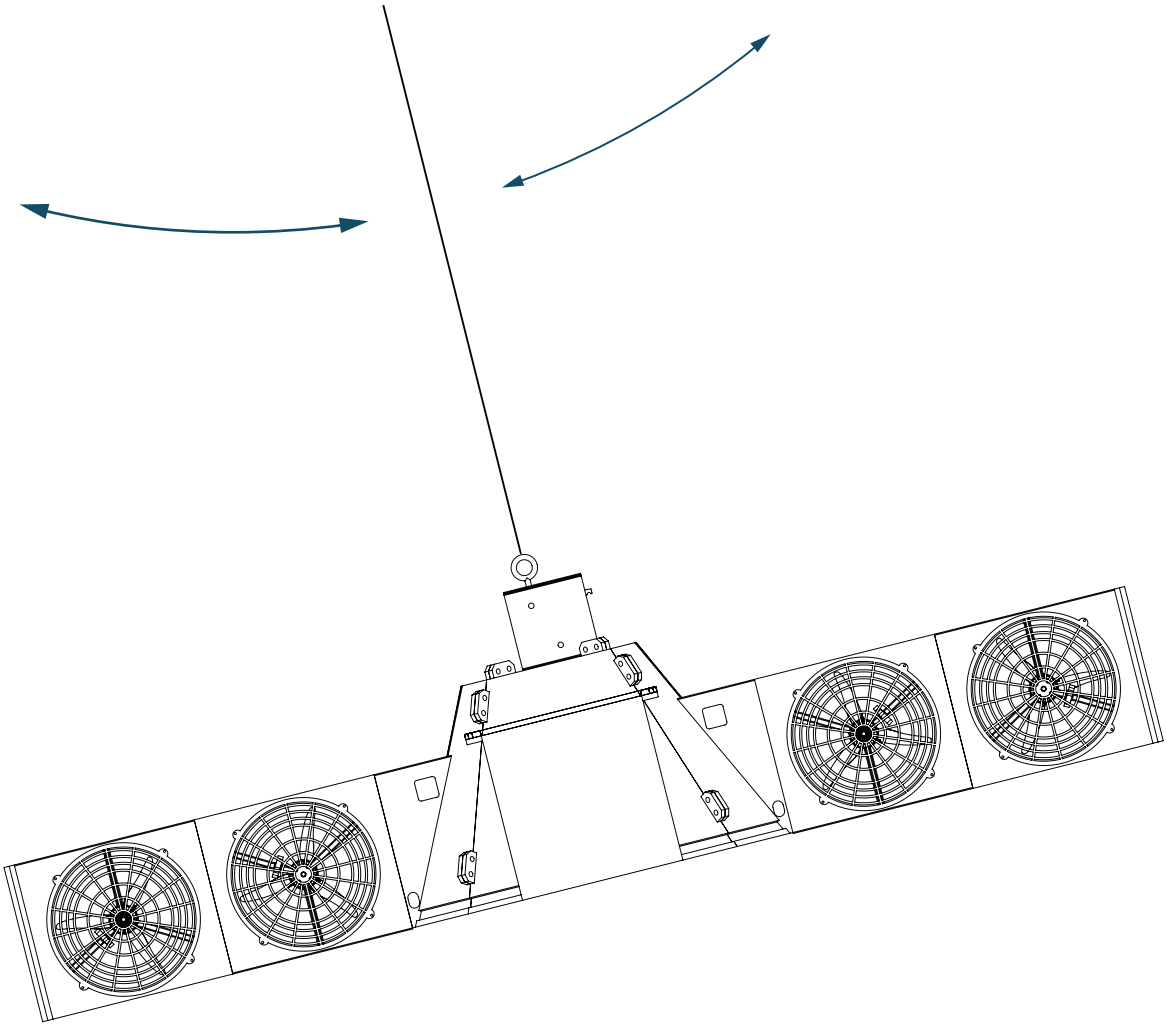
Master Thesis

DESIGN & CONTROL OF SINGLE WIRE STABILISED PAYLOAD SYSTEM

Daniel Valbjørn, Oliver Leidorf



Electro-Mechanical System Design
Materials and Production, Aalborg University

30rd of Maj 2025

Materials and Production
Electro-Mechanical System Design
Fibigerstræde 16
9220 Aalborg Øst

Title:
Design & Control of Single Wire Stabilised Payload System

Project:
Master Thesis

Project period:
2st of September 2024 to 30rd of Maj 2025

Project group:
EMSD3/4 - Group 5

Email:
dvalbj20@student.aau.dk
oleido20@student.aau.dk

Supervisor:
Dao Zhou

Co-supervisor:
Henrik C. Pedersen

Pages:
118

Appendices:
11 (129 Pages)

Participants:
Daniel Valbjørn
Oliver Leidorf

The content of the report is freely available, but publication (with source reference) may only take place in agreement with the authors.

Abstract

In response to the rising demand for sustainable energy, the construction of wind turbines is increasing, and more wind turbines will need maintenance. A significant part of the lifetime cost of operating a wind turbine is the maintenance. Half of the maintenance cost is renting and operating the cranes. Airflight proposes using flying cranes to reduce the cost of delivering lightweight components.

In this report, a solution will be designed to facilitate the use of flying cranes. The design aims to increase the distance between the flying crane and the wind turbine, which requires stabilisation of the lowered payload. The report aims to describe the development of a single-wire stabilised payload system (SPS) for delivering a payload from a flying crane to a wind turbine.

The design of the SPS is derived from the use of the House of Quality and methodological analysis. This resulted in a design for the SPS system, featuring four BLDC motors aligned along the same axis.

A model of the SPS system, comprising four BLDC motor models, a mechanical pendulum, and a yaw model, has been implemented in Simulink. The mechanical pendulum model has been derived using Euler-Lagrange mechanics, resulting in a system described by generalised coordinates. The models were verified to represent their real-world counterparts accurately. The models were linearised for use in the development of linear controllers, LQI and classical control theory were used.

A prototype of the design was constructed, and the controllers were implemented to test the system. The results of the tests show that the SPS system dampened the oscillations at different wind speeds in $\approx 10 - 15\ s$. This is a significant decrease in the dampening time compared to the natural system, which is $\approx 70 - 180\ s$ without control.

From the results, it can be concluded that a system capable of dampening oscillations at different wind speeds has been developed. However, further work is still required to develop a commercially viable product that can be effectively utilised in real-world scenarios.

Preface

This project is produced by Master Thesis, students of the group EMSD3/4 - Group 5, studying Electro-Mechanical System Design, at Aalborg University. It is written in the period 2st of September 2024 to 30rd of Maj 2025 under the supervision of Dao Zhou with some input from the CO-supervisor Henrik C. Pedersen.

We want to give our thanks to the project supervisors Dao Zhou & Henrik C. Pedersen for their support throughout the project period. We would also like to thank Airflight for their project proposal and for providing the necessary hardware for the project.

For this project, generative AI was used to proofread the report, improving its language and sentence structure. This has been done with the use of Grammarly and with Overleaf Writefull.

In some figures, a parenthesis containing a video link is accompanied by a QR code that links to the same video on YouTube. These videos are not necessary to watch to understand the rapport. They have been added to give a visual understanding of some of the tests.

During the making of this project, the following programs have been used:

- Overleaf LaTeX— For text processing and layout of this report.
- MathWorks MATLAB & Simulink — For calculations, modelling, simulation, and data processing.
- Adobe Illustrator, Inkscape — For diagrams and illustrations.
- Adobe Premiere Pro — For video editing.
- SolidWorks — For schematics and CAD drawings of the system.
- STM32CubeIDE — For microcontroller coding and communication.
- STM32CubeMonitor — For recording the parameter values on the microcontroller.
- Arduino IDE — For Arduino coding
- Bambu Studio — For 3D printing CAD parts

Daniel Valbjørn

Oliver Leidorf

Aalborg University, 30rd of Maj 2025

Nomenclature

Symbol	Definition	Unit symbol
A	System transition matrix	–
A_p	System transition matrix for model	–
A_{aug}	LQI augmented system transition matrix for model	–
$[A]$	Lift rotational matrix	–
A_p	Projected area	m^2
$A_{payload}$	Area of payload	m^2
A_{pdisk}	Area of propeller disk	m^2
A_{cap}	Area of capacitor plate	m^2
a_x	Acceleration in x,y,z	$\frac{\mathrm{m}}{\mathrm{s}^2}$
a_c	Coriolis acceleration	$\frac{\mathrm{m}}{\mathrm{s}^2}$
B	Input matrix	–
B_p	Input matrix for model	–
B_{aug}	LQI augmented input matrix for model	–
b_x	Compensated magnetic flux in x,y,z	G
b	width of sps arm	m
B_{lin}	Linearised BLDC motor constant	$\mathrm{Nm} \cdot \mathrm{s}$
C	Clarke transformation matrinc	–
C_p	Output matrix for model	–
C_{aug}	LQI augmented output matrix for model	–
c	Dampening coefficient	$\frac{\mathrm{Ns}}{\mathrm{m}}$
c_x	Damp coefficient x	$\frac{\mathrm{N} \cdot \mathrm{s}}{\mathrm{m}}$
C_D	Drag corfficent	–
$C_{D.0}$	Drag corfficent at $\Gamma = 0°$	–
$C_{D.90}$	Drag corfficent at $\Gamma = 90°$	–
C_L	Lift corfficent	–
C_x	Capacitance x	F
$C\vec{O}P$	Vector from COM to COP	–
C_v	Viscous friction in BLDC motor	$\mathrm{Nm} \cdot \mathrm{s}$
C_c	Coloumb friction in BLDC motor	Nm
C_p	Propellor resistive torque constan	$\mathrm{Nm} \cdot \mathrm{s}^2$
C_x	BLDC commutation stair wave form for phases a,b,c	–
$C_{v.\alpha}$	Viscous friction coefficient swing	$\mathrm{Nm} \cdot \mathrm{s}$
$C_{c.\Psi}$	Coloumb friction coefficient yaw	Nm
C_{BLDC}	Controller transfer function for BLDC	–
d_x	Capacitor distance x	m
d_{rest}	Capacitor rest position	m
Dx	Diode number x	–
E_x	Back-EMF phase (a,b,c)	V
e_p	Error vector	–

f	frequency	Hz
F_D	Drag force	N
F_L	Lift force	N
F_g	Gravitational force	N
F_{ex}	External force	N
F_w	Wind force	N
$\vec{F_{sum}}$	Global force vector on SPS	N
g	gravitational constant	$\frac{\mathrm{m}}{\mathrm{s}^2}$
G_{BLDC}	Transfer function for BLDC plant	–
H	Observation matrix	–
h	Hight of sps	m
H_x	Hall signal number x	m
H_k	Sum of matririal anisotropy and shape an	$\frac{\mathrm{A}}{\mathrm{m}}$
H_{ko}	Material anisotropy field	$\frac{\mathrm{A}}{\mathrm{m}}$
H_d	Shape inisotropy field	$\frac{\mathrm{A}}{\mathrm{m}}$
H_y	External magnetic field in y-axis	$\frac{\mathrm{A}}{\mathrm{m}}$
h_{acc}	Non-linear output function for accelerometer in EKF	–
h_{mag}	Non-linear output function for magnotometer in EKF	–
I	Current	A
i_x	Current in phase	A
$\dot{i}_x$	Current change in phase	$\frac{\mathrm{A}}{\mathrm{s}}$
i^ϕ	Current amplitude	A
$\bar{i}^\phi$	Current phase vector	A
i^d	Current in d-axis	A
i^q	Current in q-axis	A
J_r	Rotor inertia	$\frac{\mathrm{kg}}{\mathrm{m}^2}$
J_p	Propellor inertia	$\frac{\mathrm{kg}}{\mathrm{m}^2}$
J_{tot}	Total motor inertia	$\frac{\mathrm{kg}}{\mathrm{m}^2}$
J_Ψ	Mass moment of inertia yaw	$\frac{\mathrm{kg}}{\mathrm{m}^2}$
J_{extra}	Extra added mass moment of inertia in yaw for simulations	$\frac{\mathrm{kg}}{\mathrm{m}^2}$
K_v	Kalman gain (also used with acc and mag)	–
k_x	spring coefficient x	$\frac{\mathrm{N}}{\mathrm{m}}$
K_e	Electrical back-EMF constant	$\frac{\mathrm{V \cdot s}}{\mathrm{rad}}$
K_m	Mechanical back-EMF constant	$\frac{\mathrm{V \cdot s}}{\mathrm{rad}}$
K_m^q	Mechanical back-EMF constant in q-axis	$\frac{\mathrm{V \cdot s}}{\mathrm{rad}}$
K_I	Controller integrator gain matrix	–
K_x	Controller state gain matrix	–
l	Length of SPS	m
l_1	Length from centre of SPS to motor 1 and 2	m
l_2	Length from centre of SPS to motor 0 and 3	m
l_{tongue}	Length from centre of SPS to first arm secment tongue	m
L_s	Inductance in phase	H
L_w	Length of wire	m
M	Moment	$\mathrm{N \cdot m}$
$\bar{M}$	Saturated moment	$\mathrm{N \cdot m}$
M_α	α moment	$\mathrm{N \cdot m}$
M_γ	γ moment	$\mathrm{N \cdot m}$
M_s	Saturation magnetsation	$\frac{\mathrm{A}}{\mathrm{m}}$

