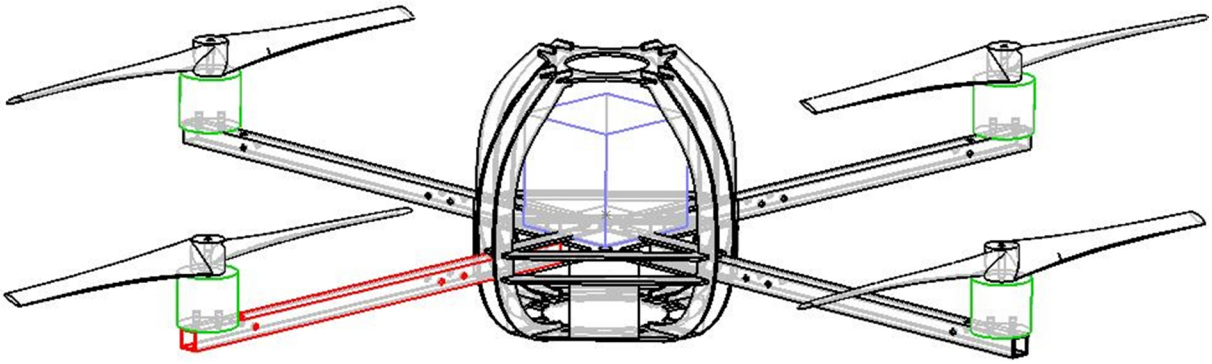
INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa



Quadrotor prototype

Jorge Miguel Brito Domingues

Dissertação para obtenção do Grau de Mestre em

Engenharia Mecânica

Júri:

Presidente:	Prof. João Rogério Caldas Pinto (DEM)
Orientadores:	Prof. José Raúl Carreira Azinheira (DEM)
	Profª. Alexandra Bento Moutinho (DEM)
Vogal:	Prof. Agostinho Rui Alves da Fonseca

Outubro de 2009

"Imagination is everything. It is the preview of life's coming attractions."

Albert Einstein

Abstract

There is currently a limited diversity of vehicles capable of Vertical Take-Off and Landing. However there is a visible interest by scholars and admirers of Unmanned Aerial Vehicles for a particular type of aircraft called quadrotor. This work aims precisely at the design and control of a quadrotor prototype having a tri-axis accelerometer and a compass as its sensors.

This project started with the design of the quadrotor having in mind a list of objectives to be fulfilled by it. Then followed the modeling of the dynamics and kinematics of the aircraft's motion, as well as of its motors and sensors. This led to the implementation of the quadrotor in a computer simulation environment called Simulink®.

Given the outputs provided by the available sensors, a Kalman filter was developed capable of estimating the angular velocities and Euler angles of the quadrotor, being therefore possible to have knowledge of its attitude. This filter was combined with a Linear-Quadratic Regulator in order to make the system follow a given attitude reference. Finally, the results of the simulations and real implementation in the quadrotor are presented.

Resumo

Existe actualmente uma variedade reduzida de veículos capazes de descolagem e aterragem vertical. No entanto existe um interesse visível por parte de investigadores e admiradores de Veículos Aéreos Não Tripulados num tipo especial de aeronave denominada de quadrirotor. Este trabalho tem precisamente como objectivo desenvolver e controlar um protótipo de um quadrirotor, possuindo apenas como sensores um acelerómetro triaxial e uma bússola.

Começou-se por conceber o quadrirotor tendo em vista uma lista de objectivos a cumprir pelo mesmo. Seguiu-se a modelação da dinâmica e cinemática da aeronave, assim como também dos seus motores e sensores. Daí resultou a implementação do sistema em ambiente de simulação computacional Simulink®.

Tendo em conta as saídas fornecidas pelos sensores disponíveis, foi desenvolvido um filtro de Kalman capaz de estimar apenas as velocidades angulares e os ângulos de Euler do quadrirotor, sendo portanto possível ter conhecimento da atitude do mesmo. Este filtro foi combinado com um controlador LQR por forma fazer o sistema seguir uma dada referência de atitude.

Finalmente, os resultados das simulações e da implementação real no quadrirotor são apresentados.

Keywords

- Quadrotor modelling
- Quadrotor simulation
- Kalman filter
- Linear-Quadratic Regulator

Acknowledgments

First of all, I would like to thank my teachers, Professor José Raul Azinheira and Professor Alexandra Bento Moutinho for their advice, support and availability to speak with me throughout the course of this thesis.

I must not forget to thank my colleague Tiago Rita, who shared with me the difficulty of working with a UAV, Tiago Gonçalves also for offering his aid and knowledge on UAVs, and Mr. Raposeiro who helped me with the quadrotor construction.

Last but not least, I want to give special thanks to my family for all their love, support, and for encouraging me to work my way through college, a path that made me gain precious life experiences, which i would have missed otherwise.

Contents

List of Tables

List of Figures

[1] http://www.thingsmagazine.net/blogimages/
[2] http://www.aviastar.org/helicopters_eng/bothezat.php
[3] http://www.aviastar.org/helicopters_eng/convertawings.php
[4] http://www.symscape.com/node/622
[5] http://aero.epfl.ch/page57689.html

Notation

Acronyms

ALIV	-	Autonomous Locomotion Individual Vehicle
COG	-	Center Of Gravity
DC	-	Direct Current
ESC	-	Electronic Speed Controller
GPS	-	Global Positioning System
LED	-	Light-emitting diode
LQR	-	Linear-Quadratic Regulator
LQ	-	Linear-Quadratic
LQG	-	Linear-Quadratic Gaussian
MIMO	-	Multiple-Input Multiple-Output
PD	-	Proportional-Derivative
PID	-	Proportional-Integral-Derivative
PWM	-	Pulse-Width Modulation
RC	-	Remote Control
SISO	-	Single-Input Single-Output
UAV	-	Unmanned Aerial Vehicle
VTOL	-	Vertical Take-Off and Landing

Subscripts

B	-	Regarding the aircraft's body centered coordinate frame
g	-	Regarding gravity
net	-	Resultant (e.g. resultant force)

Superscripts

T	-	Matrix transpose

List of symbols

Symbol	Domain	Unit	Description
ε	$\mathbb{R}$	-	Quaternion angle of rotation
θ	$\mathbb{R}$	rad	Pitch angle
$\dot{\theta}$	$\mathbb{R}$	$rad.s^{-1}$	Pitch angle rate
ρ	$\mathbb{R}$	$kg.m^{-3}$	Air density at sea level and $20°C$
τ	$\mathbb{R}$	s	Time constant
ϕ	$\mathbb{R}$	rad	Roll angle
$\dot{\phi}$	$\mathbb{R}$	$rad.s^{-1}$	Roll angle rate
ψ	$\mathbb{R}$	rad	Yaw angle
$\dot{\psi}$	$\mathbb{R}$	$rad.s^{-1}$	Yaw angle rate
ω	$\mathbb{R}$	$rad.s^{-1}$	Propeller angular velocity
$\boldsymbol{\omega}'$	$\mathbb{R}^{3\times1}$	$rad.s^{-1}$	Quadrotor's angular velocities $\begin{bmatrix} P & Q & R \end{bmatrix}$
ω_1, ω_2, ω_3, ω_4	$\mathbb{R}$	$rad.s^{-1}$	Angular velocity of each motor
$\mathbf{A}$	$\mathbb{R}^{n\times n}$	-	State matrix (n states)
$\mathbf{A^K}$	$\mathbb{R}^{n\times n}$	-	Kalman filter state matrix (n states)
$\mathbf{a}_m$	$\mathbb{R}^{3\times1}$	$m.s^{-2}$	Acceleration measured by the accelerometer
$\mathbf{a}_p$	$\mathbb{R}^{3\times1}$	$m.s^{-2}$	Absolute acceleration at the point where the sensor is located, $\mathbf{a}_p = \begin{bmatrix} a_{p_x} & a_{p_y} & a_{p_z} \end{bmatrix}$
$\mathbf{a}_B$	$\mathbb{R}^{3\times1}$	$m.s^{-2}$	Acceleration in the quadrotor's body
$\mathbf{B}$	$\mathbb{R}^{n\times m}$	-	Input matrix (n states, m inputs)
$\mathbf{B^K}$	$\mathbb{R}^{n\times m}$	-	Kalman filter input matrix
c_T	$\mathbb{R}$	none	Thrust coefficient of a propeller
c_P	$\mathbb{R}$	-	Power coefficient of a propeller
$\mathbf{C}$	$\mathbb{R}^{q\times n}$	-	Output matrix (q outputs, n states)
$\mathbf{C^K}$	$\mathbb{R}^{q\times n}$	-	Kalman filter output matrix (q outputs, n states)
D_P	$\mathbb{R}$	m	Propeller diameter
$\mathbf{D}$	$\mathbb{R}^{q\times m}$	-	Feedforward matrix (q outputs, m inputs)
$\mathbf{D^K}$	$\mathbb{R}^{q\times m}$	-	Kalman filter feedforward matrix (q outputs, m inputs)
d_{cg}	$\mathbb{R}$	m	Distance between the quadrotor's center of gravity and a motor
$\mathbf{e}_{\dot{x}}$	$\mathbb{R}^{n\times1}$	-	Error of the dynamics derivative (n states)
$\mathbf{e}_y$	$\mathbb{R}^{q\times1}$	-	Output error (q outputs)
$\mathbf{F}_{net}$	$\mathbb{R}^{3\times1}$	N	Combination of all the forces acting on the quadrotor, $\mathbf{F}_{net} = \begin{bmatrix} F_{x'} & F_{y'} & F_{z'} \end{bmatrix}$
$\mathbf{F}_P$	$\mathbb{R}^{3\times1}$	N	Total thrust generated by the propellers, $\mathbf{F}_P = \begin{bmatrix} F_{P_x} & F_{P_y} & F_{P_z} \end{bmatrix}$
f_t	$\mathbb{R}$	s	Flight autonomy
f	$\mathbb{R}$	-	Quadrotor's model function to linearize
$G(s)$	$\mathbb{R}$	-	Motor dynamics transfer function
$\mathbf{G}$	$\mathbb{R}^{n\times p}$	-	Observation matrix (n states, p process noise measurements)
g	$\mathbb{R}$	$m.s^{-2}$	Earth's gravity (constant value of $9.81m.s^{-2}$)

$\mathbf{H}$	$\mathbb{R}^{q \times p}$	-	Matrix that relates the process noise with the outputs of a system in the state space form (q outputs, p process noise measurements)
$\mathbf{I}$	$\mathbb{R}^{3 \times 3}$	$kg.m^2$	Inertia matrix of the quadrotor
$\underline{\mathbf{I}}$	$\mathbb{R}^{n \times n}$	-	Identity matrix (n states)
I_x, I_y, I_z	$\mathbb{R}$	$kg.m^2$	Elements of a motor's inertia tensor respectively related to $u_{x'}$, $u_{y'}$ and $u_{x'}$
$I^{'}$	$\mathbb{R}$	A	Electrical current consumption of the motor
I_0	$\mathbb{R}$	A	Electrical current consumption of the motor in no-load condition
J	$\mathbb{R}$	-	Propeller advance ratio
J_{LQR}	$\mathbb{R}$	-	LQR cost function
K	$\mathbb{R}$	-	Transfer function gain
K_T	$\mathbb{R}$	$kg.m^2.rad^{-2}$	Constant value to calculate the thrust provided by a propeller given its angular speed
K_M	$\mathbb{R}$	$kg.m^2.rad^{-2}$	Constant value to calculate the moment of a propeller given its angular speed
K_{TM}	$\mathbb{R}$	m	Constant value that relates thrust and moment of a propeller
$\overline{\mathbf{K}}^{m \times q}$	$\mathbb{R}$	-	LQR gain matrix (m inputs, q outputs)
k_v	$\mathbb{R}$	$RPM.V^{-1}$	Propeller's speed in revolutions per minute per unit of voltage
k	$\mathbb{R}$	-	Time instant k
$\mathbf{L}^{n \times q}$	$\mathbb{R}$	-	Kalman gain (n states, q outputs)
$\mathbf{M}_{net}$	$\mathbb{R}^{3 \times 1}$	-	Sum of all the moments acting on the quadrotor, $\mathbf{M}_{net} = \begin{bmatrix} M_x & M_y & M_z \end{bmatrix}$
M_P	$\mathbb{R}$	$N.m$	Propeller's moment
m	$\mathbb{R}$	kg	Mass of the quadrotor
N_c	$\mathbb{R}$	rad	North magnetic pole
N	$\mathbb{R}$	-	Order of the state space system
n	$\mathbb{R}$	s^{-1}	Propeller's speed in revolutions per second
n_1, n_2, n_3	$\mathbb{R}$	-	Elements of a unit vector used to build a quaternion
O_{ABC}	$\mathbb{R}^3$	-	Aircraft-Body-Centered reference frame
O_{NED}	$\mathbb{R}^3$	-	North-East-Down reference frame
$\mathcal{O}$	$\mathbb{R}^{nq \times n}$	-	Observability matrix (n states, q outputs)
P	$\mathbb{R}$	$rad.s^{-1}$	Angular speed around the $u_{x'}$ axis of the quadrotor
$\dot{P}$	$\mathbb{R}$	$rad.s^{-2}$	Angular acceleration of the quadrotor along the $u_{x'}$ axis of the inertial reference frame
P_P	$\mathbb{R}$	W	Propeller power
$\overline{\mathbf{P}}$	$\mathbb{R}^{n \times 1}$	-	Steady-state error covariance matrix (n states)
$p^{'}$	$\mathbb{R}$	-	Linearization point of the system
$\widehat{\mathbf{P}}$	$\mathbb{R}^{m \times q}$	-	Unique positive-definite solution for the LQR controller Riccati equation (m inputs, q outputs)
$\mathcal{P}$	$\mathbb{R}$	μs	Pulse width of the PWM signal driving a motor
$\mathcal{P}_0$	$\mathbb{R}$	μs	PWM signal pulse width of the motor's dead zone
Q	$\mathbb{R}$	$rad.s^{-1}$	Angular speed around the $u_{y'}$ axis of the quadrotor

$\dot{Q}$	$\mathbb{R}$	$rad.s^{-2}$	Angular acceleration of the quadrotor along the $u_{y'}$ axis of the inertial reference frame
$Q^{'}$	$\mathbb{R}$	$A.s$	Electric charge of the battery (usually presented in mAh)
Q_k	$\mathbb{R}$	-	Process noise covariance introduced through input k
$\overline{\mathbf{Q}}$	$\mathbb{R}^{k\times k}$	-	Process noise covariance matrix (k observable states)
$\widehat{\mathbf{Q}}$	$\mathbb{R}^{n\times n}$	-	LQR controller weighting matrix (n states to control)
$\hat{Q}_k$	$\mathbb{R}$	-	Diagonal elements of $\widehat{Q}$
$\mathbf{q}$	$\mathbb{R}^{4\times 1}$	-	Quaternion, $\mathbf{q} = \begin{bmatrix} q_0 & q_1 & q_2 & q_3 \end{bmatrix}$
$\dot{\mathbf{q}}$	$\mathbb{R}^{4\times 1}$	-	Angular velocity quaternion
R	$\mathbb{R}$	$rad.s^{-1}$	Angular speed around the $u_{z'}$ axis of the quadrotor
R_m	$\mathbb{R}$	Ω	Motor's copper coil electric resistance
$\dot{R}$	$\mathbb{R}$	$rad.s^{-2}$	Angular acceleration of the quadrotor along the $u_{z'}$ axis of the inertial reference frame
$\mathbf{R}^{'}(\phi)$	$\mathbb{R}^{3\times 3}$	-	Rotation matrix that expresses the orientation of the O_{ABC} frame around the u_x axis of O_{NED}
$\mathbf{R}^{'}(\theta)$	$\mathbb{R}^{3\times 3}$	-	Rotation matrix that expresses the orientation of the O_{ABC} frame around the u_x axis of O_{NED}
$\mathbf{R}^{'}(\psi)$	$\mathbb{R}^{3\times 3}$	-	Rotation matrix that expresses the orientation of the O_{ABC} frame around the u_x axis of O_{NED}
R_{acc}	$\mathbb{R}$	$m.s^{-2}$	Accelerometer reading resolution of the Arduino board
R_k	$\mathbb{R}$	-	Sensor noise covariance from sensor reading k
$\overline{\mathbf{R}}$	$\mathbb{R}^{i\times i}$	-	Measurement noise covariance matrix (i sensor measurements)
$\widehat{\mathbf{R}}$	$\mathbb{R}^{m\times m}$	-	LQR controller weighting matrix (m inputs)
$\hat{R}_k$	$\mathbb{R}$	-	Diagonal elements of $\widehat{R}$
r	$\mathbb{R}$	m	Propeller radius
$\overrightarrow{\mathbf{r}}$	$\mathbb{R}^{3\times 1}$	m	Position of the O_{ABC} coordinate frame relative to O_{NED}
$\mathbf{S}$	$\mathbb{R}^{3\times 3}$	-	Rotation matrix (also known as direction cosine matrix)
$\mathbf{S}_q$	$\mathbb{R}^{3\times 3}$	-	Quaternion equivalent of the rotation matrix
T	$\mathbb{R}$	N	Propeller thrust
$\mathbf{T}$	$\mathbb{R}^{3\times 3}$	-	Matrix that relates the body-fixed angular velocity vector with the rate of change of the Euler angles
T_P	$\mathbb{R}$	s	Time period between the passage of two propeller blades
t_s	$\mathbb{R}$	s	Sample time.
$\dot{U}$	$\mathbb{R}$	$m.s^{-2}$	Linear acceleration of the quadrotor along the u_x axis of the inertial reference frame
u_0	$\mathbb{R}$	$m.s^{-1}$	Aircraft flight velocity
u_x	$\mathbb{R}$	-	X-axis of the O_{NED} reference frame
$u_{x'}$	$\mathbb{R}$	-	X-axis of the O_{ABC} reference frame
u_y	$\mathbb{R}$	-	Y-axis of the O_{NED} reference frame
$u_{y'}$	$\mathbb{R}$	-	Y-axis of the O_{ABC} reference frame
u_z	$\mathbb{R}$	-	Z-axis of the O_{NED} reference frame.
$u_{z'}$	$\mathbb{R}$	-	Z-axis of the O_{ABC} reference frame
$\overline{\mathbf{u}}$	$\mathbb{R}^{4\times 1}$	-	Vector containing the inputs of the system at an equilibrium point

$\mathbf{u}^K$	$\mathbb{R}^{4\times1}$	-	LQR vector of control actions
$u_{k,max}$	$\mathbb{R}$	-	Highest tolerable value for input k
$\dot{V}$	$\mathbb{R}$	$m.s^{-2}$	Linear acceleration of the quadrotor along the u_y axis of the inertial reference frame
V_m	$\mathbb{R}$	V	Motor's power supply voltage
V_0	$\mathbb{R}$	V	Minimum power supply required for propeller rotation
$\mathbf{v}^{'}$	$\mathbb{R}^{3\times1}$	$m.s^{-1}$	Vector containing the quadrotor's linear velocities $\begin{bmatrix} U & V & W \end{bmatrix}$
$\mathbf{v}_p$	$\mathbb{R}^{3\times1}$	$m.s^{-1}$	Vector of linear velocities at the point where the accelerometer is located
$\mathbf{v}_{cg}$	$\mathbb{R}^{3\times1}$	$m.s^{-1}$	Vector of linear velocities of the quadrotor's center of gravity
$\mathbf{v}$	$\mathbb{R}^i$	-	Measurement noise (i sensor measurements)
$\dot{W}$	$\mathbb{R}$	$m.s^{-2}$	Linear acceleration of the quadrotor along the u_z axis of the inertial reference frame
w_v	$\mathbb{R}$	$rad.s^{-1}.V^{-1}$	Propeller angular speed in S.I. units per unit of voltage
$\mathbf{w}$	$\mathbb{R}^{p\times1}$	-	Process noise (p process noise measurements)
X	$\mathbb{R}$	m	Position of the quadrotor's center of gravity along the x-axis the inertial reference frame O_{NED}
$\dot{X}$	$\mathbb{R}$	$m.s^{-1}$	Time derivative if the O_{ABC} frame's position along u_x
$\mathbf{x}$	$\mathbb{R}^{n\times1}$	-	State vector (n states)
$\overline{\mathbf{x}}$	$\mathbb{R}^{n\times1}$	-	Vector containing the states of the system at an equilibrium point (n states)
$x_{k,max}$	$\mathbb{R}$	-	Highest tolerable value for state k
$\hat{\mathbf{x}}$	$\mathbb{R}^{n\times1}$	-	Estimated states by the Kalman filter (n states)
Y	$\mathbb{R}$	m	Position of the quadrotor's center of gravity along the y-axis the inertial reference frame O_{NED}
$\dot{Y}$	$\mathbb{R}$	$m.s^{-1}$	Time derivative if the O_{ABC} frame's position along u_y
y	$\mathbb{R}$	-	Output vector
$\hat{\mathbf{y}}$	$\mathbb{R}^{q\times1}$	-	Estimated outputs by the Kalman filter (q outputs)
Z	$\mathbb{R}$	m	Position of the quadrotor's center of gravity along the z-axis the inertial reference frame O_{NED}
$\dot{Z}$	$\mathbb{R}$	$m.s^{-1}$	Time derivative if the O_{ABC} frame's position along u_z

Chapter 1

Introduction

1.1 Definition and purpose of a quadrotor

A quadrotor, or quadrotor helicopter, is an aircraft that becomes airborne due to the lift force provided by four rotors usually mounted in cross configuration, hence its name. It is an entirely different vehicle when compared with a helicopter, mainly due to the way both are controlled. Helicopters are able to change the angle of attack of its blades, quadrotors cannot.

At present, there are three main areas of quadrotor development: military, transportation (of goods and people) and Unmanned Aerial Vehicles (UAVs) [5]. UAVs can be classified into two major groups: heavier-than-air and lighter-than-air. These two groups self divide in many other that classify aircrafts according to motorization, type of liftoff and many other parameters. Vertical Take-Off and Landing (VTOL) UAVs like quadrotors have several advantages over fixed-wing airplanes. They can move in any direction and are capable of hovering and fly at low speeds. In addition, the VTOL capability allows deployment in almost any terrain while fixed-wing aircraft require a prepared airstrip for takeoff and landing. Given these characteristics, quadrotors can be used in search and rescue missions, meteorology, penetration of hazardous environments (e.g. exploration of other planets) and other applications suited for such an aircraft. Also, they are playing an important role in research areas like control engineering, where they serve as prototypes for real life applications.

1.2 Quadrotor operation

Each rotor in a quadrotor is responsible for a certain amount of thrust and torque about its center of rotation, as well as for a drag force opposite to the rotorcraft's direction of flight. The quadrotor's propellers are not all alike. In fact, they are divided in two pairs, two pusher and two puller blades, that work in contra-rotation. As a consequence, the resulting net torque can be null if all propellers turn with the same angular velocity, thus allowing for the aircraft to remain still around its center of gravity.

In order to define an aircraft's orientation (or attitude) around its center of mass, aerospace engineers usually define three dynamic parameters, the angles of yaw, pitch and roll. This is very useful because the forces used to control the aircraft act around its center of mass, causing it to pitch, roll or yaw. Changes in the pitch angle are induced by contrary variation of speeds in propellers 1 and 3 (see Figure 1.2.1), resulting in forward or backwards translation. If we do this same action for propellers 2 and 4, we

can produce a change in the roll angle and we will get lateral translation. Yaw is induced by mismatching the balance in aerodynamic torques (i.e. by offsetting the cumulative thrust between the counter-rotating blade pairs). So, by changing these three angles in a quadrotor we are able to make it maneuver in any direction (Figure 1.2.2).

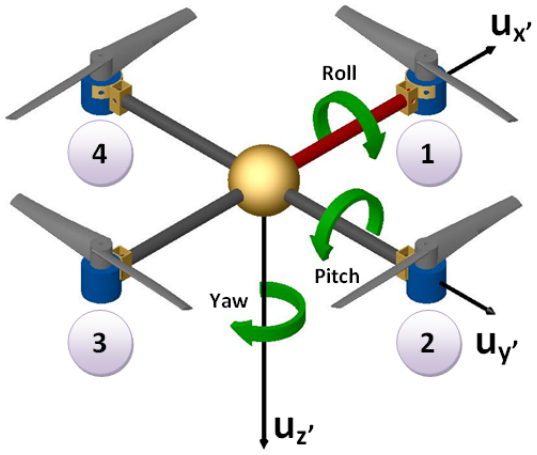


Figure 1.2.1: Yaw, pitch and roll rotations of a common quadrotor.

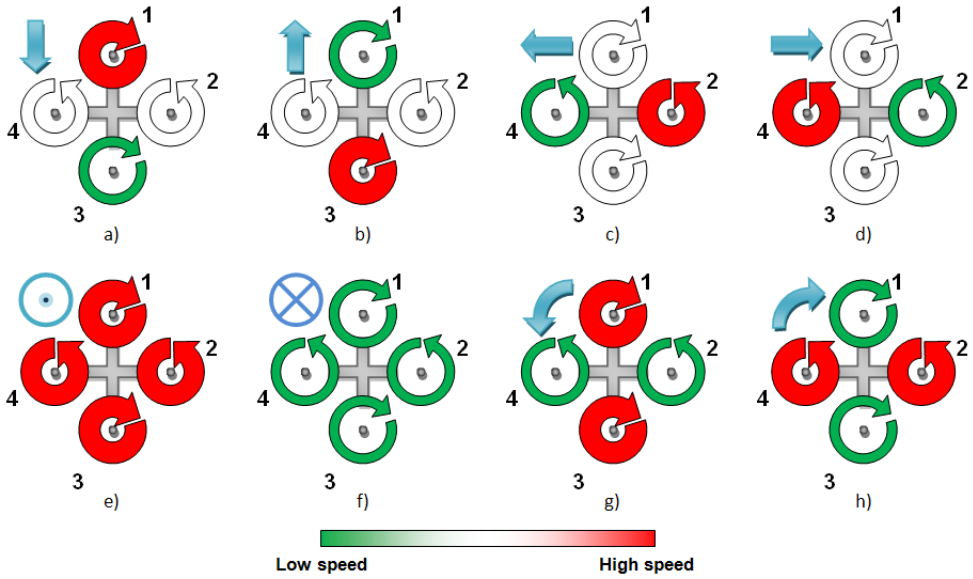


Figure 1.2.2: Illustration of the various movements of a quadrotor.

1.3 Historical role of quadrotors

This story begins in the 20th century, when Charles Richet, a French scientist and academician, built a small, un-piloted helicopter (see reference [6]). Although his attempt was not a success, Louis Bréguet, one of Richet's students, was inspired by his tutor's example. Later in 1906, Louis and his brother,

Jacques Bréguet began the construction of the first quadrotor. Louis executed many tests on airfoil shapes, proving that he had at least some basic understanding of the requirements necessary to achieve vertical flight.

In 1907 they had finished the construction of the aircraft which was named Bréguet-Richet Gyroplane No. 1 (Figures 1.3.1 and 1.3.2), a quadrotor with propellers of 8.1 meters in diameter each, weighting 578 kg (2 pilots included) and with only one 50 hp (37.3 kW) internal combustion engine, which drove the rotors through a belt and pulley transmission. Of course at that time they had no idea how they would control it, the main concern was to ensure the aircraft would achieve vertical flight. The first attempt of flight was done in between August and September of 1907 with witnesses saying they saw the quadrotor lift 1.5 m into the air for a few moments, landing immediately afterwards. Those same witnesses also mentioned the aircraft was stabilized, and perhaps even lifted by men assisting on the ground.

Discouraged by the lack of success of the Gyroplane No. 1, Bréguet and his mentor continued their pursuit to build vertical flight machines and afterwards also temporarily dedicated themselves to the development of fixed-wing aircraft, area where they became very successful. Louis never abandoned his passion for vertical flight aircraft and in 1932 he became one of the pioneers of helicopter development [6].

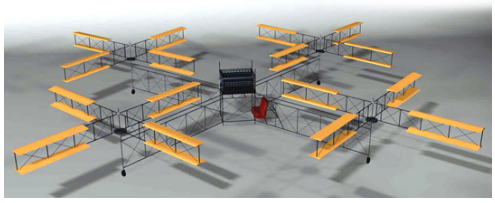


Figure 1.3.1: 3D model of the Gyroplane No. 1[1].

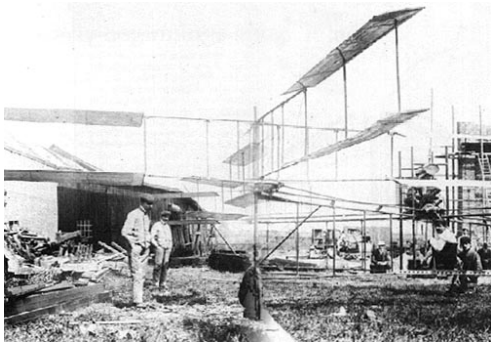


Figure 1.3.2: Bréguet-Richet Gyroplane No. 1 of 1907[2].

Etienne Oemichen, another engineer, also began experimenting with rotating-wing designs in 1920. He designed a grand total of six different vertical lift machines. The first model failed in lifting from the ground but Oemichen was a determined person, so he decided to add a hydrogen-filled balloon to provide both stability and lift. His second aircraft, the Oemichen No. 2 (Figure 1.3.3), had four rotors and eight propellers, supported by a cruciform steel-tube framework layout. Five of the propellers were meant to stabilize the machine laterally, another for steering and two for forward propulsion. Although rudimentary, this machine achieved a considerable degree of stability and controllability, having made more than a thousand test flights in the middle of that decade. It was even possible to maintain the aircraft several minutes in the air. In the 14th of May the machine was airborne for fourteen minutes and it flew more than a mile. But Oemichen was not satisfied with the poor heights he was able to fly, and the next machines had only a main rotor and two extra anti-torque rotors.

[1] http://www.thingsmagazine.net/blogimages/
[2] http://www.ctie.monash.edu.au/hargrave/images

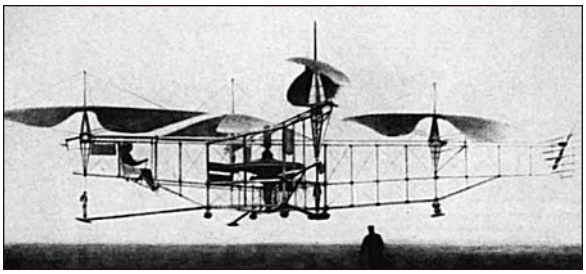


Figure 1.3.3: The Oemichen No.2 of 1922[3].

The army also had an interest for vertical lift machines. In 1921, Dr. George de Bothezat and Ivan Jerome were hired to develop one for the US Army Air Corps. The result was a 1678 kg structure with 9 m arms and four 8.1 m six-blade rotors (Figure 1.3.4). The army contract required that the aircraft would hover at 100 m high, but the best they achieved was 5 m. At the end of the project Bothezad demonstrated the vehicle could be quite stable, however it was underpowered and unresponsive, among other technical problems.

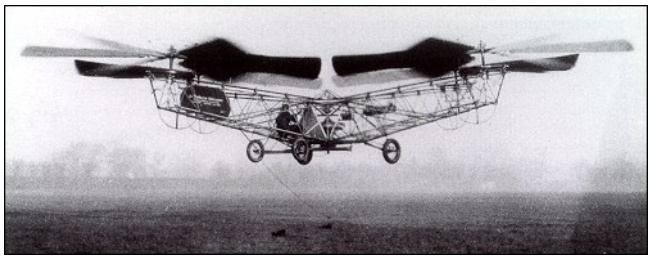


Figure 1.3.4: Quadrotor designed by Dr. Bothezat an Ivan Jerome. This photograph was taken at take-off during its first flight in October 1922[4].

Later in 1956, a quadrotor helicopter prototype called “Convertawings Model A” (see Figure 1.3.5) was designed both for military and civilian use. It was controlled by varying the thrust between rotors, and its flights were a success, even in forward flight. The project ended mainly due to the lack of demand for the aircraft.

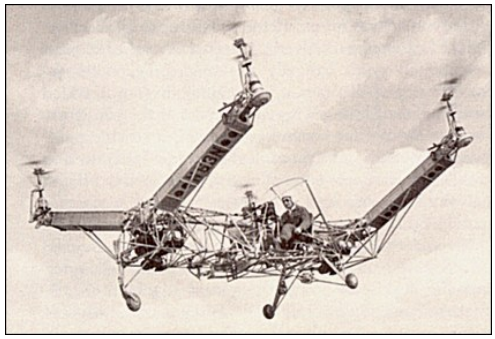


Figure 1.3.5: Convertawings Model A helicopter[5].

