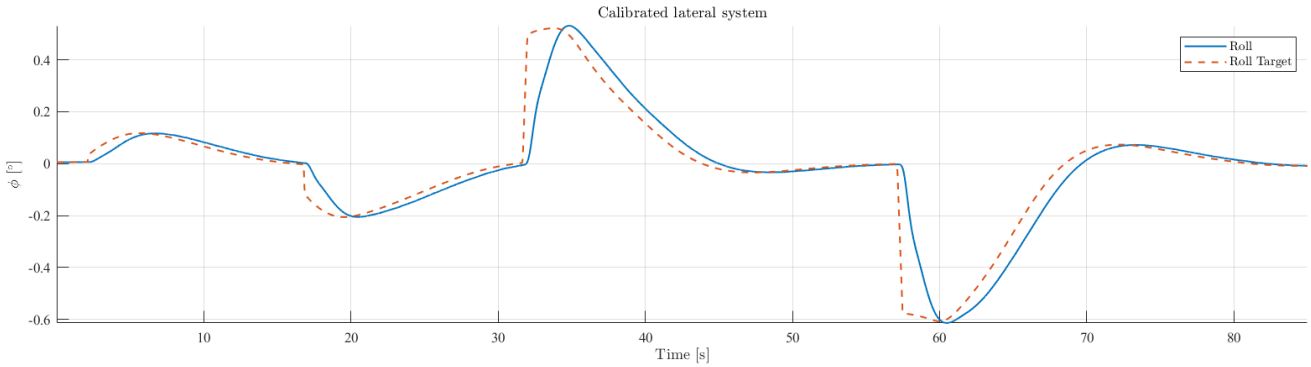
$$Q = \begin{bmatrix} 0.01 & 0 & 0 & 0 \\ 0 & 0.01 & 0 & 0 \\ 0 & 0 & 0.45 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.5}$$

$$R = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \tag{7.6}$$

The gain matrix is the following:

$$K = \begin{bmatrix} 0.110 & 0.133 & 1.485 & -1.414 \\ 0 & 0.031 & 0.025 & 0.016 \end{bmatrix} \tag{7.7}$$

Figure 7.7 shows the results with the selected matrices.