M_{mag}	Spontanious magnetisation	$\frac{\mathrm{A}}{\mathrm{m}}$
m	Mass of SPS	kg
m_x	magnetic field in x,y,z	G
m_{air}	Mass of air	kg
m_p	Proof mass	kg
$\vec{M_{total}}$	Global SPS moment about origo	$\mathrm{N} \cdot \mathrm{m}$
P	Park transformation matrix	–
P_n	Covariance matrix	–
P_T	Thrust power	W
P_{out}	BLDC motor power out	W
p_x	Static pressure zones for propeller	Pa
p	BLDC motor pole pairs	–
P^q_{put}	BLDC output power in q-axis	W
Q	State cost matrix	–
Q_v	Model variance matrix for Kalman filter	–
q_a	Anti-integral wind-up vector	–
q_p	Integrator state vector	–
R	Input cost matrix	–
R_v	Measurement variance matrix for Kalman filter	–
R_θ	Resistance for magnetic angle θ	Ω
R_s	Resistance in phase	Ω
$\vec{R}$	Pendulum position vector	m
$\dot{\vec{R}}$	Pendulum velocity vector	$\frac{\mathrm{m}}{\mathrm{s}}$
r_p	Reference vector	–
Sx	Switch number x	–
t	Thickness of sps	m
t_s	Sample time	s
t_{DZ}	Time until dead zone is reached	s
t_{diff}	Time between natural and controlled system reaching dead zone	s
T	Thrust	N
T_p	Thrust posative	N
T_n	Thrust negative	N
T_x	Thrust motor x	N
T_f	Period time	s
T_{kin}	Kinetic energy	J
T_{local}	Thrust in local frame of SPS	N
T_{m-refx}	Thrust reference for motor x	N
U_x	Voltage in phase a,b,c	V
u_x	Driving force x	N
u_y	Driving force y	N
u_p	Input states & controller signal	–
U_{bat}	Battery voltage	V
U^d	Voltage in d-axis	V
U^q	Voltage in q-axis	V
U_{ll}	Line voltage	V
u_p	Saturated controller signal	N
$\bar{u}_p$	Unsaturated controller signal	N
v	Relative wind speed	$\frac{\mathrm{m}}{\mathrm{s}}$

v_{wind}	wind speed	$\frac{\mathrm{m}}{\mathrm{s}}$
v_i	Induced velocity	$\frac{\mathrm{m}}{\mathrm{s}}$
V_{bridge}	Bridge voltage	V
V_{drive}	Drive voltage	V
V_x	Voltage capacitor x	V
V_{dc}	DC voltage	V
V_{pot}	Potential energy	J
w	Down stream velocity	$\frac{\mathrm{m}}{\mathrm{s}}$
X_{Cx}	Capacitance Reactance x	Ω
x_p	Length between COM and COP	m
x_p	State vector	–
x_I	State integral vector	–
x_n	Prediction state –	
x_{aug}	LQI augmented state vector –	
y_n	Messurement vector	–
y_p	Output vector	–
y_{aug}	LQI augmented output vector –	
y_{acc}	Accelerometer messurement vector	$\frac{\mathrm{m}}{\mathrm{s}}$
y_{mag}	Magnetometer messurement vector	mT
α_{ref}	Swing angle reference	rad
α_{error}	Error in swing angle	rad
α_{dead}	Dead zone angle for swing angle	rad
$\dot{\alpha}_{dead}$	Dead zone velocity for swing angle	$\frac{\mathrm{rad}}{\mathrm{s}}$
α	Swing angle	rad
$\dot{\alpha}$	Swing angle velocity	$\frac{\mathrm{rad}}{\mathrm{s}}$
$\ddot{\alpha}$	Swing angle acceleration	$\frac{\mathrm{rad}}{\mathrm{s}^2}$
γ	Swing diriction angle	rad
$\dot{\gamma}$	Swing diriction angle velocity	$\frac{\mathrm{rad}}{\mathrm{s}}$
$\ddot{\gamma}$	Swing diriction angle acceleration	$\frac{\mathrm{rad}}{\mathrm{s}^2}$
Γ	Angle of attack	rad
ϵ	Dielectric permittivity	$\frac{\mathrm{F}}{\mathrm{m}}$
θ	Pitch angle	rad
θ_{rot}	Global pitch angle	rad
θ_e	Electric motor angle	rad
θ_{mx}	Mechanical motor angle for motor x	rad
Θ_{am}	Attitude from accelerometer	rad
Θ_{ω}	Attitude from gyroscope	rad
Θ_{com}	Attitude from Complementary Filter	rad
$\vec{\lambda_{y'}}$	Unit vector perpendicilar to swing plane	–
$\vec{\lambda_z}$	Unit vector in z-axis	–
$\mathscr{L}$	Langrangian	J
ρ	Density of air	$\frac{\mathrm{kg}}{\mathrm{m}^3}$
$\vec{\tau}_w$	Torque from areodynamical forces	$\mathrm{N}\cdot\mathrm{m}$
τ_T	Torque from thrusters	$\mathrm{N}\cdot\mathrm{m}$
τ_e	Electrical motor torque	$\mathrm{N}\cdot\mathrm{m}$
τ_{prop}	Propellor torque	$\mathrm{N}\cdot\mathrm{m}$
τ_e^q	Electrical motor torque in q-zxis	$\mathrm{N}\cdot\mathrm{m}$
υ	Complementary filter floating value	–

ϕ	Roll angle	rad
Ψ_{ref}	Yaw angle & rotation about wire reference	rad
Ψ_{+}	Needed yaw rotation for eliminating pitch values	rad
Ψ_{error}	Error in yaw angle	rad
Ψ_{dead}	Dead zone angle for yaw angle	rad
$\dot{\Psi}_{dead}$	Dead zone velocity for yaw angle	$\frac{\text{rad}}{\text{s}}$
Ψ_{rot}	Global yaw angle	rad
Ψ	Yaw angle & rotation about wire	rad
$\dot{\Psi}$	Yaw angle velocity	$\frac{\text{rad}}{\text{s}}$
$\ddot{\Psi}$	Yaw angle acceleration	$\frac{\text{rad}}{\text{s}^2}$
Ω_z	Frame angular velocity	$\frac{\text{rad}}{\text{s}}$
ω_x	Gyroscope angular velocity in x,y,z	$\frac{\text{rad}}{\text{s}}$
$\vec{\omega}_{gyro}$	Gyroscope angular velocity vector	$\frac{\text{rad}}{\text{s}}$
ω_{mx}	Motor angular velocity for motor x	$\frac{\text{rad}}{\text{s}}$
ω_{ex}	Motor electrical angular velocity for motor x	$\frac{\text{rad}}{\text{s}}$
ω_{m-ref}	Motor reference velocity	$\frac{\text{rad}}{\text{s}}$
$\omega_{m-error}$	Motor velocity error	$\frac{\text{rad}}{\text{s}}$
$\dot{\omega}_{mx}$	Motor angular acceleration for motor x	$\frac{\text{rad}}{\text{s}}$

Abbreviations

Abbreviation	Definition
AAR	Autoreload
AC	Alternating current
AMR	Anisotropic Magnetoresistive
APB	Advanced peripheral bus
CCR	Counter compare value
COP	Center of pressure
COM	Center of mass
DC	Direct current
DOF	Degrees of freedom
DMA	Direct Memory Access
EKF	Extended Kalman Filter
GNSS	Global Navigation Satelite System
HOQ	House of quakity
LQI	Linear Quadratic Integral control
LQR	Linear Quadratic Requlator
MCU	Microcontroller unit
MR	Magnetoresistive
MIMO	Multiple input Multiple output
O&M	Operating and maintenance
PWM	Pulse-Width-Modulation
PSC	Prescaler
RTK	Real-time kinematics
SPS	Stabilized Payload System
SISO	Singel input singel output
WWL	Working load limit
ZCP	Zero crossing point

Table of Contents

Page

1 Introduction **1**
- 1.1 Initial Problem . . . 2
- 1.2 Operation routine . . . 3

2 Problem Analysis **5**
- 2.1 System description . . . 5
- 2.2 External force estimation on simple 2D model . . . 7
- 2.3 Thrust generation . . . 10
- 2.4 BLDC - Motor . . . 12
- 2.5 Sensors . . . 14
- 2.6 Methoeds of attitude estimation . . . 20
- 2.7 DJI FlyCart 30 . . . 23

3 Problem Statement **25**
- 3.1 Limitations . . . 25

4 Design **27**
- 4.1 Requirements and pecifications . . . 27
- 4.2 Morphological analysis . . . 29
- 4.3 Prototype design . . . 30

5 Modelling **37**
- 5.1 BLDC motor model . . . 37
- 5.2 Thrust model . . . 45
- 5.3 Pendulum model . . . 47

6 Controllers **61**
- 6.1 BLDC motor controller . . . 61
- 6.2 Pendulum and yaw controller . . . 66

7 Implementation **77**
- 7.1 Hardware . . . 77
- 7.2 Wireing & digital comunication . . . 78
- 7.3 Program . . . 81
- 7.4 AHRS . . . 86
- 7.5 MIMO implementation . . . 91
- 7.6 α thrust zone . . . 93

8 Test of system **95**
8.1 Test setup . . . 95
8.2 Data processing . . . 96
8.3 SPS tests results . . . 96
8.4 Simulation results . . . 104

9 Discussion **111**
9.1 Design . . . 111
9.2 Model . . . 113
9.3 Control . . . 114

10 Conclusion **117**
10.1 Future work . . . 118

Bibliography **119**

A Aero coefficents & Forces **127**

B Requeriments and Engineering specfications **131**

C Morphological Analysis **137**

D Electrical model **145**

E Mechanical model **151**

F Parameter estimations **157**

G Controller Simulation **183**

H DSHOT **187**

I Sensor implementation **189**

J Test of System - addtional test **193**

K C-code **197**

1 Introduction

Airflight is a Danish-based company that aims to innovate the industry with large flying cranes. Traditional cranes are necessary to replace most components in a wind turbine during maintenance. Traditional cranes require a long setup time and are expensive to transport and operate (Doornik 2023). A flying crane can be set up quickly, lift up to 200 kg, and is highly automatable, reducing the required employee hours, chartering vessels, and downtime (Airflight 2023).

With a growing demand for clean energy to achieve net-zero emissions by 2050, the demand for wind turbines is rapidly increasing. It is expected to grow even more in the coming years (GWEC 2024, pp. 10–11).
A large part of the lifetime cost of operating a wind turbine is the operating and maintenance cost (O&M). According to (Ren et al. 2021, p. 2), the O&M for offshore wind farms easily reaches 23% of the total cost, while for onshore wind turbines, the O&M cost is at 5%. The difference in the O&M costs is the accessibility to the turbines, which multiple factors, such as high waves and wind speeds, can limit. According to (Doornik 2023), 50% of O&M cost is renting and operating the cranes.

Flying cranes are only affected by wind speeds and can quickly take off from the shore and fly a round trip up to 30 km with 200 kg. This can be done while technicians travel to the wind turbine, so the components are ready once they arrive (Airflight 2023). The operating procedure is further clarified in Section 1.2.
Airflight has encountered difficulties operating such a large drone due to turbulence. Because of the drone's large size, the wind can affect the propellers differently, resulting in a tilt that induces horizontal movement. The significant size of the drone results in a large inertia, resulting in a slower system that takes longer to correct induced movements. This makes the drone less accurate and increases the difficulty and risk of delivering components to the wind turbine's nacelle.

Airflight aims to move the drone $10 - 15$ m above the landing pad, moving the drone to a safer distance from the nacelle. The payload will be lowered to the landing pad using a hoist system. The current implementation of the hoist system is a 3-way wire system, which behaves mechanically at longer distances like a single wire, thereby losing control authority of the payload. Airflight is therefore reducing the wire count to one and wants to implement a system at the end of the wire to regain control authority. The implemented system is called a "Stabilised Payload System" (SPS). The SPS system will use thrusters to control the payload and stabilise induced oscillations. The SPS will also aim to automate the delivery process by implementing a payload

pickup/drop-off system. An illustration of the SPS lowered 10 m below the drone can be seen in Figure 1.1.

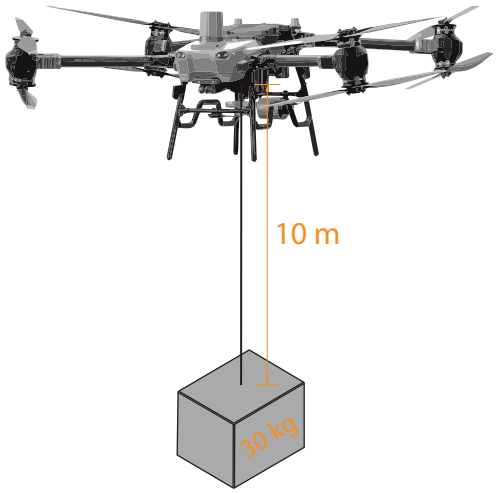


Figure 1.1. The SPS in the form of an unspecified box lowered 10m below the drone

The SPS is expected to be produced in a series with an increasing weight capacity, where the smallest one will fit on a drone capable of lifting a 30 kg payload, including the SPS system. Weight is, therefore, a crucial design specification as all mass on the SPS is parasitic and is subtracted from the payload potential. This report will focus on the design of the SPS compatible with the DJI FliCart 30. Still, it is also expected to be compatible with most other drones with a 30 kg payload capacity.

The SPS will operate autonomously, where the only input required is the activation from the drone pilot to arm the system. Therefore, SPS will have to know how far it has been lowered from the drone and how far it is to the landing location. Because the drone's battery drains more when flying with heavy loads, the SPS must be fast and precise to optimise the drone's capabilities. The attachment hook used to attach the payload to SPS is bidirectional, which requires orientation attitude control for cargo pickup missions.

1.1 Initial Problem

With an expected increase in wind turbines both on- and offshore, a sustainable solution for replacing small components is needed. Flying cranes are an attractive solution, though some problems must be resolved to make it a viable option. A system is required to maintain control authority of the payload during delivery, thereby keeping the drone at a safe distance from the nacelle.
This leads to the initial problem given by:

How to stabilise a payload lowered from a drone affected by external wind forces?

1.2 Operation routine

The SPS will be mounted on the drone with the hoist system on the ground within the drone's operating range, where two options for cargo pick-up are possible. If the cargo is on location, it can be connected to the SPS on the ground, or if the cargo is located remotely, the drone will fly to the location to pick it up.
The drone will take off with the SPS fully retracted, and the SPS will first be lowered when the drone arrives at the location, where the drone pilot will make confirmation. The SPS will lower itself down and target the landing location where it will use its thrusters to stabilise. When contact with the landing location has been made, the cargo will be dropped off, and the SPS will be hoisted back up under the drone. The drone pilot can then continue to the next task. The drone will only move when SPS is fully retracted. A sketch of the operating routine for cargo delivery missions to the nacelle of a wind turbine can be seen in Figure 1.2.