[3]http://www.thingsmagazine.net/blogimages/
[4]http://www.aviastar.org/helicopters_eng/bothezat.php
[5]http://www.aviastar.org/helicopters_eng/convertawings.php

Recently there has been an increasing interest in quadrotor designs. Bell is working on a quad tiltrotor to overcome the V-22 Ospray (see Figure 1.3.6), capable of carrying a large payload, achieving high velocity and while using a short amount of space for Vertical Take-Off and Landing (VTOL). Much of its systems come directly from the V-22 except for the number of engines. Also, the wing structure on the new design has some improvements, it has a wider wing span on the rear rotors. As a consequence, the Bell quad tiltrotor (Figure 1.3.7) aims for higher performance and fuel economy [7].

Figure 1.3.6: V-22 Ospray[6].

Figure 1.3.7: Concept of Bell's quad tiltrotor[7].

Another recent and famous quadrotor design is the Moller Skycar (Figure 1.3.8), a prototype for a personal VTOL "flying car". The Skycar has four ducted fans allowing for a safer and efficient operation at low speeds. It was a target for much criticism because the only demonstrations of flight were hover tests with the Skycar tethered to a crane [8]. It's inventor, Paul Moller already tried to sell the Skycar by auction without success. Nowadays he focuses his work on the precursor of the Skycar, the "M200G Volantor", a flying saucer-style hovercraft. This later model uses eight fans controlled by a computer and is capable of hovering up to 3 m above the ground. This limitation is imposed by the on-board computer due to regulations of the Federal Aviation Administrations, stating that any vehicle that flies above 3 m is regulated as an aircraft [9].

Figure 1.3.8: Skycar during a test flight[8].

Quadrotors are also available to the public through radio controlled toys. Some enthusiasts as well as researches have been developing their own quadrotor prototypes. This is possible due to the availability

[6] http://zerosix.wordpress.com/2007/10/06/v22-in-search-and-rescue-mode-arizona-not-iraq/
[7] http://sistemadearmas.sites.uol.com.br/ca/aco01tilt.html
[8] http://www.symscape.com/node/622

Figure 1.4.1: Ambulance stuck in a traffic jam in Lisbon, Portugal.

of cheap electronics and lightweight resistant materials available to the public. Be it for personal satisfaction, entertainment, military or civilian use, quadrotors have played an important role in the evolution of aircrafts and may prove themselves as important means of transportation in a near future.

1.4 Motivation

In situations where people's lives are at danger there is regularly the need for transportation of injured or ill patients to a hospital or any other medical facility capable of providing competent medical care. Often, when an ambulance cannot rapidly reach the emergency site, medical aircraft are usually deployed in alternative, a use which has proven to be quite valuable so far. The first record of this kind of action dates back to 1870, where during the Franco-Prussian War 160 French soldiers were transported by hot-air balloon to France [10]. Since then, the concept of medical use aircraft has evolved quite a lot, either in the military or civilian perspectives. Nowadays, medical assistance helicopters and airplanes are very well equipped (e.g. some types of equipment they have onboard: ventilators, electrocardiogram monitors, defibrillators, etc.), allowing for immediate treatment of critical cases, avoiding in many of them death before reaching the hospital. These aircrafts serve great and noble purposes, especially in remote and isolated inhabited areas, but what about practical function in developing cities? Surely there is a large concentration of medical facilities inside a big city, but there are also other factors to consider like traffic jams (Figure 1.4.1).

This causes a fast ambulance trip to become a total nightmare, taking hours to reach the destination, not speaking of the risk of accident where crew and patient can be injured severely. We can of course employ helicopters, but their dimensions are a serious obstacle due to the high diameter of the propeller which can collide into a tree or electricity pole. What if we could arrange an aircraft with the same characteristics of a helicopter, but with smaller size? A possible answer to this question is the use of a quadrotor vehicle. Here are the advantages:

- No gearing necessary between the motor and the rotor;
- No variable propeller pitch is required for altering the quadrotor angle of attack;
- No rotor shaft tilting required;

- 4 smaller motors instead of one big rotor resulting in less stored kinetic energy and thus less damage in case of accidents;
- Minimal mechanical complexity,
- Quadrotors require less maintenance compared to both helicopters and planes;
- Rotor mechanics simplification;
- Payload augmentation;
- Gyroscopic effects reduction.

Although there are the drawbacks of weight augmentation and high energy consumption, these can be reduced thanks to more efficient energy sources and materials engineering research.

Thus arose the motivation for this thesis: the design of a prototype air vehicle that can be controlled by a pilot, where some control techniques may be tested in order to study the robustness of such systems and thus contribute in some way for the development of safer quadrotors. Such an endeavor will also serve as an opportunity to experience the challenges of aircraft modelling and real life application.

1.5 State of the art

Rotorcrafts have witnessed an incredible evolution in the last years. Universities, students and researchers continuously work to introduce more robust controllers and modelling techniques, so that they can provide detailed and accurate representations of real-life quadrotors. This section introduces some of the work presented in recent years.

Progress in sensor technology, data processing and integrated actuators has made the development of miniature flying robots possible. Bouabdallah *et al.* [11] described a possible approach towards the development of an indoor micro quadrotor called “OS4” (Figure 1.5.1), including the mechanical design, dynamic modelling, sensing and control of the orientation angles. Later, Bouabdallah *et al.* [12] presented the results of classic PID (Proportional-Integral-Derivative) and LQ (Linear-Quadratic) controllers implemented in the OS4, based on a more complete model than the previously attained. The LQ controller revealed to be problematic, it was hard to find weight matrices to satisfy control stability. However, the classical approach (PID) performed favorably compared to the LQ due to the simpler method’s tolerance for model uncertainty.

In reference [13], researchers also developed a computational tool designed to simulate the dynamics of the OS4. Implemented in MatLab® using Simulink®, the software allows to test different sensors and filters, as well as various control techniques. The simulation environment was specifically used, in this case, to test obstacle avoidance approaches using ultrasound sensors, before proceeding with actual experimentation on the OS4.

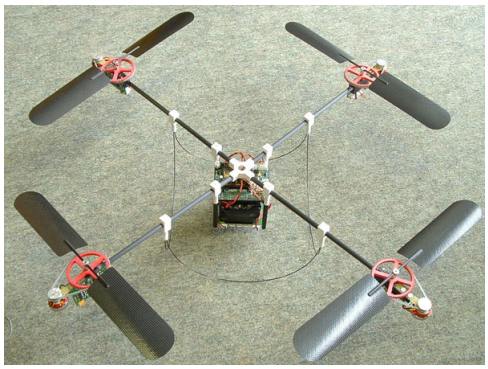


Figure 1.5.1: OS4 Unmanned Air Vehicle[9].

Aiming to design a practical quadrotor helicopter, Pounds *et al.* [1], from the Australian National University, created the "X-4 FlyerMark II" platform (Figure 1.5.2), a 4 kg structurally robust quadrotor with custom built chassis and avionics. An aero-elastic blade was designed and its performance results presented. The quadrotor also includes a sprung teetering rotor hub that allows the adjustment of the blade flapping characteristics. This effect was also introduced in the dynamic model of the rotorcraft. At this point, MatLab® simulations showed that an inverted rotorcraft configuration is beneficial to quadrotor flyers. Two years later, in reference [14] the analysis of the aircraft's attitude dynamics allowed the tuning of the mechanical design to improve disturbance rejection and control sensitivity. A linear SISO controller was then implemented for attitude control, aiming to stabilize dominant decoupled pitch and roll modes, while using a model of disturbance inputs to estimate the performance of the plant. The model of the plant also implemented blade flapping of the propellers, having the experimental results in the X-4 Flyer illustrated that blade flapping introduces significant stability effects on the vehicle. Results showed that the compensator was able to control attitude at low rotor speeds .

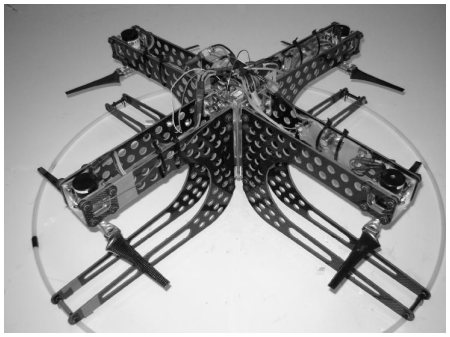


Figure 1.5.2: X-4 Flyer Mark II [1].

Mokhtari *et al.* [15] presented a mixed robust feedback linearization with linear GH_∞ controller applied to a nonlinear quadrotor. Actuator saturation and constrain on state space output was implemented to analyze the worst case scenario of control law design. Also, the controller's behavior was tested when subjugated to parameter uncertainties, external disturbances and measurement noise. Simulations revealed that the overall system becomes robust when weighting functions are chosen judiciously.

Artificial vision has also proven to be a resourceful tool when controlling quadrotors. Tournier *et*

[10]http://aero.epfl.ch/page57689.html

al. [16] presented vision-based estimation and control of a quadrotor using a single camera relative to a target that incorporates moiré patterns. The purpose of this research was to acquire six degree of freedom estimation, which is essential for the operation of vehicles in close proximity to other aircraft and landing platforms. Tests showed the usability of the system to autonomously hover the aircraft, given that the target remains within the camera's field of view.

Taking into account that current designs often consider only nominal operating conditions for vehicle control design, Hoffmann *et al.* [2] seeks to engage in the study of the issues that arise when deviating significantly from the hover flight regime. Three different quadrotor aerodynamic effects were investigated and their relation to vehicular velocity, angle of attack and airframe design. The first effect studied was the way how total thrust varies not only with input power, but with the free stream velocity and angle of attack with respect to the free stream. The second phenomenon was "blade flapping", a result of different inflow velocities experienced by the advancing and retreating blades, which induces roll and pitch moments on the rotor hub as well as a deflection of the thrust vector. Lastly, the effect of interference caused by the vehicle's body components located near the slip stream of the propellers was investigated because it is responsible for unsteady thrust, rendering attitude tracking difficult. The full extent of these aerodynamic effects was uncovered with the help of the STARMAC II quadrotor (see Figure 1.5.3), which possesses autonomous attitude and altitude control. Results proved that existing models and control techniques are inadequate for accurate trajectory tracking at high speed and in uncontrolled environments.

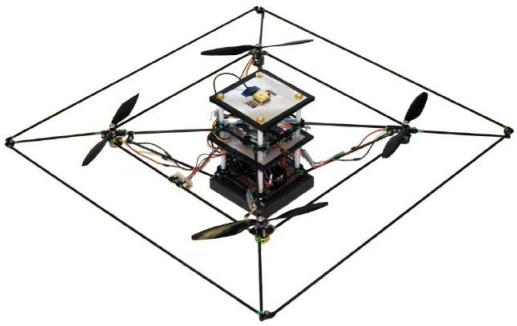


Figure 1.5.3: STARMAC II quadrotor [2].

Mehmet Efe [17] provided the full dynamical model of a commercially available quadrotor rotorcraft and presented its behavior at low altitudes through sliding mode control, which is a control technique known by its robustness against disturbances and invariance during the sliding regime. The plant was modeled as being nonlinear, with state variables tightly coupled. The assumptions made included unsaturated controls, negligible weather disturbances and reasonably fast actuation periphery. The simulations showed that the algorithm successfully drives the system towards the desired trajectory with bounded control signals.

In the University of California, Berkley, Katie Miller [18] sought to design control laws for a quadrotor helicopter to track a desired path by implementation of waypoints, more specifically a she used a PD (Proportional-Derivative) for positioning control and a PID for rotation control. The control laws were developed on a model of the quadrotor dynamics, which was obtained by linearizing its nonlinear equations of motion. The control laws were studied while in the presence of modeling uncertainties and external disturbances like wind gusts. The results showed that while the linear control laws are adequate under perfect conditions, when uncertainty is introduced, they do not perform well.

Student competitions are another way of promoting quadrotor development. AbouSleiman *et al.* [3] designed a quadrotor robot for the Sixth Annual Unmanned Aerial Systems Competition[11] from scratch (see Figure 1.5.4). The quadrotor uses GPS for waypoint tracking and altitude, on-board camera and a dual processor capable of autonomous path navigation and data exchange with the ground station. Control was made with 4 PID controllers (respectively to control pitch, roll, yaw and throttle) and their gains were found experimentally. During the start of the competition, immediately after takeoff the quadrotor was hit by strong wind turbulence but the controller performed extremely well and was able to stabilize the system.

Figure 1.5.4: Oakland University quadrotor system [3].

Sérgio Pereira [19] has recently presented his master thesis where he proposed a model to simulate the ALIV (Autonomous Locomotion Individual Vehicle) quadrotor UAV (Figure 1.5.5). He was able to test the capabilities of the ALIV by using a model built in MatLab® and Simulink®, where he designed a Kalman Filter to estimate the quadrotor state by modelling its real sensors. Then, he implemented a Linear Quadratic Regulator (LQR) controller to stabilize the rotorcraft. Positioning control was aided either through a joystick or a simulated on-board camera. A Simulink® real-time simulation of the system was successfully integrated with the FLIGHTGEAR flight simulator (Figure 1.5.6), where a user can pilot the quadrotor to a certain target using the joystick, and, after locking on the target, the rotorcraft performs autonomous flight through established waypoints.

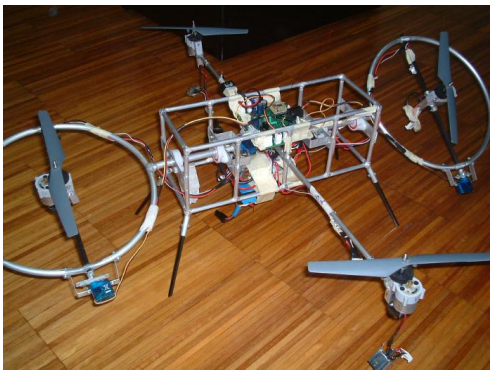


Figure 1.5.5: ALIV quadrotor [19].

Figure 1.5.6: ALIV control simulation in FLIGHTGEAR [19].

[11] http://65.210.16.57/studentcomp2010/default.html

1.6 Thesis contributions

In reference [19], the author shows how to model and control a quadrotor in a simulation environment. Control is made through two possible ways: with user input (using a joystick) and autonomously, showing how to take full advantage of the quadrotor as an UAV. The sensors used include a tri-axis accelerometer, a tri-axis magnetometer and a camera used to perform position control. This thesis uses reference [19] as a basis for the construction of a quadrotor prototype, focusing more on the issues of practical implementation. During the development of this thesis, a set of contributions were made that are worth mentioning:

- Identification of the dynamics of a motor-propeller assembly using sound acquired from the spinning propeller;
- Modeling of a brushless motor as a linear system, controlled by a PWM signal;
- Design of a Kalman filter that allows for sensor fusion of data coming from a tri-axis accelerometer and compass (instead of a tri-axis magnetometer), therefore providing an estimate for the attitude of a quadrotor.
- The implementation of a state space Kalman filter and LQR controller in the Arduino electronics prototyping platform.
- Practical demonstration of the impact of sensor noise in the control process of a quadrotor.

1.7 Structure of this thesis

In Chapter 1, Introduction, we begin by defining what a quadrotor is, and what its main applications are. After defining the maneuvering capabilities of this kind of air vehicle, its historical evolution is presented, from the beginning of the twentieth century to the present day. Next, the reasons that led to the choice of this thesis are explained, and a brief description of the main areas of research with these vehicles is made.

Moving on to Chapter 2, throughout which the design process of quadrotor is described, it starts with the definition of the objectives to be fulfilled by the aircraft. Follows the analysis and justification of the chosen components for the aircraft's assembly.

Chapter 3 focuses on quadrotor modeling. The used reference frames are defined, followed by the dynamic and kinematic analysis of the aircraft. The identification of the motor-propeller assembly is made by using recorded sound files of the propellers sound. The calculation of moments of inertia of the aircraft is also presented, as well as of the center of gravity location. This chapter ends with sensors modeling and with a brief introduction to the simulation of the quadrotor and its sensors in MATLAB® and Simulink®.

Assuming the quadrotor as a linear state space system, in Chapter 4 a Kalman filter is described according to its mathematical formulation and design for this particular case study. It is also explained how it is implementation in Simulink® as well as in the Arduino.

Chapter 5 focuses on the mathematical definition and design of a controller Linear-Quadratic Regulator to allow for control over the quadrotor states. Just as for the Kalman filter, the controller implementation in MATLAB® and embedded control platform Arduino is also revealed.

Chapter 6 begins the simulations stage, beginning with an ideal case where we have access to all of the quadrotor's states, to see if it is possible to have full control over them. Next, a set of simulations is executed where not all of the system's states are available to the controller. In this set, the sensors dynamics, Kalman filter, and finally the motors dynamics are added to the simulations, gradually increasing its degree of realism. After the simulations were completed, an attempted is made to apply the Kalman filter and LQR controller to the quadrotor prototype.

This thesis ends in Chapter 7 with the final conclusions and suggestions for future improvements.

Chapter 2

Quadrotor design

Correct execution of any aircraft design stage provides the early determination of what the drawbacks in design are, allowing us to save not only money, but also time. That way, fewer changes will need to be implemented after building the quadrotor. This chapter will provide an example on how to build a quadrotor prototype, from the mechanical components to the avionics, culminating in the construction of a functional aircraft.

2.1 General concept

The most important target of this particular design process is to arrive at the correct set of requirements for the aircraft, which are often summarized in a set of specifications. In this section we will define our mission to build a quadrotor prototype suitable for indoor flight, as well justify the decisions and equipment chosen for achieving this purpose. The specifications for our quadrotor prototype are:

- Overall mass not superior to 1 kg. When it comes to quadrotors, the heavier they get, the more expensive they are. Many quadrotors used for research do not exceed this mass. So, aiming for a maximum mass of 1 kg seems to be a suitable target;
- Flight autonomy between 10 and 20 minutes. There is no point in using the quadrotor for 2 minutes and then wait a couple of hours to recharge the batteries, wasting precious time;
- On-board controller should have its separate power supply to prevent simultaneous engine and processor failure in case of battery loss;
- Ability to transmit live telemetry data and receive movement orders from a ground station wirelessly, therefore avoiding the use of cables which could become entangled in the aircraft and cause an accident;
- The quadrotor should not fly very far from ground station so there is no need of long range telemetry hardware and the extra power requirements associated with long range transmissions.

Consequently, the main components should be:

- 4 electric motors and 4 respective Electronic Speed Controllers (ESC);

- 4 propellers;
- 1 on-board processing unit;
- 1 tri-axis analog accelerometer. Measuring acceleration in three-dimensional space aboard an aircraft is a necessary feat to understand the magnitude of the forces acting on this aircraft. It may also be possible to use this sensor to calculate the angles of yaw and pitch with knowledge of the gravity acceleration vector;
- 1 electronic compass to measure the yaw angle;
- 2 on-board power supplies (batteries), one for the motors and another to the processing unit. At this point we will assume that beyond the need to provide electrical power to the motors, we must assure that the brain of the quadrotor (i.e. the on-board processing unit) remains working well after the battery of the motors has discharged;
- 1 airframe for supporting all the aircraft's components.

2.2 Airframe

The airframe is the mechanical structure of an aircraft that supports all the components, much like a "skeleton" in Human Beings. Designing an airframe from scratch involves important concepts of physics, aerodynamics, materials engineering and manufacturing techniques to achieve certain performance, reliability and cost criteria. The main purpose of this thesis is not airframe design, so, because construction time of the quadrotor is critical, it is preferable to acquire, if possible, parts already available for sale.

The chosen airframe for the quadrotor was the "Vario-43" model (Figure 2.2.1[1]), with 258g of mass and made of GPR (Glass-Reinforced-Plastic), posessing a cage-like structure in its center that will offer extra protection to the electronics. This particular detail may prove itself very useful when it comes to the test flight stage, when accidents are more likely to happen.

Figure 2.2.1: Vario-43 quadrotor airframe.

2.3 Propellers

The typical behavior of a propeller can be defined by three parameters [4]:

- Thrust coefficient c_T;

[1] http://www.mikrokopter.de/ucwiki/Vario-Frame_43, January 2009.

- Power coefficient c_P;
- Propeller radius r.

These parameters allow the calculation of a propeller's thrust T:

$$T = c_T \frac{4\rho r^4}{\pi^2}\omega^2 \tag{2.1}$$

and power P_P:

$$P_P = c_P \frac{4\rho r^5}{\pi^3}\omega^3 \tag{2.2}$$

where ω is the propeller angular speed and ρ the density of air.

These formulas show that both thrust and power increase greatly with propeller's diameter. If the diameter is big enough, then it should be possible to get sufficient thrust while demanding low rotational speed of the propeller. Consequently, the motor driving the propeller will have lower power consumption, giving the quadrotor higher flight autonomy.

Available models of counter rotating propellers are scarce in the market of radio controlled aricrafts. The "EPP1045" (see Figure 2.3.1), a propeller with a diameter of 10" (25.4 cm) and weighting 23g, presented itself as a possible candidate for implementation in the quadrotor. To check its compatibility with the project requisites it is necessary to calculate the respective thrust and power coefficient. In reference [20] we have data from tests done on the EPP1045, from which we can extract the mean thrust and power coefficients by using equations (2.1) and (2.2):

$$c_T = 0.1154 \tag{2.3}$$

$$c_P = 0.0743 \tag{2.4}$$

In reality, neither the thrust nor the power coefficients are constant values, they are both functions of the advance ratio J [4]:

$$J = \frac{u_0}{nD_P} \tag{2.5}$$

where u_0 is the aircraft flight velocity, n the propeller's speed in revolutions per second and finally D_P is the propeller diameter. However, when observing the characteristic curves for both these coefficients (see Figures 2.3.2 and 2.3.3), it is clear that when an aircraft's flight velocity is very low (e.g. in a constant altitude hover) the advance ratio is almost zero and the two coefficients can be approximated as constants, which is the current case, for at this point there is no interest in achieving high translation velocity.

Assuming the quadrotor's maximum weight is $9.81N$ $(1kg)$ and that we have four propellers, it is mandatory that each propeller is able to provide at least $2.45N$ ($1/4$ of the quadrotor weight) in order to achieve lift-off. Taking this data into consideration leads us to wonder about the minimum propeller rotational speed involved, as well as the magnitude of the power required for flight. Figure 2.3.4 helps us with some of these questions. We can see that a propeller will have to achieve approximately $412rad.s^{-1}$, which is equivalent to 3934 revolutions per minute, to provide the minimum $2.45N$ required for lift-off. The respective propeller power is $26W$. After this short analysis we can state that the EPP1045 propellers are

suitable for implementation in the quadrotor prototype, a statement that can be proved by experimental data[2], showing that with the right motor we can produce the necessary thrust for Lift-Off.

Figure 2.3.1: EPP1045 propellers.

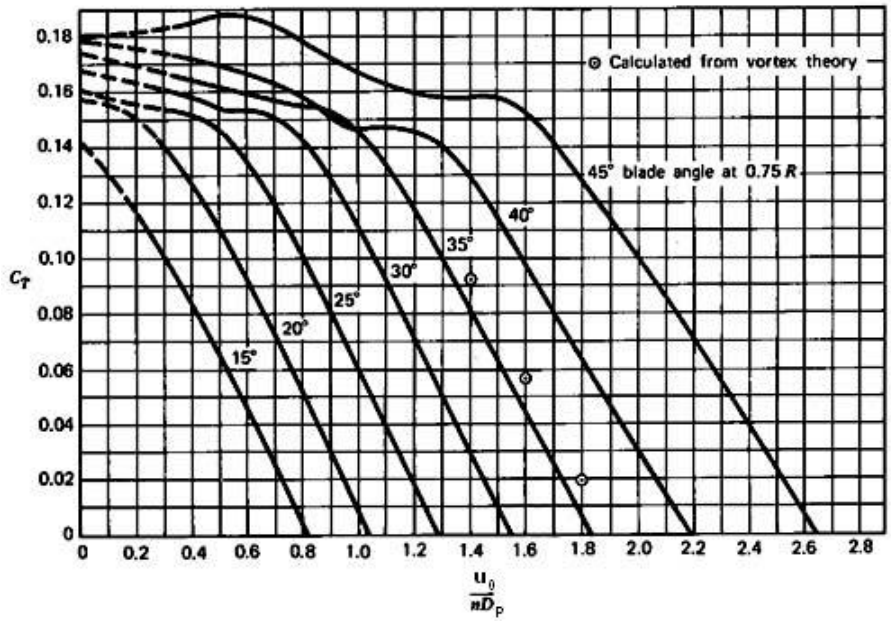


Figure 2.3.2: Typical propeller thrust curves as a function of advance ratio J and blade angle [4].

[2]www.radrotary.com/motor_test.xls, November 2008.

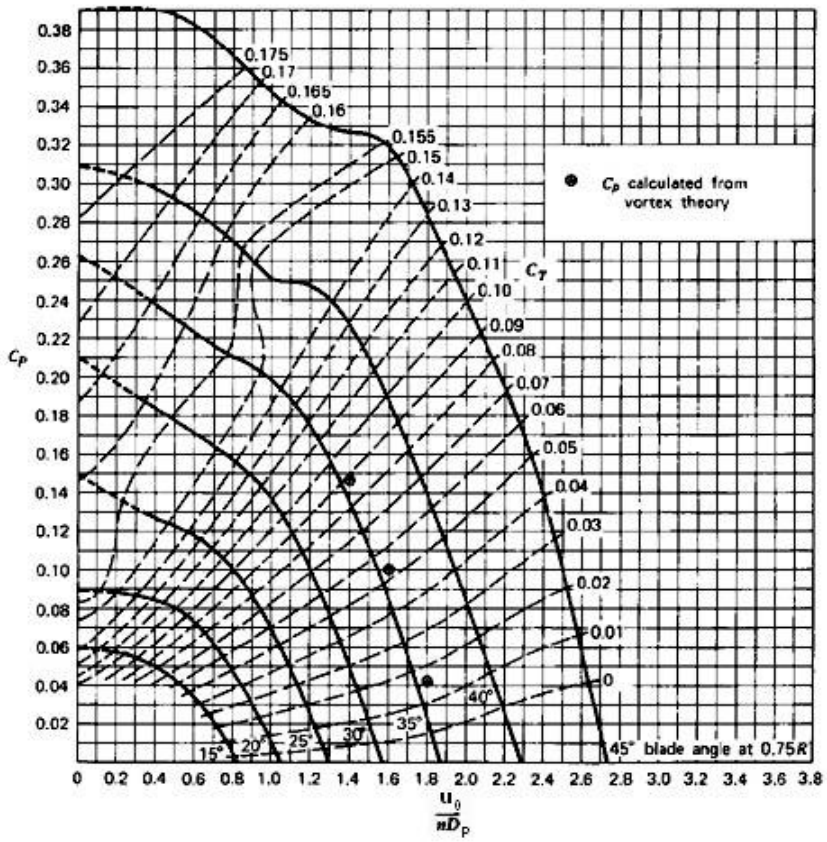


Figure 2.3.3: Typical propeller power curves as a function of advance ratio J and blade angle [4].

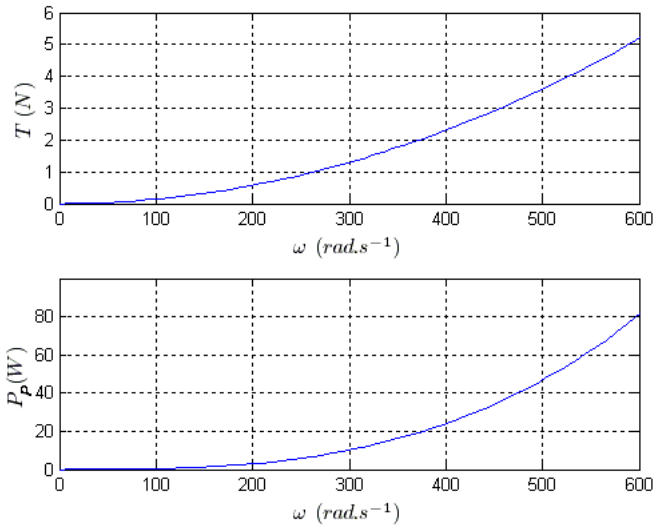


Figure 2.3.4: Theoretical thrust and power of an EPP1045 propeller.

2.4 Electronics

2.4.1 Motors and Electronic Speed Controllers

The motors usually implemented in this kind of application are electric Direct Current (DC) motors. They are lighter than combustion engines and do not need a combustible fuel, which, among other benefits, decreases the risk of explosion.

DC motors available in the radio control hobby market are either brushed or brushless. Brushless motors are expensive but have higher efficiency, power, and do not need regular maintenance. Brushed motors are cheap but have a shorter lifetime and their brushes need regular replacements. For these reasons it is preferable to use brushless motors, because loss of structural integrity of the quadrotor due to motor failure should be avoided by using more reliable equipment.

There are cases when a motor does not have the necessary torque to spin the propeller at the required speed, or even when there is the need to reduce the propeller speed to an optimum velocity inferior to that of the main drive shaft. These are situations where a PSRU (Propeller Speed Reduction Unit - a gearbox speed reduction system) is required. Although these units are available to use in RC (Radio Control) aircrafts, in the quadrotor we want to have a structure as light as possible. One way to have the benefits of high torque without using gearboxes is by using a design of brushless DC motor called “Outrunner”. The selected motor was the “BL-Outrunner 2824-34” model from the manufacturer Robbe ROXXY (Figure 2.4.2). This motor is able to rotate at 12100 RPM when free of load, weights $48g$ and has a maximum efficiency of 79%[3].

The speed of a brushless motor is controlled by an Electronic Speed Controllers (or ESC). This hardware receives the power from the battery and drives it to the motor according to a PWM (Pulse-Width Modulation) signal that is provided by the controller unit. The “thunderbird-9” ESC from Castle Creations is well suited for the job at hand (Figure 2.4.1). It has a mass of $9g$ and is capable of providing up to $9A$ of current[4] (which is also the maximum allowable current of the BL-Outrunner 2824-34 motor).

Figure 2.4.1: Thunderbird-9 ESC.

Figure 2.4.2: Electric DC Brushless motor Robbe ROXXY BL-Outrunner 2824-34.

2.4.2 Microcontroller Unit

A stabilization system is necessary to drive the quadrotor because it is a naturally unstable vehicle. The implementation of stabilization control algorithms has not been possible until very recently with the

[3] http://data.robbe-online.net/robbe_pdf/P1101/P1101_1-4777.pdf, February 2009.
[4] http://www.castlecreations.com/support/documents/Thunderbird-9-user-guide.pdf, February 2009.

development of small size microcontrollers.

One microcontroller that has gotten special attention from the robotics community world-wide is the Arduino. This microcontroller platform is quite inexpensive and has a C-based language development environment that is very intuitive to use. From the different versions of the Arduino, the selected one for this project was the Arduino Duemilanove (Figure 2.4.3). With a mass of $35g$, the Duemilanove has 6 analog inputs with 10 bit resolution, serial port communication, 14 I/O pins (of which 6 provide PWM output) and many other features [5].

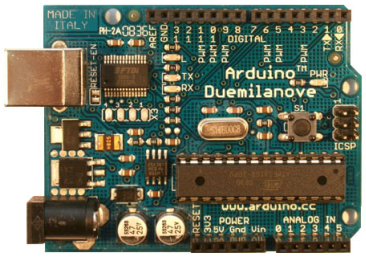


Figure 2.4.3: Arduino Duemilanove.

2.4.3 Wireless communication

Wireless communications are always a challenge. One has to weight important factors like power consumption, weight, transmission speed and reliability. Fortunately it is possible to use hardware with the Arduino that allows to satisfy all the previous conditions, e.g. the XBee 802.15.4 1mW (Figure 2.4.4). This module can be wrapped into a serial command set (useful because the Arduino can use serial communication), consumes only $50mA$ of current, has a maximum data rate of $250kbps$ (using radio frequency) and a range of $100m$ (larger range telemetry is not a requirement of this thesis)[6]. Two XBee modules are going to be used: one for the quadrotor and another in the ground station computer (that will handle all telemetry for system identification and control purposes).

Figure 2.4.4: XBee 802.15.4 1mW wireless module.