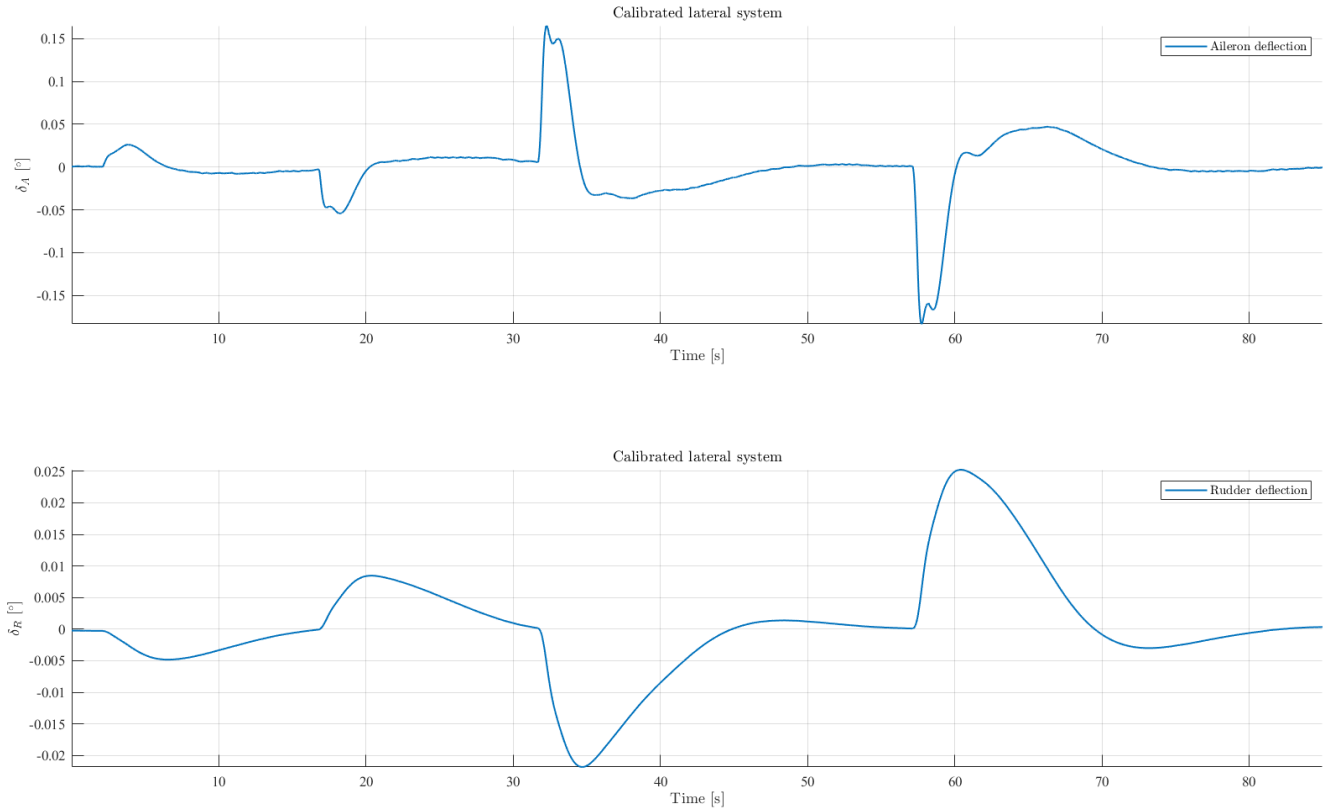


Figure 7.7: Calibrated lateral system simulation results. Source: own

7.2 Comparison with regular implementation

In this section the results will be compared with the PID implementation. A more complex mission is followed, which includes a climb phase, an acceleration phase, turns at constant height and turns at variable height.

Moreover, for this section the PID is calibrated with values obtained by HORUS. These can be seen in Table 7.1.

PID values					
Roll		Pitch		Yaw	
P	0.05	P	0.005	P	0.02
I	0.03	I	0.005	I	0.1
D	0	D	0.005	D	0
FF	0.4	FF	0.4	FF	0.3

Table 7.1: PID calibration values

7.2.1 Error-based indices

The following indices will be used to compare the performance between algorithms:

1. Integral of Absolute Error (IAE):

$$IAE = \int_0^\infty |e(t)|\, dt$$

Measures the total error over time. Lower values indicate better tracking performance.

2. Integral of Squared Error (ISE):

$$ISE = \int_0^\infty e^2(t)\, dt$$

Penalizes larger errors more heavily, emphasizing the elimination of large deviations.

3. - Integral of Time-weighted Absolute Error (ITAE):

$$ITAE = \int_0^\infty t|e(t)|\, dt$$

Penalizes errors at later times, encouraging faster error correction.

4. Root Mean Square Error (RMSE):

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} e(t_i)^2}$$

Measures the average or root-mean-square value of the error.

7.2.2 Pitch error

Figure 7.8 shows the comparison of pitch error over time. However, it may be because of poor PID calibration as higher constant error over time is the responsible for this.

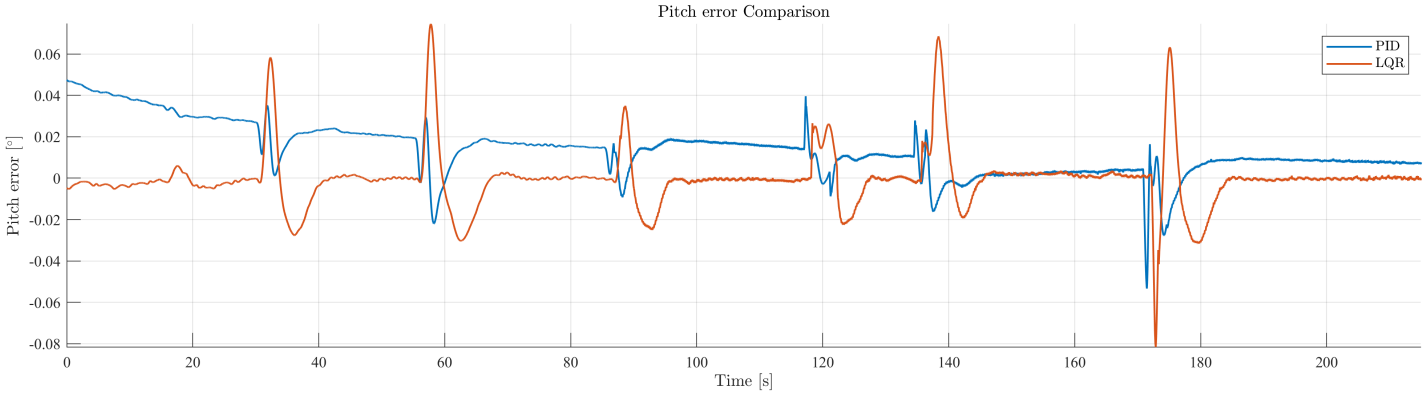


Figure 7.8: Comparison of pitch error between PID and LQR implementations. Source: own

Table 7.2 shows the error-based indices for pitch, and the percentage difference between them.