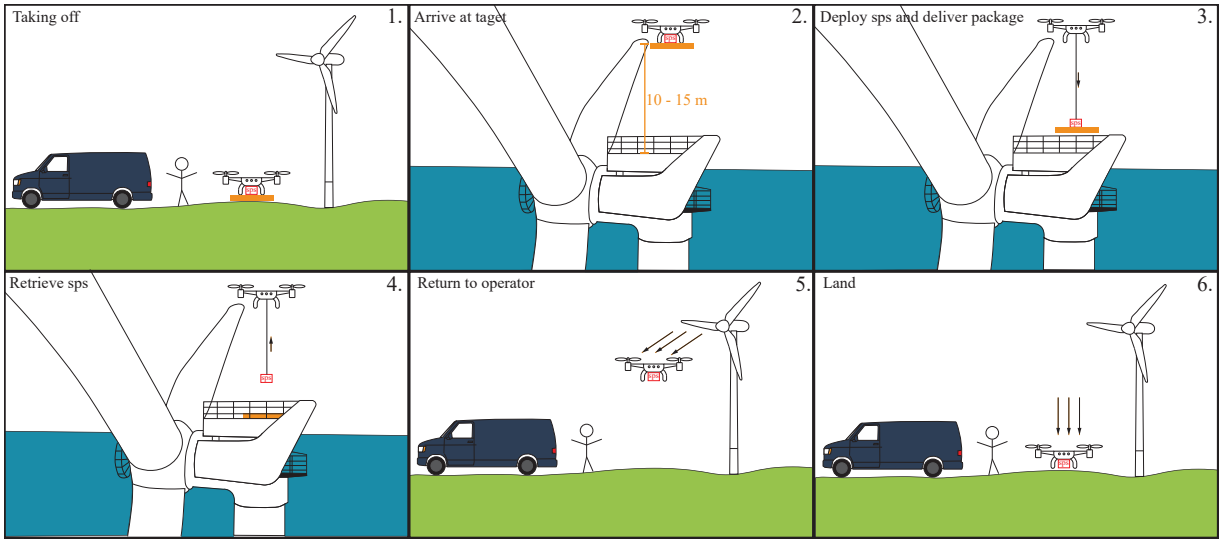


Figure 1.2. Operating routine for cargo delivery mission to onshore wind turbine.

Problem Analysis 2

This chapter will further investigate the initial problem stated in Chapter 1, where an expected increase in wind turbine construction demands a cheaper and more sustainable method of delivering a small payload. Airfligt proposes a flying crane which requires a method to stabilise the payload during delivery. This chapter will look at a simplified mechanical system and investigate the external forces to which the system is exposed. Furthermore, the chapter will look at how the system can counteract the forces. The attitude must be measured to control the system. Therefore, the necessary sensors are also investigated. Lastly, research on the DJI FlyCard 30 is conducted to better understand its specifications and capabilities.

2.1 System description

The drone with a payload lowered below it can be described as a 3D dynamically mounted double pendulum. The drone has 6 degrees of freedom (DOF) (3 rotational, 3 translational). The payload also has 6 DOF (3 rotational, 3 translational), where it can swing independently from the wire. A wire connects the drone and payload with 5 DOF (2 rotational, 3 translational). Due to its length, the wire cannot transfer a twist. This results in a system with 17 DOF, which can be simplified to a 12 DOF system by neglecting the wire. The wire can be neglected because its mass and inertia are much lower than the payload. Therefore, it can be assumed that the wire will follow the payload swing, resulting in the payload being perpendicular to the wire and always in tension. The described model is elaborated in further detail in Section 5.3. A simplified 2D model is illustrated in Figure 2.1 to investigate the system and the external forces applied.

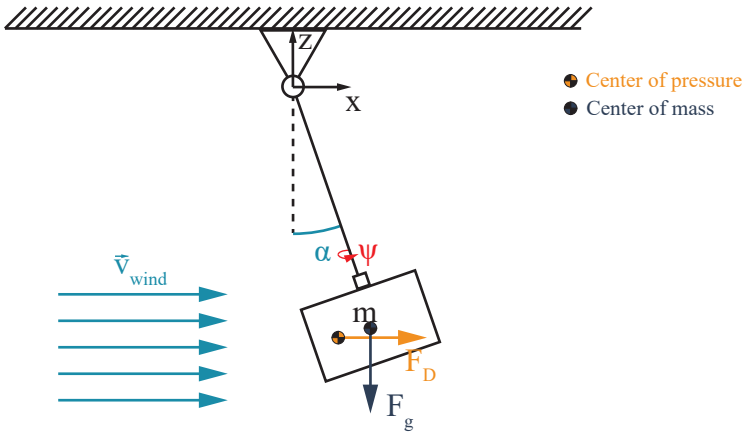


Figure 2.1. Sketch of the simplified system, where the forces and coordinates can be seen

The position of the payload can be identified by the generalised coordinate α describing the swing

angle. In 3D, the payload would be able to rotate around the wire, changing the aerodynamic properties depending on the orientation of the payload. The orientation of the payload is described by the angle Ψ, and its influence will also be investigated. The 2D model, therefore, has the following assumptions:

- **Drone is stationary:**
  The drone is assumed to be stationary, so only aerodynamic forces are present. In Table 2.4 a hovering accuracy of $\pm 10\ cm$ is specified at a maximum wind speed of 12 $\frac{m}{s}$ for the DJI FlyCart 30, validating the assumption that drone movement can be neglected due to relatively small movement compared to the length of the wire.
- **COM of the payload is directly below the fixation point:**
  It is assumed that the payload is balanced, so the COM is located directly below the wires' fixation point.
- **Wire Always in tension:**
  It is assumed that the wire is always in tension. Since the COM is located directly below the wire fixation point, the payload should always be perpendicular to it. With the drone assumed to be stationary, the payload rotates around a hinged support located on the drone.
- α **is a small angle:**
  Some things can be simplified if α is a small angle. When the payload rotates with α, the change of aerodynamic properties varies little and is negated. The drag forces are thereby calculated based on the aerodynamic properties at $\alpha = 0°$, which can be affected by the payloads' orientation Ψ about the wire. The small angle α also means that the moments on the payload from the aerodynamic forces that affect the centre of pressure (COP) will be small and can be negated.

From these assumptions, the system can be described as a 1 DOF pendulum system described by the angle α, where ψ can be changed to alter the aerodynamic property of the system. The pendulum is affected by gravitational forces F_g and aerodynamic forces such as drag F_D and lift F_L, which are calculated based on $\alpha = 0°$ and some arbitrary orientation Ψ.

The forces affecting the simplified 2D model as seen in Figure 2.1 are drag and lift, which affect the COP, where gravity affects the COM. The equations for calculating drag and lift can be found in equations (2.1) (Mandø and Rosendahl 2010, p. 7) and (2.2) (Jensen and Hærvig 2014, p. 25) respectively. The gravitational force is calculated by equation (2.3).

$$\vec{F_D} = \frac{1}{2} \cdot C_D \cdot \rho \cdot A_p \cdot |\vec{v}| \cdot \vec{v} \tag{2.1}$$

$$\vec{F_L} = \frac{1}{2} \cdot C_L \cdot \rho \cdot A_p \cdot [A] \cdot |\vec{v}| \cdot \vec{v} \tag{2.2}$$

$$\vec{F_g} = m \cdot \vec{g} \tag{2.3}$$

Equations (2.1) and (2.2) describe the aerodynamic forces as vectors that show both direction and amplitude. $|\vec{v}| \cdot \vec{v}$ is the relative velocity of the wind to the payload squared while keeping

the direction information intact. C_D and C_L are dimensionless coefficients that describe several complex conditions such as shape, air conditions, and inclination (Benson n.d.). $[A]$ is a rotational matrix added for $\vec{F_L}$ to orient it perpendicular to the drag force $\vec{F_D}$. A_p is the projected reference area of the box orthogonal to $\vec{v}_{wind}$ this can be seen in Figure 2.5 but in the xy plane. ρ is the density of the air, m is the mass of the object, and $\vec{g}$ is the gravitational constant pointing along the z-axis. The external forces applied to the system are investigated in Section 2.2.

2.2 External force estimation on simple 2D model

The system is investigated in two scenarios to examine external forces: the deflection of α in the xz-plane and the moment about the wire created by the aerodynamic forces in the xy-plane. The forces calculated in this section will only be a rough estimate of the expected forces and will be used as a guide for the design in Chapter 4. The payload size is determined by the cargo case size of the DJI FlyCart 30 as described in Table 2.4.

Deflection in xz-plane

For simplicity, it has been decided to keep the angle α constant at 0° for all aerodynamic calculations in the xz-plane. This is because α is expected to be small and, therefore, have a negligible influence compared to the rotation Ψ. The payload can rotate about Ψ, resulting in a different projected area A_p depending on the angle of Ψ. A_p is described in Figure 2.5 and depends on the length and thickness of the payload. When $\Psi = 0°$ then $A_p = t \cdot h$, and when $\Psi = 90°$ then $A_p = l \cdot h$. The projected area A_p can be calculated by Equation (2.4) as a function of Ψ.

$$A_p(\Psi) = (|l \cdot sin(\Psi)| + |t \cdot cos(\Psi)|) \cdot h \tag{2.4}$$

Because of the complexity of determining the drag coefficient, a regression analysis has been used to estimate C_D using a lookup table from (Munsen et al. 2013, p. 529). The table is for 2D rectangles with sharp corners at Reynolds numbers 10^5, where the ratio between height and depth is calculated as $ratio = \frac{h}{d}$, which gives a corresponding drag coefficient. The regression results can be seen in Figure 2.2, where the analysis can be found in Appendix A.1.

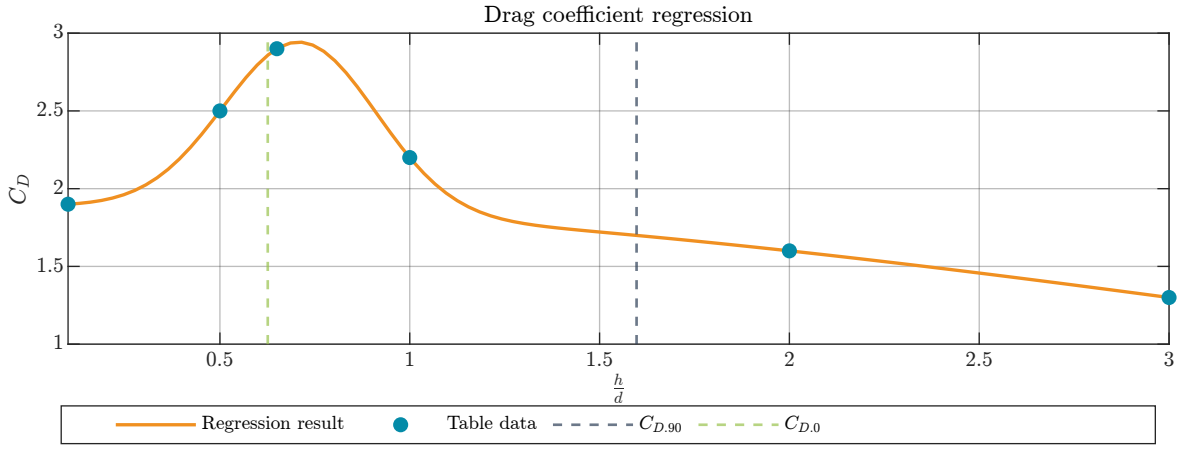


Figure 2.2. Regression results for C_D values, and datapoints used for regression analysis

By utilising a smoothing function, it is possible to estimate the drag coefficients to any angle of Ψ between 0° and 90° with the use of $C_{D.0}$ and $C_{D.90}$. The payload is a rectangle and can be split into four ídentical segments of 90° through symmetry. The smoothing functions can therefore be used for the full 360° rotation of the payload. From (Mandø and Rosendahl 2010), two smoothing functions by Pesavento and Rosendahl can be used. These can be seen in Table 2.1. From the regression results, $C_{D.0}$ is calculated by the ratio $\frac{l}{t} = 1.59$ and is found to be 1.7 and for $C_{D.90}$ the ratio is calculated as $\frac{t}{l} = 0.62$ and is found to be 2.9.

Author	Function
Pesavento	$C_D(\Psi) = C_{D.0} \cdot cos^2(\Psi) + C_{D.90} \cdot sin^2(\Psi)$
Rosendahl	$C_D(\Psi) = C_{D.0} + (C_{D.90} - C_{D.0}) \cdot sin^3(\lvert\Psi\rvert))$

Table 2.1. Pesavento's smoothing function for free-falling plate and Rosendahl's smoothing function for non-spherical particles (Mandø and Rosendahl 2010)

Figure 2.3 shows the calculated C_D values in the interval $\Psi = 0°$ to $\Psi = 90°$ with Pesavento and Rosendahl's smoothing functions. The corresponding forces are computed with equation (2.1) and (2.4). F_D is calculated at a relative velocity of the air equal to the DJI FlyCard 30 max wind speed resistance of 12 $\frac{m}{s}$ with a payload of 30 kg, see Table 2.4. The air density is set at $\rho = 1.293\ \frac{kg}{m^3}$ under the assumption that the air is dry, pure, at $0°C$, and 1 bar of pressure (NASA 2024). In Appendix A.2, the ratios for $C_{D.0}$ and $C_{D.90}$ have been calculated and the corresponding change in C_D by the smoothing functions if α were to be changed from 0° to 90°. Here, it can be seen that the assumption of calculating the aerodynamic forces at $\alpha = 0$ is valid when α is between $0 - 30°$.

The max forces from drag are found with the use of Pesavento at 85 N at $\Psi = 71.8°$, and with Rosendahl it is 83.6 N at $\Psi = 74.5°$. This can be seen in figure 2.3.