2.4.4 Sensors

The sensors of a rotorcraft are a key element of the control loop. They are responsible for providing information like aircraft attitude, acceleration, altitude, global position, and other relevant data. Modern aircraft often carry robust and expensive sensor technology, which is often a synonymous of big and

[5] http://arduino.cc/en/Main/ArduinoBoardDuemilanove, February 2009.
[6] http://www.inmotion.pt/store/product_info.php?cPath=7&products_id=63, February 2009.

heavy electro-mechanical systems. In our quadrotor prototype, the keywords are “light and small”, thus when it comes to small sensing we should consider MEMS (Micro-Electro-Mechanical Systems) technology. The use of low-cost sensors of this type implies less efficient data processing and thus a bad orientation data prediction in addition to a weak drift rejection, making the control process a lot more challenging. Also, and in spite of the latest advances in miniature actuators, the scaling laws[7] are still unfavorable and one has to face sometimes the problem of actuator saturation.

Our prototype has two sensors, a triple axis accelerometer providing three accelerations (one for each axis of a Cartesian coordinate system) and a dual axis magnetometer providing the magnetic north direction. The accelerometer of choice is the “ADXL330” (Figure 2.4.4) and some of its main characteristics are: maximum current consumption of $320\mu A$, an acceleration range of $\pm 29.4 m.s^{-2}$ and a mass of approximately $2g$. The magnetometer is an “HMC6352” (Figure 2.4.4), with a mass of $0.14g$, a $1mA$ current consumption, update rate up to $20Hz$ and a selectable heading resolution of either $0.5°$ or $1°$.

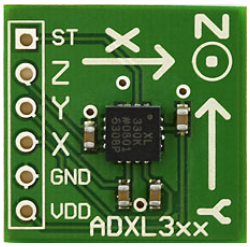


Figure 2.4.5: ADXL330 accelerometer.

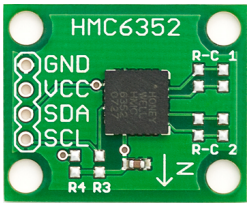


Figure 2.4.6: HMC6352 compass.

2.4.5 Batteries

Power storage has experienced great advances in the last decades, mainly due to the search of lighter and long-lasting power sources for the mobile devices industry and recently, for the emerging electric car market. The aircraft industry however as not yet had a noticeable interest in electric power sources. Interested parties in this field of development are mostly related to a few research projects (some of which make use of solar energy as a power source) or even some hobby developers that are already on their way to selling their electric airplanes (e.g. the ElectraFlyer-C[8] and the Sunseeker II[9]). In the RC market, electric batteries have proven to be a long term cheaper solution to combustion engines. Specifically, Lithium-ion Polymer (or LiPo) batteries, a recent advance in power storage technology, provide high capacity, light and robust power source that has a large market spectrum of applications, including RC aircraft. For these reasons, the quadrotor prototype will use two separate LiPo batteries, one for the motors and another for Arduino (a separate power source for the controller unit assures that the controller has power, even if the main motor battery reaches critical levels which is useful for emergency situations).

The Arduino is therefore powered with a $15g$ LiPo battery, capable of providing $240mAh$ of current at $7.4V$. As for the motors, they are powered by a $179g$ battery, electric charge of $2200mAh$ at $11.1V$ (Figure

[7] Scaling laws are an engineering concept that refers to variables which change drastically depending on the scale being considered. For example, if we try to use a mechanical gyroscope aboard an aircraft subject to a strong electromagnetic field, we may experience different readings than when using a small piezo electric gyroscope (whose readings are highly susceptible to electromagnetic fields).

[8] http://www.electraflyer.com/electraflyerc.php

[9] http://solar-flight.com/

2.4.7).

Figure 2.4.7: Vislero LiPo 11,1V 2200mAh 20C battery.

2.4.6 Electronic circuit schematics

After describing the parts used in the construction of the quadrotor, it is important to mention how the electronic components are assembled (the electronic circuit schematics can be found in Appendix A). Each one of the motors has three wires which are directly connected to the respective electronic speed controller (or ESC). The speed controllers were attached to the main battery through a MPX connector. These speed controllers also require a connection to a $5V$ DC power supply, ground wiring, and the use of the PWM pins on the processing board (all of these pins are available on the Arduino).

The wiring of the accelerometer uses the $3.3V$ pin as the sensor power supply, a ground pin, and three analog inputs for the accelerations. Turning our attention to the compass, its power supply uses the $5V$ pin on the Arduino, a ground pin and the two analog inputs for establishing data transactions by I2C-Bus protocol.

Supplying the Arduino with DC means we need to use at least a $6V$ battery. To monitor this power source, a circuit was devised to light up a LED in case the voltage drops below $6.2V$. The basic philosophy in this battery check system is the fact that electricity follows the easiest way possible to the ground. So, while voltage is above $6.2V$, electric current flows through a Zener diode and activates the base collector of a NPN transistor, and allows current to flow from the collector to the emitter of this component, directly to the ground. If the voltage drops below $6.2V$ then no more current passes through the Zener diode, meaning it has to go through the LED, lighting it up in the process. While testing the quadrotor, it was discovered that the Arduino remained powered, even when its battery was not connected. This happened because the chosen electronic speed controllers have the particular ability to use power from the motors battery to feed $5V$ to the hardware connected to it, usually a RC receiver but in our case, an Arduino board. This made the battery for the Arduino dispensable, but at the same time useful if we want to keep the entire system online for a few more minutes.

Regarding the autonomy of the main battery, a voltage divider was implemented so that an analog port on the Arduino could be used to measure its voltage. This approach was discarded due to the fact that the speed controllers already possess a built in security system that slowly cuts power from the motors as soon as they detect it's time to recharge the main battery (note that there is no risk of the Arduino stop working before the motors because the ESCs prevents the motors of working when the battery reaches 9V, which is more than enough for our processing unit).

2.5 Flight autonomy

Flight autonomy is an important specification when designing a quadrotor. The major factor directly influencing the flight autonomy is the high power consumption of the motors, which increases with the value of the propeller angular velocity. An electric DC motor does not have a linear behavior (Figure 2.5.1). Usually, it is characterized by angular speed saturation, as well as a dead zone preceding the minimum voltage required for propeller rotation [19]. The saturation, prevents to surpass the maximum speed allowable by the motor.

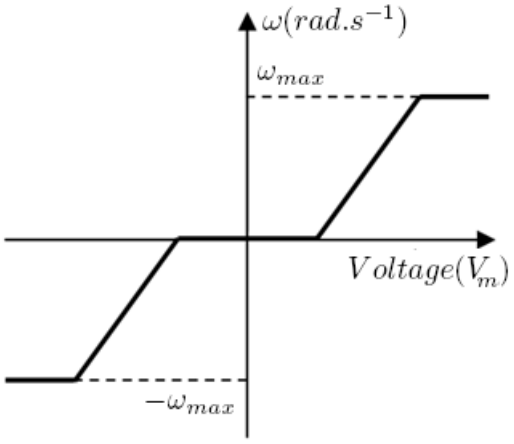


Figure 2.5.1: Example illustration of the angular velocity of a DC motor as a function of power supply voltage.

The property that characterizes how the angular velocity of a certain DC motor evolves with power supply voltage is the k_v (pronounced exactly as "kv"). This property simply allows to know how many RPM will the motor provide per each Volt of supplied electricity. Thus, to calculate how a motor's angular velocity ω changes with voltage we can use:

$$\omega_v = \frac{2\pi k_v}{60} \tag{2.6}$$

$$\omega = \omega_v(V_m - V_0) \tag{2.7}$$

where ω_v is the propeller's angular speed per Volt, V_m is the motor's power supply voltage and V_0 is the power supply voltage corresponding to the motor's dead zone.

At this point, it is only a matter of combining equations (2.1) (assuming that thrust is equal to the total weight of the vehicle) and (2.7) with Ohm's law to get the total electric current consumption $I^{'}$ required to maintain a quadrotor of a given mass m in the air:

$$I^{'} = \frac{V_0}{R_m} + \frac{1}{R_m k_v}\left(\frac{\pi^2 mg}{4c_T\rho r^4}\right)^{\frac{1}{2}} \tag{2.8}$$

For practical reasons we can rewrite 2.8 into:

$$I^{'} = I_0 + \frac{\pi}{2R_m k_v r^2}\left(\frac{mg}{c_T\rho}\right)^{\frac{1}{2}} \tag{2.9}$$

where g is the acceleration of gravity, $R_m$the motor's copper coil electric resistance and I_0 the electric current consumption of the motor in no-load condition. Now that we can calculate the motor's electrical consumption needs, it is time to conjugate this information with certain battery properties that will allow the calculation of the flight autonomy f_t, given the electric charge $Q^{'}$ of the battery:

$$f_t = \frac{Q^{'}}{I^{'}} \tag{2.10}$$

The BL-Outrunner 2824-34 motor has a coil resistance of 0.3136Ω and is capable of rotating at 1100 RPM per Volt. At this point it is not possible to predict the exact mass of the aircraft neither to have knowledge of the no-load current, so let us assume a mass of $1kg$ and a minimum electrical current of $3A$ (a conservative value for this type of motor). Equation (2.9) therefore shows that each motor consumes $3.23A$. If we insert this data into equation (2.7) (not forgetting that we have four motors) we can predict a flight autonomy of approximately $614s$ (10.2 minutes), a reasonable time given that we are uncertain about some properties of the motors and aircraft at this stage.

Chapter 3

Quadrotor modelling

The first step before the control stage is the adequate modeling of the system dynamics. This phase will hopefully facilitate the control of the aircraft as it will provide us with a better understanding of the overall system capabilities and limitations. The current chapter will guide us through the equations and techniques used to model our quadrotor and its sensors, providing the mathematical basis for the application of the system dynamics in a simulation environment.

3.1 System dynamics

Writing the equations that portray the complex dynamics of an aircraft implies first defining the system of coordinates to use. Only two reference frames are required, an earth fixed frame and a mobile frame whose dynamic behavior can be described relative to the fixed frame. The earth fixed axis system will be regarded as an inertial reference frame: one in which the first law of Newton[1] is valid. Experience indicates this to be acceptable even for supersonic airplanes but not for hypersonic vehicles[2] [21]. We shall designate this reference frame by O_{NED} (North-East-Down) because two of its axis (u_x and u_y) are aligned respectively with the North and East direction, and the third axis (u_z) is directed down, aligned towards the center of the Earth (Figure 3.1.1). The mobile frame is designated by O_{ABC}, or Aircraft-Body-Centered, and has its origin coincident with the quadrotor's center of gravity (Figure 3.1.2).

[1] An object that is not moving will not move until a net force acts upon it. An object that is moving will not change its velocity (accelerate) until a net force acts upon it.

[2] The rotational velocity of Earth must not be neglected for hypersonic flight.

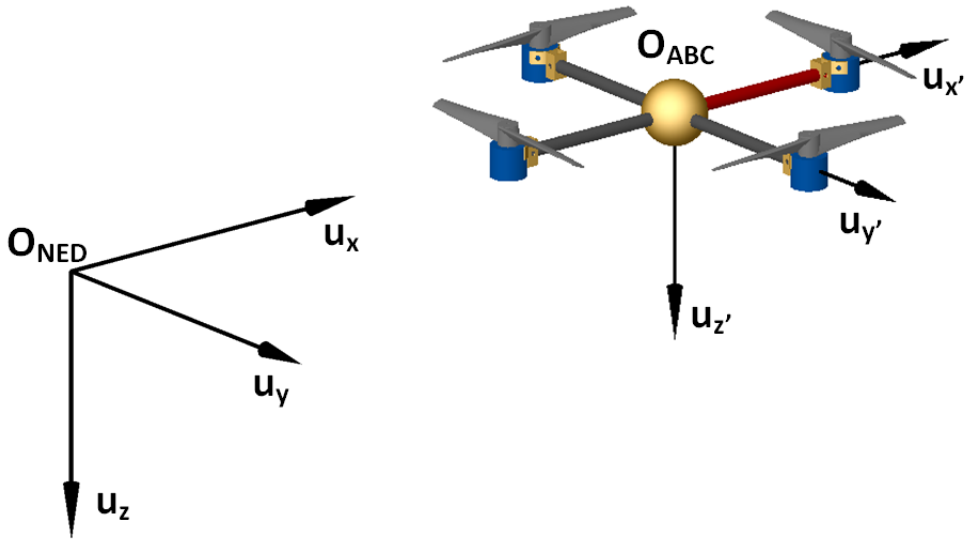


Figure 3.1.2: NED and ABC reference frames.

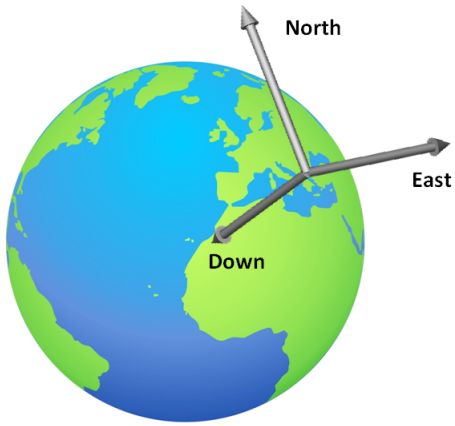


Figure 3.1.1: Visualization of the North-East-Down reference frame.

In control theory, knowledge about the dynamic behavior of a given system can be acquired through its states. For a quadrotor, its attitude about all 3 axis of rotation is known with 6 states: the Euler angles $[\phi\ \theta\ \psi]$ (Roll – Pitch – Yaw as seen before in Figure (1.2.1)) and the angular velocities around each axis of the O_{ABC} frame $[P\ Q\ R]$.

Yet another 6 states are necessary: the position of the center of gravity (or COG) $[X\ Y\ Z]$ and respective linear velocity components $[U\ V\ W]$ relative to the fixed frame. In sum, the quadrotor has 12 states that describe 6 degrees of freedom.

Unsurprisingly, we must deduce the equations describing the orientation of the mobile frame relative to the fixed one, which can be achieved by using a rotation matrix. This matrix results of the product between three other matrices ($\mathbf{R}'(\phi)$, $\mathbf{R}'(\theta)$ and $\mathbf{R}'(\psi)$), each of them representing the rotation of the ABC frame around each one of the O_{NED} axis [22]:

$$\mathbf{R}^{'}(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \mathbf{R}^{'}(\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \mathbf{R}^{'}(\psi) = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3.1)$$

$$\mathbf{S} = \mathbf{R}^{'}(\phi)\,\mathbf{R}^{'}(\theta)\,\mathbf{R}^{'}(\psi) \quad (3.2)$$

$$\mathbf{S} = \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \sin\psi\sin\theta\cos\psi - \cos\phi\sin\psi & \cos\phi\cos\psi + \sin\phi\sin\theta\sin\psi & \sin\phi\cos\theta \\ \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi & \sin\theta\cos\phi\sin\psi - \sin\phi\cos\psi & \cos\theta\cos\phi \end{bmatrix} \quad (3.3)$$

where $\mathbf{S}$ is the rotation matrix that expresses the orientation of the coordinate frame O_{ABC} relative to the reference frame O_{NED}. To mathematically write the movement of an aircraft we must employ Newton's second law of motion. As such, the equations of the net force and moment acting on the quadrotor's body (respectively $\mathbf{F}_{net}$ and $\mathbf{M}_{net}$) are provided:

$$\mathbf{F}_{net} = \frac{d}{dt}[m\mathbf{v}]_B + \boldsymbol{\omega}^{'} \times [m\mathbf{v}]_B \quad (3.4)$$

$$\mathbf{M}_{net} = \frac{d}{dt}[\mathbf{I}\boldsymbol{\omega}^{'}]_B + \boldsymbol{\omega}^{'} \times \left[\mathbf{I}\boldsymbol{\omega}^{'}\right]_B \quad (3.5)$$

where $\mathbf{I}$ is the inertia matrix of the quadrotor, $\mathbf{v}$ is the vector of linear velocities and $\boldsymbol{\omega}^{'}$ $\text{the vector of angular velocities}$. If the equation of Newton's second law is to be as complete as possible, we should add extra terms such as the Coriolis, Euler and aerodynamic forces (e.g. wind), but to keep the model simple, and also because the quadrotor is not supposed, at this stage, to go very far away from the ground station, these forces will not be incorporated in the modeling process. The force of gravity ($\mathbf{F}_g$) is too significative to be neglected, thus it is defined by:

$$\mathbf{F}_g = m\mathbf{S}\begin{bmatrix} 0 & 0 & g \end{bmatrix}^T = mg\begin{bmatrix} -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix}_B^T \quad (3.6)$$

The force of gravity together with the total thrust generated by the propellers ($\mathbf{F}_P$) have therefore to be equal to the sum of forces acting on the quadrotor:

$$\mathbf{F}_g + \mathbf{F}_P = \mathbf{F}_{net} \quad (3.7)$$

Combining equations (3.4), (3.6) and (3.7) we can write the vector of linear accelerations acting on the vehicle's body:

$$\begin{bmatrix} \dot{U} \\ \dot{V} \\ \dot{W} \end{bmatrix} = \frac{1}{m}\begin{bmatrix} F_{Px} \\ F_{Py} \\ F_{Pz} \end{bmatrix} + g\begin{bmatrix} -\sin\theta \\ \cos\theta\sin\phi \\ \cos\theta\cos\phi \end{bmatrix} - \begin{bmatrix} QW - RV \\ RU - PW \\ PV - QU \end{bmatrix} \quad (3.8)$$

where $[F_{Px}\ F_{Py}\ F_{Pz}]$ are the vector elements of $\mathbf{F}_P$. Assuming the aircraft is in a hovered flight, in such a scenario there are forces acting only in the z axis of quadrotor, corresponding to the situation where

we have the engines trying to overcome the force of gravity to keep the aircraft stable at a given altitude:

$$F_{Pz} = -(T_1 + T_2 + T_3 + T_4) \tag{3.9}$$

Note that the minus sign means the lifting force is acting upwards, away from the surface (note that the positive axis of the O_{NED} is point downwards).

Working now on Newton's second law for rotation, the inertia matrix is given by:

$$\mathbf{I} = \begin{bmatrix} I_{11} & -I_{12} & -I_{13} \\ -I_{21} & I_{22} & -I_{23} \\ -I_{31} & -I_{32} & I_{33} \end{bmatrix} \tag{3.10}$$

Assuming the quadrotor is a rigid body with constant mass and axis aligned with the principal axis of inertia, then the tensor $\mathbf{I}$ becomes a diagonal matrix containing only the principal moments of inertia:

$$\mathbf{I} = \begin{bmatrix} I_{11} & 0 & 0 \\ 0 & I_{22} & 0 \\ 0 & 0 & I_{33} \end{bmatrix} \tag{3.11}$$

Joining equations 3.11 and 3.5 we get:

$$\mathbf{M}_{net} = \begin{bmatrix} I_{11} & 0 & 0 \\ 0 & I_{22} & 0 \\ 0 & 0 & I_{33} \end{bmatrix} \begin{bmatrix} \dot{P} \\ \dot{Q} \\ \dot{R} \end{bmatrix} + \begin{bmatrix} P \\ Q \\ R \end{bmatrix} \times \begin{bmatrix} I_{11} & 0 & 0 \\ 0 & I_{22} & 0 \\ 0 & 0 & I_{33} \end{bmatrix} \begin{bmatrix} P \\ Q \\ R \end{bmatrix} \tag{3.12}$$

Consequently, we will have:

$$\mathbf{M}_{net} = \begin{bmatrix} I_{11}\dot{P} \\ I_{22}\dot{Q} \\ I_{33}\dot{R} \end{bmatrix} + \begin{bmatrix} (I_{33} - I_{22})\, QR \\ (I_{11} - I_{33})\, RP \\ (I_{22} - I_{11})\, PQ \end{bmatrix} \tag{3.13}$$

$$\begin{bmatrix} \dot{P} \\ \dot{Q} \\ \dot{R} \end{bmatrix} = \begin{bmatrix} \frac{M_x}{I_{11}} \\ \frac{M_y}{I_{22}} \\ \frac{M_z}{I_{33}} \end{bmatrix} - \begin{bmatrix} \frac{(I_{33}-I_{22})QR}{I_{11}} \\ \frac{(I_{11}-I_{33})RP}{I_{22}} \\ \frac{(I_{22}-I_{11})PQ}{I_{33}} \end{bmatrix} \tag{3.14}$$

Information on the moments acting on the aircraft can be provided by:

$$M_x = (T_4 - T_2)\, d_{cg} \tag{3.15}$$

$$M_y = (T_1 - T_3)\, d_{cg} \tag{3.16}$$

$$M_z = (T_1 + T_3 - T_2 - T_4)\, K_{TM} \tag{3.17}$$

where d_{cg} is the distance to the aircrafts COG (see Table 3.4) and K_{TM} is a constant that relates moment and thrust of a propeller (see Section 3.4).

3.2 Kinematic equations and Euler angles

In this section we will study the kinematics of the quadrotor. The first stage of kinematics analysis consists of deriving position to obtain velocity. Let us then consider the position vector $\overrightarrow{\mathbf{r}}$, which indicates the position of the origin of the O_{ABC} frame relative to O_{NED}:

$$\overrightarrow{\mathbf{r}} = X\overrightarrow{\mathbf{i}} + Y\overrightarrow{\mathbf{j}} + Z\overrightarrow{\mathbf{k}} \tag{3.18}$$

If we derive each of the components in $\overrightarrow{\mathbf{r}}$ we can obtain the instantaneous velocity of O_{ABC} relative to O_{NED}:

$$\dot{\overrightarrow{\mathbf{r}}} = \dot{X}\overrightarrow{\mathbf{i}} + \dot{Y}\overrightarrow{\mathbf{j}} + \dot{Z}\overrightarrow{\mathbf{k}} \tag{3.19}$$

To find out the components of the aircraft's linear velocity $\mathbf{v}'$ in the fixed frame we can use:

$$\begin{bmatrix} U \\ V \\ W \end{bmatrix} = \mathbf{S} \begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{bmatrix} \tag{3.20}$$

where we can take full advantage of the orthogonality of $\mathbf{S}$, meaning its inverse is equal to its transpose:

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{bmatrix} = \mathbf{S}^T \begin{bmatrix} U \\ V \\ W \end{bmatrix} \tag{3.21}$$

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{bmatrix} = \begin{bmatrix} \cos\theta\cos\psi U + (\sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi)V + (\cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi)W \\ \cos\theta\sin\psi U + (\cos\phi\cos\psi + \sin\phi\sin\theta\sin\psi)V + (\sin\theta\cos\phi\sin\psi - \sin\phi\cos\psi)W \\ -\sin\theta U + \sin\phi\cos\theta V + \cos\theta\cos\phi W \end{bmatrix} \tag{3.22}$$

The flight path of the quadrotor in terms of $[X\ Y\ Z]$ can be found by integration of equation 3.22. To perform this integration, the Euler angles ϕ, θ and ψ must be known. However, the Euler angles themselves are functions of time: the Euler rates $\dot{\phi}$, $\dot{\theta}$, and $\dot{\psi}$ depend on the body axis angular rates P, Q and R. To establish the relationship between $\left[\dot{\phi}\ \dot{\theta}\ \dot{\psi}\right]$ and $[P\ Q\ R]$ the following equality must be satisfied:

$$\overrightarrow{\boldsymbol{\omega}} = P\overrightarrow{i} + Q\overrightarrow{\mathbf{j}} + R\overrightarrow{\mathbf{k}} = \dot{\phi} + \dot{\theta} + \dot{\psi} \tag{3.23}$$

Note that although it may appear that the Euler rates are the same as angular velocities, this is not the case. If a solid object is rotating at a constant rate, then its angular velocity $\overrightarrow{\boldsymbol{\omega}}$ will be constant, however the Euler rates will be varying because they depend on the instantaneous angles between the coordinate frame of the body and the inertial reference system (e.g. between O_{ABC} and O_{NED}). The Euler angle sequence is made up of three successive rotations: Roll, Pitch and Yaw. In other words, the angular rate $\dot{\phi}$ needs one rotation, $\dot{\theta}$ needs two and $\dot{\psi}$ needs three:

$$\overrightarrow{\boldsymbol{\omega}} = R(\phi)\,R(\theta)\,R(\psi)\begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} + R(\phi)\,R(\theta)\begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + R(\phi)\begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} \tag{3.24}$$

Therefore:

$$\begin{bmatrix} P \\ Q \\ R \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \sin\phi\cos\theta \\ 0 & -\sin\phi & \cos\theta\cos\phi \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{3.25}$$

Solving for the Euler angular rates yields the desired differential equations [23]:

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \mathbf{T} \begin{bmatrix} P \\ Q \\ R \end{bmatrix} \tag{3.26}$$

$$\mathbf{T} = \begin{bmatrix} 1 & \tan\theta\sin\phi & \tan\theta\cos\phi \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi/\cos\theta & \cos\phi/\cos\theta \end{bmatrix} \tag{3.27}$$

where $\mathbf{T}$ is the matrix that relates the body-fixed angular velocity vector $\overrightarrow{\omega}$ and the rate of change of the Euler angles.

3.3 Quaternion differential equations

A problem with the implementation of Euler angles ϕ, θ and ψ is that for $\theta = 90^\circ$ the Roll angle ϕ loses its meaning. In other words, If the aircraft pitches up 90 degrees, the aircraft roll axis becomes parallel to the yaw axis, and there is no axis available to accommodate yaw rotation (one degree of freedom is lost). This phenomenon is designated by gimbal lock. In simulations where complete looping maneuvers may have to be performed that is not acceptable. To overcome this problem, the so called quaternion method may be used. Quaternions are vectors in four-dimensional space that offer a mathematical notation that allows the representation of three dimensional rotations of objects. Quaternions also avoid the problem of gimbal lock and, at the same time, are numerically more efficient and stable when compared with traditional rotation matrices (e.g. $\mathbf{S}$) [24]. A rotation quaternion is then presented by:

$$\mathbf{q} = \begin{bmatrix} \cos(\varepsilon/2) \\ \sin(\varepsilon/2)\, n_1 \\ \sin(\varepsilon/2)\, n_2 \\ \sin(\varepsilon/2)\, n_3 \end{bmatrix} \tag{3.28}$$

where $\mathbf{q}$ represents a rotation about the unit vector $[n_1\ n_2\ n_3]$ through an angle ε. The time derivative of the rotation quaternion is also provided by [23]:

$$\dot{\mathbf{q}} = \frac{1}{2}\mathbf{\Omega}_q\mathbf{q} + \gamma\mathbf{q} = \frac{1}{2}\begin{bmatrix} 0 & -P & -Q & -R \\ P & 0 & R & -Q \\ Q & -R & 0 & -P \\ R & Q & -P & 0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} + \gamma \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} \tag{3.29}$$

$$\gamma = 1 - \left(q_1^2 + q_2^2 + q_3^2 + q_4^2\right) \tag{3.30}$$

For practical reasons, the initialization of equation (3.29) requires that we express the quaternion components in terms of Euler angles because doing it with quaternions is not so straightforward. Therefore, expressing the quaternion vector elements as a function of Euler angles yields:

$$\begin{cases} q_0 = \cos\left(\frac{\psi}{2}\right)\cos\left(\frac{\theta}{2}\right)\cos\left(\frac{\phi}{2}\right) + \sin\left(\frac{\psi}{2}\right)\sin\left(\frac{\theta}{2}\right)\sin\left(\frac{\phi}{2}\right) \\ q_1 = \cos\left(\frac{\psi}{2}\right)\cos\left(\frac{\theta}{2}\right)\cos\left(\frac{\phi}{2}\right) - \sin\left(\frac{\psi}{2}\right)\sin\left(\frac{\theta}{2}\right)\sin\left(\frac{\phi}{2}\right) \\ q_2 = \cos\left(\frac{\psi}{2}\right)\sin\left(\frac{\theta}{2}\right)\cos\left(\frac{\phi}{2}\right) + \sin\left(\frac{\psi}{2}\right)\cos\left(\frac{\theta}{2}\right)\sin\left(\frac{\phi}{2}\right) \\ q_3 = \sin\left(\frac{\psi}{2}\right)\cos\left(\frac{\theta}{2}\right)\cos\left(\frac{\phi}{2}\right) - \cos\left(\frac{\psi}{2}\right)\sin\left(\frac{\theta}{2}\right)\sin\left(\frac{\phi}{2}\right) \end{cases} \tag{3.31}$$

Otherwise, if we want to extrapolate the Euler angles from the quaternion differential equations we can achieve just that by taking intermediate steps of the rotation tensor $\mathbf{S}$ and relate them with the elements of the rotation quaternion matrix, rather than calculating them from the quaternion directly:

$$\mathbf{S}_q = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2\left(q_1 q_2 + q_0 q_3\right) & 2\left(q_1 q_3 - q_0 q_2\right) \\ 2\left(q_1 q_2 - q_0 q_3\right) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2\left(q_2 q_3 + q_0 q_1\right) \\ 2\left(q_1 q_3 + q_0 q_2\right) & 2\left(q_2 q_3 - q_0 q_1\right) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} \tag{3.32}$$

where $\mathbf{S}_q$ is the quaternion equivalent of the rotation matrix (3.3). The Euler angles can therefore be calculated by using:

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \arctan\left(\frac{2(q_2 q_3 + q_0 q_1)}{q_0^2 - q_1^2 - q_2^2 - q_3^2}\right) \\ \arcsin\left(-2\left(q_1 q_3 - q_0 q_2\right)\right) \\ \arctan\left(\frac{2(q_2 q_3 + q_0 q_1)}{q_0^2 + q_1^2 - q_2^2 - q_3^2}\right) \end{bmatrix} \tag{3.33}$$

In analogy to (3.21), we can also get the absolute velocity using the quaternion rotation tensor:

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{bmatrix} = \mathbf{S}_q^T \begin{bmatrix} U \\ V \\ W \end{bmatrix} \tag{3.34}$$

To employ the quaternion method we must first use the initial Euler angles in (3.31), and use the resulting quaternion elements in (3.32). It is also possible to combine the previous quaternion equations with the dynamics equations to compose the vehicle acceleration on the aircraft local frame:

$$\mathbf{F}_g = m\mathbf{S}_q \begin{bmatrix} 0 & 0 & g \end{bmatrix}^T = mg \begin{bmatrix} 2\left(q_1 q_3 - q_0 q_2\right) & 2\left(q_2 q_3 + q_0 q_1\right) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}_B^T \tag{3.35}$$

$$\begin{bmatrix} \dot{U} \\ \dot{V} \\ \dot{W} \end{bmatrix} = \frac{1}{m}\begin{bmatrix} F_{Px} \\ F_{Py} \\ F_{Pz} \end{bmatrix} + g\begin{bmatrix} 2\left(q_1 q_3 - q_0 q_2\right) \\ 2\left(q_2 q_3 + q_0 q_1\right) \\ q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} - \begin{bmatrix} QW - RV \\ RU - PW \\ PV - QU \end{bmatrix} \tag{3.36}$$

3.4 Modelling of a motor-propeller assembly

The motor-propeller assemblies are essential components of our aircraft since they are responsible for the production of the lifting force that allows flight. This section focuses on modeling the dynamics of

these components.

As we have seen previously in the quadrotor design section there are many variables to account for when choosing the right motor-propeller assembly. This is mostly because the thrust is a function of several properties such as the propeller's diameter, thrust coefficient, air density and the motor's k_v. Moreover, although we have 4 "equal" motors, in reality each one of them has a different dynamic behavior. This dissimilarity requires a more precise modelling of the motors, a process that often starts by establishing the relation between the voltage fed to the motor and the matching speed of rotation.

A permanent magnet DC motor transfer function is usually described by a gain and two real poles (also known as a second-order model), respectively associated with mechanical and electrical time constants, being the mechanical constant commonly higher than the electrical one by at least one order of magnitude. In other words, the pole associated with the electrical constant is the faster one and as a consequence the dynamic behavior of such a motor is dominated by the slower pole. Having this knowledge into consideration, a common way to define the dynamic behavior of a DC motor is by using a first order transfer function, having only a gain and a pole associated with the mechanical time constant.

To identify the transfer function of each motor we have to conceive a method of data acquisition, particularly between the input and output of each motor. The propeller rotation speed (output) can be acquired through a tachometer device. For the input we could use the voltage provided to a motor but for practical reasons it is by far simpler to use the PWM signal because it is a variable we have direct access to (measuring voltage is also possible using the analog pins on the Arduino board, but in this case these are already being used for sensor readings). Figure 3.4.1 shows an idealization of the identification process.

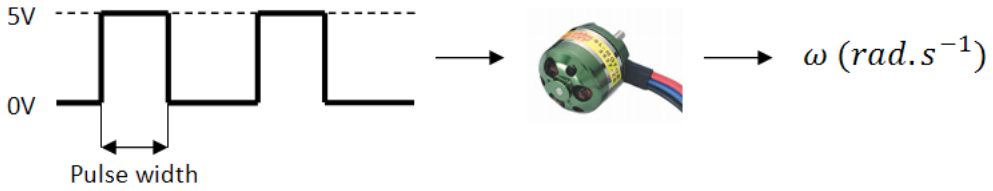


Figure 3.4.1: Schematic of the input-output sequence for identification purposes.