Index	PID	LQR	Percentage
IAE	3.1954	1.4517	-54.57%
ISE	0.0720	0.0419	-41.81%
ITAE	234.2405	151.5056	-35.32%
RMSE	0.0183	0.0139	-24.04%

Table 7.2: Pitch error-based indices for PID and LQR implementation

It can be concluded that the new implementation outperforms the regular one regarding pitch.

7.2.3 Roll error

Figure 7.9 shows the comparison of roll error over time.

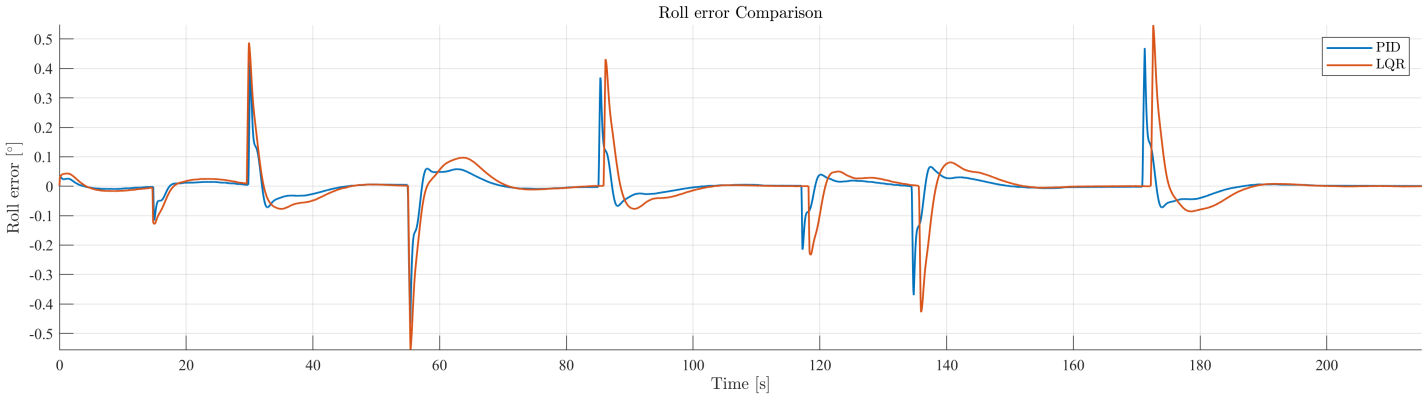


Figure 7.9: Comparison of roll error between PID and LQR implementations. Source: own

Table 7.3 shows the error-based indices for roll, and the percentage difference between them.

Index	PID	LQR	Percentage
IAE	4.8105	7.5204	+56.33%
ISE	0.5160	1.1785	+128.39%
ITAE	461.7153	715.3954	+54.94%
RMSE	0.0485	0.0740	+52.58%

Table 7.3: Roll error-based indices for PID and LQR implementation

It is concluded that the regular implementation outperforms the experimental one regarding roll.

7.2.4 Trajectory error

Figure 7.10 shows the comparison of trajectory error over time.

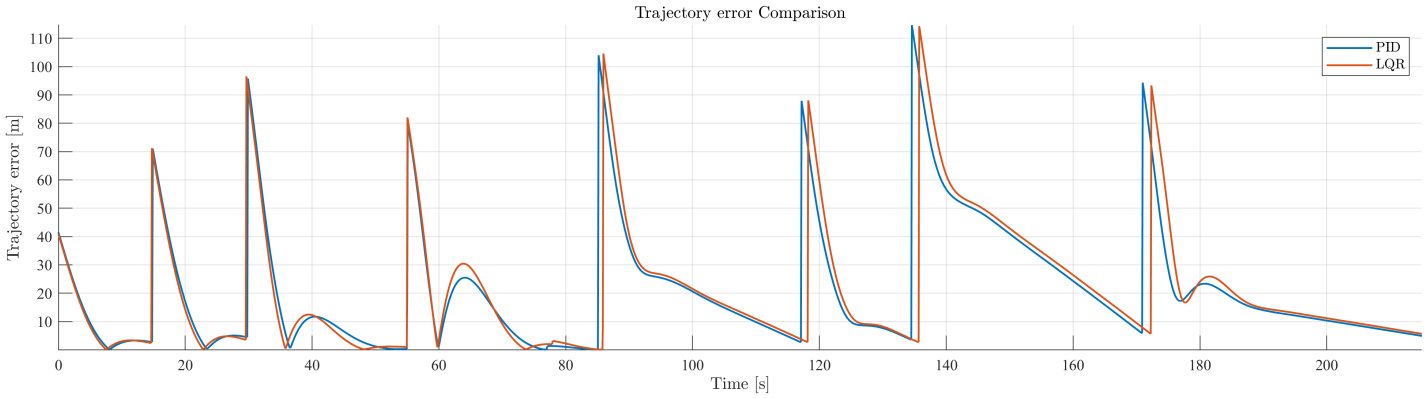


Figure 7.10: Comparison of trajectory error between PID and LQR implementations. Source: own

Table 7.4 shows the error-based indices for trajectory, and the percentage difference between them.