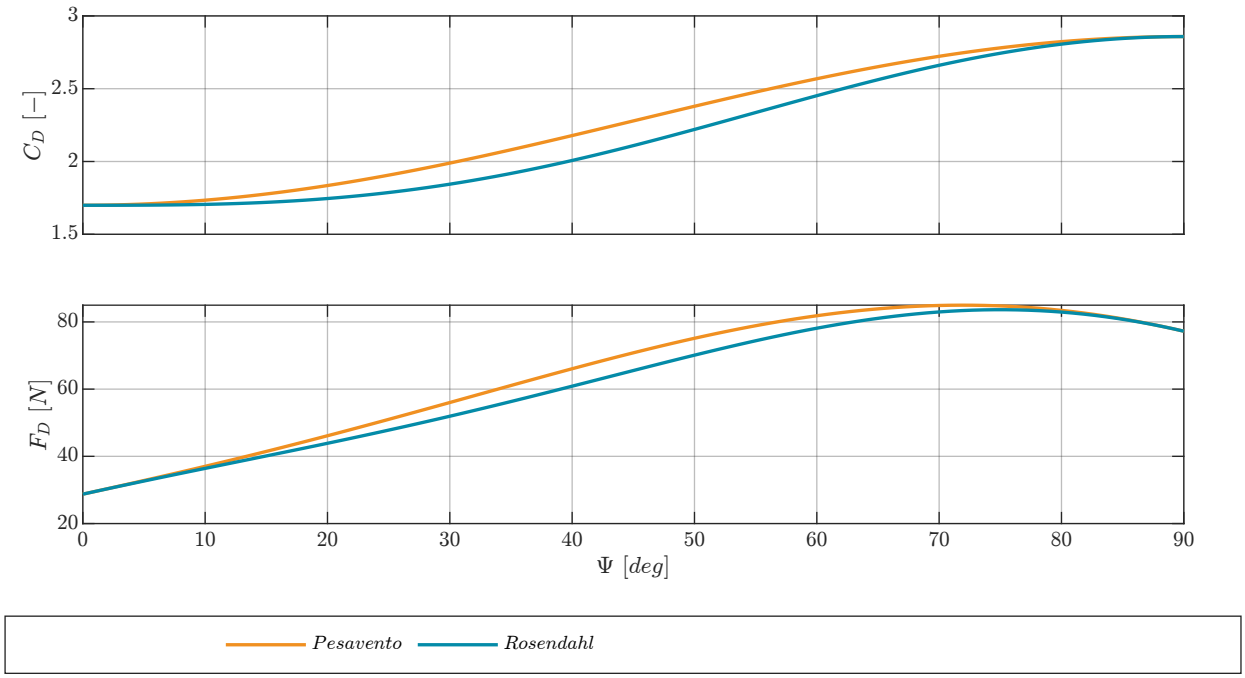


Figure 2.3. The drag coefficient and drag forces for a Ψ rotation

Estimating the maximum deflection in α when the maximum F_D is applied to the payload is interesting to investigate for further validation of the assumptions and expected maximum deflection. This is done by setting up an equilibrium for drag and gravity in the xz-plane. Drag F_D and gravity F_g will work against each other, resulting in the equilibrium shown in Equation (2.5). Figure 2.4 shows a sketch of the force contributions.

$$F_D \cdot cos(\alpha) - F_g \cdot sin(\alpha) = 0 \tag{2.5}$$

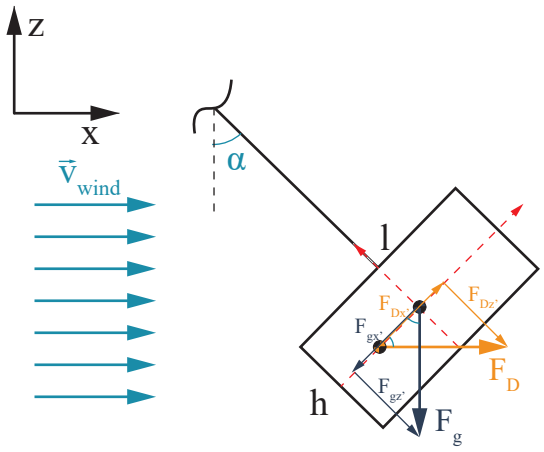


Figure 2.4. Forces in the xz-plane

The maximum deflection in α is calculated using the results from Pesavento at $\alpha_{max} = 16.1°$ and for Rosendahl $\alpha_{max} = 15.8°$. Both results are less than $\alpha = 30°$, validating it as acceptable to assume that α is small.

Moment in xy-plane

The payload is affected by rotations around the wire. The rotation occurs when a moment is applied around the COM. The moment contributions come from the aerodynamic forces affecting the COP, with a moment at the COM of length x_p. It would be interesting to find the moment applied to the payload. A sketch of the system in the xy-plane and the forces applied can be seen in Figure 2.5.

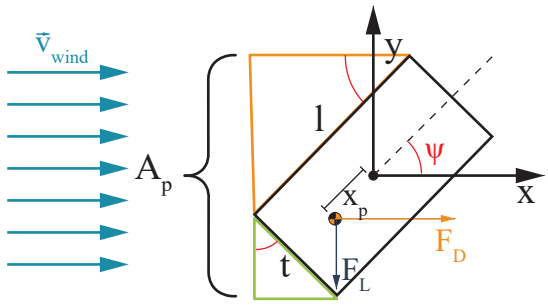


Figure 2.5. The project area A_p based on Ψ and forces in the xy-plane

Here, it is also assumed that $\alpha = 0$ and the wire force cancels out the gravitational forces. Therefore, the only remaining forces would be the aerodynamic forces that act in the COP. The

equation for torque induced by the aerodynamic forces, drag and lift, can be seen in equation (2.6).

$$\vec{\tau_w} = \vec{COP} \times (\vec{F_D} + \vec{F_L}) \tag{2.6}$$

The drag is known from Section 2.2, and the lift coefficient C_L is calculated from the C_D and Ψ in Equation (2.7) (Mandø and Rosendahl 2010, p. 8).

$$C_L = sin^2(\Psi) \cdot cos(\Psi) \cdot C_D \tag{2.7}$$

The lift coefficient and lift force are calculated in Appendix A.3 using Equation (2.7) and (2.2). The results are shown in Figure A.2. The results will be used for calculating the moment in the xy-plane for an angle Ψ.

The distance between COM and COP is called x_p. Using equation (2.8) by (Mandø and Rosendahl 2010), the distance from COM to COP is given by the length of the box and is dependent on the angle of attack Ψ. Based on the length x_p and the angle of attack Ψ, the vector from COM to COP can be calculated.

$$\frac{x_p}{l} = 0.25(1 - sin^3(\Psi)) \tag{2.8}$$

The vector $\vec{COP}$ points from the COM to the COP and has the length x_p. Using the equations for calculating F_D and F_L from $\Psi = 0°$ to 90°, the resulting torque is calculated by Equation (2.6). The results using both Presevento and Rosendahl's smoothing functions for calculating C_D are shown in Figure 2.6. Here, it can be seen that according to Pesavento's method, there occurs a maximum moment of 8.26 Nm at 43.5° and according to Rosendahl's, a max moment of 7.64 Nm at 44.5°.

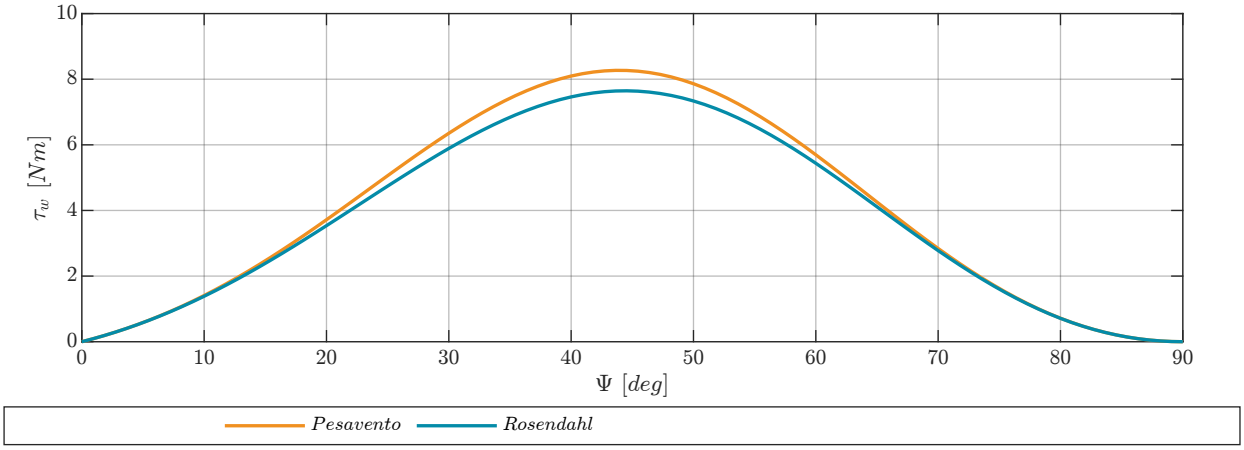


Figure 2.6. The moment about the z-axis as a function of Ψ

2.3 Thrust generation

In Section 2.2, it was identified that the wind force will affect the payload with a deflection in α and a rotation in Ψ. The forces were estimated to be $\approx 83 - 85$ N, and the moments could be up to $\approx 7.6 - 8.2$ Nm. To create a counterforce to cancel out the aerodynamic forces, thrust

is needed. As described by Newton's third law, thrust can be viewed as a reaction force. The thrust is generated by accelerating a mass in one direction, resulting in an equal force applied to the system in the opposite direction. Typical ways to generate thrust are propellers, jet engines, and rocket engines (NASA 2021). The latter two can be considered overkill for this report. Therefore, only propellers will be considered.

The thrust generated by a propeller can be described using momentum theory, a simplified aerodynamic theory that describes ideal propellers. This means that this theory does not account for the losses or geometry of the propeller. According to (Kumar 2021), there is typically an error of $10 - 15\%$ in the power calculations based on momentum theory.

When the propeller spins, it pushes air behind it, which generates thrust. This requires the transfer of energy from the propeller to the air. The air interacting with the propeller can be described as a finite volume in which mass, momentum, and energy conservation are considered. This volume can be seen in Figure 2.7 and is called a stream tube.

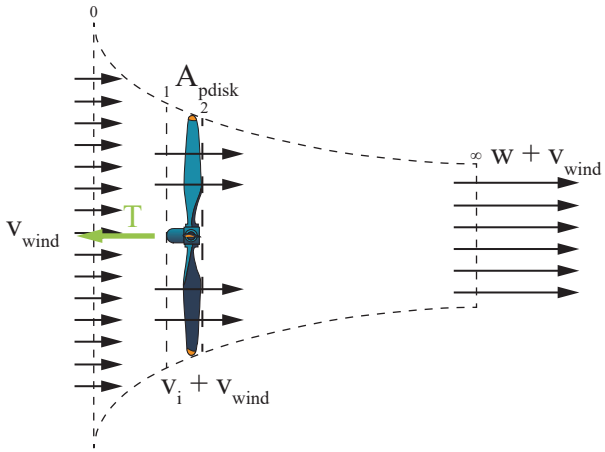


Figure 2.7. Stream tube around idealised propeller used for deriving the thrust equation

The mass entering the volume can be described by Equation (2.9). This equation describes the mass conservation $\dot{m}_{air}$ since the mass entering must be equal to the mass exiting the volume.

$$\dot{m}_{air} = \rho \cdot A_{pdisk}(v_{wind} + v_i) \tag{2.9}$$

In the equation, ρ is the density of the fluid and A_{pdisk} is the area of the propeller disk. v_i is the induced velocity of the fluid passing the propeller. v_{wind} is the velocity of the fluid entering the volume perpendicular to the propeller. The thrust created by the momentum can be described by Newton's second law, seen in equation (2.10).

$$T = \dot{m}_{air} \cdot w \tag{2.10}$$

Here, w is the downstream velocity and the mass described by $\dot{m}_{air}$ is exhausted, creating the thrust T. The energy of this system can be represented by Equation (2.11). In this expression, the first term describes the energy of the air exiting the volume and inducing thrust. The second term is the energy of the air entering the volume. Subtracting the two terms determines the

energy added to the air by the propeller. Because the mass changes over time, the equation describes the power added to the system.

$$\begin{aligned} T(v_{wind} + v_i) &= \frac{1}{2}\dot{m}_{air}(v_{wind} + w)^2 - \frac{1}{2}\dot{m}_{air} \cdot v_{wind}^2 \Rightarrow \\ T(v_{wind} + v_i) &= \frac{1}{2}\dot{m}_{air} \cdot w(w + 2v_{wind}) \end{aligned} \tag{2.11}$$

By dividing Equation (2.9) with (2.11), the following relation can be found $w = 2v_i$. This can be combined with Bernoulli's principle, which can describe the relation between a velocity in a slipstream and the pressure at a point. Bernoulli's principle determines the total pressure in the volume based on the sum of the static and dynamic pressure. It can be said that the pressure outside of the slipstream given by p_0 is static, and that the total pressure on the left side of the propeller is p_1 and the right side is p_2. The pressure in p_1 can be found by the velocity v_{wind}. The pressure in p_2 can be found by the velocity $(w + v_{wind})$. The pressure difference can be found over the propeller, by Equation (2.12). The pressure difference can be described as the thrust divided by the area of the propeller T/A_{pdisk}.

$$p_0 + \frac{1}{2}\rho(v_{wind} + w)^2 - p_0 + \frac{1}{2}\rho v_{wind}^2 = p_2 - p_1 = T/A_{pdisk} \tag{2.12}$$

In this expression, w can be replaced with $2v_i$ and solved for v_i, resulting in Equation (2.13).

$$v_i = -\frac{v_{wind}}{2} + \sqrt{\left(\frac{v_{wind}}{2}\right)^2 + \frac{T}{2\rho A_{pdisk}}} \tag{2.13}$$

As explained earlier, the power of the propeller to generate thrust is given by Equation (2.14).

$$P_T = T(v_{wind} + v_i) \tag{2.14}$$

Inserting Equation (2.13) into (2.14) provides an equation in which thrust depends on the power that the ideal propeller delivers.

2.4 BLDC - Motor

A motor is needed to generate the power required to produce thrust. For this purpose, a couple of *Brushless direct current* (BLDC) *Motors* have been made available for the report.

Although they are called DC motors, BLDC commutation is done electronically, often by using driving circuits. BLDCs are constructed with magnets on the rotor and windings on the stator. The available motors are radial-flux motors with an outer rotor configuration. A depiction of this configuration can be seen in Figure 2.8. BLDCs can vary in the number of phases, but the most typical is three-phase BLDC motors, which also coincide with the ones available for this report (Kim 2017, pp. 389–415).