Here arises our first big problem: we know our inputs because we can program the pulse width of the signals that control the speed of the motors, but how can we know our outputs if one does not have direct access to a digital tachometer device? The devised solution for this problem was to build a rudimentary tachometer to get the job done. This device consisted of a simple microphone placed approximately $3cm$ away from the tip of the propeller and high enough not to get hit by it (see Figure 3.4.2). The technology itself does not seem to be new, a simple search on the Internet reveals that there are already a few digital microphone tachometers available to the general public. Nonetheless, the concept behind it is quite simple: at each pass of the propeller, air gets pulled downwards of the propeller plane and a microphone's membrane may be able to register the consequent suction effect. Another advantage of using a microphone is the fact that modern day computers can capture sound data of a microphone in real-time, which makes it very useful for any MATLAB® data processing or Simulink® data acquisition that may be required. Following this logic, by using Simulink® it was possible to provide orders to an Arduino program and capture the sound data of each propeller.

Figure 3.4.2: Microphone positioned to record the propeller sound.

Sound processing of the captured sound files was made as simple as possible. Using a MATLAB® script it was possible to plot the sound data and measure the period between each passage of the propeller blades T_p (see Figure 3.4.3), calculating the speed of rotation immediately afterwards:

$$\omega = \frac{\pi}{T_P} \tag{3.37}$$

It is important to mention that although this method for measuring the propeller output is very cheap and accessible, it does not work very well for low speeds of rotation as was verified during experimental trials, mostly due to very low signal-to-noise ratios (i.e. noise amplitude is so high that it gets very hard to distinguish what is ambient noise from propeller sound). Figure 3.4.3 shows a very clear signal form of the propeller sound, where we can clearly identify the sound pressure waves of the passage of the propeller. Proceeding with the motor-propeller system identification, measurements took place to find the respective dead zone (Table 3.1) and dynamic response (Table 3.2).

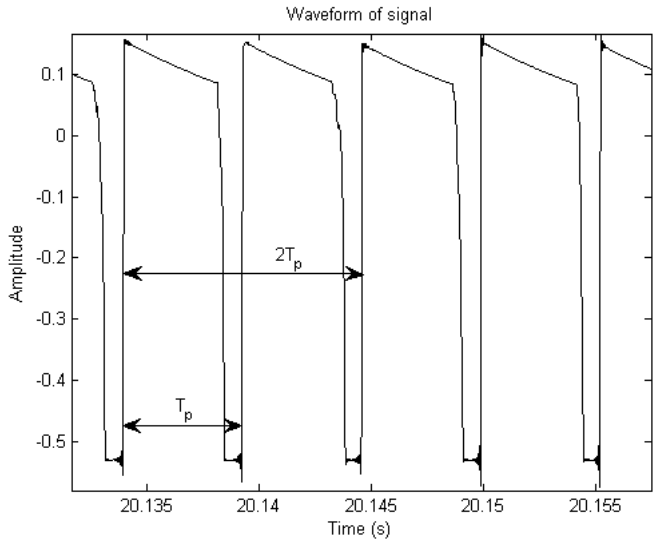


Figure 3.4.3: Propeller sound. Each step in amplitude is a consequence of a propeller blade passage.

Table 3.1: Dead zone pulse width of each motor.

Motor i	Dead zone pulse width (μs)
1	1262
2	1252
3	1250
4	1252

Table 3.2: Data points gathered for understanding the motor-propeller dynamics.

Input PWM signal (μs)	1500	1700	1800	2000
Motor 1 output ($rad.s^{-1}$)	4268.6	5926.5	6504.8	6729.5
Motor 2 output ($rad.s^{-1}$)	4083.3	5732.8	6361.3	6645.9
Motor 3 output ($rad.s^{-1}$)	4336.5	6132.5	6746.1	6925.2
Motor 4 output ($rad.s^{-1}$)	4329.6	6064.2	6684.5	6890.2

Figure 3.4.4 shows plotted data of tables 3.1 and 3.2 revealing that the dynamics of the motors is nonlinear, meaning that direct implementation of a first-order transfer function to model the dynamic of each motor is not adequate.

To solve this issue it was decided that to obtain a linear model (to allow implementation of first-order transfer functions) the model should be linearized around an adequate operation point. Initial experimental flight tests were executed and revealed that shortly beyond the dead zone the quadrotor took off from the ground. Taking this observation into consideration led to the choice of a relatively low operation point (pulse width) of $1300\mu s$.

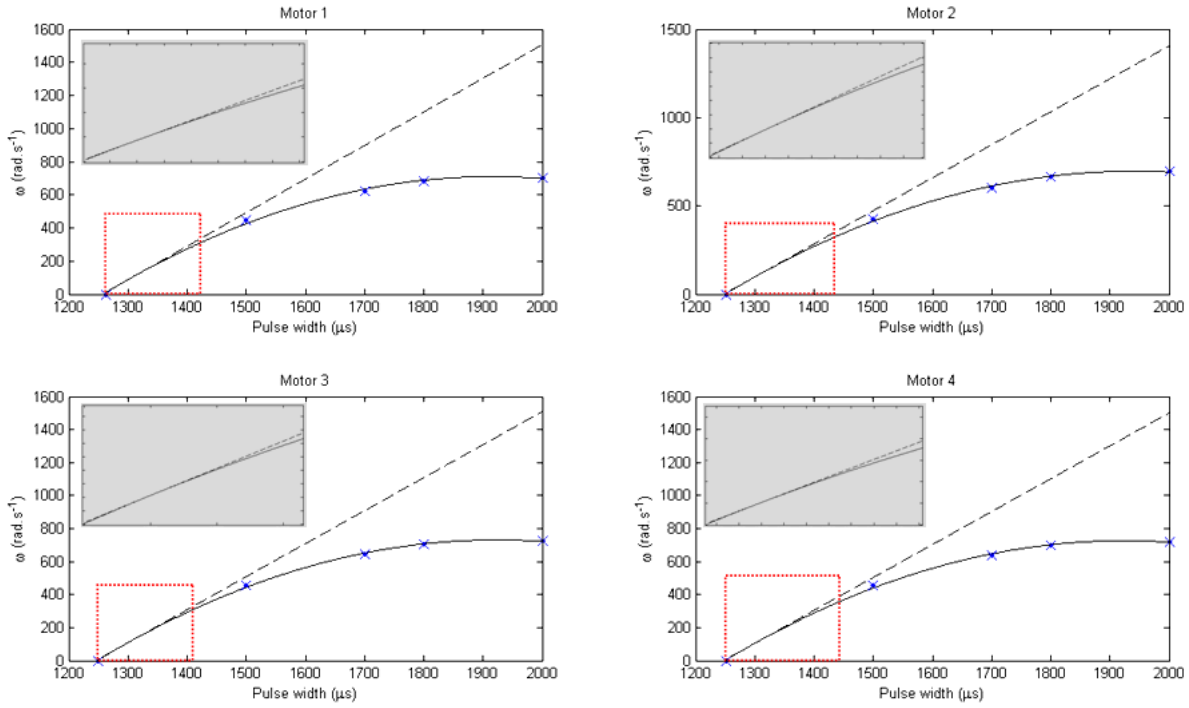


Figure 3.4.4: Dynamic behaviour of the motors. The grey areas show a zoom on of the linearization zone in red.

(x collected data points, - second order polynomial data fit, – linearization)

First order transfer functions require a gain K and a time constant τ:

$$G(s) = \frac{K}{\tau s + 1} \tag{3.38}$$

The gains are equal to the slope of the linearized dynamic response of each motor (dashed lines of Figure 3.4.4; slope values in Table 3.3). Input-output data was gathered for motor 1 and the Open System Identification Tool Graphical User Interface in MATLAB® was used to estimate the respective time constant, with value $0.136s$. A sample of the input-output data used can be observed in Figure 3.4.5. Time constants for the four motors were assumed to be all equal to the one of motor 1, which also serves as future test for robustness of the controller.

Table 3.3: Gains of the transfer functions describing the motors dynamics.

	Motor 1	Motor 2	Motor 3	Motor 4
Gain K	2.0276	1.8693	2.0018	1.996

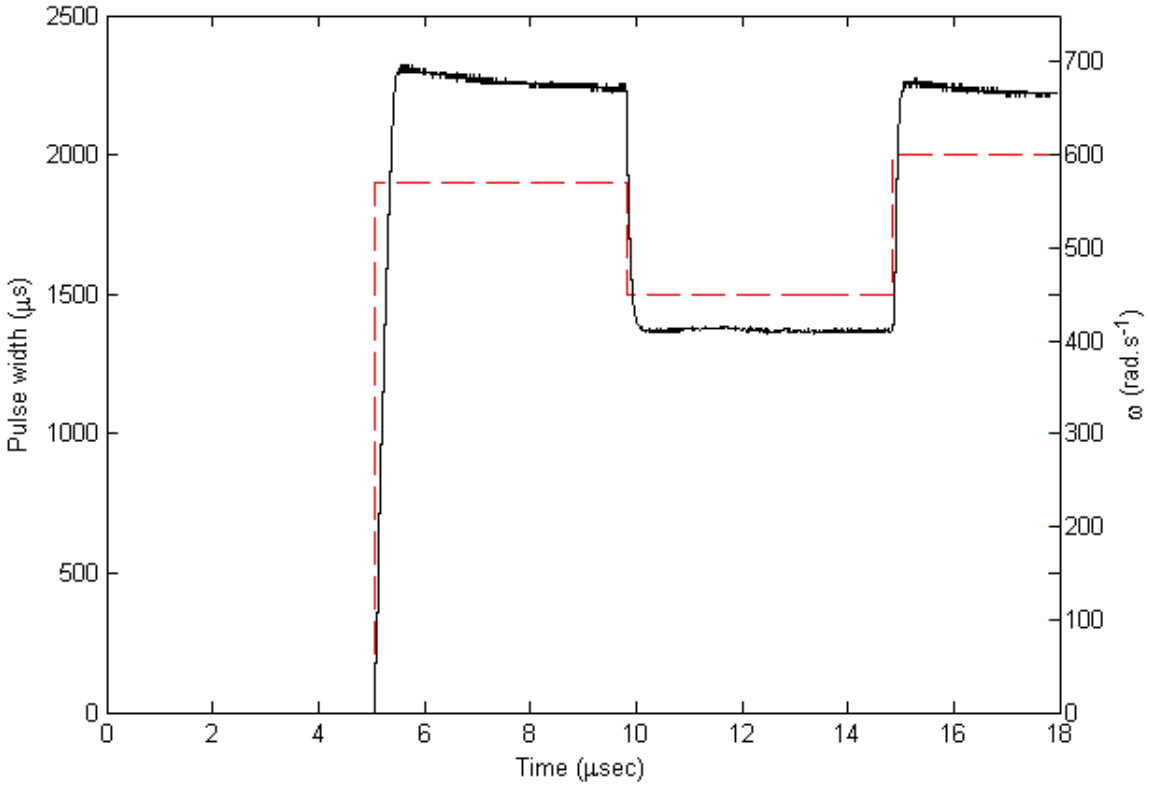


Figure 3.4.5: Input-output data used with the system identification toolbox to calculate the time constant. (– pulse width of the input signal; - speed of rotation)

By using Simulink® to simulate the motors (Figure 3.4.6) we can extend the quality of it if we add extra parameters like saturation of the motors speed (i.e. maximum and lowest possible speeds), the dead zone of each motor, and sampling frequency (i.e. the Arduino is a digital platform and therefore it is not a continuous-time processing device).

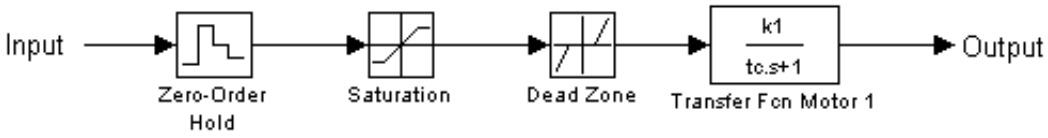


Figure 3.4.6: Blocks used in Simulink® to simulate motor 1.

Now that we have a way to calculate the speed of rotation of a motor-propeller assembly in a simulation environment, we also need to calculate the force and moments produced by each one of these assemblies. Total force can be calculated by adding the thrust produced by a propeller as in equation (2.1). As for the moment M_P produced by the same propeller we have:

$$P_P = \omega M_P \tag{3.39}$$

Replacing this equation (3.39) in (2.2) leads to:

$$M_P = c_P \frac{4\rho r^5}{\pi^3}\omega^2 \tag{3.40}$$

Assuming all variables in eqs. (2.1) and (3.40) are constant with exception for angular speed, propeller moment and thrust, we can rewrite these equations:

$$T = K_T \omega^2 \tag{3.41}$$

$$K_T = c_T \frac{4\rho r^4}{\pi^2} = 0.1154 \times \frac{4 \times 1.2 \times (5 \times 25.4 \times 10^{-3})^4}{\pi^2} = 1.46 \times 10^{-5} kg.m.rad^{-2} \tag{3.42}$$

$$M_T = K_M \omega^2 \tag{3.43}$$

$$K_M = c_P \frac{4\rho r^5}{\pi^3} = 0.0743 \times \frac{4 \times 1.2 \times (5 \times 25.4 \times 10^{-3})^5}{\pi^3} = 3.8 \times 10^{-7} kg.m^2.rad^{-2} \tag{3.44}$$

where K_M and K_T are constants that respectively relate a propeller moment and thrust with angular speed. Moment and thrust can therefore also be related through the constant K_{TM}:

$$M_T = \frac{K_M}{K_T} T = K_{TM} T = \frac{c_P r}{c_T \pi} T \tag{3.45}$$

$$K_{TM} = 0.026m \tag{3.46}$$

3.5 Moments of inertia

Mass and its geometric distribution in an aircraft is something of extreme importance because it affects the entire dynamics of the system. Then, after building the quadrotor, it is time to evaluate some of its most important features like its moments of inertia and mass.

We will assume the inertia matrix is diagonal and positive-definite, with the purpose of simplifying

the calculations and also due to the particular symmetric geometry of quadrotors (see Figure 3.5.1). As such, the calculation of inertia moments will only include the geometry and mass of the motors, as well as their geometric position on the quadrotor.

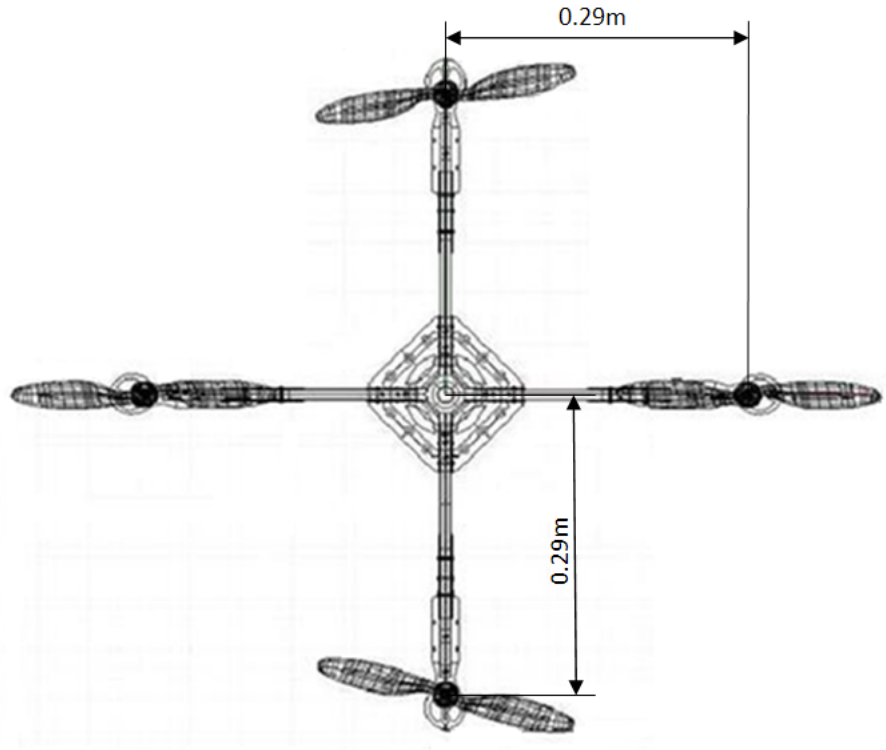


Figure 3.5.1: Distance from the motors to the center of gravity.

Each motor weights approximately $0.048kg$. Other important variables related to the aircraft such as the elements of each motor inertia tensor (l_x, l_y and l_z) can be consulted in table 3.4.

Table 3.4: Quadrotor's mass and main geometric variables.

Variable	Value
$l_x\ (m)$	0.0288
$l_y\ (m)$	0.0288
$l_z\ (m)$	0.026
$d_{cg}\ (m)$	0.29
$m\ (kg)$	0.82

It follows that the inertia matrix elements of the aircraft are:

$$\begin{cases} I_{x_1} = I_{x_3} = \frac{1}{12} m_m (l_y^2 + l_z^2) = 6.0218 \times 10^{-6}\ kg.m^2 \\ I_{x_2} = I_{x_4} = \frac{1}{12} m_m (l_y^2 + l_z^2) + m_m d_{cg}^2 = 0.004\ kg.m^2 \\ I_{11} = 2I_{x_1} + 2I_{x_2} = 0.0081\ kg.m^2 \end{cases} \tag{3.47}$$

$$\begin{cases} I_{y_1} = I_{y_3} = \frac{1}{12} m_m (l_x^2 + l_z^2) + m_m d_{cg}^2 = 0.004\ kg.m^2 \\ I_{y_2} = I_{y_4} = \frac{1}{12} m_m (l_x^2 + l_z^2) = 6.0218 \times 10^{-6}\ kg.m^2 \\ I_{22} = 2I_{y_1} + 2I_{y_2} = 0.0081\ kg.m^2 \end{cases} \tag{3.48}$$

$$\begin{cases} I_{z_1} = I_{z_2} = I_{z_3} = I_{z_4} = \frac{1}{12} m_m (l_x^2 + l_y^2) + m_m d_{cg}^2 = 0.004\ kg.m^2 \\ I_{33} = 4I_{z_1} = 0.0162\ kg.m^2 \end{cases} \tag{3.49}$$

And the inertia matrix itself is:

$$\mathbf{I} = \begin{bmatrix} I_{11} & 0 & 0 \\ 0 & I_{22} & 0 \\ 0 & 0 & I_{33} \end{bmatrix} = \begin{bmatrix} 0.0081 & 0 & 0 \\ 0 & 0.0081 & 0 \\ 0 & 0 & 0.0162 \end{bmatrix} kg.m^2 \tag{3.50}$$

3.6 Calculation of the center of gravity

It has been clear so far that we have assumed the quadrotor to have its center of gravity located in the center of the XY plane of the O_{ABC} reference frame. Nonetheless it is also important to calculate its position along the Z axis because we will need it to know the relative position of the accelerometer in the next section. This task was carried out in the most practical way possible through the use of the SolidWorks® software, in which some of the most heavy parts of the quadrotor were modeled (e.g. motors, sensors, main battery and aluminum beams) and assembled. Figure 3.6.1 shows the computed position of the COG, located $6.7cm$ above the base of the quadrotor.

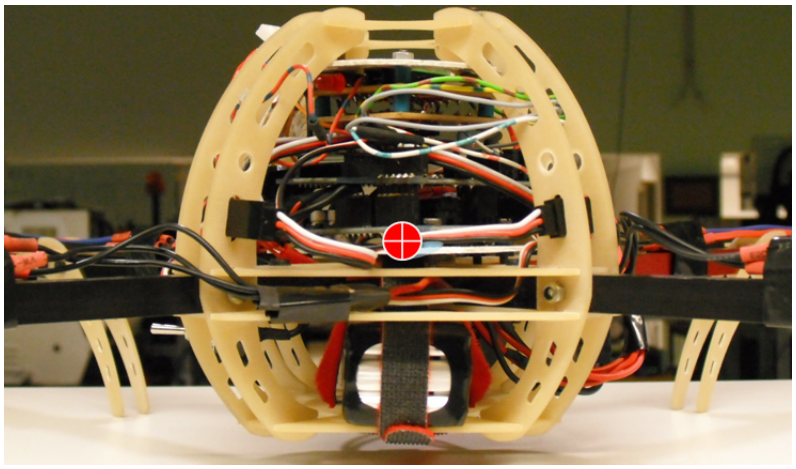


Figure 3.6.1: The red dot in the center of the figure indicates the center of gravity.

3.7 Sensors modelling

Inertial navigation is implemented in cases where the use of an external reference to measure position is impractical. Typical inertial navigation systems used in real-life applications such as aircraft or space rockets are highly advanced, and expensive, pieces of technology. However, inexpensive sensoring

equipment can be used to make a less accurate inertial navigation unit, hence the need for sensor modelling to increase the precision of computational simulations.

3.7.1 Accelerometer

As the name implies, the accelerometer is a sensor that measures accelerations. Typically, accelerometers like the one installed onboard our quadrotor work through piezo resistive materials, that when deformed by an external force, change the intensity of the electric current flowing through them, providing in this manner a quantification of that force.

Ideally, an accelerometer should be located near the center of gravity of the aircraft. In the case of our quadrotor it is positioned in the coordinates indicated in table 3.5.

Having the position of the accelerometer relative to the center of gravity, the measured acceleration $\mathbf{a}_m$ is given by [19]:

$$\mathbf{a}_m = \mathbf{a}_p - \mathbf{S}\begin{bmatrix} 0 & 0 & g \end{bmatrix}^T \tag{3.51}$$

where $\mathbf{a}_p$ is the absolute acceleration at point p, usually the point where the accelerometer is located. Also, the instantaneous velocity $\mathbf{v}_p$ at point p is:

$$\mathbf{v}_p = \mathbf{v}_{cg} + \boldsymbol{\omega}_B \times \mathbf{r}_s \tag{3.52}$$

where $\mathbf{r}_s$ is the position of the accelerometer relative to the quadrotor's center of gravity (see Table 3.5 for data on the accelerometer location) and $\mathbf{v}_{cg}$ is the vector of linear velocities at the COG. Applying the Coriolis theorem yields:

$$\mathbf{a}_p = \mathbf{a}_{cg} + \dot{\boldsymbol{\omega}}_B \times \mathbf{r}_s + \boldsymbol{\omega}_B \times (\boldsymbol{\omega}_B \times \mathbf{r}_s) + 2\boldsymbol{\omega}_B \times \mathbf{v}_{cg} \tag{3.53}$$

Given that:

$$m\mathbf{a}_{cg} = m\mathbf{a}_B + m\mathbf{S}\begin{bmatrix} 0 & 0 & g \end{bmatrix}^T \tag{3.54}$$

we can rewrite equation (3.53) to provide absolute acceleration of the accelerometer located at point p:

$$\mathbf{a}_p = \frac{\mathbf{a}_B}{m} + \mathbf{S}\begin{bmatrix} 0 & 0 & g \end{bmatrix}^T + \dot{\boldsymbol{\omega}}_B \times \mathbf{r}_s + \boldsymbol{\omega}_B \times (\boldsymbol{\omega}_B \times \mathbf{r}_s) \tag{3.55}$$

Equation (3.55) does not contain the Coriolis acceleration. The reason for this is that the Coriolis Force is proportional to the velocity of the aircraft. Because we do not want to achieve extreme velocities, we are going to neglect this term of the equation. As we have a tri-axis accelerometer, we need to read three components of acceleration:

$$\begin{bmatrix} a_{p_x} \\ a_{p_y} \\ a_{p_z} \end{bmatrix} = \frac{1}{m}\begin{bmatrix} F_{x'} \\ F_{y'} \\ F_{z'} \end{bmatrix} + g\begin{bmatrix} -\sin\theta \\ \cos\theta\sin\phi \\ \cos\theta\cos\phi \end{bmatrix} + \begin{bmatrix} \dot{Q}r_{s_z} - \dot{R}r_{s_y} \\ \dot{R}r_{s_x} - \dot{P}r_{s_z} \\ \dot{P}r_{s_y} - \dot{Q}r_{s_x} \end{bmatrix} + \begin{bmatrix} Q(Pr_{s_y} - Qr_{s_x}) - R(Rr_{s_x} - Pr_{s_z}) \\ R(Qr_{s_z} - Rr_{s_y}) - P(Pr_{s_y} - Qr_{s_x}) \\ P(Rr_{s_x} - Pr_{s_z}) - Q(Qr_{s_z} - Rr_{s_y}) \end{bmatrix} \tag{3.56}$$

Table 3.5: Relevant accelerometer data.

Variable	Value
Number of bits used by the Arduino for A/D conversion N_b	10
Sensor measurement range y_s	$\pm 3g$
Sensitivity	$0.3V.g^{-1}$
Position along the x-axis relative to the center of gravity r_{s_x}	$0.003m$
Position along the y-axis relative to the center of gravity r_{s_y}	$0m$
Position along the z-axis relative to the center of gravity r_{s_z}	$-0.049m$

Using data from Table 3.5, the sensor reading resolution of the Arduino board will be:

$$R_{acc} = \frac{y_{smax} - y_{smin}}{2^{N_b}} = \frac{3 - (-3)}{2^{10}} = 0.00586g = 0.0575m.s^{-2} \tag{3.57}$$

Other variables such as variation of sensor sensibility with temperature can be simulated with the addition of white Gaussian noise.

As already noticed, we do not have of gyros onboard the quadrotor. Their absence also makes the determination of the angles of roll and pitch quite tricky because gyros are used for sensor fusion together with accelerometers, providing very steady outputs, thus making them suitable for vibration intensive applications.

A rather less robust but direct path to obtain these angles is through the vector of gravitational acceleration provided by the accelerometer, based on the initial approach that we do not intend to move the quadrotor at high speeds (otherwise very high accelerations will indicate roll and pitch angles with very high error) :

$$\phi = \arctan\left(\frac{a_{p_y}}{a_{p_z}}\right) \tag{3.58}$$

$$\theta = -\arctan\left(\frac{a_{p_x}}{a_{p_z}}\right) \tag{3.59}$$

3.7.2 Compass

Compasses are scientific instruments used to measure the direction of the magnetic field in the vicinity of the sensor. Among other applications, they are used in aircraft to provide a directional bearing component centered on the north or south poles. The prototype of this thesis uses a compass which serves this exact purpose. Electronic compasses operate using the Hall-effect technology, which is to say that in its interior there are materials which vary the voltage at its terminals when crossed by a magnetic field. Let us assume that the sensor can be modeled as follows:

$$\psi = N_c \tag{3.60}$$

where N_c is the direction of the magnetic North Magnetic Pole mapped between $-\pi$ and π radians. Table 3.6 shows the key characteristics of the sensor to improve the simulation veracity.

Similarly to the accelerometer we can add white Gaussian noise to the compass readings to simulate the effects of parasite magnetic fields.

Table 3.6: Relevant compass data

Variable	Value
Sensor measurement range	360°
Selected resolution	1°

3.8 Model implementation in MATLAB® and Simulink®

Starting with the definition of all the variables that are important to the simulations, we can load them all into the MATLAB® environment by using the file *loadvars.m* (see Appendix B.1). As for the dynamics of the quadrotor's body, its modeling was accomplished by applying the equations previously explained in this section through a function named quadsys.m (see Appendix B.2). Here's how this function works, step by step:

1. Use the provided state vector of the quadrotor to build the quaternion rotation matrix $\mathbf{S}_q$ necessary for the next steps using equation 3.32;

2. Calculate the forces and moments applied on the quadrotor by the propellers using equations (3.41) and (3.43);

3. Compute the linear and angular accelerations using equations (3.36) and (3.14);

4. Compute the linear and quaternion angular velocities through equations (3.34) and (3.29);

5. Output the states vector derivative $\begin{bmatrix} \dot{U} & \dot{V} & \dot{W} & \dot{P} & \dot{Q} & \dot{R} & \dot{X} & \dot{Y} & \dot{Z} & \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}$.

Now, when we import the function into the Simulink® environment we must use an integrator block to produce linear and angular velocities from the linear and angular accelerations, and we can also use the same logic to obtain the linear position and quaternion angle representation respectively from the linear velocities and angular velocity quaternion. Of course angles in the form of a quaternion are not very intelligible, so in order to transform a quaternion into Euler angles we can use a transformation block available in Simulink® for this task (the name of the block is *Quat2Ang*). One should be very careful not to mistake the angles order of rotation, in our case we are working with the order ZYX (or Yaw-Pitch-Roll) and therefore if we wish to obtain, as an example, the roll, we should get it in the third output of this block. A diagram illustrating the *quadsys.m* function is presented in figure 3.8.1.

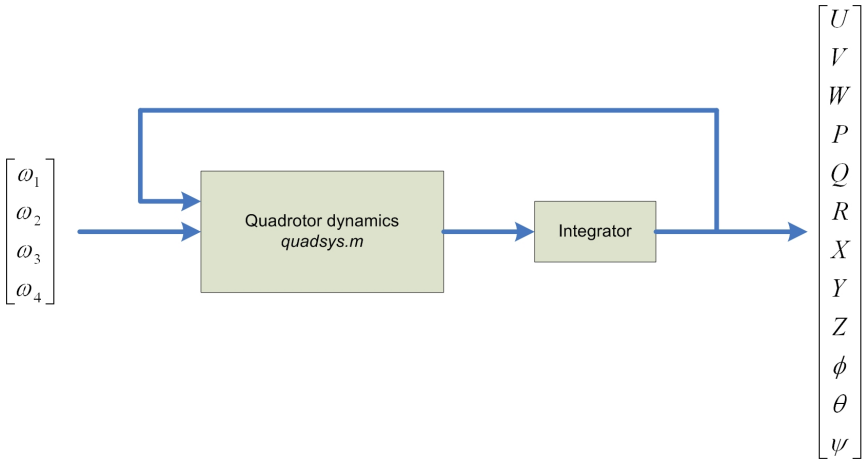


Figure 3.8.1: Block diagram of quadrotor dynamics.

Turning our attention to the sensors, the *quadsens.m* function was designed to simulate the sensor readings. It reads the outputs (i.e. the state vector) from the quadrotor dynamics blocks and outputs the sensor readings. Here is a description of the algorithm:

1. Use the provided state vector of the quadrotor to assemble the rotation matrix $\mathbf{S}$ essential for the next steps from equation (3.3);
2. Calculate the forces and moments applied on the quadrotor by the propellers using equations (3.41) and (3.43);
3. Compute the accelerations felt by the accelerometer using equation (3.56);
4. Use the read accelerations to determine the roll and pitch angles using equations (3.58) and (3.59);
5. Get the compass reading from the yaw angle of the provided state vector using equation (3.60);
6. Output the sensor readings $\begin{bmatrix} a_{p_x} & a_{p_y} & a_{p_z} & \phi & \theta & \psi \end{bmatrix}$.

Now that we have the sensor readings it is important not to forget the application of the sensor characteristics, which can be easily done in the Simulink® environment. This can be seen in figure 3.8.2 and the Simulink® implementation in figure 3.8.3.

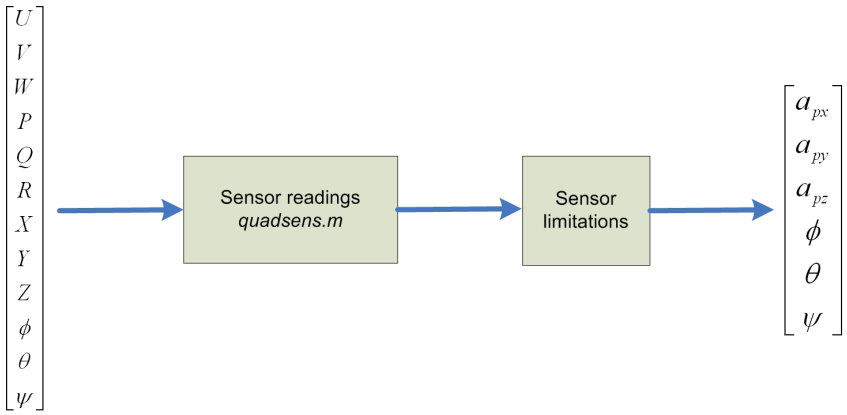


Figure 3.8.2: Block diagram of the sensors.