Index	PID	LQR	Percentage
IAE	4539.7368	4552.7832	+0.29%
ISE	194321.7812	195698.7188	+0.71%
ITAE	512496.0312	518481.7500	+1.17%
RMSE	30.2578	30.3489	+0.30%

Table 7.4: Trajectory error-based indices for PID and LQR implementation

Trajectory following is the final goal of the algorithm. In this case, LQR has worse performance, but the difference is minimum.

7.2.5 Comparison conclusion

PID outperformed LQR in two of the three errors. However, regarding trajectory error they were similar. Nevertheless, the experimental implementation has room for improvement: With the position controller being substituted, having data for angle of attack and angle of drift, and an optimal calibration methodology, the algorithm would perform better.

Furthermore, LQR control is distinguished by its ability to produce smoother trajectories, as seen in Figures 7.8 and 7.9. These figures show that the error is reduced more smoothly with LQR compared to PID implementation. Since LQR incorporates all system states in its formulation, disturbances affecting any state are rejected in a more gradual and controlled manner than in the PID approach.

7.3 Objectives evaluation

After the comparison of both implementations the project is considered complete. Recalling the steps:

1. Adaptation of PX4 code to be suited for control mode switchability.
2. Comprobation of correct behaviour of the regular implementation and of control mode switchability.
3. Development of experimental LQR code.
4. Linearization of Bryan equations with aircraft dynamic model around an operating point.
5. Q and R matrices assignment.
6. Riccati's equation calculation.
7. LQR parameters insertion and simulation with H200 model.
8. Validation of experimental controller performance.
9. Comparison with original PID implementation.

The code has been adapted to suit an experimental version and its behaviour is checked in Section 6.2.6. The LQR code is shown in the same section. The linearization of Bryan equations to obtain the aircraft model is performed in Section 6.2.3. The calibration process is shown in 7.1. The Riccati equation calculation is presented in Section 6.2.4 and done in Section 7.1. The LQR parameters insertion is shown in Section 6.2.7. Finally, the validation of experimental controller and its comparison with the original implementation is done in Section 7.1. Therefore, all the objectives have been accomplished.

Chapter 8

Conclusion

During this project, an experimental implementation of a control algorithm into the existing PX4 platform has been developed and tailored for an specific aircraft. This was achieved by understanding the behaviour of the PX4 stack and modifying it without disrupting its usual workflow.

Furthermore, after validating the algorithm and refining its implementation, the system can be tested using non-destructive simulations based on Software-in-the-Loop (SITL). These simulations provide an approximation to real-world conditions, enabling performance validation in realistic scenarios. This is a step towards deployment on the physical aircraft and subsequent validation through actual flight testing.

Additionally, a linear model of the H200 aircraft has been developed and tailored to meet the exact requirements of the project. This model has been validated through simulations, ensuring its accuracy and applicability to the control algorithm's design and testing processes.

However, some limitations persist. The integration of the control algorithm within existing PX4 modules and files imposes constraints on flexibility and may limit the performance of future updates due to the coexistence of legacy and experimental code. Furthermore, the implemented control algorithm focuses solely on managing aircraft attitude, leaving room for enhanced performance through the incorporation of broader control strategies. The absence of data regarding the direction of air velocity during flight also reduces the algorithm's effectiveness, as well as the lack of an optimal calibration method. In spite of that, LQR implementation has proved to have similar performance to the PID implementation. Nevertheless, LQR implementation demonstrated smoother trajectory rejection due to its nature.

Lastly, any changes in the aircraft's design or geometry would necessitate updates to both the dynamic model and the control variables to maintain optimal performance.

Considering everything, all the objectives set for the project have been accomplished.

Chapter 9

Annex

9.1 Sustainable development goals: Agenda 2030

Sustainable development goals	High	Medium	Low	Not applicable
SDG 1. No Poverty			x	
SDG 2. Zero Hunger			x	
SDG 3. Good Health and Well-being			x	
SDG 4. Quality Education		x		
SDG 5. Gender Equality			x	
SDG 6. Clean water and sanitation		x		
SDG 7. Affordable and Clean Energy		x		
SDG 8. Decent Work and Economic Growth		x		
SDG 9. Industry, Innovation, and Infrastructure	x			
SDG 10. Reduced Inequality			x	
SDG 11. Sustainable Cities and Communities		x		
SDG 12. Responsible Consumption and Production		x		
SDG 13. Climate Action		x		
SDG 14. Life Below Water		x		
SDG 15. Life on Land		x		
SDG 16. Peace, Justice, and Strong Institutions			x	
SDG 17. Partnerships for the Goals			x	

Table 9.1: Sustainable development goals relation with the project

The Sustainable and Development Goal 9. Industry, Innovation, and Infrastructure is the most related to the project as it works on the development of UAS technology.