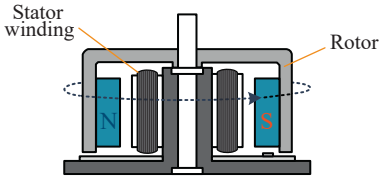


Figure 2.8. BLDC motor in a radial-flux configuration with an outer rotor

The construction of a BLDC is very similar to a *Permanent Magnet Synchronous Motor* (PMSM), where the main difference is the direction of the flux. A BLDC is designed to have a trapezoidal back-electromotive force (back-EMF) compared to the sinusoidal in PMSMs. A comparison of BLDC and PMSM back-EMF can be seen in Figure 2.9. A rectangular stator current must be applied to develop a constant torque in a BLDC compared to the sinusoidal in PMSMs (Kim 2017, pp. 389–415). It must be noted that for a Y-connected BLDC, the flat top is 120 deg and the slopes are 60 deg. For a Delta-connected BLDC, the flat top is 60 deg and the slopes are 120 deg (Lee, Seo, et al. 2015, pp. 167–168) (Lee, Song, et al. 2014, pp. 2202–2203). Figure 2.11 shows these back-EMF profiles.

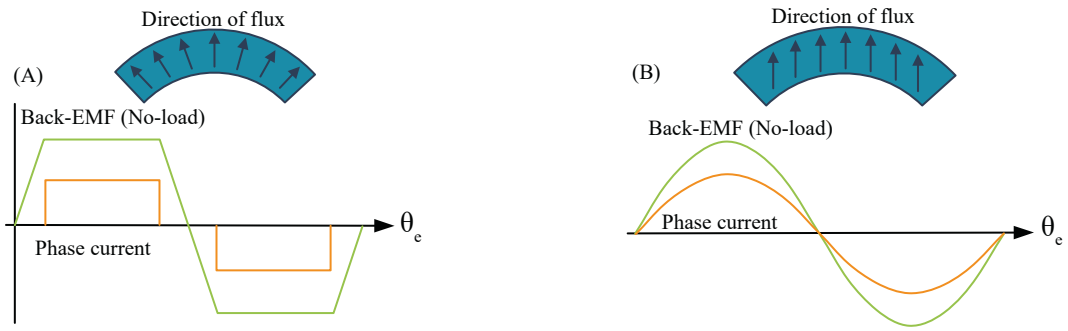


Figure 2.9. Back-EMF in a (A) BLDC (B) PMSM

This means that for a three-phase BLDC motor, the phase currents only flow in two phases simultaneously, compared to a PMSM, where currents flow in all three phases simultaneously. A BLDC typically uses a conduction interval of 120° compared to the conduction interval of 180° commonly used in PMSMs. Therefore, only two phases must be excited to operate a BLDC motor in a Y-connection with a 120° conduction interval. This gives six switching states where a DC voltage is applied over two phases simultaneously. The switching states need to be activated according to the position of the magnets in the rotor. A three-phase inverter is used to control the switching states. The inverter must be operated in a specific order to produce DC voltage in only two lines at a time (Kim 2017, pp. 389–415). A single swinging state is depicted in Figure 2.10 for a Y-connected motor. The same switching states are used for a delta-connected motor, and the voltage is applied similarly to a Y-connected motor.

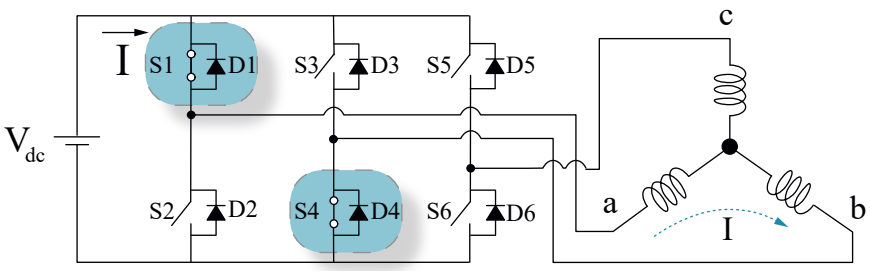


Figure 2.10. A Switching state in a three-phase inverter connected to a BLDC in a Y-connection (120° conduction interval)

In the delta connection, three phases are always excited, resulting in currents in all phases at any time during commutation. Even though the windings are configured differently for star and

delta, the input line currents share the same waveform for both winding types. A figure of all the switching states on a Delta-connected BLDC can be seen in Appendix D in Figure D.2 (Lee, Seo, et al. 2015, pp. 167–168).

To control the inverter and thereby the motor's commutation, some BLDCs have built-in Hall sensors to detect the rotor's magnetic field, and thereby its position, and activate the relevant switching state to rotate the rotor. The motors provided for this report do not have hall sensors. Therefore, sensorless commutation must be used. This works by measuring the back-EMF of the motor since this includes information about the magnetic flux that can be used to determine the position of the motor.(Kim 2017, pp. 389–415).

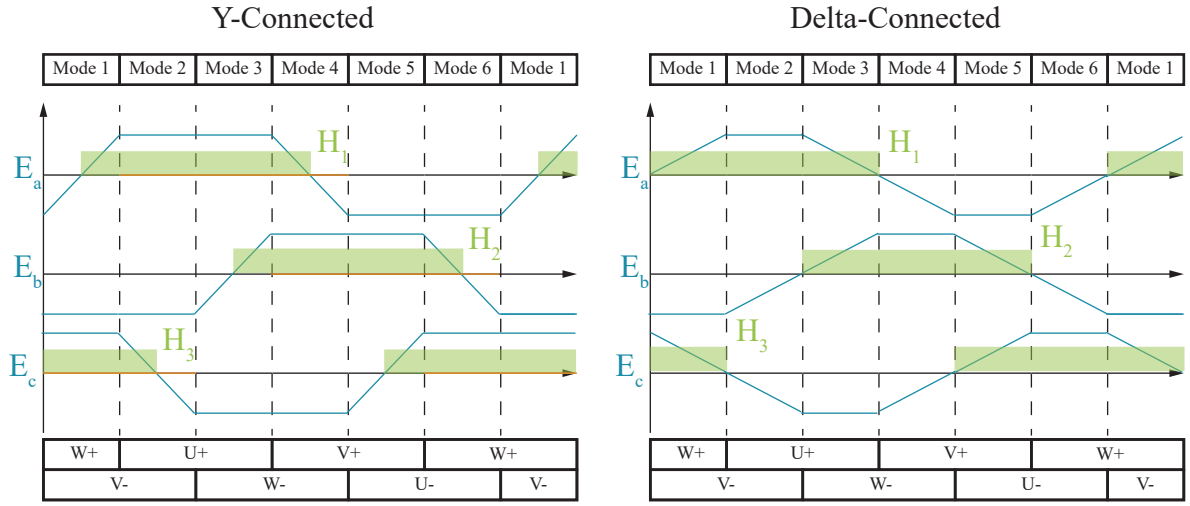


Figure 2.11. Hall sensor signal and back-EMF comparison between star and delta wound BLDC

When the rotor rotates, the back-EMF will cross 0 V when not powered, making it possible to use the zero crossing point (ZCP) to determine position. Therefore, a back-EMF sensing circuit can be used to determine position. For a Y-connected motor, the problem with this method is that the ZCP is 30 electrical degrees ahead of the commutations. This means the states are not activated at the perfect moment, making low-speed operation difficult (Kim 2017, pp. 389–415). For a Delta-connected motor, the ZCP coincides with the commutation (Lee, Seo, et al. 2015, p. 172). In Figure 2.11, the back-EMF profiles are shown by blue lines, the virtual Hall signals are the green boxes, and an orange line marks the proper Hall signal locations for the Y-connected motor, indicating the 30 electrical delay.

2.5 Sensors

As described in Section 2.2, the system is a pendulum where thrusters are necessary to stabilise the payload. To effectively obtain stability, sensors are required to measure the attitude of the pendulum. The sensors used to obtain attitude for position feedback are an accelerometer, a gyroscope, and a magnetometer.

2.5.1 Accelerometer

An accelerometer is a device that measures vibration and acceleration of motion. Different types of accelerometers excel in various use cases.

A piezoelectric accelerometer contains a sensing crystal to which a seismic weight is attached. An acceleration exerts a force on the crystal, generating an electric charge corresponding to the acceleration. Piezoelectric accelerometers are highly effective for measuring shocks and vibrations.

Piezoresistance accelerometers change their resistance when an acceleration is exerted. The rate of acceleration can be determined by measuring the change in resistance. Piezoresistance accelerometers are primarily used in high-amplitude applications such as vehicle crash testing.

Capacitive accelerometers measure the acceleration by measuring the change in capacitance due to the capacitor plates being squeezed by the exerted acceleration. Capacitive micromechanical system (MEMS) accelerometers are the most used in smartphones and other consumer electronics (Omega n.d.). It's not the most precise type of accelerometer and has difficulty sampling high-frequency acceleration. But it has an excellent price-to-performance ratio (Connectivity 2017).

Working principle of capacitive accelerometer

A capacitive accelerometer contains a known proof mass m_p and is connected to the sensor frame as a mass-spring-damper system. When an external force F_{ex} is exerted on the sensor frame, the proof mass will fall behind because of inertia, and thereby the proof mass has a relative position change Δd to the sensor frame from its original rest position d_{rest}. An electrode is connected to the proof mass, and two fixed electrodes are connected to the sensor frame, creating two capacitors. When proof mass moves in a direction, the capacitance in the capacitors changes, where one increases and the other decreases. The capacitance can be calculated by Equation (2.15) where the d_x is the distance between the fixed and movable electrodes. d_x changes with the position of the proof mass. ϵ is the dielectric permittivity. A sketch of the sensor frame and proof mass system for a capacitive accelerometer can be seen in Figure 2.12.

$$C_x = \epsilon \cdot \frac{A_{cap}}{d_x} \tag{2.15}$$

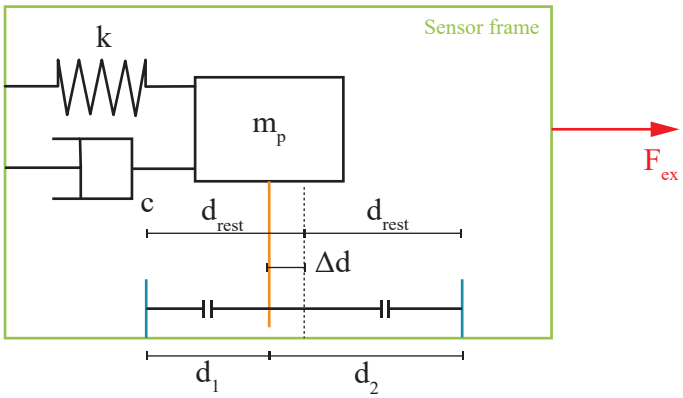


Figure 2.12. Mass-spring-damper structure where an external force is exerted on the sensor frame, resulting in a displacement Δd

By measuring the capacitance, Equation (2.15) can be calculated backwards to determine d_x and thereby Δd from which an acceleration can be calculated for the mass-spring-damper system. To measure the change in capacitance, a technique called synchronous demodulation is applied, where a simplified electrical diagram of the circuit used by the method can be seen in Figure 2.13.

A square wave is applied to the two fixed electrodes with the same amplitude but with a 180° phase shift. The capacitors are connected as a capacitive voltage divider. When the proof mass is in its resting position, the output at V_{bridge} is a DC signal, and a capacitive voltage divider cannot pass a DC signal, resulting in a voltage at $V_{bridge} = 0$. Moving the proof mass closer to one of the fixed electrodes means the voltage at V_{bridge} is no longer DC, and a voltage can be passed through. The fixed electrode, to which the proof mass is closer, will excite a larger voltage and be in phase with V_{bridge}. This way, it is possible to know the direction of the acceleration by comparing which square wave V_{bridge} is in phase with, and the amplitude is a linear function of the displacement of the mass.

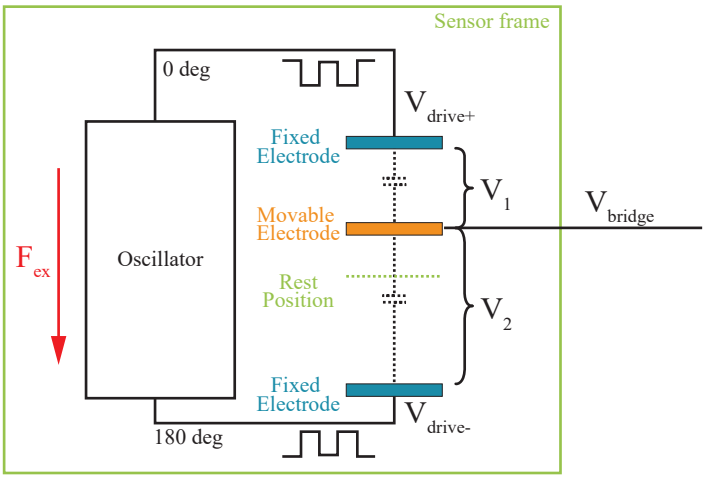


Figure 2.13. Simplified electrical circuit for utilising synchronous demodulation technique.

The amplitude of V_{bridge} can be derived based on the system being a capacitive voltage divider. A square wave AC signal goes through the capacitors, which will build up a reactance. The reactance for a capacitor can be calculated with Equation (2.16). For a capacitive voltage divider, the relationship between input and output for both V_{drive+} and V_{drive-} can be derived by Equation (2.17). The derivation of the voltage divider will only be one for V_1, but the same process can be done for V_2.

$$X_{Cx} = \frac{1}{2 \cdot \pi \cdot f \cdot C_x} \tag{2.16}$$

$$\frac{V_1}{V_{drive+}} = \frac{X_{C2}}{X_{C1} + X_{C2}} \tag{2.17}$$

Combining Equation (2.16) and (2.17), Equation (2.18) can be derived.

$$\frac{V_1}{V_{drive+}} = \frac{\frac{1}{2 \cdot \pi \cdot f \cdot C_2}}{\frac{1}{2 \cdot \pi \cdot f \cdot C_1} + \frac{1}{2 \cdot \pi \cdot f \cdot C_2}} \rightarrow \frac{C_2}{C_1 + C_2} \tag{2.18}$$

The capacitance is a function of the distance d_x between the electrodes, which can be calculated by equation (2.15). Where ϵ is the permittivity of the dielectric material between the electrodes and A_{cap} is the cross-sectional area of the electrodes. If both ϵ and A_{cap} are equal in both capacitors, they will cancel each other out when Equation (2.15) is inserted into Equation (2.18), which results in Equation (2.19)

$$V_1 = \frac{d_1}{d_1 + d_2} \cdot V_{drive+} \tag{2.19}$$

Where

$$\begin{aligned} d_1 &= d_{rest} - \Delta d \\ d_2 &= d_{rest} + \Delta d \\ V_{drive+} &= -V_{drive-} \end{aligned} \tag{2.20}$$

The voltage at V_{bridge} is then the sum of V_1 and V_2, which means that the voltage at V_{bridge} can be calculated by Equation (2.21).

$$V_{bridge} = V_1 + V_2 = V_{drive+} \cdot \frac{d_{rest} + \Delta d}{2 \cdot d_{rest}} - V_{drive+} \cdot \frac{d_{rest} - \Delta d}{2 \cdot d_{rest}} \rightarrow \frac{\Delta d}{d_{rest}} \cdot V_{drive+} \tag{2.21}$$

By utilising two capacitors, a linear relation between V_{drive+} and V_{bridge} is possible through the movement of the mass. From Equation (2.21), Δd can be calculated and used to calculate the acceleration of the mass-spring-damper system (Dr. Arar 2021).