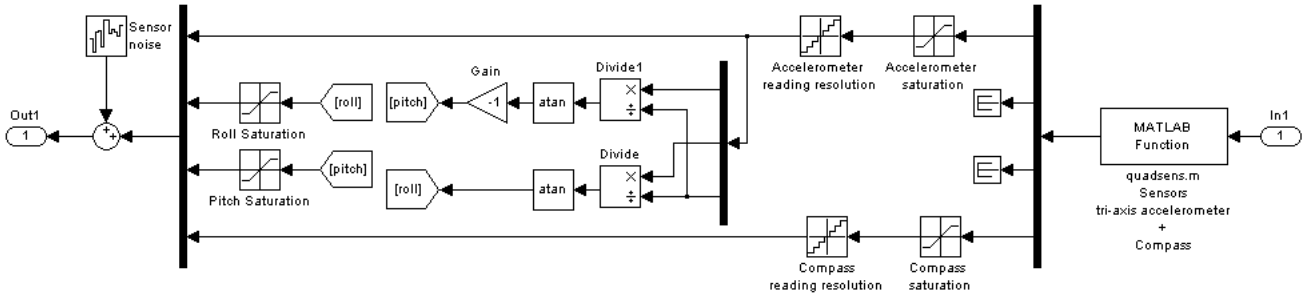


Figure 3.8.3: Inside view of sensors Simulink® block.

Chapter 4

Kalman Filter

The Kalman filter [25] is a recursive filter[1] created in 1960 by engineer Rudolf Emil Kalman that estimates the states of a linear system from readings corrupted by noise (e.g. sensor readings).

The word recursive in the former description means that, distinctly from certain data processing concepts, the Kalman filter does not need all previous data to be kept in storage and reprocessed every time a new measurement is taken. This will be of vital importance to the practicality of filter implementation (e.g. Arduino memory restrictions).

Regardless of the typical connotation of filter as a "black-box" containing electrical networks, the fact is that in most practical applications, the Kalman filter is just a computer program in a central processor. As such, it inherently incorporates discrete-time measurement samples rather than continuous time inputs. Often, the variables of interest, some finite number of quantities to describe the state of the system cannot be measured correctly, and some means of estimating these values from the available data must be generated (i.e. sometimes not all of the states are available). Thus deduction is problematical by the facts that a system driven by inputs other than our own known controls and the relationships among the various state variables and measured outputs are known only with some degree of uncertainty. Furthermore, any measurement will be degraded to some degree by noise, biases, and some device inaccuracies, and so a means of extracting valuable information from a noisy signal must be provided also. There may as well be a number of different measurement devices, each with its own particular dynamics and error characteristics, that supply some information about a particular variable, and it would be desirable to combine their outputs in a methodical and optimal manner. A Kalman filter combines any available measurement data, plus prior knowledge about the system and measuring devices, to generate an estimate of the desired variables in such a manner that the error is minimized statistically (e.g. if we were to run a number of candidate filters many times for the same application, then the average results of the Kalman filter would be better than the average of any other) [26].

In resume, the Kalman filter algorithm incorporates all information that can be provided to it, regardless of the precision, to estimate the current value of the variables of interest, with use of:

- Knowledge of the system ;
- Measurement device dynamics ;
- Statistical description of the system noises;

[1] Filter that uses one or more of its outputs as inputs.

- Measurement errors;
- Uncertainty in the dynamics models;
- Any available information about initial conditions of the variables of interest.

Provided that we can expect noisy measurements from our sensors, in this chapter we aim to design a Kalman filter to estimate as many of the system states as possible from those sensor readings.

4.1 State-space system linearization

The present equations in section 3 that describe the dynamics of the system are not linear. As already mentioned before, in such a situation it is common to linearize the equations around an operating point. Linearizing the system will facilitate the construction of the Kalman filter and quadrotor control, as we shall see later.

Let us take as a starting point for the process of linearization the following state vector $\bar{\mathbf{x}}$, which illustrates nothing less than the quadrirotor in flight, stabilized at a height z from the ground, what is usually known as an equilibrium point:

$$\bar{\mathbf{x}} = [U\ V\ W\ P\ Q\ R\ X\ Y\ Z\ \phi\ \theta\ \psi] = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ -z\ 0\ 0\ 0] \tag{4.1}$$

where the R, X, Y and ψ can assume any random value because they do not interfere in the linearization process. Let us also consider the following equilibrium input vector $\bar{\mathbf{u}}$ (i.e. the vector with the speed of rotation of each one of the motors):

$$\bar{\mathbf{u}} = \begin{bmatrix} \omega_1 & \omega_2 & \omega_3 & \omega_4 \end{bmatrix} \tag{4.2}$$

As an example, in the case of the quadrotor $\bar{\mathbf{u}}$ can be built by finding the angular speed of each motor so that the total thrust produced is equal to the force of gravity. Implementing the first order Taylor series expansion, the linearization of the quadrotor's model:

$$\begin{cases} \dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) \\ \mathbf{y} = h(\mathbf{x}, \mathbf{u}) \end{cases} \tag{4.3}$$

around an operating point $(\bar{\mathbf{x}}, \bar{\mathbf{u}})$ is [27]:

$$\begin{cases} \dot{\mathbf{x}} \approx f(\bar{\mathbf{x}}, \bar{\mathbf{u}}) + \frac{\partial f(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}} (\mathbf{x} - \bar{\mathbf{x}}) + \frac{\partial f(\mathbf{x},\mathbf{u})}{\partial \mathbf{u}} (\mathbf{u} - \bar{\mathbf{u}}) \\ \mathbf{y} \approx h(\bar{\mathbf{x}}, \bar{\mathbf{u}}) + \frac{\partial h(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}} (\mathbf{x} - \bar{\mathbf{x}}) + \frac{\partial h(\mathbf{x},\mathbf{u})}{\partial \mathbf{u}} (\mathbf{u} - \bar{\mathbf{u}}) \end{cases} \tag{4.4}$$

where $f(\mathbf{x}, \mathbf{u})$ are the set of equations providing the dynamic behavior of the system (see equations (3.36), (3.14), (3.34) and (3.29)) and $h(\mathbf{x}, \mathbf{u})$ are the equations that simulate the sensor outputs (see equations (3.56), (3.58), (3.59) and (3.60)). Assuming that the presence of very small disturbances around the equilibrium point $(\bar{\mathbf{x}}, \bar{\mathbf{u}})$, $(\mathbf{x} - \bar{\mathbf{x}})$ and $(\mathbf{u} - \bar{\mathbf{u}})$ are both almost zero, we can verify:

$$\begin{cases} \dot{\mathbf{x}} = 0 \Rightarrow f(\bar{\mathbf{x}}, \bar{\mathbf{u}}) = 0 \\ \mathbf{y} = 0 \Rightarrow h(\bar{\mathbf{x}}, \bar{\mathbf{u}}) = 0 \end{cases} \tag{4.5}$$

Therefore:

$$\begin{cases} \dot{\mathbf{x}} = \frac{\partial f(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}} (\mathbf{x} - \bar{\mathbf{x}}) + \frac{\partial f(\mathbf{x},\mathbf{u})}{\partial \mathbf{u}} (\mathbf{u} - \bar{\mathbf{u}}) \\ \mathbf{y} = \frac{\partial h(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}} (\mathbf{x} - \bar{\mathbf{x}}) + \frac{\partial h(\mathbf{x},\mathbf{u})}{\partial \mathbf{u}} (\mathbf{u} - \bar{\mathbf{u}}) \end{cases} \tag{4.6}$$

with:

$$\mathbf{A} = \frac{\partial f(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}} \tag{4.7}$$

$$\mathbf{B} = \frac{\partial f(\mathbf{x},\mathbf{u})}{\partial \mathbf{u}} \tag{4.8}$$

$$\mathbf{C} = \frac{\partial h(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}} \tag{4.9}$$

$$\mathbf{D} = \frac{\partial h(\mathbf{x},\mathbf{u})}{\partial \mathbf{u}} \tag{4.10}$$

Equation (4.6) is the generic state space representation:

$$\begin{cases} \dot{\widetilde{\mathbf{x}}} = \mathbf{A}\widetilde{\mathbf{x}} + \mathbf{B}\widetilde{\mathbf{u}} \\ \widetilde{\mathbf{y}} = \mathbf{C}\widetilde{\mathbf{x}} + \mathbf{D}\widetilde{\mathbf{u}} \end{cases} \tag{4.11}$$

with $\widetilde{\mathbf{x}} = \mathbf{x} - \bar{\mathbf{x}}$ and $\widetilde{\mathbf{y}} = \mathbf{y} - \bar{\mathbf{y}}$.

4.2 System analysis

If we wish the Kalman filter to be convergent, we must analyze some properties of discrete state space systems such as reachability, controllability and observability. By convergence we mean the Kalman filter estimated states error tries to go to zero while under the influence of noise. That noise cannot push a state very far and we can expect the variance of that state to remain bounded, and this is precisely what happens when the system is stable. If the system is not stable, then the variance of a state will explode when the system gets affected by noise. Hence, if we wish the Kalman filter to be convergent, we must analyze some properties of discrete state space systems such as reachability, controllability and observability [28].

By definition, reachability tells us if it is possible to find a control sequence such that an arbitrary state can be reached from any initial state in finite time. There are several methods for determining if a system is reachable, depending on whether the system matrix $\mathbf{A}$ is or isn't singular (i.e. if $\mathbf{A}$ isn't or is invertible). This property can be investigated by checking the determinant of matrix $\mathbf{A}$, which in this case equals zero, meaning this matrix is nonsingular. In this case we can immediately say the system is not reachable.

Controllability aims to tell us if it is possible to find a control sequence such that the origin (i.e. when a state is equal to zero) can be reached in finite time. For systems in which matrix $\mathbf{A}$ is nonsingular, we can say the system is controllable if there is a value for i such that $\mathbf{A}^i$ equals zero. In this case this is

true for $i = 4$, therefore our system is fully controllable.

Observability refers to the possibility of a given state being determined in finite time using only the system outputs. This property can be verified if the rank of the observability matrix $\mathcal{O}$:

$$\mathcal{O} = \begin{bmatrix} \mathbf{C} \\ \mathbf{CA} \\ \mathbf{CA}^2 \\ \vdots \\ \mathbf{CA}^{N-1} \end{bmatrix} \tag{4.12}$$

is equal to the system's order. The equivalent MATLAB® command is:

rank((obsv($\mathbf{A}$,$\mathbf{C}$*));*

The result is 6 meaning we have 6 unobservable states (from the total 12) in the system.

4.3 Mathematical formulation

To construct a Kalman filter we must meet 3 conditions:

1. The model of the system must be linear. Very regularly, even in the presence of nonlinearities, the common approach is to engage in system linearization around an operating point, given that it is easier to work with linear systems;

2. The noise is white, which means it has equal power over all frequencies. This also simplifies the math involved in the filter, for white noise is very similar to the actual noise affecting the system, within limited range of frequencies;

3. The noise has Gaussian density. Typically there are two statistical properties that are easily ascertainable and that characterize a noise signal, the mean and variance, to assume that the noise is Gaussian and simplify the process of filtering from the mathematical point of view.

Starting with the linearized continuous-time state space form of the system:

$$\begin{cases} \dot{\mathbf{x}} = \mathbf{Ax} + \mathbf{Bu} + \mathbf{Gw} \\ \mathbf{y} = \mathbf{Cx} + \mathbf{Du} + \mathbf{Hw} + \mathbf{v} \end{cases} \tag{4.13}$$

where the stochastic variables $\mathbf{v}$ and $\mathbf{w}$ respectively represent the noise affecting the sensor measurements and the aircraft, characterized by its respective noise covariances R_k and Q_k:

$$Q_k = E\left\{\mathbf{ww}^T\right\} \tag{4.14}$$

$$R_k = E\left\{\mathbf{vv}^T\right\} \tag{4.15}$$

The process noise covariances Q_k and measurement covariances R_k might change with each time step or measurement, however, here we assume they are constant. The optimal solution for the Kalman filter is then provided by:

$$\begin{cases} \hat{\mathbf{x}}(n) = \mathbf{A^k}\hat{\mathbf{x}}(n-1) + \mathbf{B^k}\begin{bmatrix} \mathbf{u}(n) \\ \mathbf{y}(n) \end{bmatrix} \\ \hat{\mathbf{y}}(n) = \mathbf{C^k}\hat{\mathbf{x}}(n-1) + \mathbf{D^k}\begin{bmatrix} \mathbf{u}(n) \\ \mathbf{y}(n) \end{bmatrix} \end{cases} \tag{4.16}$$

$$\mathbf{A^k} = \underline{\mathbf{I}} - \mathbf{LC} \tag{4.17}$$

$$\mathbf{B^k} = \begin{bmatrix} -LD & L \end{bmatrix} \tag{4.18}$$

$$\mathbf{C^k} = \mathbf{C}(\underline{\mathbf{I}} - \mathbf{LC}) \tag{4.19}$$

$$\mathbf{D^k} = \begin{bmatrix} (\underline{\mathbf{I}} - \mathbf{CL})\mathbf{D} & \mathbf{CL} \end{bmatrix} \tag{4.20}$$

where the Kalman gain $\mathbf{L}$ updates the estimate $\hat{\mathbf{x}}(n-1)$ using the new measurement $\mathbf{y}(n)$:

$$\hat{\mathbf{x}}(n) = \hat{\mathbf{x}}(n-1) + \mathbf{L}(\mathbf{y}(n) - \mathbf{C}\hat{\mathbf{x}}(n-1) - \mathbf{Du}(n)) \tag{4.21}$$

Notice that this last equation is the same equation used for states estimation in (4.16), only rewritten in different form. The main target of the Kalman estimator design is to minimize the errors:

$$\mathbf{e}_x = \mathbf{x} - \hat{\mathbf{x}} = \mathbf{C}(\mathbf{x} - \hat{\mathbf{x}}) + \mathbf{v} \tag{4.22}$$

$$\mathbf{e}_y = \mathbf{y} - \hat{\mathbf{y}} = (\mathbf{A} - \mathbf{LC})(\mathbf{x} - \hat{\mathbf{x}}) + \mathbf{Gw} - \mathbf{Lv} \tag{4.23}$$

This is accomplished by finding the filter gain $\mathbf{L}$ that provides the state-estimate $\hat{\mathbf{x}}$ that minimizes the steady-state error covariance $\overline{\mathbf{P}}$:

$$\overline{\mathbf{P}} = E\left\{(\mathbf{x} - \hat{\mathbf{x}})(\mathbf{x} - \hat{\mathbf{x}})^T\right\} \tag{4.24}$$

Therefore, the gain $\mathbf{L}$ can be calculated by solving an algebraic Riccati simplified equation [29]:

$$\mathbf{L} = \overline{\mathbf{P}}\mathbf{C}^T\left(\mathbf{C}\overline{\mathbf{P}}\mathbf{C^T} + \overline{\mathbf{R}}\right)^{-1} \tag{4.25}$$

where $\overline{\mathbf{R}}$ is the measurement noise covariance matrix.

4.4 Kalman filter in MATLAB® and Simulink®

To produce a Kalman filter in MATLAB® we need the state space matrices as well as the covariance matrices of the process and sensor noises. The numerical calculus of the state space matrices was

achieved by designing a MATLAB® function named *quadss.m*. This function works with a very basic algorithm:

1. Use the *quadsys.m* function to get matrices $\mathbf{A}$ and $\mathbf{B}$;

2. Use the *quadsens.m* function to calculate matrices $\mathbf{C}$ and $\mathbf{D}$.

After executing this code we get the following matrices:

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -9.81 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 9.81 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.0005 & -0.0005 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.26}$$

$$\mathbf{B} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -0.0132 & -0.0132 & -0.0132 & -0.0132 \\ 0 & -0.3881 & 0 & 0.3881 \\ 0.3881 & 0 & -0.3881 & 0 \\ 0.0174 & -0.0174 & 0.0174 & -0.0174 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.27}$$

$$\mathbf{C} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -19.62 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 19.62 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.001 & -0.001 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.28}$$

$$\mathbf{D} = \begin{bmatrix} 0 & 0.019 & 0 & -0.019 \\ 0.0001 & -0.0191 & 0.0001 & 0.019 \\ -0.0144 & -0.0132 & -0.0121 & -0.0132 \\ 0 & -0.0019 & 0 & 0.0019 \\ 0 & -0.0019 & 0 & 0.0019 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.29}$$

Matrix $\mathbf{D}$ shows that the Euler angles pitch and roll depend on the inputs, a consequence of the accelerometer not being positioned at the COG of the quadrotor. Now we have to eliminate all unobservable or uncontrollable states. The MATLAB® command for this operation is:

sysm=minreak(ss($\mathbf{A}$,$\mathbf{B}$,$\mathbf{C}$,$\mathbf{D}$*));*

The resultant state space realization of the system is:

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{4.30}$$

$$\mathbf{B} = \begin{bmatrix} 0 & -0.3881 & 0 & 0.3881 \\ 0.3881 & 0 & -0.3881 & 0 \\ 0.0174 & -0.0174 & 0.0174 & -0.0174 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.31}$$

$$\mathbf{C} = \begin{bmatrix} 0 & 0 & 0 & 0 & -19.62 & 0 \\ 0 & 0 & 0 & 19.62 & 0 & 0 \\ 0 & 0 & 0 & -0.001 & -0.001 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.32}$$

where matrix $\mathbf{D}$ suffers no changes from equation (4.29). This realization also has no reachable states, but is fully controllable and observable. After close observation we can also see that the eliminated states are the linear velocities and linear positions, leaving us with the attitude of the quadrotor and angular velocities. The linear positions are no surprise due to the absence of position sensors. About the linear velocities, would be possible to extract them by integration of the accelerations? Well, this is something we can not expect to do with current filtering techniques because by integrating the accelerations we are also doing the same to the noise associated with it, thus the linear velocities drift.

The state space form of this system can be implemented in MATLAB® using the command:

sys = ss($\mathbf{A}$*,[*$\mathbf{B}$ $\mathbf{G}$*],*$\mathbf{C}$*,[*$\mathbf{D}$ $\mathbf{H}$*]);*

Where the matrices $\mathbf{G}$ and $\mathbf{H}$ are defined as

$\mathbf{G}$ *= eye(12,6);*
$\mathbf{H}$ *= zeros(6,6);*

Matrix $\mathbf{G}$ basically says that we can make measurements on only 6 states (in a perfect situation where all 12 states measured, $\mathbf{G}$ would be an identity matrix of size $[12 \times 12]$). Matrix $\mathbf{H}$ is defined as null because we are assuming the process noise has no influence in the system outputs (see equation (4.13)). As for the covariance matrices, we assume the noise covariances of both the system inputs and sensor readings are the same (because we do not know its real values):

$$\overline{\mathbf{Q}} = 10^{-4}\underline{\mathbf{I}}(6) \tag{4.33}$$

$$\overline{\mathbf{R}} = 10^{-4}\underline{\mathbf{I}}(6) \tag{4.34}$$

Notice that $\overline{\mathbf{R}}$ is a $[6 \times 6]$ matrix, because we can only measure 6 outputs (3 accelerations and the Euler angles yaw, pitch and roll). Also, the process noise covariance matrix $\overline{\mathbf{Q}}$ has size $[6 \times 6]$ because we are working only with the 6 controllable and observable states of the system.

When designing a Kalman filter, we can also choose from two possible types of estimators: continuous-time or discrete-time. As we plan to use the Kalman estimator on the Arduino (a digital system), we must choose the discrete-time option, meaning we have to use another very important parameter, the sample time t_s. In MATLAB® the discrete Kalman filter is obtained with:

Kest = kalmd(sys,$\overline{\mathbf{Q}}$*,*$\overline{\mathbf{R}}$*,*t_s*);*

where Kest provides contains the filter matrices $\mathbf{A}^k$, $\mathbf{B}^k$ $\mathbf{C}^k$ and $\mathbf{D}^k$.

Trying to produce a discrete Kalman filter with the current state space matrices and covariance matrices results in the following state space realization of the filter:

$$\mathbf{A}^k = \begin{bmatrix} 1 & 0 & 0 & -0.5985 & 0 & 0 \\ 0 & 1 & 0 & 0 & -0.5985 & 0 \\ 0 & 0 & 1 & 0 & 0 & -0.0479 \\ 0.05 & 0 & 0 & 0.3384 & 0 & 0 \\ 0 & 0.05 & 0 & 0 & 0.3384 & 0 \\ 0 & 0 & 0.05 & 0 & 0 & 0.9147 \end{bmatrix} \tag{4.35}$$

$$\mathbf{B}^k = \begin{bmatrix} 0 & -0.0188 & 0 & 0.0188 & 0 & 0.0302 & 0 & 0.0031 & 0 & 0 \\ 0.0194 & 0.0006 & -0.0194 & -0.0006 & -0.0302 & 0 & 0 & 0 & 0.0031 & 0 \\ 0.0009 & -0.0009 & 0.0009 & -0.0009 & 0 & 0 & 0 & 0 & 0 & 0.0479 \\ 0 & 0.0002 & 0 & -0.0002 & 0 & 0.0334 & 0 & 0.0034 & 0 & 0 \\ 0.0005 & 0.0006 & -0.0005 & -0.0006 & -0.0334 & 0 & 0 & 0 & 0.0034 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.0853 \end{bmatrix} \tag{4.36}$$

$$\mathbf{C}^k = \begin{bmatrix} 0 & 0 & 0 & 0 & -7.2272 & 0 \\ 0 & 0 & 0 & 7.2272 & 0 & 0 \\ 0 & 0 & 0 & -0.0004 & -0.0004 & 0 \\ 0 & 0 & 0 & 0.7367 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.7367 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.917 \\ 1 & 0 & 0 & -0.5985 & 0 & 0 \\ 0 & 1 & 0 & 0 & -0.5985 & 0 \\ 0 & 0 & 1 & 0 & 0 & -0.0479 \\ 0 & 0 & 0 & 0.3684 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.3684 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.917 \end{bmatrix} \tag{4.37}$$

$$\mathbf{D}^k = \begin{bmatrix} 0 & 0.007 & 0 & -0.007 & 0.6251 & 0 & 0 & 0 & 0 & -0.0637 \\ 0 & -0.007 & 0 & 0.007 & 0 & 0.6251 & 0 & 0.0637 & 0.0637 & 0 \\ -0.0144 & -0.0132 & -0.0121 & -0.0132 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -0.0007 & 0 & 0.0007 & 0 & 0.0637 & 0 & 0.0065 & 0.0065 & 0 \\ 0 & -0.0007 & 0 & 0.0007 & -0.0637 & 0 & 0 & 0 & 0 & 0.0065 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.0006 & 0 & -0.0006 & 0 & 0.0302 & 0 & 0.0031 & 0.0031 & 0 \\ 0 & 0.0006 & 0 & -0.0006 & -0.0302 & 0 & 0 & 0 & 0 & 0.0031 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.0006 & 0 & -0.0006 & 0 & 0.0319 & 0 & 0.0032 & 0.0032 & 0 \\ 0 & 0.0006 & 0 & -0.0006 & -0.0319 & 0 & 0 & 0 & 0 & 0.0032 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.38}$$

Now that we have our state space Kalman filter we can implement it in Simulink® using the block disposition in figure 4.4.1, through a discrete state space block.

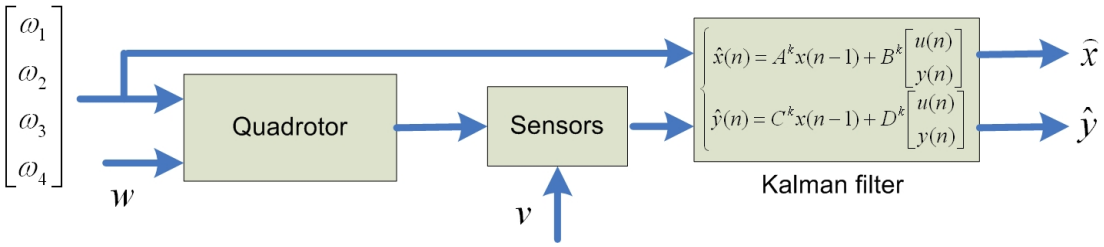


Figure 4.4.1: Kalman filter schematic.

4.5 Kalman filter for Arduino

Implementation of a state space Kalman filter in the Arduino is quite easy, mostly because we only need matrices $\mathbf{A}^k$ and $\mathbf{B}^k$ to estimate the states $\begin{bmatrix} P & Q & R & \phi & \theta & \psi \end{bmatrix}$. The equation we are going to use for this purpose is:

$$\mathbf{x}(k) = \mathbf{A^k x}(k-1) + \mathbf{B}^k \mathbf{u}(k) \tag{4.39}$$

where we estimate the states at time instant k are calculated by using the states at the time instant $k-1$ and inputs at the current time instant k. Constantly trying out different state space matrices in the Arduino can be a dilatory work. To ease this task a MATLAB® function was developed to convert any matrix variable to Arduino code ready for copy/paste operations. We also need to remember the necessity of providing the initial states $\mathbf{x}(0)$ for the filter's startup calculation. In our case we are going to assume the quadrotor is leveled and heading north, which makes us assume the initial state vector as:

$$\mathbf{x}(0) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.40}$$

The estimated states will be used to feed a LQR controller, which will be the main subject of the next chapter.

Chapter 5

Optimal Control - The Linear-Quadratic Regulator

In optimal control one endeavors on finding a controller that provides the best possible performance with respect to some given measure of performance (e.g. the controller that uses the least amount of control-signal energy to take the output to zero). In this case the measure of performance (also called optimal criterion) would be the control-signal energy. In this section we try to implement optimal control through the use of a LQR controller (Linear-Quadratic Regulator) in which the control signal energy is measured by a cost function containing weighting factors provided by the controller designer.

Moreover, the LQR controller is applicable to MIMO (Multiple-Input Multiple-Output) systems (e.g. in the quadrotor we have the speeds of 4 motors as inputs, and the sensor readings as outputs) for which classical designs are difficult to apply (see reference [30]).

5.1 Mathematical formulation

For a continuous-time linear system described by:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{5.1}$$

The cost function J_{LQR} is [29]:

$$J_{LQR} = \int_0^\infty \left(\mathbf{x}^T \hat{\mathbf{Q}} \mathbf{x} + \mathbf{u}^T \hat{\mathbf{R}} \mathbf{u}\right) dt \tag{5.2}$$

where $\hat{\mathbf{Q}}$ (size $n \times n$, where n corresponds to the number of states to control) and $\hat{\mathbf{R}}$ (size $m \times m$, with m the number of process inputs) are both symmetric positive-definite weighting matrices. Consequently, the equivalent discrete-time feedback control law (note that the controller is to be implemented in the Arduino, which by it self is a digital platform, hence it is not a continuous-time system) that minimizes the cost value is:

$$\mathbf{u}^K = -\overline{\mathbf{K}}\mathbf{x} \tag{5.3}$$

where $\mathbf{u}^K$ is the vector of control actions. The LQR gain matrix $\overline{\mathbf{K}}$ is provided by:

$$\overline{\mathbf{K}} = \hat{\mathbf{R}}^{-1}\mathbf{B}^T\hat{\mathbf{P}} \tag{5.4}$$

and $\hat{\mathbf{P}}$ is derived by means of the algebraic Riccati equation:

$$\mathbf{A}^T\hat{\mathbf{P}} + \hat{\mathbf{P}}\mathbf{A} - \hat{\mathbf{P}}\mathbf{B}\hat{\mathbf{R}}^{-1}\mathbf{B}^T\hat{\mathbf{P}} + \hat{\mathbf{Q}} = 0 \tag{5.5}$$

The LQR design technique presented so far assumes the order of the controller will be equal to the order of the system. This may not be our case. In fact, because, as we have seen before, we have only 6 states available to control from the Kalman filter (angular velocities and Euler angles), the order of our controller will be lower than the order of the system. This suggests that we have two possible case studies for the LQR controller: 12 states control and 6 states control. The first one assumes we have no Kalman filter and have full access to all states, as for the second we will be interested in testing the performance of the controller using the Kalman filter. This implies respectively either we use a $[12 \times 12]$ matrix $\mathbf{A}$ and $[12 \times 6]$ matrix $\mathbf{B}$ (for 12 state control) as opposite to the lower size $\mathbf{A}$ and $\mathbf{B}$ matrices when using Kalman filtering (see section 4.4). As for matrix $\hat{\mathbf{Q}}$ we will have to assume the same logic respectively for 12 or 6 states control, where it will have different size accordingly to each case.

A first choice for matrices $\hat{\mathbf{Q}}$ and $\hat{\mathbf{R}}$ is provided by the Bryson's rule which states these matrices as being diagonal:

$$\hat{Q_k} = \frac{1}{x_{i,max}^2} \tag{5.6}$$

$$\hat{R_k} = \frac{1}{u_{i,max}^2} \tag{5.7}$$

where $x_{i,max}$ is the highest tolerable value for the state x_i, and $u_{i,max}$ is the highest tolerable value for the input u_i. The core objective of Bryson's rule is therefore to scale the variables in the J_{LQR} so that the highest tolerable value for each term is one. Particularly, this is very important when $\mathbf{x}$ and $\mathbf{u}$ are numerically very different from each other [31].

5.2 LQR control in MATLAB® and Simulink®

A discrete LQR controller can be produced in MATLAB® using the command:

Klqr=lqrd($\mathbf{A}$,$\mathbf{B}$,$\hat{\mathbf{Q}}$,$\hat{\mathbf{R}}$,t_s*);*

which will return the gain matrix $\overline{\mathbf{K}}$, according to the sample time t_s. Tables 5.1 and 5.2 present the values chosen to building the weight matrices $\hat{\mathbf{Q}}$ and $\hat{\mathbf{R}}$. The values presented in these tables are not very easy to choose. In fact, they were sought iteratively, showing that the implementation of Bryson's rule is not very intuitive, and does not guarantee a working controller on the first try.

Table 5.1: Maximum state values for designing matrix $\hat{\mathbf{Q}}$.

States	U_{max}	V_{max}	W_{max}	P_{max}	Q_{max}	R_{max}	X_{max}	Y_{max}	Z_{max}	ϕ_{max}	θ_{max}	ψ_{max}
12	0.1	0.1	0.1	0.001	0.001	0.001	0.4	0.4	0.4	0.1	0.1	0.1
6	-	-	-	1	1	1	-	-	-	0.1	0.1	0.1

Table 5.2: Maximum input values for designing matrix $\hat{\mathbf{R}}$.

	$\omega_{1\,max}$	$\omega_{2\,max}$	$\omega_{3\,max}$	$\omega_{4\,max}$
12 states control	3	3	3	3
6 states control	3	3	3	3

Now all we have to do is to build matrices $\hat{\mathbf{Q}}$ and $\hat{\mathbf{R}}$ by appropriately using equations (5.6) and (5.7). The following gain matrices are obtained for each control scenario of table 5.1 by using a sample time of $0.05s$ $(20Hz)$:

$$\overline{\mathbf{K}}_{12\,\text{states}} = \begin{bmatrix} -0.6649 & -0.0001 & -18.7694 & 0.0001 & 32.9579 & 338.8566 \\ 0.0001 & -0.6652 & -18.7694 & -32.9577 & 0.0001 & -338.8566 \\ 0.6652 & -0.0001 & -18.7694 & 0.0001 & -32.9577 & 338.8566 \\ 0.0001 & 0.6649 & -18.7694 & 32.9579 & 0.0001 & -338.8566 \end{bmatrix}$$

$$\begin{bmatrix} -0.0803 & 0 & -3.6567 & 0.0057 & 20.6969 & 3.3871 \\ 0 & -0.0804 & -3.6567 & -20.6855 & 0.0057 & -3.3871 \\ 0.0804 & 0 & -3.6567 & 0.0057 & -20.6855 & 3.3871 \\ 0 & 0.0803 & -3.6567 & 20.6969 & 0.0057 & -3.3871 \end{bmatrix} \tag{5.8}$$

$$\overline{\mathbf{K}}_{6\,\text{states}} = \begin{bmatrix} 0 & 7.103 & 20.4163 & 0 & 18.2839 & 14.4659 \\ -7.103 & 0 & -20.4163 & -18.2839 & 0 & -14.4659 \\ 0 & -7.103 & 20.4163 & 0 & -18.2839 & 14.4659 \\ 7.103 & 0 & -20.4163 & 18.2839 & 0 & -14.4659 \end{bmatrix} \tag{5.9}$$

For simulation purposes, a LQR controller can be implemented in Simulink® with a simple gain block as schematically portrayed in figure 5.2.1. When a LQR controller is combined with a Kalman filter, they form a LQG (Linear-Quadratic-Gaussian) controller (see Figure 5.2.2). Both elements of an LQG controller can be designed and computed independently, as in this case.