Bibliography

[1] SA. Costea Andronache, JD. Cerdán Torres, JA. Such García, LM. García-Cuevas González, and S. García-Nieto Rodríguez. Diseño Conceptual Proyecto H200. *Universitat Politècnica de València*, 2023.

[2] José Luis Musoles Aguña. Design and implementation of angle-of-attack control system using active disturbance rejection control algorithms for fixed-wing unmanned aerial vehicles. *Universitat Politècnica de València*, 2024.

[3] Valyra Aerospace. Series vx overview. `https://valyraaerospace.com/series-vx`. Accessed: 2024-12-05.

[4] ICAO. Circular 328. *Unmanned Aircraft Systems (UAS)*, 2011.

[5] FAA. Unmanned aircraft definition. `https://www.faa.gov/faq/what-unmanned-aircraft-system-uas#:~:text=It%20is%20defined%20by%20statute,for%20general%20questions%20and%20comments.` Accessed: 2024-06-10.

[6] Wikipedia. Kettering bug. `https://commons.wikimedia.org/wiki/File:Kettering_Bug.jpg`. Accessed: 2024-07-04.

[7] Greg Goebel. Oq-2a radioplane. `https://commons.wikimedia.org/wiki/File:OQ-2A-Radioplane.jpg`. Accessed: 2024-07-05.

[8] Ben_pcc. V-1 flying bomb. `https://commons.wikimedia.org/wiki/File:V1Musee.jpg`. Accessed: 2024-07-05.

[9] Frank Garzelnick. Bqm-34f firebee ii rato launch. `https://commons.wikimedia.org/wiki/File:BQM-34F_launch_Tyndall_AFB_1982.JPEG`. Accessed: 2024-07-06.

[10] ZLEA. Project aquiline image. `https://commons.wikimedia.org/w/index.php?curid=122384187`. CC BY-SA 4.0 Accessed: 2024-06-12.

[11] ZLEA. Mcdonnell douglas aquiline. `https://commons.wikimedia.org/wiki/File:McDonnell_Douglas_Aquiline_(8-22-2022).jpg`. Accessed: 2024-07-07.

[12] Bobbi Zapka. Rq-4 global hawk. `https://commons.wikimedia.org/wiki/File:Global_Hawk_1.jpg`. Accessed: 2024-07-07.

[13] A. Savin. Dji phantom 4 pro image. `https://commons.wikimedia.org/w/index.php?curid=58430612`. CC FAL Accessed: 2024-06-12.

[14] Louis V. Schmidt. *Introduction to Aircraft Flight Dynamics*. AIAA Education, 1998.

[15] PX4. Px4 user guide. https://docs.px4.io/main/en/. Accessed: 2024-06-16.

[16] Arturo Urquizo. Pid scheme, cc by-sa 3.0. https://commons.wikimedia.org/w/index.php?curid=17633925. Accessed: 2024-06-30.

[17] MATLAB. Pole placement scheme. https://es.mathworks.com/help/control/ref/place.html?lang=en#mw_a10dbf9a-d759-4f09-88a2-dd5b4a50240d. Accessed: 2024-06-29.

[18] MATLAB. Lqi scheme. https://ch.mathworks.com/help/control/ref/ss.lqi.html. Accessed: 2024-06-29.

[19] Md. Sohel Rana, Hemanshu Pota, and I. Petersen. Model predictive control of atomic force microscope for fast image scanning. *Proceedings of the IEEE Conference on Decision and Control*, 12 2012.

[20] Mehmet Usta, Ömür Akyazi, and Adem Akpinar. Aircraft roll control system using lqr and fuzzy logic controller. *Innovations in Intelligent Systems and Applications (INISTA)*, 06 2011.

[21] Vicken Kasparian and Celal Batur. Model reference based neural network adaptive controller. *ISA Transactions*, 37(1):21–39, 1998.

[22] Christopher Loom. Introduction to Full State Feedback Control. https://www.youtube.com/watch?v=1zIIcYfp5QA&t=0s, May 2019.

[23] Christopher Loom. Introduction to Linear Quadratic Regulator (LQR) Control. https://www.youtube.com/watch?v=wEevt2a4SKI, Dec 2018.

[24] Marcos Carreres Talens. 11901 – Ampliación de Mecánica de Vuelo, Tema 02. Dinámica lineal longitudinal, Parte A. Linealización de las ecuaciones de Bryan. Aplicación al movimiento longitudinal. Accessed: 2024-07-06.