2.5.2 Gyroscrope

Gyroscopes are sensors that can sense angular velocity if the frame on which the Gyroscopes are mounted is rotating. Different types of gyroscopes operate on different physical principles. Four types are particularly used. These are mechanical gyroscopes, optical gyroscopes, fiberoptical gyroscopes and MEMS gyroscopes. These types of gyroscopes differ in terms of their performance and price. The most expensive and accurate gyroscopes are mechanical gyroscopes, the more common gyroscopes are MEMS gyroscopes that are cheaper and are often used in consumer hardware and robotics (Passaro et al. 2017). The workings of a MEMS gyroscope will be explained since this is used in this report.

MEMS gyroscope

A MEMS gyroscope is often constructed as a mass-spring-damper system. This system is located on a frame that rotates with a given angular velocity Ω_z. This angular velocity makes the fictitious coriolis and some implicated centripetal and tangential accelerations affect the mass-spring-damper system. The absolute acceleration from these effects can, according to (Cao 2023, pp. 21–26), be found from Equation (2.22) when the frame is only rotating about the z-axis.

$$\begin{bmatrix} a_x \\ a_y \end{bmatrix} = \underbrace{\begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix}}_{Relative} + \underbrace{\begin{bmatrix} 0 & -2\Omega_z \\ 2\Omega_z & 0 \end{bmatrix}}_{Coriolis} \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} + \underbrace{\begin{bmatrix} -\Omega_z^2 & -\dot{\Omega}_z \\ \dot{\Omega}_z & -\Omega_z^2 \end{bmatrix}}_{Centripetal\ \&\ Tangential} \begin{bmatrix} x \\ y \end{bmatrix} \tag{2.22}$$

Here, a relative acceleration is an acceleration that occurs locally in the frame. The Coriolis acceleration is determined from $a_c = 2\Omega_z \times [\dot{x}\ \dot{y}]^T$. The centripetal and tangential accelerations are determined based on the local location of the affected object, which is the mass-spring-damper system that can be seen in Figure 2.14

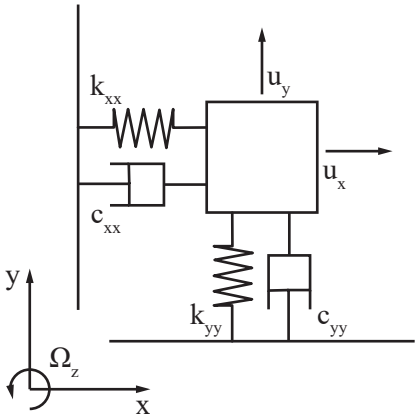


Figure 2.14. Mass-spring-damper system of a MEMS gyroscope

From this system, the forces that affect the mass can be found. First, the forces resulting from the angular velocity are found by Newton's second law. Then there are the forces from the spring and damper. A driving force can be added to both the x- and y-axes; these are given by u_x and u_y. This results in Equation (2.23) for the mass block.

$$\begin{bmatrix} u_x \\ u_y \end{bmatrix} = m_p \begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} + \begin{bmatrix} c_x & -2m_p\Omega_z \\ 2m_p\Omega_z & c_y \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} + \begin{bmatrix} k_x - m_p\Omega_z^2 & -m_p\dot{\Omega}_z \\ m_p\dot{\Omega}_z & k_y - m_p\Omega_z^2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{2.23}$$

In this equation, some simplifications can be made. It can be assumed that the displacement in the x and y-axis is very small, and if $m_p \cdot \Omega_z^2 << k$ then the following terms can be neglected $m_p\Omega_z^2 x$, $m_p\Omega_z^2 y$, $m_p\dot{\Omega}_z x$, $m_p\dot{\Omega}_z y$ and if the driving force u_x is very large $2m_p\Omega_z\dot{y}$ can be neglected. If the driving force u_y is set to 0, Equation (2.24) can be expressed.

$$\begin{cases} m_p\ddot{x} + c_x\dot{x} + k_x x = u_x \\ m_p\ddot{y} + c_y\dot{y} + k_y y = -2m_p\Omega_z\dot{x} \end{cases} \tag{2.24}$$

This equation shows that the mass only moves in the y direction if an angular velocity is present, and the angular velocity will scale the force acting in the y direction. The angular velocity can therefore be determined based on the amplitude of the y-axis excitation. To get a 3-dimensional gyroscope, three are placed inside the chip to sense the angular velocity in 3 dimensions.

2.5.3 Magnetometer

A magnetometer is a sensor that can detect and measure magnetic fields. In general, magnetometers are split into two variants: a scalar magnetometer that can precisely measure the strength of the magnetic field expressed in flux and vector magnetometers that measure, as the name implies, the direction and magnitude of a magnetic field. Since a magnetometer in this project will be used for navigation, vector magnetometers will be investigated further.

To create a vector magnetometer, magnetoresistive (MR) sensors are often utilised for multi-axis magnetic sensors in a single microchip according to (Sreevidya et al. 2018, pp. 1–2). They work by exploiting the magnetoresistive effect, which is that the resistance of a current-carrying material changes when it is exposed to a magnetic field. Often, Anisotropic Magnetoresistive (AMR) sensors are used, which are MR sensors where the resistance property of the current-carrying material differs based on the angle between the magnetic field and the direction of the current running through the material.

The AMR sensor consists of a thin ferromagnetic film, e.g. $Ni_{81}Fe_{19}$ (permalloy). The film is in a strip geometry, when the current I flows along the length of the strip, a spontaneous magnetisation M_{mag} occurs in the same direction. So that the magnetisation and current are parallel, this direction will be considered the x-axis, see Figure 2.15. When an external magnetic field is applied in the y-axis H_y, the magnetisation will change the angle between the current and magnetisation, causing the resistance value to change. The change of resistance can be found by $\Delta R = R_{\theta=0^\circ} - R_{\theta=90^\circ}$ where $R_{\theta=0^\circ}$ is the maximum resistance when the current and magnetisation are parallel and $R_{\theta=90^\circ}$ is the minimum resistance when the current and magnetisation is perpendicular. According to (Sreevidya et al. 2018, pp. 1–2), it is used to find the resistance of the film when the magnetisation is rotated uniformly over the angle. This can be seen in Equation (2.25).

$$R_\theta = R_{\theta=0^\circ} - \Delta R \cdot \left(\frac{H_y^2}{H_k^2}\right) \tag{2.25}$$

Here H_k is the sum of the material anisotropy field H_{ko} and shape anisotropy field $H_d = (t/w) \cdot M_s$ where M_s is the saturation magnetisation, t is the thickness of the film, and w is the width.

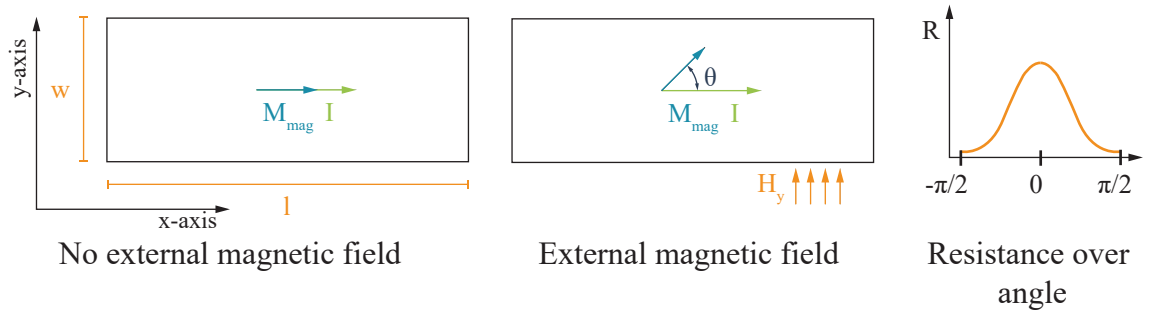


Figure 2.15. Sketch of AMR sensor with and without External magnetic field and Resistance over angle behaviour

This change in resistance is the basis for the AMR sensor. By changing the layout of the strip, the sensor can be further improved to get a more linear change in resistance and make it bidirectional. Three of these sensors can then be mounted in a microchip to get a three-axis magnetometer, which can be used to determine the direction of the magnetic north pole if no significant disturbances are present.

2.6 Methoeds of attitude estimation

None of the sensors described in Section 2.5 can directly sense the attitude, but can be used to estimate the attitude of the rigid body. The different sensors have different drawbacks when estimating the attitude, so often a combination of various sensors is needed to obtain an improved estimate of the attitude. This is frequently referred to as sensor fusion.

The attitude can be expressed in either a three-component vector representation or a four-component quaternion representation. The main disadvantage of the vector representation is discontinuities or singularities, making it challenging to make filters when the middle angle in the rotation sequence comes near 90°, gimbal lock occurs (F. Landis Markley 2019, pp. 236–240). It also requires a lot of trigonometric evaluations, making it more computationally heavy than using the quaternion representation. The main advantage of the three-component vector representation is that it is easier to interpret. The quaternion representation allows for a less computationally heavy estimation of the attitude, with the advantage of removing many singularity problems.

This report will use vector representation to keep attitude estimation intuitive and straightforward. This is done since large angles near 90° are not expected to occur in either roll ϕ or pitch θ. The only angle that can exceed 90° is the yaw Ψ. Using vector representation should allow for simpler filters that give a reasonable attitude estimation. It is just essential to ensure that possible singularities are correctly handled.

In the vector representation, the attitude is often given in terms of Euler angles expressed in terms of roll ϕ, pitch θ, and yaw Ψ, which can be seen depicted in Figure 2.16. A complementary or Kalman filter can blend the measurements from an accelerometer, magnetometer, and gyroscope (AHRS 2024) to estimate the attitude in Euler's angles.

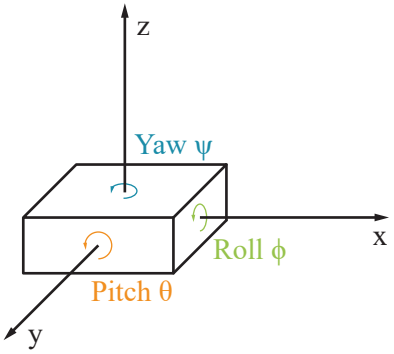


Figure 2.16. Definition of Roll, Pitch & Yaw

Attitude estimation from Accelerometer and magnetometer

A 3-axis accelerometer can estimate the rigid body's roll ϕ and the pitch θ, but not yaw Ψ. With Equation (2.26), the measured accelerometer data $a_{measured} = \begin{bmatrix} a_x & a_y & a_z \end{bmatrix}^T$ is used to calculate the roll and pitch angles. It can be noted that arctan2 allows for angles between $-\pi$ & π, which means that a discontinuity will be present at $-\pi$ & π.

$$\Theta_{am} = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^T = \begin{bmatrix} arctan2(a_y, a_z) & arctan2(-a_x, \sqrt{a_y^2 + a_z^2}) & 0 \end{bmatrix}^T \tag{2.26}$$

To find the yaw angle the magnetometer data $m_{messured} = \begin{bmatrix} m_x & m_y & m_z \end{bmatrix}^T$ can be used. The magnetometer data must first be compensated for roll and pitch to do this. This can be done by equation (2.27) (AHRS 2024).

$$\vec{b} = \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} = \begin{bmatrix} cos(\phi) & sin(\phi)sin(\theta) & sin(\phi)cos(\theta) \\ 0 & cos(\theta) & -sin(\theta) \\ -sin(\phi) & cos(\phi)sin(\theta) & cos(\phi)cos(\theta) \end{bmatrix} \begin{bmatrix} m_x \\ m_y \\ m_z \end{bmatrix} \tag{2.27}$$

The yaw can be obtained from equation (2.28) using the compensated magnetometer data.

$$\psi = arctan2(-b_y, b_x) \tag{2.28}$$

From this, the attitude can be estimated using an accelerometer and a magnetometer. However, it should be noted that this method often has high-frequency noise that needs to be addressed.

Attitude estimation from gyroscope

The attitude can also be determined from a gyroscope using the angular velocity to calculate the angle based on numerical integration. Here the measured angular velocity $\omega = \begin{bmatrix} \omega_x & \omega_y & \omega_z \end{bmatrix}^T$ and the time between samples Δt can be used to find the attitude of the rigid body the sensor is mounted on. This equation can be seen in equation (2.29).

$$\Theta_\omega = \begin{bmatrix} \phi_i \\ \theta_i \\ \psi_i \end{bmatrix} = \begin{bmatrix} \phi i - 1 + \omega_x \Delta t \\ \theta_{i-1} + \omega_y \Delta t \\ \psi_{i-1} + \omega_z \Delta t \end{bmatrix} \tag{2.29}$$

The attitude measurements are obtained from this, but these measurements often suffer from low-frequency drift due to measurement bias. This means that if data are collected over a more extended period, the data starts to deviate slowly from the correct angles.

Complementary Filter

To combat the weakness in the two methods, a complementary Filter can be used. This filter takes the attitude measurements from the accelerometer and magnetometer. It blends it with the attitude measurements from the gyroscope, eliminating the drift from the gyroscope and the noise from the accelerometer. The equation for the Complementary filter can be seen in Equation (2.30) where υ is a floating value in the range $[0, 1]$.

$$\Theta_{com} = \upsilon\Theta_\omega + (1 - \upsilon)\Theta_{am} \tag{2.30}$$

Here υ controls how much of each measurement method is used. If υ is 0, only the accelerometer data is used, and if υ is 1, then only the gyroscope data is used. The disadvantage of the complementary filter is that it requires choosing the value of υ, which is often done through trial and error.