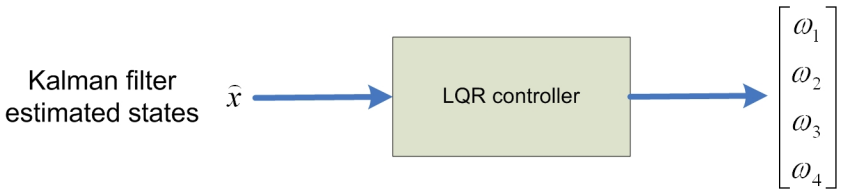


Figure 5.2.1: LQR control.

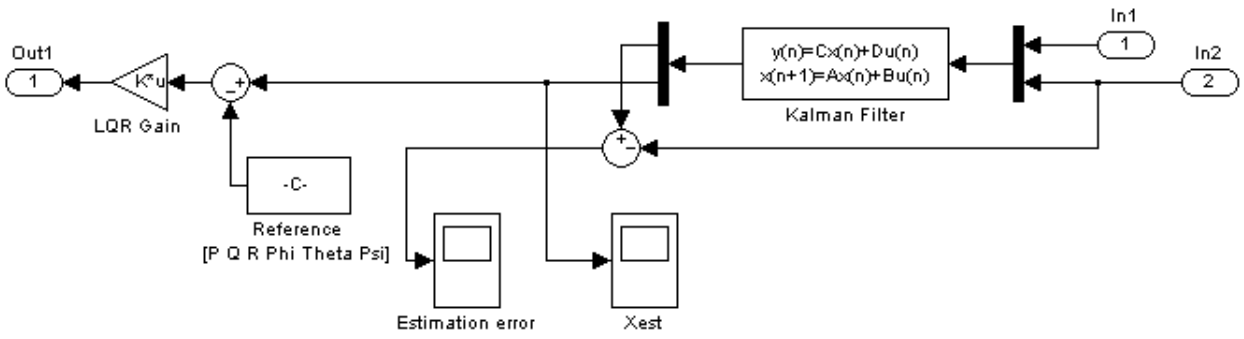


Figure 5.2.2: LQG control block diagram in Simulink®.

5.3 LQR control in the Arduino

The controller implementation in the Arduino is quite easy. We only have to define the gain matrix $\overline{\mathbf{K}}$ in the Arduino program environment and multiply it by the estimated states produced by the Kalman filter as in equation (4.39). We have to be careful as the control actions of the LQR controller are the angular speeds of the motors. This value has to be converted back into a PWM signal. We can achieve this goal by using data in equation (3.38) to convert angular speeds to pulse widths and compensating with the respective motors' dead zone:

$$\mathcal{P} = \frac{\omega}{K} + \mathcal{P}_0 \tag{5.10}$$

where $\mathcal{P}_0$ is the pulse width of a motor's dead zone, K the gain of its first order transfer function, ω the angular speed (from the LQR controller) and $\mathcal{P}$ is the consequent pulse width to drive the motor.

Chapter 6

Implementation and results

This chapter shows the evolution of our attempts to control the quadrotor system. The addressed case studies are:

- 12 states LQR control. Starting from the ideal case, we will study the situation where a LQR controller handles all 12 states of the system. This will show us the best scenario possible with this kind of controller.

- LQG control with sensors. Although we would like to have all 12 states of the quadrotor system at our disposal, this is not always possible. In this case, a Kalman filter is used to estimate three angular velocities and three Euler angles from the sensors outputs. These six states are afterwards fed to a LQR controller.

- LQG control with sensors and motors dynamics. This case is the continuation of the previous one, but this time the controller robustness is put to the test when the simulation includes the dynamics of the motor-propeller assemblies.

- Practical implementation. Taking our controller and Kalman filter into our quadrotor prototype is the next logical step, where we can test how the LQG controller performs in the real world.

- Using gyroscopes to improve state estimation. Sometimes the intensity of sensor noise is underestimated in simulation environments. In this case study, the use of gyroscopes (together with the tri-axis accelerometer and compass) is evaluated as a means of providing extra information to the Kalman filter and observe what is its effect on the quality of the state estimates (which accuracy is crucial for system control).

6.1 Full 12 states control

One of the most important lessons to be drawn from the process of simulating any control loop is that it is unwise to start immediately with the most complex situation where we have the system as complete as possible. This case is often more difficult to control, which in case of the emergence of some unexpected results (e.g. the apparent inability to stabilize a given system in the simulation environment) can make the detection of any implementation errors very hard to detect. There is nothing better than to start with a simple system and gradually progress to its most complete (and complex) form.

Following this philosophy, we shall start with by analyzing whether or not it is possible to control our system using only the dynamics of quadrotor together with the LQR controller. This case is considered to be ideal because we are assuming that all 12 states of the system are available for control purposes, which does not always correspond to a real situation. The control loop in Simulink used for this task is presented in Figure 6.1.1, where we can see that the LQR controller (in orange) receives the error between the actual states and the desired ones, and produces four angular speeds (one for each motor) that will make the quadrotor follow a given reference.

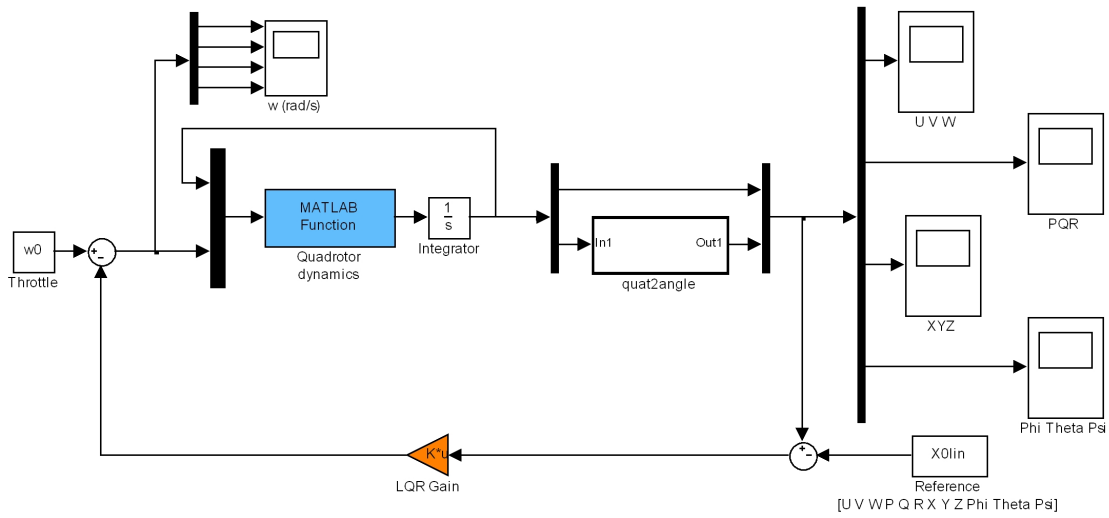


Figure 6.1.1: 12 states LQR control loop.

Knowing that this control loop uses the controller in equation (6.1.1), we also have to specify a reference for our system in order to analyze the performance of this controller. Taking into account that we want to take the aircraft from a situation where it is landed on the ground, with the initial state vector equal to $\mathbf{x} = [0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0]$ to a reference $\mathbf{x} = [0\,0\,0\,0\,0\,0\,1\ -1\ -1\,0\,0\,0]$ (i.e. one meter up, one meter West and one meter North). Figure 6.1.2 illustrates the results from the controller performance using these references.

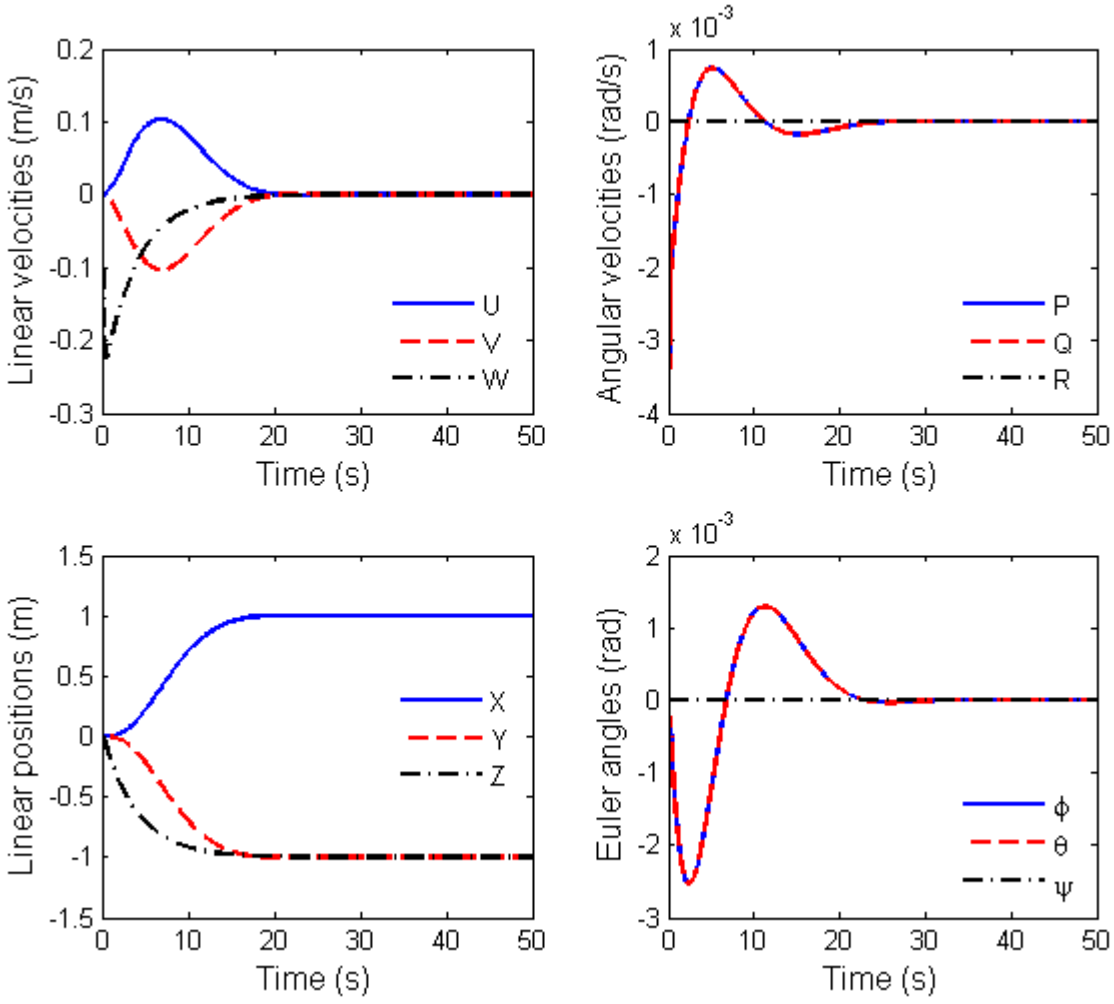


Figure 6.1.2: Time response of the LQR control loop using with 12 states feedback.

We can see that with this controller, the system can follow the reference without any problems, starting with Take-Off and then moving Northeast while always keeping the North heading. The time response in terms of the pitch and roll angles is the same. Figures 6.1.3 and 6.1.4 show further analysis on this controller, allowing us to see what happens when we tune the $\hat{\mathbf{Q}}$ and $\hat{\mathbf{R}}$ matrices.

Figure 6.1.3 illustrates that increasing the value of the weighting matrix $\hat{\mathbf{Q}}$ also augments the system speed of response, while also making it a little bit under damped. On the other hand, Figure 6.1.4 reflects that an increase in matrix $\hat{\mathbf{R}}$ leads to a decrease in the control action of the motors, as it also appears to have a significant influence only in the position of the quadrotor along u_Z.

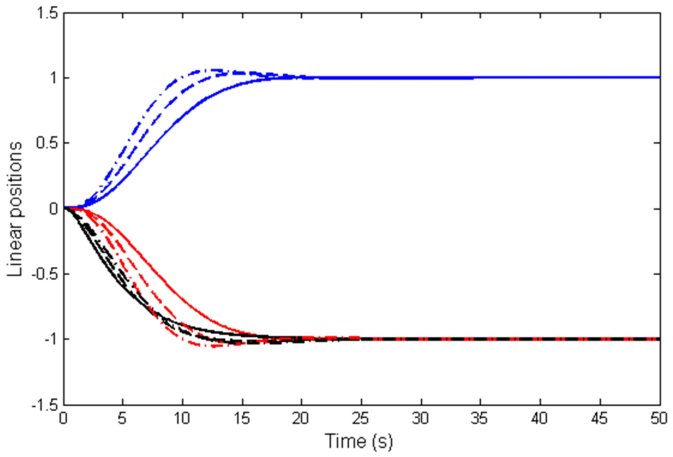


Figure 6.1.3: Linear positions with increasing weight matrix $\hat{\mathbf{Q}}$.
(weight matrices $\hat{\mathbf{Q}}_1$>$\hat{\mathbf{Q}}_2$>$\hat{\mathbf{Q}}_3$: - roll $\hat{\mathbf{Q}}_1$; - - roll $\hat{\mathbf{Q}}_2$; -. roll $\hat{\mathbf{Q}}_3$; - pitch $\hat{\mathbf{Q}}_1$; - - pitch $\hat{\mathbf{Q}}_2$; -. pitch $\hat{\mathbf{Q}}_3$; - yaw $\hat{\mathbf{Q}}_1$; - - yaw $\hat{\mathbf{Q}}_2$; -. yaw $\hat{\mathbf{Q}}_3$)

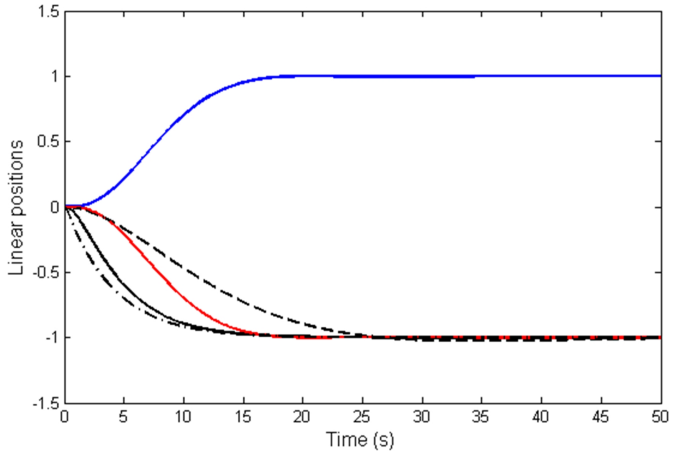


Figure 6.1.4: Linear positions with increasing weight matrix $\hat{\mathbf{R}}$.
(weight matrices $\hat{\mathbf{R}}_1$>$\hat{\mathbf{R}}_2$>$\hat{\mathbf{R}}_3$: - roll $\hat{\mathbf{R}}_1$, $\hat{\mathbf{R}}_2$ and $\hat{\mathbf{R}}_3$; - pitch $\hat{\mathbf{R}}_1$, $\hat{\mathbf{R}}_2$ and $\hat{\mathbf{R}}_3$; - yaw $\hat{\mathbf{R}}_2$; - - yaw $\hat{\mathbf{R}}_3$; -. yaw $\hat{\mathbf{R}}_1$)

6.2 6 states control

6.2.1 LQG control with sensors

After our analysis on the design of the Kalman estimator for our system (see Chapter 4) we concluded that we can only work with 6 of the 12 states available from our system, the angular velocities and and the Euler angles. It is therefore appropriate to investigate the behavior of our system in the presence of the LQR controller in equation (5.9), which is designed to control only these 6 states.

The 6 states control loop is displayed in Figures 6.2.1 (Simulink®) and 6.2.2, where our state vector is $\mathbf{x} = \begin{bmatrix} P & Q & R & \phi & \theta & \psi \end{bmatrix}$ and our reference is $\mathbf{x} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & \pi/4 \end{bmatrix}$ (i.e. we want to keep the quadrotor leveled and rotated at an angle $\psi = 0.7854$ radians, or $45°$).

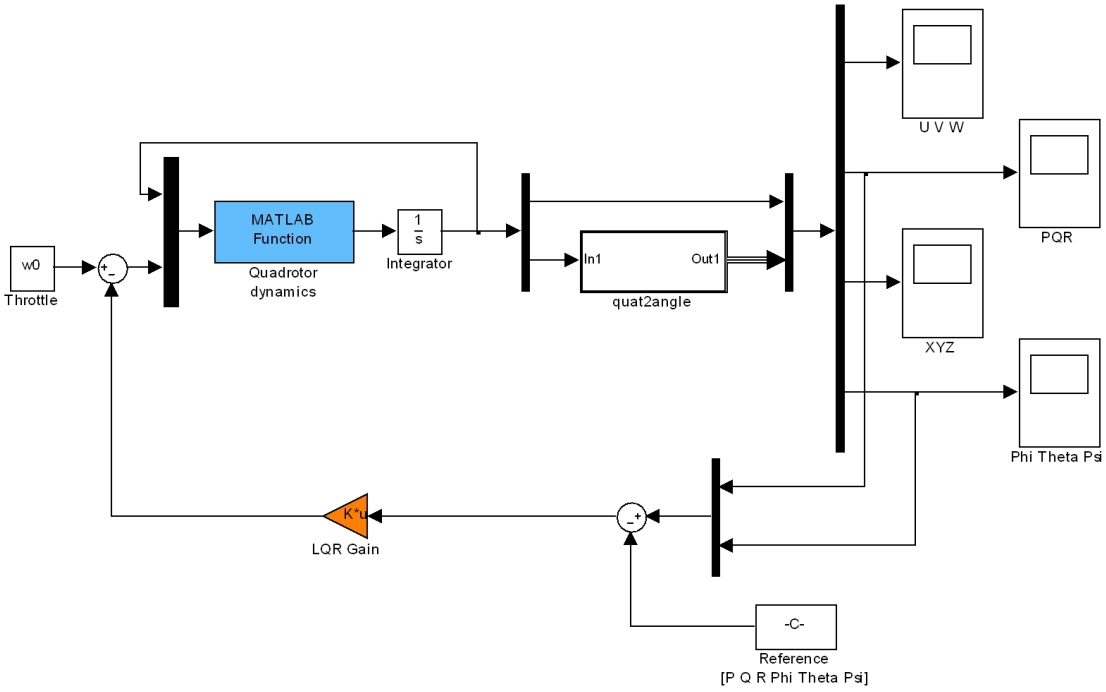


Figure 6.2.1: 6 states Simulink® LQR control loop.

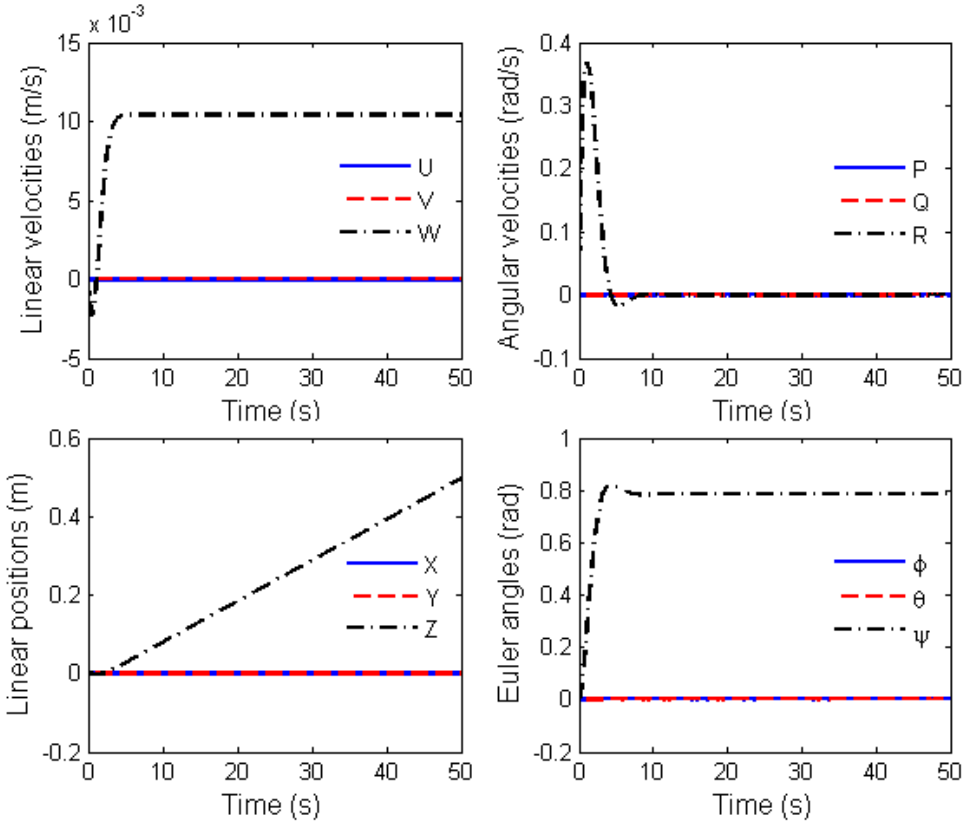


Figure 6.2.2: Time response of the LQR control loop using 6 states.

The Figure above shows that the system follows the reference reasonably fast (less than 10 seconds). At the same time we can see that both the linear velocities and linear positions are not under the influence of the controller, since they are not controllable states. In a real case, if we would like to control the altitude, we would have to do it ourselves only by controlling the power provided to the motors. This would not be necessary if, for example, we had an altitude sensor.

Figure 6.2.3 illustrates Simulink block diagram of the next stage of simulations, where we include the sensors (with noise) in the control loop to see how the LQR control performs (see Figure 6.2.4). The contents of the LQG block of Figure 6.2.3 are also presented in Figure 5.2.2.

We can see (in Figure 6.2.4, where the Euler angles are now graphically represented in degrees to improve readability) that the LQR controller continues to render the system stable, without any need of adjustments to the Q and R matrices (note that the pitch and yaw angles are overlapped).

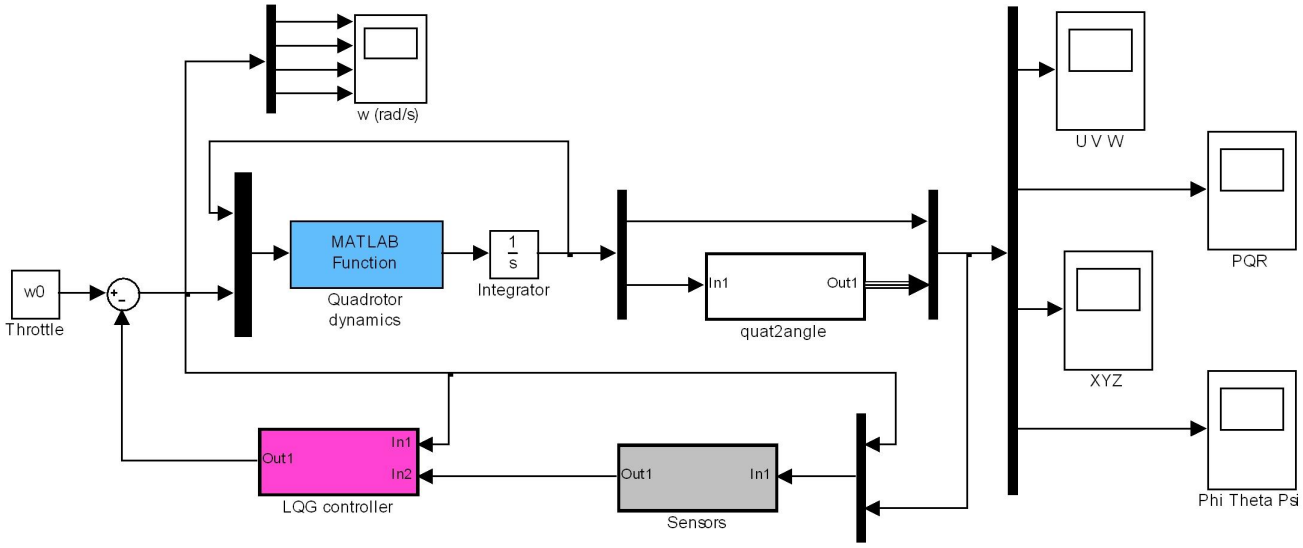


Figure 6.2.3: 6 states Simulink® LQR control loop with sensors.

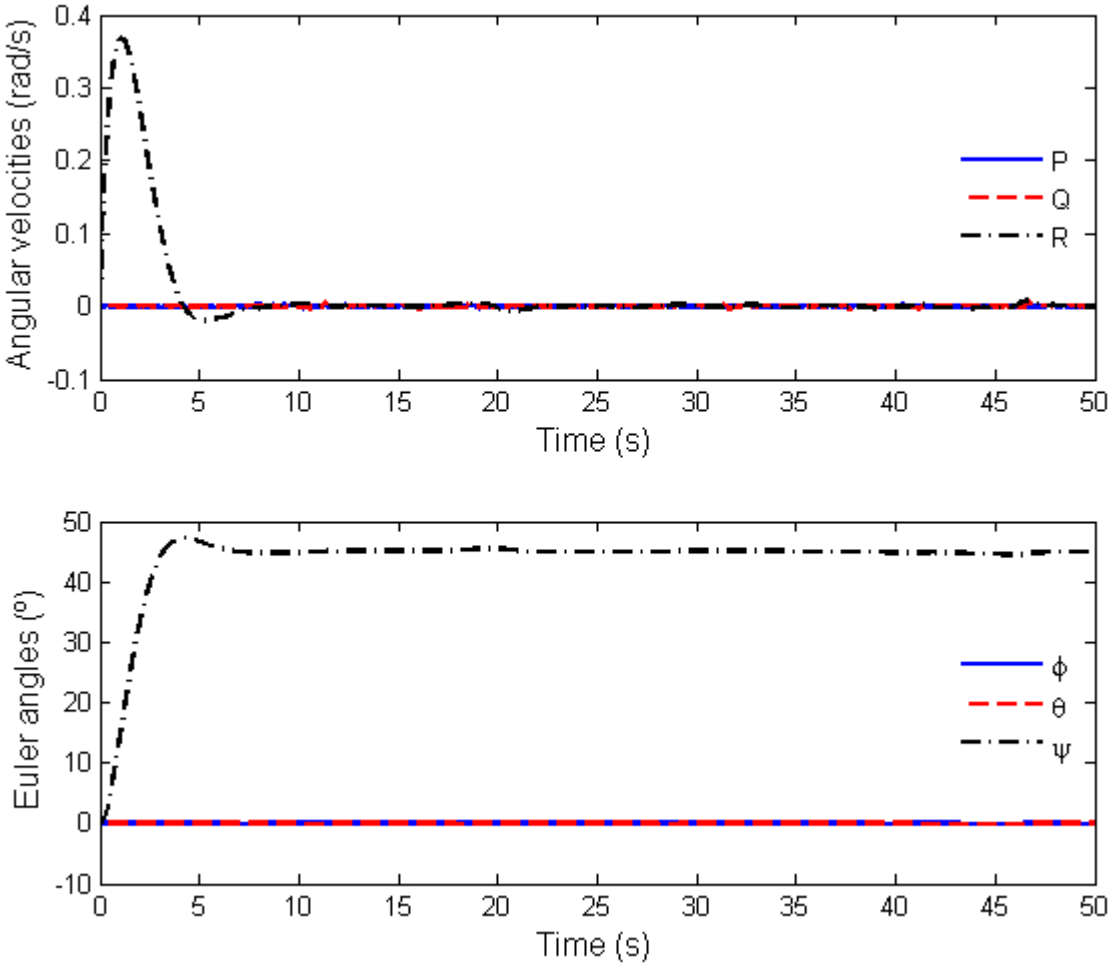


Figure 6.2.4: Time response of the LQR control loop using 6 states and sensors.

6.2.2 LQG control with sensors and motors dynamics

In this section we will observe what is the effect produced by the introduction of the motors dynamics in the control loop (see Figure 6.2.5), and check if the LQR controller continues to maintain the quadrotor leveled. The reference used in this case continues to be $\mathbf{x} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & \pi/4 \end{bmatrix}$ and the simulation results are available in Figure 6.2.6.

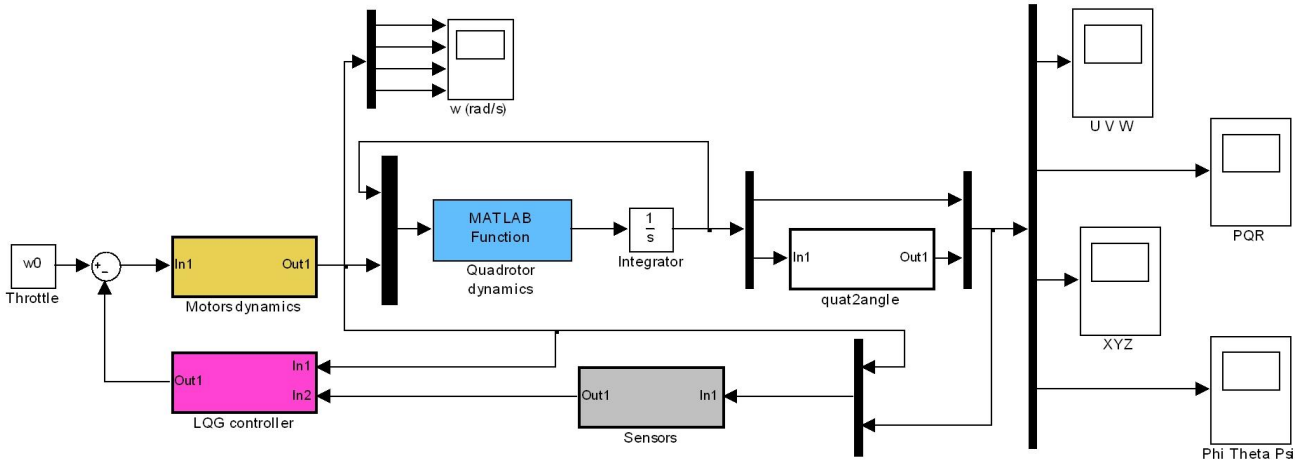


Figure 6.2.5: 6 states Simulink® LQR control loop with sensors and motors dynamics.

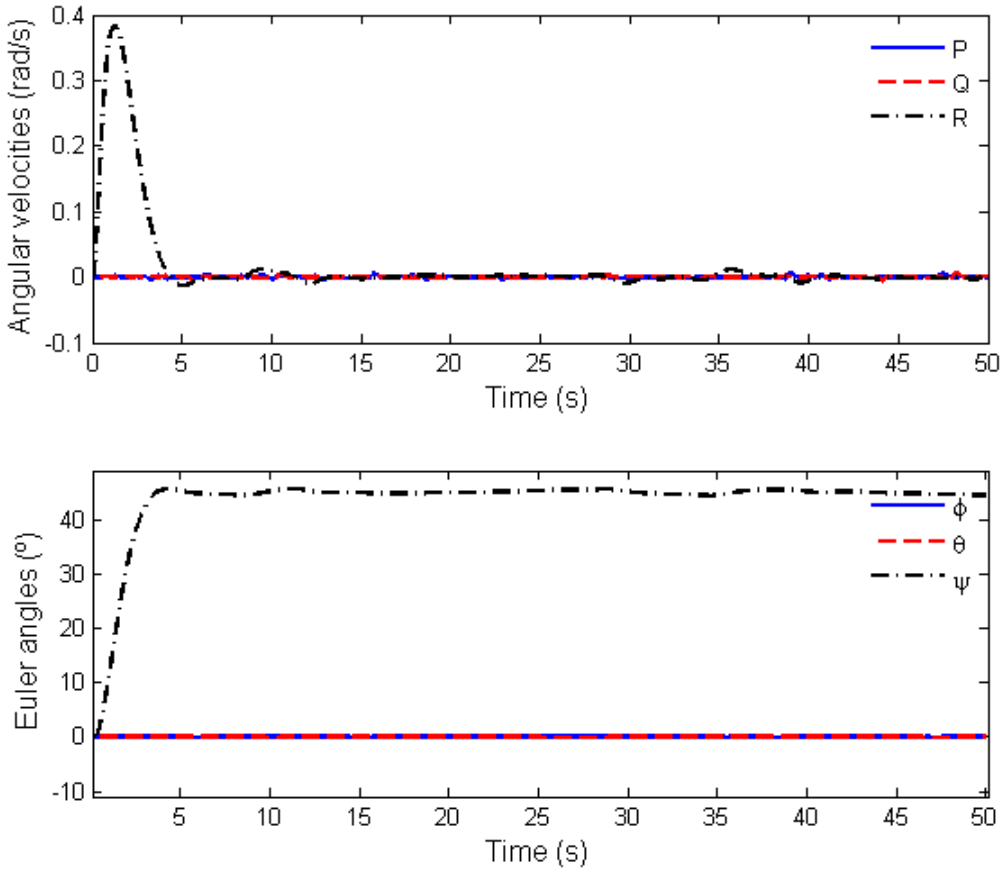


Figure 6.2.6: Time response of the LQR control loop using 6 states, sensors and motors dynamics.