[25] DMMT. 11901_Mecánica de vuelo (II) Tema_02_A Apuntes Método de estudio y de cálculo de la dinámica del movimiento no estacionario de un avión. Linealización de las ecuaciones de Bryan. Movimiento Longitudinal. Accessed: 2024-07-15.

[26] Arnau García Cases. Diseño de un sistema de simulación de misiones autónomas para sistemas aéreos no tripulados. aplicación al proyecto horus upv. *Universitat Politècnica de València*, 2021.

[27] José Domingo Cerdán Torres. Design, analysis and manufacturing of an unmanned aircraft with distributed propulsion and hydrogen fuel cell power supply. *Universitat Politècnica de València*, 2023.

[28] Kiril Boychev. Pixhawk SIL Connector for Simulink.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ETSI Aeroespacial y Diseño Industrial

Design and implementation of multivariable control strategies for autopilots based on the PX4 control stack

TERMS OF REFERENCE

AUTHOR: Iborra Lérida, Alex

TUTOR: García-Nieto Rodríguez, Sergio

Universitat Politècnica de València

Escuela Técnica Superior de Ingeniería Aeroespacial y de Diseño Industrial

Grado en Ingeniería Aeroespacial

2024-2025 Course

Contents

1 Introduction

This document outlines the terms of reference for the project, detailing the technical and legal specifications required for its execution. It provides a comprehensive framework for the project's development. Additionally, it delineates the assignment of responsibilities, rights, and duties for both the client and the contractor, ensuring clear communication and understanding of each party's obligations to facilitate successful project completion.

2 Project description

This project consists of the following parts:

1. Modification of PX4 standard code to suit the experimental algorithm implementation. The client will be able to switch between control modes.
2. Implementation of the LQR algorithm code.
3. Comprobation of correct behaviour of the original implementation after its manipulation.
4. Linearization of the aircraft model. The dynamic model of the aircraft will be provided by the client, including all the necessary variables of geometry, aerodynamics or propulsion
5. Q and R matrices terms assignment.
6. Riccati's equation solving to obtain controller gain matrix.
7. Run software in the loop simulation with gain matrix terms.
8. System validation with results.
9. Comparison with original PID implementation.

3 General terms of reference

In this section the project contents are enumerated and the relationship between client and contractor and its administrative aspects are defined.

3.1 Project documents

The documents provided in the project are:

1. Memory
2. Costs
3. Terms of reference

3.2 General conditions

In this section the relationship between client and contractor is detailed.

Client functions

The responsibilities that lie on the client are the following:

- Mathematical model of the aircraft, including geometric, aerodynamic and propulsive information required to use control theory on it.
- Simulink® model of the aircraft to be able to perform Software in the Loop (SIL) simulations.
- Regular implementation of the autopilot.
- Control specifications of the desired experimental implementation.

Contractor functions

The responsibilities that lie on the contractor are the following:

- Accomplish the different sections of the project with good engineering practices and provide the documents indicated in section 3.1

However, as the dynamic model is a responsibility of the client, the contractor cannot guarantee its exactitude or the correct behaviour of the control.

3.3 Conditions of the project accomplishment

Project submission

All of the output of the project, including documents specified in section 3.1, and generated code, will be submitted to the client.

The code will be functional without the need of modifications or corrections, and will be properly commented for easy comprehension in case of future extensions of it. In case there is dissatisfaction of the final result by the client, modifications of it can be requested as long as it lies inside the project specifications. As previously stated, the dynamic model or experimental control specifications are responsibilities of the client. In case of dissatisfaction regarding these aspects, it is not the contractor responsibility to modify these.

For the project output to be replicated in other computer system, the required programs and tools will not be provided by the contractor. Instead, these will be detailed in section 4.

Deadline and time extensions

The project deadline is agreed between both sides of the contract. It is detailed in section 4, where the different stages of the development are also mentioned. If it is requested by the client, it will be needed to provide written evidence of it.

If beginning or ending a project section inside the deadline is not feasible due to force majeure, a time extension will be provided for the sake of the contract. This time extension will be agreed between both sides of the contract.

3.4 General economical conditions

This section will detail the economical conditions of the project. The Costs document will provide the complete version of this section.

Prices

For price calculations three different aspects are being considered:

- Direct costs: they include the labour cost, tools, material and software licenses.
- General costs: they include costs that are not considered in direct costs, such as energetic, reparation or maintenance costs. These are considered to be 15% of the direct costs.
- Industrial benefit: it is the economic surplus obtained by a business after eliminating costs associated to production and distribution of goods or services. They are considered to be 6% of the direct costs.

4 Technical terms of reference

4.1 Motivation

This section is responsible for describing the materials and tools used for the development of the project, which, due to the project nature, will include software licenses. Moreover, the terms of use of these are also described.