Kalman Filter

The Kalman filter is shown as a mathematical model in discrete time, which allows for estimating system states. The complementary filter and Kalman filter essentially do the same thing. Still, the Kalman filter, through knowledge of the system and measurement noise, calculates Kalman

gain K, which from the complementary filter equals the blending value v. The Kalman filter combines a model and measurements to obtain the optimal state estimation. The model might not fully represent the system, e.g., simplification and neglect of disturbances. Therefore, the model has an uncertainty in its prediction, which can be defined through a zero-mean Gaussian distribution with a variance Q_v. Likewise, the measurement also has noise and errors, which are also represented through a zero-mean Gaussian distribution with a variance of R_v. The Kalman filter merges the model and measurement variances to obtain the optimal state prediction through the Kalman gain K_v. Figure 2.17 shows how the Kalman filter merges the prediction and measurement to obtain the optimal state estimation (Mathworks 2017) (Scherr et al. 2014).

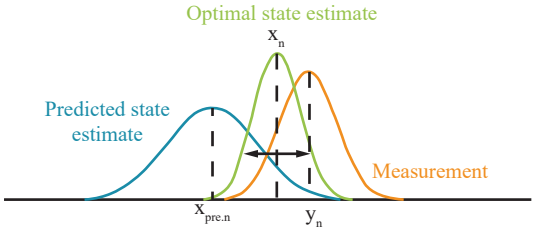


Figure 2.17. The messurment distabution in a Kalman filter

The Kalman filter is represented as a state space model $\dot{x} = Ax + Bu$ and $y = Cx$. The prediction state shown in Equation (2.31) applies the model and the input u_n to obtain an initial prediction of the states x.

$$x_{pre.n} = A \cdot x_{n-1} + B \cdot u_n \tag{2.31}$$

The last step in the prediction is to predict the covariance matrix P, which is a prediction of the combined model error by using the state space model and the model variance Q_v, and is shown in Equation (2.32).

$$P_{pre.n} = A \cdot P_{n-1} \cdot A^T + Q_v \tag{2.32}$$

With the states and covariance predicted, the predictions can be updated through measurements. The Kalman gain is calculated in Equation (2.33), which is done with the predicted covariance matrix P_{pre} and the observation matrix H. H is a matrix which transforms the state vector x_n to the measurements y_n.

$$K_{v.n} = P_{pre.n} \cdot H^T \cdot (H \cdot P_{pre.n} \cdot H^T + R_v)^{-1} \tag{2.33}$$

With the Kalman gain calculated, the states can be updated to obtain the optimal state estimate. This is done by updating the predicted state $x_{n.pre}$ with the Kalman gain multiplied by how well the state prediction filters the measurements. The state update is shown in Equation (2.34).

$$x_n = x_{pre.n} + K_{v.n} \cdot (y_n - H \cdot x_{pre.n}) \tag{2.34}$$

The last update step is to use the Kalman gain to update the covariance matrix P_n as shown in Equation (2.35).

$$P_n = (I - K_{v.n} \cdot H) \cdot P_{pre.n} \tag{2.35}$$

The gyroscope measurements in u_n to calculate the state prediction $x_{pre.n}$, which then can be updated by the accelerometer and magnetometer measurement data y_n to update the optimal state prediction x_n. By tuning the Q_v and R_v matrices, the "trust" in the sensors can be modified (Scherr et al. 2014). Extensions to the Kalman filter exist to obtain more complex state predictions, one of which is the Extended Kalman filter and is explored further in Chapter 7.

2.7 DJI FlyCart 30

Airflight wants to incorporate the SPS onto a DJI FlyCart 30 drone, although it will not be limited to it. The FlyCart 30 drone is a commercially available heavy lifter that can go long distances. The drone can be configured in two modes, with a single battery or dual batteries. The dual battery has a longer flight time but with a lower max payload capacity, whereas the single battery mode has a greater payload capacity with a shorter flight time. The specifications for the two modes can be found in Table 2.2 for single battery mode and Table 2.3 for dual battery mode. Table 2.4 contains general data for FlyCart 30 for both single and dual battery configurations. The data specified is gathered from (DJI n.d.), which was collected in controlled conditions at zero altitude and a windless environment.

Single battery specifications	value	unit
Payload capacity	40	kg
Flight distance w/o payload	12	km
Flight distance w. payload	8	km
Hovering endurance w/o payload	15	min
Hovering endurance w payload	8	min
Max flight time w. payload	9	min

Table 2.2. DJI Flycart 30 specifications in single battery configuration. Note all w. payload data is at max payload capacity of 40kg (DJI n.d.).

Dual battery specifications	value	unit
Payload capacity	30	kg
Flight distance w/o payload	28	km
Flight distance w. payload	16	km
Hovering endurance w/o payload	29	min
Hovering endurance w payload	18	min
Max flight time w. payload	18	min

Table 2.3. DJI Flycart 30 specifications in dual battery configuration. Note all w. payload data is at max payload capacity of 30kg (DJI n.d.).

Flight specifications	value	unit
Mass w. dual batteries	65	kg
Max horizontal speed w. 30kg payload	20	$\frac{m}{s}$
Cruising speed	15	$\frac{m}{s}$
Max wind speed resistance w. 30kg payload	12	$\frac{m}{s}$
Hovering accuracy with RTK - vertical[1]	±10	cm
Hovering accuracy with RTK - horizontal[1]	±10	cm
Max flight altitude w/o payload	6000	m
Altitude detection	1.5-200	m
Cargo case size (l/t/h)	754/472/385	mm
Operating temperature	-20 - 45	°C

[1] RTK is an abbreviation for Real-Time Kinematics. A strong Global Navigation Satellite System (GNSS) signal is required to achieve this accuracy.

Table 2.4. General flight specifications for DJI FlyCart 30 (DJI n.d.)

From Table 2.4, it can be seen that the Flycart 30 drone has a hovering accuracy of $\pm 10\ cm$, proving the assumption that the drone is stationary in section 2.1. This means that the drone should not move much while hovering. At the same time, it is described that the drone can resist an applied wind speed of 12 $\frac{m}{s}$, meaning that the drone should be able to hold its position at wind speeds up to 12 $\frac{m}{s}$.

Problem Statement 3

This chapter defines a problem statement, based on chapter 2 it can be seen that when the payload hangs in a wire aerodynamic forces can affect the payload these forces are strong enough that they can result in a deflection of up to 16° *at a wind of the maximum windspeed of* $12\ \frac{m}{s}$*. Depending on the payload's yaw angle, torque can also be present, leading to the payload rotating about the wire. A thrust needs to be applied to cancel out these forces so that the payload can be stabilised. To do this, the system needs to know its attitude, therefore, various sensor devices have been researched to establish control feedback. The system should be designed to fit multiple drones, but this report will focus on the DJI Flychart 30, the primary drone system Airflight works with. From this, the following problem statement can be defined:*

How can a single wire stabilised payload system be implemented on a DJI FlyCard 30?

To implement a single-wire stabilised payload system on the DJI FlyCard 30, the following questions also need to be answered:

- How can a single-wire stabilised payload system be designed per Airflight's specifications for a DJI FlyCard 30?
- How can the designed system be modelled for controller design and implemented on the stabilised payload system?

3.1 Limitations

Because time and workforce are limited, not everything can be achieved within the report's timeframe. The scope of the report, therefore, needs to be limited so that it matches the timeframe. It has thus been decided to focus on the core functionality of the SPS system that also matches the study curriculum. The report focuses on designing a prototype system as a proof of concept to validate the controller and design choices as a possible solution for the problem specified in Chapter 1.

The following limitations are applied:

- Not fully automatic
- No Hoist system

This means that the system does not need to be fully automatic in finding a payload target and automatically adjusts the position of the SPS to match the payload's position accordingly. The Hoist system does not need development since a system already exists for the DJI FlyCart 30. Communication between the SPS and FlyCart 30 would be required, but the drone is not available for this report and exceeds the budget of the report.

Design 4

This chapter will describe the design process of the SPS prototype. To evaluate the design ideas for the prototype, design requirements with corresponding engineering requirements are made and subsequently weighted by Airflight and its customers. The SPS will be split into multiple design elements that are used to generate concepts, which, when combined, will result in full solutions for the SPS prototype. The requirements will be the basis for the evaluation of the prototype solutions. The best-scoring solution will be expanded upon and manufactured.

4.1 Requirements and pecifications

Table 4.1 lists all requirements set for the design of the SPS, and a complete description of all the requirements can be found in Appendix B.1. The table also contains weights for the customer and the manufacturer based on the importance of each requirement. Airflight weighs the customer and manufacturer as they speak with potential customers and know their priorities. The customer has the most significant influence on weighing the requirements as they are the source of income and therefore their weighting is 60% of the weighted sum. The manufacturer is the remaining 40% of the weighting, as the weighting focuses on producing a complete product available for a broad customer base. The weighted sum is the weighting used when calculating the importance of the engineering requirements.

#	Requirments	Weight customer 0.6	Weighted manufacturer 0.4	Weight sum	Normalised weight 1
1	Damp pendlum osilations	11	8	9.8	0.098
2	Land on target	15	12	13.8	0.138
3	Measure attitude	4	8	5.6	0.056
4	Control position	13	11	12.2	0.122
5	Maintenance	4	3	3.6	0.036
6	Minimal input	8	6	7.2	0.072
7	Motor position	0	6	2.4	0.024
8	Compatibility	5	12	7.8	0.078
9	Maximise Payload capacity	13	8	11	0.110
10	Fit within DJI FlyCart30 limitations	10	10	10	0.100
11	Price	10	7	8.8	0.088
12	Safety	7	9	7.8	0.078
	Sum:	100	100	100	1

Table 4.1. Requirements and weight by customers and manufacturer

The customer's prioritisation in Table 4.1 extensively focuses on value for the money. Although the price is not weighted the highest, the weighting shows that the customers require it to stabilise the payload and land on the target accurately. Also, the customer is weighing the

payload capacity and fit within the DJI FlyCart 30's limitations high to maximise the amount of payload that can be carried per trip and fully utilise DJI FlyCart 30's limitations.

The manufacturer also wants to fully utilise DJI FlyCart 30's limitations, as this will be a selling point. In addition, compatibility with other drones is also weighted high, increasing the number of potential customers. The manufacturer also weighs *"land on target"* and *"control position"* highly because these are of great importance to the customer and, therefore, a selling point too. While *"damp pendulum oscillations"* are essential to the customer, it is not as crucial to the manufacturer if the system can land on the target and has proper position control.

The requirements with the highest influence can be seen in the weighted sum column in Table 4.1 and are *"land on target"*, *"control position"*, and *"maximise payload capacity"*. The requirements indicate that a design should have great position control and, thereby, be able to land on the target. Chapter 1 suggests that the ability to pick up remote cargo is an important factor for potential customers, indicating a landing area less than 0.5 m^2. The SPS system should be as light as possible to maximise the payload capacity.

#	Engineering specification	Min	Max	Weighted prioritization
1	SPS weight	–	5 kg	2.98
2	Payload capacity	25 kg	–	1.43
3	Wind speed	10 $\frac{m}{s}$	–	4.42
4	Max swing distance	–	1 m pr. 15 m	3.16
5	Landing pad size	–	2 x 2 m	3.39
6	SPS size	350/350/200 mm	754/472/385 mm	4.8
7	Payload size	450/450/500 mm	–	3.54
8	Measured attitude accuracy	–	1°	4.28
9	Control attitude accuracy	–	1°	4.58
10	Torque about Ψ	8.26 Nm	–	3.14
11	Exposed hazardous parts	–	0	1.62
12	Ease of access	0 min	10 min	0.61
13	Time between service	120 min	–	1.12
14	Number of wear parts	–	6	1.19
15	Number of sensors	1	10	1.8
16	Time for swapping battery	–	2 min	0.59
17	Max wire torsion	–	0 Nm	0.95
18	Automatic docking	–	Manual oversight	1.75
19	Propulsion efficiency	–	–	1.85
20	Number of motors	4	8	4.53
21	Operating time	18 min	–	1.93
22	Time until stability	–	15 s	3.99

Table 4.2. Engineering specification with boundary values

Based on the requirements in Table 4.1, the engineering requirements and the boundary values have been specified as seen in Table 4.2. An explanation of the engineering specifications and the boundary values can be found in Appendix B.2. These specifications have been implemented in a house of quality (HOQ) and can be found in Appendix Figure B.1. The strength of the relationship between the requirements and the specifications has been evaluated. A weighted

prioritisation has been calculated for each specification based on the weight of the requirements and the specification's relevance to that requirement. The weighted prioritisation can be seen in Table 4.2.

Appendix Figure B.2 evaluates the relations between the engineering specifications. The relationship between engineering specifications is shown as an arrow, visualising the change that would occur if the numerical value of the engineering specifications in the column increases. This evaluation is important for understanding the relationship between engineering specifications in a design and an understanding of how two design choices impact the overall specification of the final design.

4.2 Morphological analysis

In this section, an overview of the morphological analysis is given. A full description can be found in Appendix C. A morphological analysis can, in general, be split into four parts:

- Define dimensions
- Create solutions
- Combine
- Evaluate

The overall stabilised single-wire payload delivery system is divided into dimensions, each defining a single element of the overall design. To design the SPS system, it is required to produce thrust through propellers, which must be mounted on the system. The SPS system must also be able to be lowered from the drone. The system also needs to sense distances, attitude, and motor speed. From this the system is divided into the following dimensions: "*Motor position*", "*Payload coupling*", "*Hoist mount*", "*Hoist system*", "*XYZ attitude sensing*", "*SPS to drone distance sensing*", "*SPS to ground distance sensing*", "*Motor speed sensing*", and "*Frame construction/type*".

For each dimension, solutions are brainstormed, resulting in several possible solutions. Each generated solution for every dimension is sketched in Appendix Figure C.1.

A total of $4,665,600$ unique combinations can be created from the solutions. From the possible combinations, ten different concepts are developed that incorporate as many of the solutions as possible. Appendix Figure C.2 shows a sketch of each of the ten concepts. However, multiple solutions were generated for the "*Hoist system*", solutions 1 were chosen for each concept. This decision was made because the prototype will be implemented on the DJI FlyCart 30, which has an implemented hoist system.

The concepts' naming scheme describes the chosen solutions. A unique code refers to which solution is used for each dimension in the order in which they were named. For example, concept 1 has the code $4-3-4-1-4-4-4-3-3$.