We continue to verify that the system remains stable, noting that the time response is very similar to the one where the motors dynamics had not yet been included (see Figure 6.2.4). This effect can only be explained by the introduction of the motors dynamics, which despite not being considered in the controller design, continues to be dominated by it.

6.3 Practical implementation

Now that we have proved that it is possible to stabilize the quadrotor in a simulated environment with the current combination of sensors, we will try to validate our results through the implementation of both the LQR controller and Kalman filter in our quadrotor prototype (see Figure 6.3.1), and see if we can have control over its attitude.

Figure 6.3.1: Quadrotor prototype.

6.3.1 Telemetry and joystick control software

In this phase of our work we can expect to be necessary the development of a software module which can collect information from the Arduino and, if necessary, send control orders to it. With this objective in mind, a small software module was built to assist us. This application is designated by Quadjoy and was designed to collect not only ASCII text messages sent by the Arduino, as it can also send commands from a game controller that has 2 joysticks, which together have a sufficient number axis to control the Euler angles and power supplied to the motors (i.e. we need four axes, three for the Euler angles and 1 for motors power). The application interface is visible in Figure 6.3.2 and the control scheme in Figure 6.3.3.

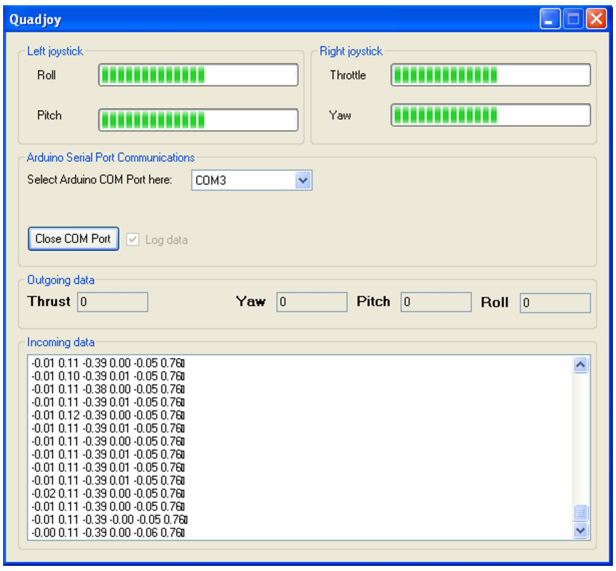


Figure 6.3.2: Quadjoy user interface.

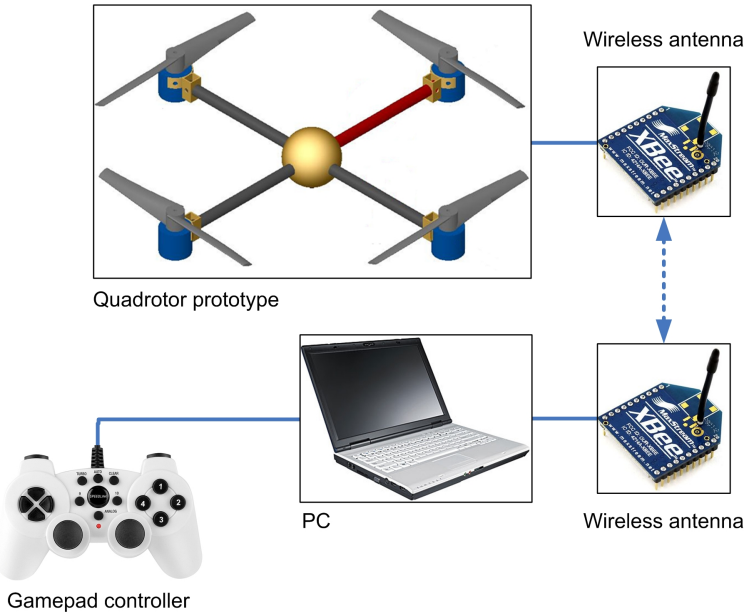


Figure 6.3.3: Quadrotor control diagram.

The presented interface has been planned to be as intuitive as possible, also allowing us to acquire data to a file that can be loaded in Matlab®. As an example for a session of data acquisition, we initiate communications with the Arduino by first choosing the COM port that is connected to wireless antenna, and then selecting the checkbox that indicates whether we want to write the collected data to a file. Finally, we have to press the button “Open” to open the COM port and start giving orders to the quadrotor using our joystick. After we finished our flying session we close the COM port and find out that the program directory will now have a data file named *qdata.txt*. To load the data contained in this file in Matlab®, we simply use the command:

load qdata.txt

6.3.2 Kalman filter and LQR controller

The first phase of prototype tests consists in verifying the behavior of the Kalman filter. The code that was developed to implement the Kalman filter and LQR controller in the Arduino can be found in Appendix C. This test consists in grabbing the quadrotor with both hands and tilting it around the roll, pitch and yaw axis to ensure that it is providing an acceptable estimation of the 6 states wanted. This experience went fairly well, showing that the Kalman filter can estimate the desired 6 state. As an example, Figure 6.3.4 illustrates one of those situations where we can see the estimation of the yaw angle when the quadrotor is rotated 45 degrees from North to West.

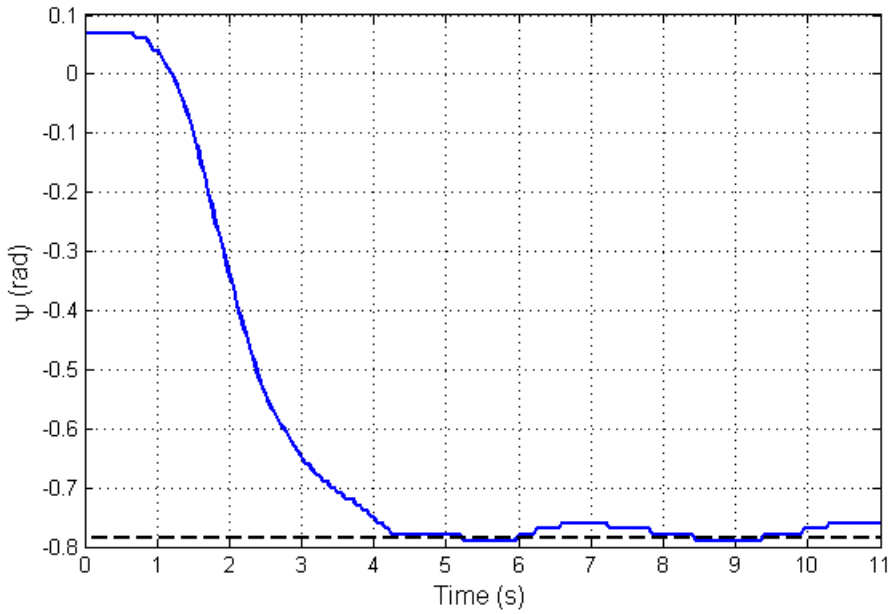


Figure 6.3.4: Kalman filter estimation of the yaw angle in the quadrotor prototype.
(- Yaw angle; - Yaw angle reference)

Knowing that the Kalman filter seems to be functional, we proceed to the stability test of our quadrotor using the LQR controller. The intended reference to follow is $\mathbf{x} = [\ 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0\ ]$. Thus this reference was programmed in the Arduino and, for safety reasons, the quadrotor was placed at a safe distance from any physical obstacle. Then, with the help of the joystick, power was provided to the motors so that the aircraft was able to gain altitude (note that the code implemented in the Arduino already includes a security system that stops the motors in the absence of a signal from the joystick).

It was observed that the quadrotor, just before reaching Take-Off, began to slowly spin around the u_z axis, suggesting that the control of the yaw angle was not functioning as expected. Giving more power to the motors resulted in an uncontrolled flight. It is therefore preferable to investigate what is the cause of this instability. We started by verifying the orders that the controller was giving to the motors. For this purpose, the quadrotor was lifted into the air, and without giving power to the motors, we tried to check if the control orders corresponded to those shown in Figure 1.2.2 of section 1.2. Based on this Figure, if for

example, in a situation where the quadrotor is leveled but with $\psi = -\pi/4\, rad$, It is therefore expected that motors two and four have higher angular speed than motors one and three in order for the quadrotor to reach our reference. This situation was indeed verified, however it was observed that the changes in the angular speed of the motors (produced by the controller) did not exceed two radians per second, which leads us to believe that a there is an important variable that was not taken into account: the resolution of the PWM signal. This means that, for example, if the motors are not sensitive to variations of 2 radians per second and the controller only issues control orders which do not vary by more than 1 radian per second (which is lower than the resolution of the PWM signal) then control over the quadrotor might be very difficult. We need to check if the controller is aggressive enough to deal with this new aspect, and if necessary, make adjustments to its weighting matrices. After consulting the resolution of the servo signal of the Arduino, it was concluded that it is only sensitive to changes of 4 microseconds. This new feature was introduced in the Simulink® block of the motors and, according to the results observed in Figure 6.3.5, we can still see that the controller can keep the system stable, although there are now wide fluctuations in the pitch and yaw angles.

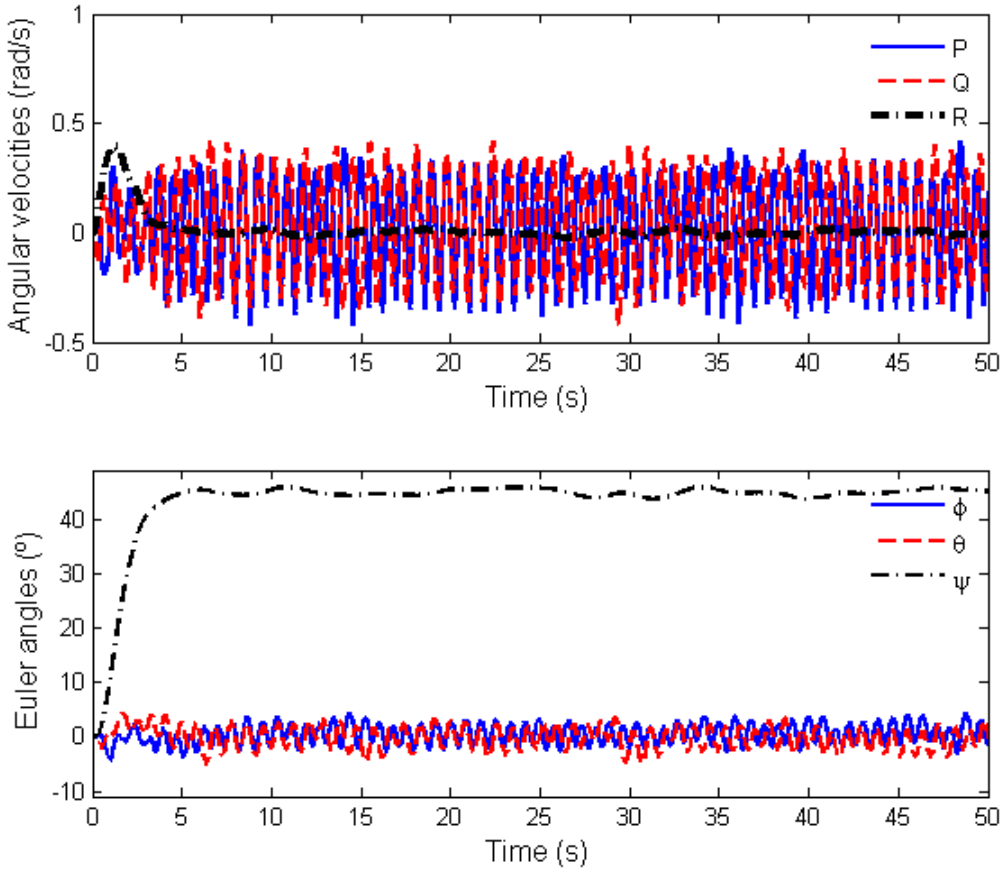


Figure 6.3.5: Time response of the LQR control loop using 6 states, sensors and motors dynamics with PWM resolution.

This new data means we need to make the following question: If by manipulation of the quadrotor

attitude with the motors turned off we have control orders and estimated states that appear to be framed within values that seem appropriate to the stability scenario we wish to have, then what is happening when the motors are working that leads to instability? The Arduino code was therefore changed to send us data directly from the sensors in a situation where the motors were working. Figure 6.3.6 shows the accelerometer data collected, where we can see the noise appearing 4 seconds from the start of data collation, moment at which the motors started to work, producing vibrations across the structure of the quadrotor.

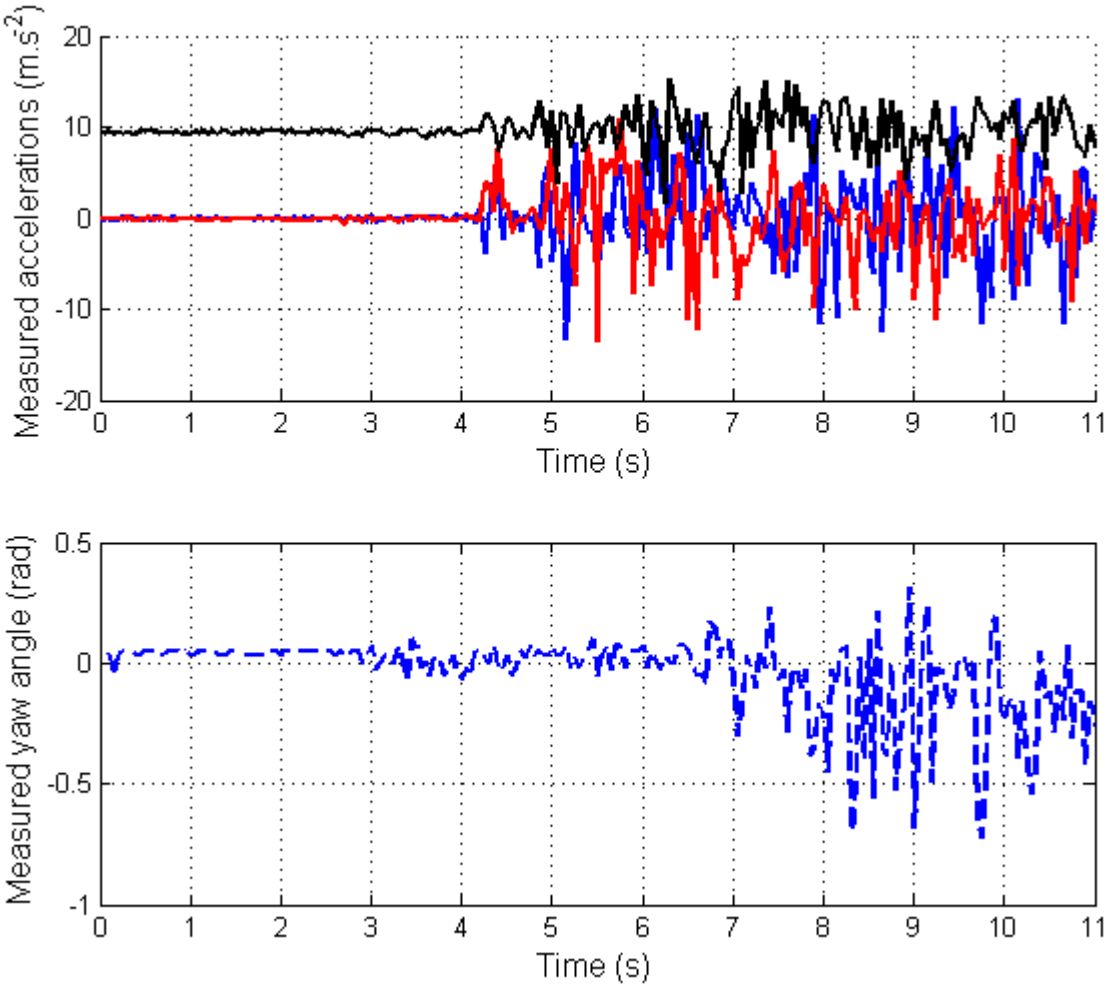


Figure 6.3.6: Sensor noise data.
(- x-axis acceleration; - y-axis acceleration; - z-axis acceleration; – compass yaw angle)

At this moment we can only say our biggest problem is noise (as shown by the Figure above). This is particularly serious for the accelerometer, whose data is used to estimate the pitch and yaw angles. To try to reduce the noise coming from the sensors, a new Kalman filter was designed with a covariance matrix $\overline{\mathbf{R}}$ calculated from the collected noise data. Consequently, the noise was also introduced in the simulation block of sensors in order to better portray reality. The estimation error proved to be so high that it was impossible to find a functional LQR controller for the system. Even the addition of a low pass

filter after the sensors, such as a Butterworth filter, rendered the system impossible to control.

6.4 Using gyroscopes to improve state estimation

We have seen that we have too much noise for our Kalman filter to handle. It is therefore time to approach the problem from a different perspective in order to achieve a stabilized flight. This chapter focus on exploring the option of sensor fusion using the current sensor configuration (the tri-axis accelerometer and magnetic compass) together with three angular rate gyroscopes (one for each axis of rotation) to provide more accurate state estimation from noisy measurements.

Angular rate sensors, also known as gyroscopes (or gyros), serve the purpose of measuring the rate of rotation around the sensor axis. Theoretically, when integrating the signal from the gyro, one can acquire the angular change over a period of time, but one has to consider the problem of knowing the initial angle at time zero, and also the signal bias which varies in an unpredictable way. As a consequence, long term gyro signal integration is very poor, and the calculated angle drifts away from its real value. However, gyros provide very good readings of the angular rates, and once we get some knowledge on the sensor signal bias, we can accurately integrate the signal over a short period of time to determine reliable angular measurements. This means that a gyroscope has a very good response for high frequencies, but a very poor one for low frequencies (corresponding to the opposite case of an accelerometer). By fusing the low frequency response of the accelerometer with the high frequency measurements from the gyros yields accurate estimates for the attitude of an aircraft. Furthermore, we can also use the compass to help the Kalman filter getting an estimate on the yaw angle.

Having collated a series of noise data on the sensors (see Subsection 6.3.2), we can use this information to build a more accurate covariance matrix $\overline{\mathbf{R}}$, as presented:

$$\overline{\mathbf{R}} = \begin{bmatrix} 29.3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 22.2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 8 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.18 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.15 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.05 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.05 \end{bmatrix} \tag{6.1}$$

Notice that the matrix above was not built with angular gyro data, but with data from the accelerometer and compass. We will also assume that the covariance of the yaw gyro reading is the same as the one from the compass. This approach is not very accurate in the aspect that we should have gyro data to calculate its covariance, but because we do not have any at the time, we shall use the current data as a means of testing the behavior of the Kalman filter in the simulations. As usual so far, we shall use a reference where the quadrotor is in stationary hover while rotated 45° degrees around its yaw axis. Figure 6.4.1 shows the simulation results, where the simulation now includes noisy measurements based on the later collated sensor data. It is clear the our triad of sensors is very powerful in the sense that it provides sufficient data to the Kalman filter to at least keep the quadrotor stabilized, although we can clearly see some disturbances in the signal that are nevertheless handled by the LQR controller.

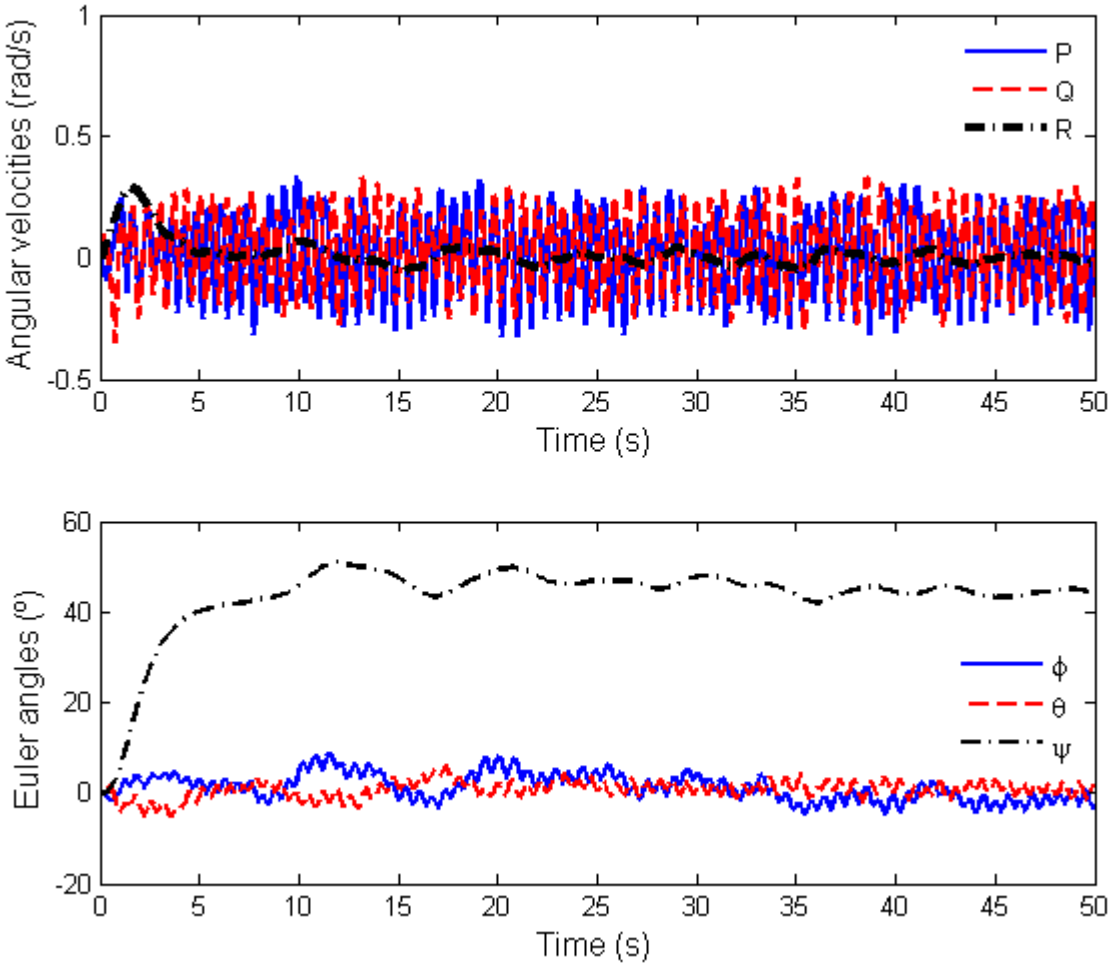


Figure 6.4.1: Time response of the LQR control loop using 6 states and gyros while in the presence of very noisy sensor readings.

6.5 Technical issues

During the construction and testing of the quadrotor, one is faced with many technical issues, such as the problem of fitting the avionics inside a protective hull and improving vibration absorption of the structure amongst others. One problem in particular, concerning the battery feeding the motors, is worth of mention. Through the course of control testing with the quadrotor prototype, it was verified that after a batch of tests, the battery was discharged, as expected. During the following test flight, after the battery was recharged, it was observed that the quadrotor was no longer able to achieve takeoff, even though the joystick controlling the propellers power was at full throttle. Measuring the voltage in each one of three cells of the battery revealed that two of them were reading 3 Volts and the third one was reading 0 Volts. LiPo batteries are the top of the line when it comes to power-to-weight ratio, but they are extremely sensitive and require special care. It is not advisable to let the voltage of each cell to drop below 3 Volts

because it is impossible to recharge the damaged cell again without an extreme risk of causing a battery fire. This fire is extremely dangerous in nature, and it cannot be extinguished with water. Therefore, it is advisable to always keep the voltage of each cell between 3 and 3.7 Volts. As the battery could no longer be used, the remaining cells were discharged by connecting a LED and a small resistor in series with each remaining battery cell. The battery was safely discarded afterwards and replaced with a model having similar characteristics.

Chapter 7

Conclusions and future work

The main objective of this thesis was to design and control a quadrotor prototype, having a tri-axis accelerometer and a compass as its sensors.

In the first phase of this thesis, objectives were defined for the quadrotor design, and based on them a study was carried out on the electronics that would be part of the aircraft. This analysis resulted in the construction of a quadrotor capable of achieving Take-Off without using the motors at full throttle, leaving plenty of extra space for maneuvering.

Knowing the characteristics of our quadrotor, we proceeded to the modeling of the aircraft's dynamics and kinematics for implementation of the resulting equations in a computer simulation environment. The modeling of the motors was achieved due to the use of data collected for identification which originated from the capture of sound of the propellers. This technique was based on the fact that a microphone is capable of sensing the pressure wave resulting from the passage of a propeller, and therefore, by measuring the period of each rotation, it is possible determine the angular speed of the motor-propeller assembly. This technique proved to be effective only for high angular velocities. Following this logic, the accelerometer and the compass were also modeled so that the computer simulations could better portray reality.

Anticipating the need to estimate as many states as possible from the available sensors, a Kalman filter was designed, limited to the hypothesis that the quadrotor is a linear system. It was verified that the Kalman filter could only estimate 6 of the 12 states initially considered for aircraft. In this stage, it was also discovered that none of the system's states is reachable.

Followed the construction of two LQR controllers for the system, one for the situation where all 12 states are observable (perfect situation) and another for the real situation where we only have 6 states available from the Kalman filter (i.e. we only have the angular speeds and Euler angles).

Finally we proceeded to the implementation phase, which evolved from the simplest case of control in the Simulink environment, to the real application in the quadrotor. Thus, starting with the 12 states LQR controller, we found that the system is completely controllable when all states are available. Continuing to the situation where we can only read 6 of those 12 states, the LQR controller designed for this case was also able to control the available states, while the other 6 visibly drifted with time. The introduction of the Kalman filter, sensors and motors dynamics in the control loop was also dominated by the same 6 states LQR controller, without any need for adjustment to its weighting matrices. We then started with the actual implementation, where a software program was developed to allow for communication between a

joystick and the quadrotor's control unit, the Arduino. Unfortunately, the quadrotor displayed an unstable behavior. It was assumed that the lack of inclusion of the Pulse-Width Modulation signal resolution in the simulations could the reason why this instability had not been foreseen. After introducing this new element in the simulation, we found nevertheless that the controller could still drive the system to the intended reference, although with a little more difficulty. This information led us to analyze the noise coming from the sensors with the motors running, which proved to be the source of the problem. Not even the Kalman filter was able to filter out that much noise. This may be explained by the fact that the system is not reachable. In other words, if the sensor data is corrupted with noise of such high in intensity, it may be impossible for the Kalman filter to estimate the desired states.

All the work done so far shows only that there are several aspects of the control process that need to be corrected/improved in order to make a controlled flight for this prototype possible. In particular, the inclusion of 3 gyroscopes (one for each axis of rotation) should be considered, which together with the tri-axis accelerometer, form a combination of sensors that has already been tested in quadrotors, allowing for very good control over its pitch and roll angles. The idea is that with gyroscopes we can determine orientation, but they drift over time (long-term errors), which is something we can correct with accelerometers (short-term errors, i.e. they are very noisy). But even then, we would need something else to correct the drifting yaw angle from the gyroscope, which could be done with a magnetic compass. The long-term idea is that adding a greater number of sensors, in addition to those suggested, can only improve the quality and quantity of states estimated by the Kalman filter.

It would also be interesting to develop the simulations further by including new elements such as aerodynamic forces (i.e. wind), collisions and other types of control methods, such as Proportional Derivative controllers. The quadrotor prototype should also be taken to an outside environment to see its performance tested under more extreme conditions.

Bibliography

[1] P. Pounds, R. Mahony, J. Gresham, P. Corke, and J. Roberts. Towards Dynamically-Favourable Quad-Rotor Aerial Robots. 2004.

[2] G.M. Hoffmann, H. Huang, S.L. Waslander, and C.J. Tomlin. Quadrotor Helicopter Flight Dynamics and Control: Theory and Experiment. 2007.

[3] Gjioni E. AbouSleiman R., Korff D. and Yang H. The oakland university unmanned aerial quadrotor system. 2008.

[4] W. Barnes and W. McCormick. *Aerodynamics, Aeronautics and Flight Mechanics*. New York: Wiley, 2nd. edition, 1995.

[5] http://en.wikipedia.org/wiki/Quadrotor, October 2009.

[6] J.G. Leishman. The Breguet-Richet Quad-Rotor Helicopter of 1907. *Vertiflite*, 47(3):1–4, 2001.

[7] http://www.nationmaster.com/encyclopedia/Quadrotor, January 2009.

[8] http://en.wikipedia.org/wiki/Moller_Skycar_M400, January 2009.

[9] http://en.wikipedia.org/wiki/M200G_Volantor, January 2009.

[10] http://www.accessmylibrary.com/article-1G1-83761490/air-medical-transport-taking.html, September 2009.

[11] S. Bouabdallah, P. Murrieri, and R. Siegwart. Design and control of an indoor micro quadrotor. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 5.

[12] S. Bouabdallah, A. Noth, and R. Siegwart. PID vs LQ control techniques applied to an indoor micro quadrotor. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3.

[13] M. Becker, S. Bouabdallah, and R. Siegwart. Desenvolvimento de um controlador de desvio de obstáculos para um mini-helicóptero Quadrirotor autónomo - 1ª fase: Simulação.

[14] P. Pounds, R. Mahony, and P. Corke. Modelling and Control of a Quad-Rotor Robot. 2006.

[15] A. Mokhtari, A. Benallegue, and B. Daachi. Robust feedback linearization and gh8 controller for a quad rotor unmanned aerial vehicle. *Journal of Electrical Engineering*, 57(1):20–27, 2006.

[16] G.P. Tournier, M. Valenti, J.P. How, and E. Feron. Estimation and control of a quadrotor vehicle using monocular vision and moirè patterns. 2006.

[17] E. Courses and T. Surveys. Robust low altitude behavior control of a quadrotor rotorcraft through sliding modes. pages 1–6, 2007.

[18] Katie Miller. Path tracking control for quadrotor helicopters. July 2008.

[19] Sérgio Eduardo Aurélio Pereira da Costa. Controlo e simulação de um quadrirotor convencional. Master's thesis, october 2008.

[20] www.radrotary.com/motor_test.xls, November 2008.

[21] J. Roskam. *Airplane flight dynamics and automatic flight controls*. DARcorporation, 2001.

[22] http://mathworld.wolfram.com/EulerAngles.html, May 2009.

[23] HZ Peter. Modeling and simulation of aerospace vehicle dynamics. *Virginia: American Institute of Aeronautics and Astronautics*, pages 17–242, 2000.

[24] http://en.wikipedia.org/w/index.php?title=Quaternions_and_spatial_rotation&oldid=296655109, June 2009.

[25] A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[26] P.S. Maybeck. Stochastic Models, Estimation and Control, Volume 1 Chapter 1, 1979.

[27] Seth Hutchinson. Lecture notes on linearization via taylor series, university of illinois, Spring 2009.

[28] Jean Walrand. Kalman filter: Convergence. Lecture Notes, U. C. Berkeley, August 2009.

[29] G.F. Franklin, M.L. Workman, and D. Powell. *Digital control of dynamic systems*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1997.

[30] B. D. O. Anderson and J. B. Moore. *Optimal Control: Linear Quadratic Methods*. Prentice-Hall, Englewood Cliffs, New Jersey, 1990.

[31] http://www.ece.ucsb.edu/ roy/classnotes/147c/lqrlqgnotes.pdf, July 2009.

Appendix A

Electronic Circuits

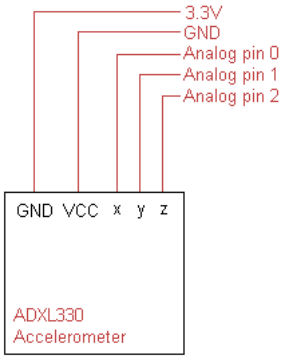


Figure A.1: Accelerometer wiring.