4.2 Materials and tools conditions

The materials and tools used are the different software programs needed for the code development and simulation. The terms of use of each of them are described:

- MATLAB® and Simulink®: For the former, version R2023b is used, and for the latter it is version 23.2. In addition, to be able to perform the required simulations, the following toolboxes are required, all with version 23.2:
  - Aerospace Blockset
  - Aerospace Toolbox

- Control System Toolbox
- Symbolic Math Toolbox

MATLAB is not open-source, it is a commercial software product that requires a paid license to use. Special academic licenses are available for students and educators at a discounted rate. These licenses may have additional restrictions, such as non-commercial use only. In this case, an educational version is used.

- QGroundControl: it is a software dedicated to plan an autonomous mission , communicate with the autonomous aircraft, and visualize the environment in a simulation or real mission. The source code of QGroundControl is dual-licensed under Apache 2.0 and GPLv3 (or later), the artwork/images are licensed under CC by SA. The version being used is v4.3.0.

- PX4 source code: It is an open-source flight control software for drones and other unmanned vehicles. It is the base code frame of the autopilot of the aircraft which will be modified to obtain an experimental version compatible with the regular implementation. PX4 is distributed under the BSD 3-Clause License. This is a permissive open-source license that allows for free use, modification, and distribution of the software, with minimal restrictions. The main conditions are that the original copyright notice, list of conditions, and disclaimers must be retained in all copies or substantial portions of the software. The version used is v1.14.0.

- Microsoft Visual Studio Code (VS Code): It is a free, open-source code editor developed by Microsoft. It is used to modify the PX4 source C++ code and develop the experimental control. Visual Studio Code is distributed under the Microsoft Software License Terms. The source code for Visual Studio Code is available on GitHub under the MIT License, which allows for free use, modification, and distribution of the source code. However, the official binary distributions provided by Microsoft come with a proprietary license that includes additional terms and conditions, particularly around redistribution and usage. The version used is 1.91.0.

- Texmaker: It is a free, cross-platform LaTeX editor, which is the software used to create the documents of the project. Texmaker is distributed under the GNU General Public License (GPL) v2. This license allows users to freely use, modify, and distribute the software, provided that any distributed versions or derivative works also come under the GPL. The version used is 5.1.4.

- MiKTeX: It is a distribution of the TeX/LaTeX typesetting system. It includes a package manager that automatically installs missing packages from the Internet. It is required for Texmaker to work, so the project documents also rely on this software. MiKTeX is distributed under the MiKTeX Project Public License, which is similar to the LaTeX Project Public License (LPPL). The source code for MiKTeX is available, and users can contribute to its development. The version used is 4.12.

4.3 Description of tools usage

This section provides the link between the procedures and tools needed for the project accomplishment:

- Modification of PX4 standard code and implementation of LQR algorithm code: Using Microsoft Visual Studio and PX4 software, the C++ PX4 code is modified to suit the experimental LQR code and allow the different implementations to be switched from QGroundControl console.
- Aircraft dynamic model linearization and LQR parameter calculation: the model is implemented in Simulink® making use of the Aerospace Toolbox, Aerospace Blockset, Symbolic Math Toolbox and Control System Toolbox. The implementation is based on Bryan equations which approximate the behaviour of the flight of an aircraft. It is linearized and then using MATLAB® the LQR parameters are calculated. Then, using Microsoft Visual Studio and PX4 software they are implemented in the code.
- Software in the Loop simulation: the purpose is to run the experimental code and check that the original implementation works as well. With the Simulink® model connected to QGroundControl, the mission is planned, launched and monitored. In this scenario, the regular and experimental implementations can be switched.
- System validation: by looking at the error between the desired trajectory and the followed one from QGroundControl, the system performance is assessed.

4.4 Final checks and adjustments

Upon completion of project components, the client may assess the developed simulation system through testing with various autonomous missions. If the results of these tests are unsatisfactory, the client may request modifications to the simulation system, always falling between the project scope. This phase corresponds to the final adjustments of the project. Ultimately, the client will receive all documentation generated throughout the project, along with the code developed for each component, as per the terms of execution.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ETSI Aeroespacial y Diseño Industrial

Design and implementation of multivariable control strategies for autopilots based on the PX4 control stack

COSTS

AUTHOR: Iborra Lérida, Alex

TUTOR: García-Nieto Rodríguez, Sergio

Universitat Politècnica de València

Escuela Técnica Superior de Ingeniería Aeroespacial y de Diseño Industrial

Grado en Ingeniería Aeroespacial

2024-2025 Course

Contents

List of Tables

1 Introduction

In this document the different costs that are needed for the accomplishment of the project "Design and implementation of multivariable control strategies for autopilots based on the PX4 control stack" are presented.

Costs are divided into labour, material costs and software license costs. Global cost will be the addition of these, plus 15% due to general costs and 6% of industrial benefit, as stated in Terms of Reference document. Finally, an addition of 21% of VAT due to Spanish regulation is required.

2 Partial costs

Time spent in each project stage is crucial to define partial costs. To do so, the hour of labour cost will be calculated to obtain the price of each stage of the project.

2.1 Labour

Considering a workday that consists of 8 hours during 22 days a month over 11 months of a year (considering 1 month of vacation), **yields a total of 1,936 hours.**

Assuming the labour to be a junior engineer (Salary level 2), according to "Disposición 5873 del BOE núm. 73 de 2024" from the Spanish state bulletin, in which the XX collective agreement of engineering companies and technical offices is revised to change salary amounts, the annual salary in 2024 would be of 22,224.26€. To this amount, a 30% increase to pay Social Security contribution will be added.

The price of an hour is obtained with annual salary and annual worked hours, which in this case is **14.91€**. This is summarized in Table 1.

Annual salary [€]	**Worked hours**	**Hour salary [€/h]**
22,224.26€	1,936	11.48

Annual salary (SS) [€]	**SS contribution**	**Final hour salary [€/h]**
28,869.30	6,645.05	14.91

Table 1: Summary of labour hour cost

2.1.1 Project phases

The stages are defined in Terms of Reference document and are summarized here:

1. Modification of PX4 standard code and implementation of LQR algorithm code: Using Microsoft Visual Studio Code and PX4 software.

2. Aircraft dynamic model linearization and LQR parameter calculation: Using MATLAB®, Simulink®, Aerospace Toolbox, Aerospace Blockset, Symbolic Math Toolbox, Control System Toolbox, Microsoft Visual Studio and PX4 software.

3. Software in the Loop simulation: Using MATLAB®, Simulink®, Aerospace Toolbox, Aerospace Blockset, Symbolic Math Toolbox, Control System Toolbox and QGroundControl.

4. System validation and comparison with original implementation: using MATLAB® and QGroundControl.

Total labour costs are represented in Table 2, where time spent in each project phase is assigned and calculated its cost with the previous section work.

Project phase	**Time spent [h]**	**Total [€]**
1	165	2460.15
2	30	447.30
3	35	521.85
4	25	372.75
Documents creation	100	110.20
TOTAL	355	5293.05

Table 2: Project phases costs

2.2 Tools

The main tool used for the project is the computer in which the code is modified and the simulations are run. In this case, an *HP VICTUS 16-e1007ns*. It is considered to be amortized in a period of 5 years, which is 10,080 hours.

Tool	**HP VICTUS 16-e1007ns**
Acquisition cost [€]	849.50
Hours amortized	10,080
Hours spent in the project	355
Cost of project utilisation [€]	29.92

Table 3: Tools costs

The consulted bibliography is of free access so there are not associated costs to it.

2.3 Software licenses

As there is only one employee doing the work requested by the client, software licenses costs will be considered for one computer, the one described in the previous section. Again, the different software licenses are considered to be amortized during a period of time, so cost calculation will be based on the portion of time from the amortization period. In Table 4 software licenses costs are collected. MATLAB® related costs are obtained from `https://es.mathworks.com/pricing-licensing.html`.

Software	**Annual cost [€]**	**Time used [h]**	**Cost [€]**
MATLAB®	900	90	41.84
Simulink®	1360	65	45.66
Aerospace Toolbox	560	65	18.80
Aerospace Blockset	780	65	26.19
Symbolic Math Toolbox	448	65	15.04
Control System Toolbox	520	65	17.46
Microsoft Visual Studio Code	0	195	0
PX4	0	195	0
QGroundControl	0	55	0
TexMaker	0	100	0
MikTex	0	100	0
TOTAL			164.99

Table 4: Software licenses costs

3 Global costs

Total cost calculation, as stated previously, will be comprised of the addition of project phase, tools, and software license costs. This is shown in Table 5 before applying the percentage increase due to general costs, industrial benefit or VAT.

Cost Type	**Total [€]**
Labour	5293.05
Tools	29.92
Software Licenses	164.99
TOTAL	5487.96

Table 5: Total gross costs

Tender budget is obtained after the percentage increase of general costs, industrial benefit and VAT. It is shown in Table 6.

	Total [€]
Total gross cost	5487.96
+15% general costs	823.20
+6% industrial benefit	329.28
Investment costs	6640.43
+21% VAT	1394.49
Tender budget	8034.92

Table 6: Tender budget

Therefore, the total cost of this project is in EUROS:

EIGHT THOUSAND THIRTY-FOUR and NINETY-TWO