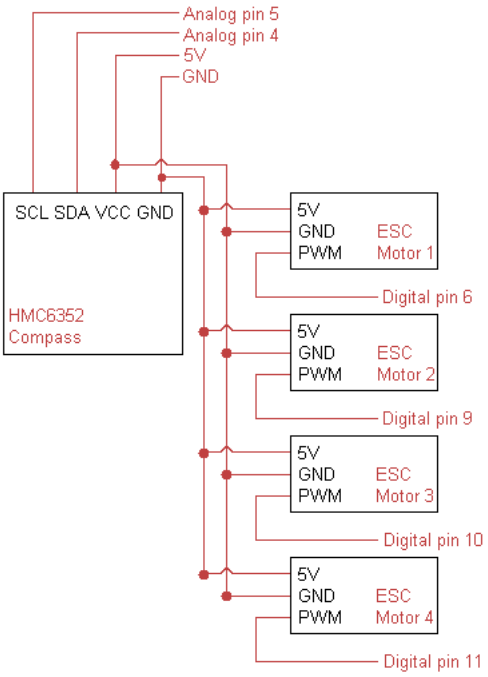


Figure A.2: Compass and electronic speed controllers wiring.

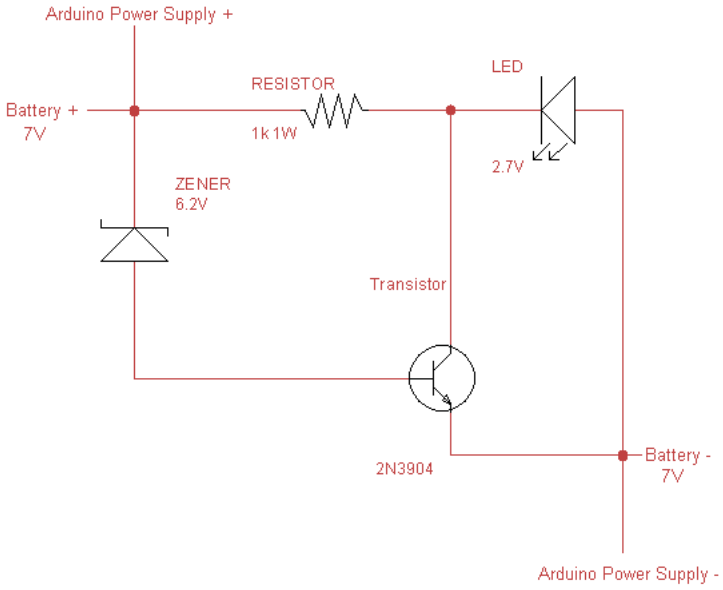


Figure A.3: Arduino's battery check circuit.

Appendix B

Matlab® files

B.1 loadvars.m

```
%Description: This m-file initializes all necessary variables required to
%buid the dynamic model of a quadrotor in the state-space form.
%The motor configuration is as follows:
%                  X axis
%                 _|_
%                /   \
%               |  1  |
%                \___/
%                  |
%                  |
%   ___            |           ___
%  /   \       |-----|       /   \
% |  4  |------|  xZ |------|  2  |-Y axis
%  \___/       |-----|       \___/
%                  |
%                  |
%                 _|_
%                /   \
%               |  3  |
%                \___/

clear all;

%% Physical properties of the environment

g=9.81; %Acceleration of gravity (m)
rho=1.2; %Density of air (m^3.kg^-1)

%% Physical properties of the quadrotor

mq=0.82; %Total mass of the quadrotor

mm=0.048; %Mass of a motor (kg). All motors have equal mass.

lx=28.8e-3; %Motor length along x-axis (m). All motors have equal sizes.
ly=28.8e-3; %Motor length along y-axis (m)
lz=26e-3;  %Motor length along z-axis (m)
dcg=0.29; %Distance from the center of gravity to the center of a motor (m).
          %The quadrotor is symmetric regarding the XZ and YZ planes, so
          %dcg is the same for all motors.

Ix1=(1/12)*mm*(ly^2+lz^2); %Moment of inertia (x-axis) for motors 1 and 3
%(kg.m^2).
```

```
Ix2=(1/12)*mm*(ly^2+lz^2)+mm*dcg^2; %Moment of inertia (x-axis) for motors
%2 and 4 (kg.m^2).
Ixx=2*Ix1+2*Ix2; %Total moment of inertia along the x-axis (kg.m^2)

Iy1=(1/12)*mm*(lx^2+lz^2)+mm*dcg^2; %Moment of inertia (y-axis) for motors
%1 and 3 (kg.m^2).
Iy2=(1/12)*mm*(lx^2+lz^2); %Moment of inertia (y-axis) for motors 2 and 4
%(kg.m^2).
Iyy=2*Iy1+2*Iy2; %Total moment of inertia along the y-axis (kg.m^2)

Iz1=(1/12)*mm*(lx^2+ly^2)+mm*dcg^2; %Moment of inertia (z-axis) for motors
%1 and 3 (kg.m^2).
Iz2=(1/12)*mm*(lx^2+ly^2)+mm*dcg^2; %Moment of inertia (z-axis) for motors
%2 and 4 (kg.m^2).
Izz=2*Iz1+2*Iz2; %Total moment of inertia along the z-axis (kg.m^2)

I=diag([Ixx Iyy Izz]); %Inertia matrix

cp=0.0743; %Thrust coefficient of the propeller
ct=0.1154; %Power coefficient
rp=5*25.4e-3; %Propeller radius (m)

Km=cp*4*rho*rp^5/pi()^3; %Constant value to calculate the moment provided
%by a propeller given its angular speed (kg.m^2.rad^-1)
Kt=ct*4*rho*rp^4/pi()^2; %Constant value to calculate the thrust provided
%by a propeller given its angular speed (kg.m.rad^-1)

Ktm=Km/Kt; %Constant that relates thrust and moment of a propeller.

k1=2.028; %Transfer-function gain of motor 1
k2=1.869; %Transfer-function gain of motor 2
k3=2.002; %Transfer-function gain of motor 3
k4=1.996; %Transfer-function gain of motor 4

tc=0.136; %Time-constant of the motors (assumed equal for all motors)

dz1=1247.4; %PWM dead-zone of motor 1 given its first-order transfer
%function (micro-seconds)
dz2=1247.8; %PWM dead-zone of motor 2 given its first-order transfer
%function (micro-seconds)
dz3=1246.4; %PWM dead-zone of motor 3 given its first-order transfer
%function (micro-seconds)
dz4=1247.9; %PWM dead-zone of motor 4 given its first-order transfer
```

```
%function (micro-seconds)

g_m1=tf(k1,[tc 1]); %First-order transfer-function of motor 1
g_m2=tf(k2,[tc 1]); %First-order transfer-function of motor 2
g_m3=tf(k3,[tc 1]); %First-order transfer-function of motor 3
g_m4=tf(k4,[tc 1]); %First-order transfer-function of motor 4
%Note: If the behavior of the motors is not linear, linearization around
%an operating point is required to attain these transfer-functions.

PWM_max=2200; %Upper limit of the PWM signal provided to
%the motors (micro-seconds)


%% Arduino variables

freq = 20; %Sampling frequency (Hz);
ardres = 10; %Arduino analog resolution (bits)

%% Sensors

daccel_x = 0.003; %Accelerometer distance to the quadrotor's center of
%gravity along the x-axis (m)
daccel_y = 0; %Accelerometer distance to the quadrotor's center of
%gravity along the y-axis (m)
daccel_z = -0.0490; %Accelerometer distance to the quadrotor's center of
%gravity along the z-axis (m)
daccel=[daccel_x daccel_y daccel_z];

accel_max = 3*g; %Accelerometer maximum possible reading
accel_min = -3*g; %Accelerometer minimum possible reading

ares = (accel_max-accel_min)/(2^ardres); %Accelerometer reading resolution

cres = 1*pi()/180;
%Compass reading resolution (rad) = i degree x 180 degrees / pi() rad.
%Note: this can be configured in the Arduino to be either 1 or 0.5 degrees


%% Initial states

k1 = 2.0276; %Constant that relates Rotation speed of propeller 1 with the
%corresponding PWM pulse of the Arduino
k2 = 1.8693; %Constant that relates Rotation speed of propeller 2 with the
```

```
%corresponding PWM pulse of the Arduino
k3 = 2.0018; %Constant that relates Rotation speed of propeller 3 with the
%corresponding PWM pulse of the Arduino
k4 = 1.9986; %Constant that relates Rotation speed of propeller 4 with the
%corresponding PWM pulse of the Arduino
PWMdz1 = 1256; %Dead zone of the motor 1 (PWM pulse-width in microseconds)
PWMdz2 = 1264; %Dead zone of the motor 2 (PWM pulse-width in microseconds)
PWMdz3 = 1256; %Dead zone of the motor 3 (PWM pulse-width in microseconds)
PWMdz4 = 1256; %Dead zone of the motor 4 (PWM pulse-width in microseconds)

%Calculate the PWM pulse necessary to beat the force of gravity
syms PWMg1 PWMg2 PWMg3 PWMg4
PWMg1=solve((PWMg1-PWMdz1)^2-(1/4)*mq*g/(Kt*(k1^2)),PWMg1);
PWMg1=double(PWMg1);
PWMg1=PWMg1(PWMg1>PWMdz1);
PWMg2=solve((PWMg2-PWMdz2)^2-(1/4)*mq*g/(Kt*(k2^2)),PWMg2);
PWMg2=double(PWMg2);
PWMg2=PWMg2(PWMg2>PWMdz2);
PWMg3=solve((PWMg3-PWMdz3)^2-(1/4)*mq*g/(Kt*(k3^2)),PWMg3);
PWMg3=double(PWMg3);
PWMg3=PWMg3(PWMg3>PWMdz3);
PWMg4=solve((PWMg4-PWMdz4)^2-(1/4)*mq*g/((Kt*k4^2)),PWMg4);
PWMg4=double(PWMg4);
PWMg4=PWMg4(PWMg4>PWMdz4);

%Compute the necessary rotation speed (rad.s^-1) of each propeller to beat
%the force of gravity
wm1 = double(k1*(PWMg1-PWMdz1));
wm2 = double(k2*(PWMg2-PWMdz2));
wm3 = double(k3*(PWMg3-PWMdz3));
wm4 = double(k4*(PWMg4-PWMdz4));

w0=[wm1 wm2 wm3 wm4]; %Initial angular speed of motors 1 to 4 (rad.s^-1)

U0=0; %Initial linear velocity along the x-axis (m.s^-1)
V0=0; %Initial linear velocity along the y-axis (m.s^-1)
W0=0; %Initial linear velocity along the z-axis (m.s^-1)
P0=0; %Initial angular velocity around the x-axis of the quadrotor
%(rad.s^-1)
Q0=0; %Initial angular velocity around the y-axis of the quadrotor
%(rad.s^-1)
R0=0; %Initial angular velocity around the z-axis of the quadrotor
%(rad.s^-1)
```

```
X0=0; %Initial quadrotor position along the x-axis of the inertial frame of
%reference (m)
Y0=0; %Initial quadrotor position along the y-axis of the inertial frame of
%reference (m)
Z0=0; %Initial 1uadrotor position along the z-axis of the inertial frame of
%reference (m)
PHI0=0; %Initial pitch angle of the quadrotor (rad)
THETA0=0; %Initial roll angle of the quadrotor (rad)
PSI0=0; %Initial yaw angle of the quadrotor (rad)

q0=angle2quat(PHI0,THETA0,PSI0); %Initial state quaternion

X0=[U0 V0 W0 P0 Q0 R0 X0 Y0 Z0 q0(1) q0(2) q0(3) q0(4)]; %Initial state
%vector

Z0lin = -1; %Linearization point with the quadrotor stabilized (m)
PHIlin = 0; %Linearization point pitch angle of the quadrotor (rad)
THETAlin = 0; %Linearization point roll angle of the quadrotor (rad)
PSIlin = 0; %Linearization point yaw angle of the quadrotor (rad)
qlin=angle2quat(PHIlin,THETAlin,PSIlin); %Initial state quaternion
X0lin=[0 0 0 0 0 0 0 0 Z0lin PHIlin THETAlin PSIlin]; %Linearization point
%for generating the state-space system representation of the quadrotor

[A,B,C,D]=quadss(X0lin,mq,I,w0,Kt,Ktm,dcg,daccel); %Get linearized state
%space matrices of the quadrotor
Y0ref = [0 0 0 0 0 0]; %Initial quadrotor reference

quadkalm; %Get Kalman filter

quadLQRcontrol; %Get LQR controller
%quadLQRcontrol_12states %Uncomment this line if you wish 12 states control
%with pure feedback (i.e. no Kalman filter nor sensors)

damp(A-B*[zeros(4,3) Klqr(:,1:3) zeros(4,3) Klqr(:,4:6)]) %Eigen values of
%the closed loop. If an eigenvalue is not stable, the dynamics of this
%eigenvalue will be present in the closed-loop system which therefore will
%be unstable.
```

B.2 quadsys.m

```
function [Xout] = quadsys(Xin,m,I,wm,Kt,Ktm,dcg)
%Function to simulate the quadrotor dynamics and outputs its current
%state space vector.
%The motor configuration is as follows:
%                 X axis
%                _|_
%               /   \
%              |  1  |
%               \___/
%                 |
%                 |
%   ___           |           ___
%  /   \       |-----|       /   \
% |  4  |------|  xZ |------|  2  |-Y axis
%  \___/       |-----|       \___/
%                 |
%                 |
%                _|_
%               /   \
%              |  3  |
%               \___/
%    Inputs:
%
%    Xin - Initial state vector of the quadrotor:
%         Xin = [U V W P Q R X Y Z q1 q2 q3 q4]
%         U - Linear velocity in the x-axis direction of the inertial
%           reference frame (m.s^-1);
%         V - Linear velocity in the y-axis direction of the inertial
%           reference frame (m.s^-1);
%         W - Linear velocity in the z-axis direction of the inertial
%           reference frame (m.s^-1);
%         P - Angular speed around the x-axis of the quadrotor (rad.s^-1);
%         Q - Angular speed around the y-axis of the quadrotor (rad.s^-1);
%         R - Angular speed around the z-axis of the quadrotor (rad.s^-1);
%         X - Position of body frame of the quadrotor along the x-axis of
%           the inertial reference frame (m);
%         Y - Position of body frame of the quadrotor along the x-axis of
%           the inertial reference frame (m);
%         Z - Position of body frame of the quadrotor along the x-axis of
%           the inertial reference frame (m);
%         q1...q4 - Quaternion containing the quadrotor's attitude
%           (conversion made from Euler angles).
%
```

```
%     m - Quadrotor's mass (kg).
%
%     I - Inertia matrix assuming the quadrotor is a rigid body with
%           constant mass of which its axis are aligned with the principal
%           axis of inertia. The matrix I becomes a diagonal matrix
%           containing only the principal moments of inertia:
%
%         I = [I1 0 0;0 I2 0;0 0 I3]
%         I1 - Principal moment of inertia along the x-axis of the
%           quadrotor's body frame (kg.m^2);
%         I2 - Principal moment of inertia along the y-axis of the
%           quadrotor's body frame (kg.m^2);
%         I3 - Principal moment of inertia along the z-axis of the
%         quadrotor's body frame (kg.m^2).
%
%     wm - Angular speed array of the propellers:
%         wm = [w1 w2 w3 w4]
%         w1 - Current angular speed of propeller 1 (North propeller)
%           (rad.s-1);
%         w2 - Current angular speed of propeller 2 (East propeller)
%           (rad.s-1);
%         w3 - Current angular speed of propeller 3 (South propeller)
%           (rad.s-1);
%         w4 - Current angular speed of propeller 4 (West propeller)
%           (rad.s-1).
%
%     Kt - Constant used to express the thrust and prop. angular speed
%
%     Ktm - Constant used to express the thrust with prop. momentum
%
%     dcg - Distance from the quadrotor's COG to the motor
%
%      Outputs:
%
%      Xout - Current system states:
%         Xout = [U V W P Q R X Y Z q1 q2 q3 q4]
%
%     Example:
%     Xin = [0 0 0 0.1 0 0 0 0 0 1 0 0 0];
%     wm = [0 0 0 0];
%     m = 1;
%     I = [0.1 0 0;0 0.1 0;0 0 0.1];
%     Kt=0.1;
```

```
%     Ktm=0.01;
%     dcg=0.5;
%
%     Xout = quadsys(Xin,m,I,wm,Kt,Ktm,dcg) outputs:
%
%     Xout = [0 0 9.8100 0 0 0 0 0 0 0 0.0500 0 0]

U=Xin(1); %Linear velocity x-axis (m)
V=Xin(2); %Linear velocity y-axis (m)
W=Xin(3); %Linear velocity z-axis (m)
P=Xin(4); %Angular velocity around the x-axis (rad.s^-1)
Q=Xin(5); %Angular velocity around the y-axis (rad.s^-1)
R=Xin(6); %Angular velocity around the z-axis (rad.s^-1)
X=Xin(7); %Position of the body axes frame along the x-axis of the inertial
%reference system (m)
Y=Xin(8); %Position of the body axes frame along the y-axis of the inertial
%reference system (m)
Z=Xin(9); %Position of the body axes frame along the z-axis of the inertial
%reference system (m)
q0=Xin(10); %Quaternion element 1 calculated from Euler angle conversion
q1=Xin(11); %Quaternion element 2 calculated from Euler angle conversion
q2=Xin(12); %Quaternion element 3 calculated from Euler angle conversion
q3=Xin(13); %Quaternion element 4 calculated from Euler angle conversion

Rm=[q0^2+q1^2-q2^2-q3^2 2*(q1*q2+q0*q2) 2*(q1*q3-q0*q2);
   2*(q1*q2-q0*q3) q0^2-q1^2+q2^2-q3^2 2*(q2*q3+q0*q1);
   2*(q1*q3+q0*q2) 2*(q2*q3-q0*q1) q0^2-q1^2-q2^2+q3^2]; %rotation matrix R
%using quaternions

Tm(1:4)=Kt.*(wm(1:4).^2); %calculate thrust produced by each propeller (N)

Fx=0; %x-axis force acting on the quadrotor while hovering steady at
%constant altitude
Fy=0; %y-axis force acting on the quadrotor while hovering steady at
%constant altitude
Fz=-(Tm(1)+Tm(2)+Tm(3)+Tm(4)); %z-axis force acting on the quadrotor while
%hovering steady at constant altitude

Mx=(Tm(4)-Tm(2))*dcg; %Momentum responsible for rotation around the x-axis
%(N.m)
My=(Tm(1)-Tm(3))*dcg; %Momentum responsible for rotation around the y-axis
%(N.m)
Mz=Ktm*(Tm(1)+Tm(3)-Tm(2)-Tm(4)); %Momentum responsible for rotation around
```

```
%the z-axis (N.m)

g=9.81; %Acceleration of gravity (m.s^-2)
Lin_a=(1/m).*[Fx Fy Fz]'+Rm*[0 0 g]'-([Q*W-R*V R*U-P*W P*V-Q*U])'; %Calc.
%the linear acceleration of the quadrotor

AngP=(Mx/I(1,1))-(I(3,3)-I(2,2))*Q*R/I(1,1); %Angular acceleration around
%the x-axis of the body reference frame
AngQ=(My/I(2,2))-(I(1,1)-I(3,3))*R*P/I(2,2); %Angular acceleration around
%the y-axis of the body reference frame
AngR=(Mz/I(3,3))-(I(2,2)-I(1,1))*P*Q/I(3,3); %Angular acceleration around
%the z-axis of the body reference frame

Pos=Rm'*[U V W]'; %Calculate the position of the quadrotor

Omq=[0 -P -Q -R; %Rotation matrix to calculate the time drivative of the
     P 0 R -Q;   %rotation quaternion
     Q -R 0 P;
     R Q -P 0];

epsilon=1-[q0 q1 q2 q3]*[q0 q1 q2 q3]';

dq=(1/2)*Omq*[q0 q1 q2 q3]'+epsilon*[q0 q1 q2 q3]'; %Time derivative of the
%rotation quaternion

Xout=[Lin_a(1) Lin_a(2) Lin_a(3) AngP AngQ AngR Pos(1) Pos(2) Pos(3)...
    dq(1) dq(2) dq(3) dq(4)]; %Output state vector

end
```

Appendix C

Arduino source code

```
#include <Wire.h> //Library for I2C implementation using analog pins 4 (SDA) and 5 (SCL)

#include <MegaServo.h>

#define NBR_SERVOS 4 // the maximum number of servos (up to 12 on Arduino Diecimilia)

#define FIRST_SERVO_PIN 6 // define motor pins on the Arduino board

#define SECOND_SERVO_PIN 9

#define THIRD_SERVO_PIN 10

#define FOURTH_SERVO_PIN 11

MegaServo Servos[NBR_SERVOS] ; // max servos

//Accelerometer variables

float ax,ay,az; //Acceleration values in m/s^2 for all three axis

const float as=0.00488; //Arduino analog resolution = 5Volts/1024 (10bits)

const float zg=1.63; //Sensor zero g bias = Supply Voltage/2 =3.3/2 (V)

const float sR=0.3; //Sensor resolution (V/g)

//Compass variables

int compassAddress = 0x42 >> 1; // compass I2C slave adress:  0x42

// because the wire.h library only uses the 7 bits most significant bits

// a shift necessary is to get the most significant bits.

int psi = 0; // Compass heading = yaw (direct readings from the compass in degrees)

float psirad; // Compass heading = yaw in radians

//Euler angles Roll and pitch

float phi; // (rad)

float theta; // (rad)

//Kalman filter data

const float A[6][6] = {

{1.0000,0.0000,0.0000,-0.5985,-0.0000,0.0000},

{0.0000,1.0000,-0.0000,-0.0000,-0.5985,0.0000},

{-0.0000,-0.0000,1.0000,-0.0000,-0.0000,-0.0479},
```

```
{0.0500,0.0000,0.0000,0.3384,-0.0000,-0.0000},

{0.0000,0.0500,-0.0000,-0.0000,0.3384,0.0000},

{-0.0000,0.0000,0.0500,0.0000,0.0000,0.9147}

};//(Kalman state-space matrix Ak)

const float B[6][10] = {

{-0.0000,-0.0188,-0.0000,0.0188,0.0000,0.0302,-0.0000,0.0031,-0.0000,-0.0000},

{0.0194,0.0006,-0.0194,-0.0006,-0.0302,-0.0000,-0.0000,-0.0000,0.0031,-0.0000},

{0.0009,-0.0009,0.0009,-0.0009,-0.0000,0.0000,0.0000,0.0000,0.0000,0.0479},

{-0.0000,0.0002,-0.0000,-0.0002,0.0000,0.0334,-0.0000,0.0034,-0.0000,0.0000},

{0.0005,0.0006,-0.0005,-0.0006,-0.0334,-0.0000,-0.0000,-0.0000,0.0034,-0.0000},

{0.0000,-0.0000,0.0000,-0.0000,0.0000,0.0000,0.0000,0.0000,-0.0000,0.0853}

}; //(Kalman state-space matrix Bk)

float x[6] = {0,0,0,0,0,0}; //state vector:  Roll speed, Pitch Speed, Yaw speed, Roll angle, Pitch angle, Yaw angle

float u[10] = {0,0,0,0,0,0,0,0,0,0}; //input vector:  w1 (motor 1 speed),w2,w3,w4,ax,ay,az,phi,theta,yaw

float At[6] = {0,0,0,0,0,0}; //temp variable

float Bt[6] = {0,0,0,-0.01,-0.04,0}; //temp variable

//LQR gain matrix

const float K[4][6] = {

{0.0000,7.1030,20.4163,0.0000,18.2839,14.4659},

{-7.1030,0.0000,-20.4163,-18.2839,0.0000,-14.4659},

{0.0000,-7.1030,20.4163,0.0000,-18.2839,14.4659},

{7.1030,0.0000,-20.4163,18.2839,0.0000,-14.4659}

};

float w1,w2,w3,w4; //Motor inputs from LQR controller (rad.s^-1)

float w0; //Throttle to the motors (from joystick)

//Motor inputs for PWM signal

float wc1=0;
```

```
float wc2=0;

float wc3=0;

float wc4=0;

//PWM signals

int p1 = 200;

int p2 = 200;

int p3 = 200;

int p4 = 200;

//Reference

float yr[6]={0,0,0,0,0,0};

void setup()

{

Wire.begin();

Serial.begin(9600); // Initialize serial communications with setup

Servos[0].attach( FIRST_SERVO_PIN, 800, 2200); //Motor 1 - North - Clockwise rotation

Servos[1].attach( SECOND_SERVO_PIN, 800, 2200); //Motor 2 - East - Counterclockwise rotation

Servos[2].attach( THIRD_SERVO_PIN, 800, 2200); //Motor 3 - South - Clockwise rotation

Servos[3].attach( FOURTH_SERVO_PIN, 800, 2200); // Motor 4 - West - Counterclockwise rotation

//Put the motors in full stop

Servos[0].write(0);

Servos[1].write(0);

Servos[2].write(0);

Servos[3].write(0);

}

void loop()

{

// get the most recent acceleration values for all three axis
```

```
ax = ((analogRead(0)*as-zg)/sR)*9.81; //acceleration x-axis (m.s^-2)

ay = ((analogRead(1)*as-zg)/sR)*9.81; //acceleration y-axis (m.s^-2)

az = ((analogRead(2)*as-zg)/sR)*9.81; //acceleration z-axis (m.s^-2)

ax=-ax; //This correction allows the accelerometer to provide positive acceleration in the x-axis direction

// Compass task 1:  connect to the HMC6352 sensor

Wire.beginTransmission(compassAddress);

Wire.send('A'); // The ascii character "A" tells the compass sensor to send data

Wire.endTransmission();

// Compass task 2:  wait for data processing

delay(50); // Compass datasheet says we need to wait at least 6000 microsegundos (0.006s) for data processing in the sensor

//We can also take this opotunity to implement the sampling time (20Hz)

// Compass task 3:  Request heading

Wire.requestFrom(compassAddress, 2); // request 2 bytes of data

// Compass task 4:  Get heading data

if(2 <= Wire.available()) // if 2 bytes are available

{

//16bit numbers can be broken in two 8 bit chunks.  The first is called high byte and the second low byte

psi = Wire.receive(); // get high byte

psi = psi << 8; // shift high byte

psi += Wire.receive(); // get low byte

psi /= 10; // comment this line if you wish to get the heading in decidegrees instead of degrees

}

psirad = deg2rad((float)psi); // Map the compass reading from 0 to 359 degrees to -pi radians to pi radians

phi=atan(ay/az); //Calculate roll angle from the accelerations provided by the accelerometer (rad)

theta=-atan(ax/az); //Calculate pitch angle from the accelerations provided by the accelerometer (rad)

//Perform Kalman filtering

At[0]=A[0][0]*x[0]+A[0][1]*x[1]+A[0][2]*x[2]+A[0][3]*x[3]+A[0][4]*x[4]+A[0][5]*x[5];
```

```
At[1]=A[1][0]*x[0]+A[1][1]*x[1]+A[1][2]*x[2]+A[1][3]*x[3]+A[1][4]*x[4]+A[1][5]*x[5];

At[2]=A[2][0]*x[0]+A[2][1]*x[1]+A[2][2]*x[2]+A[2][3]*x[3]+A[2][4]*x[4]+A[2][5]*x[5];

At[3]=A[3][0]*x[0]+A[3][1]*x[1]+A[3][2]*x[2]+A[3][3]*x[3]+A[3][4]*x[4]+A[3][5]*x[5];

At[4]=A[4][0]*x[0]+A[4][1]*x[1]+A[4][2]*x[2]+A[4][3]*x[3]+A[4][4]*x[4]+A[4][5]*x[5];

At[5]=A[5][0]*x[0]+A[5][1]*x[1]+A[5][2]*x[2]+A[5][3]*x[3]+A[5][4]*x[4]+A[5][5]*x[5];

//Get throttle from the joystick

if (Serial.available() > 0)

{

// read the incoming byte:

w0 = map((float)Serial.read(), 0, 100, 0, 500);

w0=constrain(w0,0,500); //Limit maximum velocity to 600 rad/s

Serial.flush();

}

else

{

w0 = 0;

}

//place known inputs and measurements in the input vector u

u[0]=wc1-w0;

u[1]=wc2-w0;

u[2]=wc3-w0;

u[3]=wc4-w0;

u[4]=ax-yr[0];

u[5]=ay-yr[1];

u[6]=az-yr[2];

u[7]=phi-yr[3];

u[8]=theta-yr[4];
```

```
u[9]=psirad-yr[5];


Bt[0]=B[0][0]*u[0]+B[0][1]*u[1]+B[0][2]*u[2]+B[0][3]*u[3]+B[0][4]*u[4];

Bt[0]=Bt[0]+B[0][5]*u[5]+B[0][6]*u[6]+B[0][7]*u[7]+B[0][8]*u[8]+B[0][9]*u[9];


Bt[1]=B[1][0]*u[0]+B[1][1]*u[1]+B[1][2]*u[2]+B[1][3]*u[3]+B[1][4]*u[4]

Bt[1]=Bt[1]+B[1][5]*u[5]+B[1][6]*u[6]+B[1][7]*u[7]+B[1][8]*u[8]+B[1][9]*u[9];


Bt[2]=B[2][0]*u[0]+B[2][1]*u[1]+B[2][2]*u[2]+B[2][3]*u[3]+B[2][4]*u[4]

Bt[2]=Bt[2]+B[2][5]*u[5]+B[2][6]*u[6]+B[2][7]*u[7]+B[2][8]*u[8]+B[2][9]*u[9];


Bt[3]=B[3][0]*u[0]+B[3][1]*u[1]+B[3][2]*u[2]+B[3][3]*u[3]+B[3][4]*u[4]

Bt[3]=Bt[3]+B[3][5]*u[5]+B[3][6]*u[6]+B[3][7]*u[7]+B[3][8]*u[8]+B[3][9]*u[9];


Bt[4]=B[4][0]*u[0]+B[4][1]*u[1]+B[4][2]*u[2]+B[4][3]*u[3]+B[4][4]*u[4]

Bt[4]=Bt[4]+B[4][5]*u[5]+B[4][6]*u[6]+B[4][7]*u[7]+B[4][8]*u[8]+B[4][9]*u[9];


Bt[5]=B[5][0]*u[0]+B[5][1]*u[1]+B[5][2]*u[2]+B[5][3]*u[3]+B[5][4]*u[4]

Bt[5]=Bt[5]+B[5][5]*u[5]+B[5][6]*u[6]+B[5][7]*u[7]+B[5][8]*u[8]+B[5][9]*u[9];

//Get estimated states x(k+1)=A.x(k)+B.u(k)

x[0]=At[0]+Bt[0];

x[1]=At[1]+Bt[1];

x[2]=At[2]+Bt[2];

x[3]=At[3]+Bt[3];

x[4]=At[4]+Bt[4];

x[5]=At[5]+Bt[5];

//Controlo LQR

w1=K[0][0]*x[0]+K[0][1]*x[1]+K[0][2]*x[2]+K[0][3]*x[3]+K[0][4]*x[4]+K[0][5]*x[5];
```

```
w2=K[1][0]*x[0]+K[1][1]*x[1]+K[1][2]*x[2]+K[1][3]*x[3]+K[1][4]*x[4]+K[1][5]*x[5];

w3=K[2][0]*x[0]+K[2][1]*x[1]+K[2][2]*x[2]+K[2][3]*x[3]+K[2][4]*x[4]+K[2][5]*x[5];

w4=K[3][0]*x[0]+K[3][1]*x[1]+K[3][2]*x[2]+K[3][3]*x[3]+K[3][4]*x[4]+K[3][5]*x[5];

//Adjust speeds with the throttle

wc1=w0-w1;

wc2=w0-w2;

wc3=w0-w3;

wc4=w0-w4;

//Calculate and send pulse width to the motors

p1=(int)((wc1/2.0276)+1252); // (speed of motor 1 (rad/s) / transfer function gain) + Dead-zone pulse width

p2=(int)((wc2/1.8693)+1262);

p3=(int)((wc3/2.0018)+1255);

p4=(int)((wc4/1.9986)+1255);

if(w0==0) //Forçar paragem

{

p1=400;

p2=400;

p3=400;

p4=400;

}

Servos[0].write(p1);

Servos[1].write(p2);

Servos[2].write(p3);

Servos[3].write(p4);

// output states

Serial.print(x[0]);

Serial.print(" ");
```

```
Serial.print(x[1]);

Serial.print(" ");

Serial.print(x[2]);

Serial.print(" ");

Serial.print(x[3]);

Serial.print(" ");

Serial.print(x[4]);

Serial.print(" ");

Serial.print(x[5]);

Serial.println("");

}

float deg2rad (float x) //function that receives an angle between 0 and 359 degrees and resturns the same angle between -pi
and pi radians

{

if(x>=0 && x<=180)

{

x=-x*3.14/180;

}

else

{

x = (360-x)*3.14/180;

}

return x;

}
```