When evaluating the concepts, an expected median concept is chosen as a reference, and each concept is compared to the reference for each engineering specification, where a value is given based on the concept being better, the same or worse than the reference concept. The final and weighted sum is calculated by multiplying the points by the weighted prioritisation of the engineering specifications. This evaluation can be seen in Appendix Table C.3. A segment of the table can be seen in Table 4.3.

#	Engineering specification	Weight	Concept 1 4-3-4-1-4-4-4-3-3	Concept 5 4-4-4-1-1-4-4-3-4	Concept 6 8-4-2-1-1-2-6-3-4	Concept 7 3-3-2-1-1-2-5-3-4	Concept 11 3-4-4-1-1-2-6-3-4	Concept 12 4-4-4-1-1-2-6-3-4
1	SPS weight	2.98	1	1	0	1	1	0
2	Payload capacity	1.43	1	1	0	1	1	1
3	Wind speed	4.42	1	1	0	1	1	1
4	Max swing distance	3.16	1	1	0	1	1	1
5	Landing pad size	3.39	1	1	0	0	0	1
6	SPS size	4.80	-1	-1	0	0	0	-1
7	Payload size	3.54	0	0	0	0	0	0
8	Measured attitude accuracy	4.28	0	0	0	0	0	0
9	Control attitude accuracy	4.58	3	3	0	1	1	3
10	Torque about Ψ	3.14	1	1	0	1	1	1
11	Exposed hazardous parts	1.62	-1	-3	0	0	0	0
12	Ease of access	0.61	1	1	0	1	1	1
13	Time between service	1.12	-3	-3	0	0	0	0
14	Number of wear parts	1.19	-1	-1	0	0	0	-1
15	Number of sensors	1.80	-1	0	0	0	0	0
16	Time for swapping battery	0.59	1	1	0	1	1	1
17	Max wire torsion	0.95	1	1	0	0	1	1
18	Automatic docking	1.75	-1	0	0	-1	0	0
19	Propulsion efficiency	1.85	1	0	0	0	0	1
20	Number of motors	4.53	-1	-1	0	0	0	-1
21	Operating time	1.93	0	-1	0	0	0	0
22	Time until stability	3.99	1	1	0	0	0	1
Sum:			5	3	0	7	9	10
Weigted sum:			21.24	17.76	0	19.17	21.86	26.77

Table 4.3. A segment of Appendix Table C.3 where the evaluation of the top concepts and the new modified concepts can be seen

The evaluation provided three concepts with a higher weighted sum than the rest. Due to how the concepts were generated, an additional review of the three best concepts was made to identify the dimensions in which a change in solution could improve the concept. The final two concepts combined the best characteristics of the three best concepts and thereby got a better score. The difference between the two final concepts is the number of motors, where concept 11 has four motors pointing in the same direction. Concept 12 has two additional motors pointing perpendicular to the rest and is the concept with the highest score. Due to limitations of available components, concept 11 was selected for the design process.

4.3 Prototype design

From the morphological analysis in Section 4.2, concept 11 is used as the baseline for the design. This report does not necessarily aim to design a finalised commercial version. Therefore, a prototype is designed and produced to be tested and determine if the concept is feasible. It has been decided to mainly 3D print the prototype in PLA, as it allows for a fast prototyping process. The frame will only transfer forces from the motors, where the locking mechanism will bear the payload mass. The 3D printers available for this report are “Bambu Lab P1S”, with a

print area of 256 *mm* x 256 *mm* x 256 *mm*. The prototype, therefore, needs to be printed in segments that fit within the print area.

From concept 11, the frame of the design should consist of a cone in the middle. From Airflight, it was specified that the cone should have a top diameter of 160 *mm*, a bottom diameter of 350 *mm* and a height of 200 *mm*. The cone is larger than the print area, resulting in the cone being split into four segments joined by bolts. Two of the segments have connections for the arms to mount on. The other two segments have a box that holds the electronics and the batteries, respectively.

From the cone, two arms extend that hold the motors with propellers. The arms consist of segments, and each arm segment contains a motor with a propeller. The arm segments are bidirectional, so only one segment needs to be designed. The segments have also been designed to mount a mesh to shield the propeller and motor. Each segment also has a mount for a hall sensor for testing purposes. The segments are joined by a tongue and groove joint connection with added bolts to join the segments securely.

On top of the cone, the locking mechanism is mounted via sheet metal. Bolts join together the locking mechanism, sheet metal, and cone. An eyebolt is placed on top of the sheet metal above the COM and will be connected to the wire from the drone. This way of connecting the wire to the payload results in the locking mechanism and sheet metal carrying all the load from the payload. The CAD model of the SPS prototype can be seen in Figure 4.1, and in Figure 4.2, the frontal measurements of the prototype can be seen.

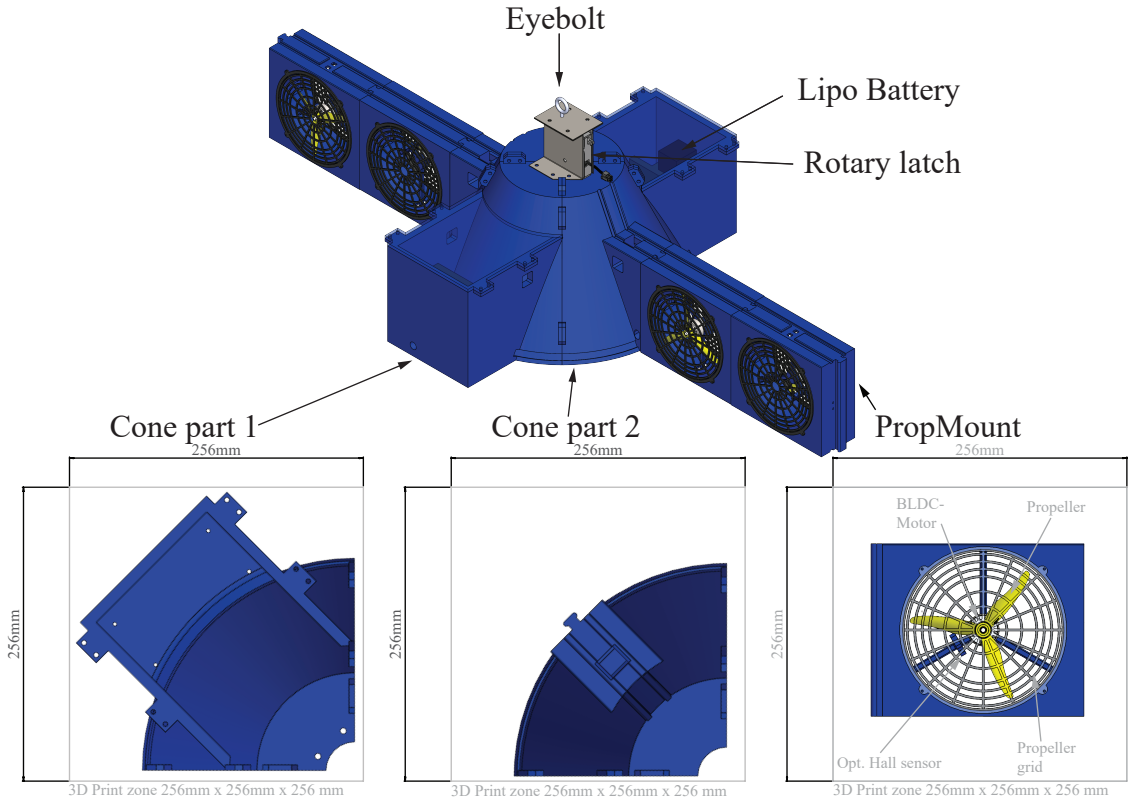


Figure 4.1. CAD model of SPS prototype and visualisation of parts inside 3D printer work area of 256 *mm* x 256 *mm* x 256 *mm*

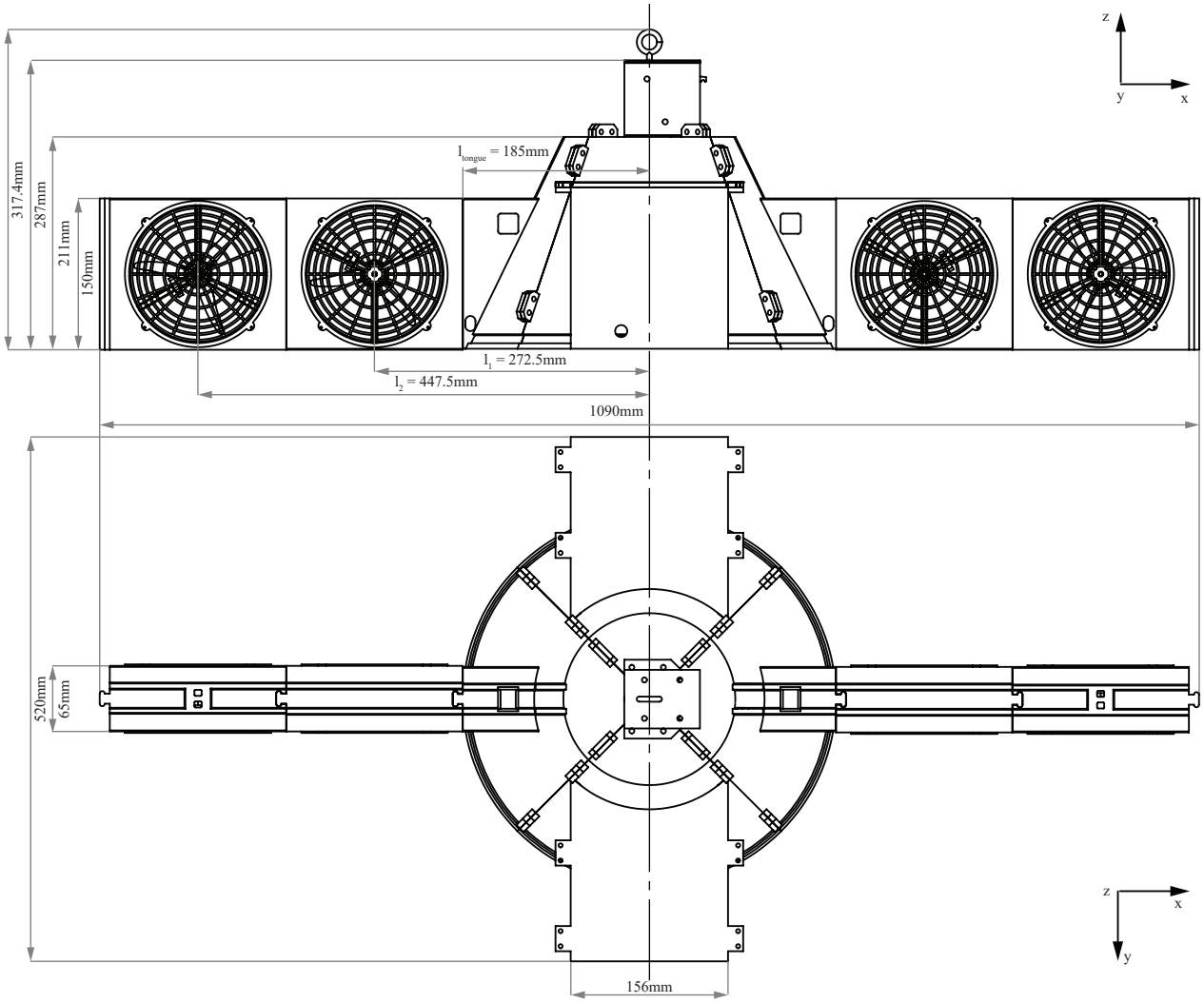


Figure 4.2. CAD linedrawing of SPS prototype with measurements

4.3.1 Design requirements analysis

As the design of the prototype is determined, it can be calculated if the design complies with some of the engineering specifications in Table 4.3. The engineering specifications requiring testing or simulation will be tested in Chapter 8 and evaluated in Chapter 9.

Strength analysis

A strength analysis will determine whether the design, including the system, can carry the required 30 kg payload. A strength analysis will be performed on these parts as the prototype transfers all its weight through the sheet metal and the eyebolt. It will also be determined whether the arms can withstand the maximum torque the thrusters produce.

A wire is necessary to join the drone with the SPS system, and many options are available. The testing setup will use a steel wire, which needs to carry 30 kg, which equals 0.29 kN. The steel wire with the lowest diameter from "Certex.dk" is a 1 mm rust-free steel wire in a 1x19 configuration. This wire has a minimum ultimate strength of 0.83 kN, which is 2.85 times the load. Therefore, a steel wire with 1 mm in diameter should be suitable for the prototype. The

wire weighs 0.49 $kg/100m$ and at 15 m the wire would weigh approximately 0.0735 kg, which is negligible compared to the SPS system and payload (CERTEX.dk 2025).

The wire is mounted to an eyebolt with a working load limit (WWL) of 100 kg, which is far above the total system mass. The eye bolt is placed in the middle of a 2 mm steel sheet. There is $L_{bolt} = 38.5\ mm$ between the bolts fastening the plate. The force is located in the middle of the bolts. The equations for a simple supported beam estimate the stress since the sheet can be assumed as a long, slim beam. The stress depends on the area moment of inertia, where the width of the sheet is $w = 75\ mm$ and height $h = 2\ mm$. The area moment of inertia is calculated as $I_x = \frac{w \cdot h^3}{12} = 50\ mm^4$. The stress in the plate is then found by combining Equation (4.1) for the moment in a supported beam with Equation (4.2) the maxium stress in a beam is found at $y = \frac{h}{2}$.

$$M = \frac{1}{4} \cdot F_g \cdot L_{bolt} \qquad (4.1) \qquad\qquad \sigma = \frac{M \cdot y}{I_x} \qquad (4.2)$$

The maximum moment applied to the sheet is 2.835 Nm, and the maximum stress is 56 MPa. The exact steel type used is unknown, but a typical universal structural steel type, such as S235, has a yield strength of $\approx 235\ MPa$, which results in a safety factor of 4.2. The sheet is mounted on top of two other plates in tension. These plates are $t = 1\ mm$ thick, with a width of $w = 71\ mm$. The area in which the stress is spread spans $A = w \cdot t \cdot 2 = 142\ mm^2$, resulting in a stress of 2 MPa, giving a safety factor of 117.

The plates are mounted on a rotary latch (the locking mechanism), rated with an ultimate strength of 6.77 kN or 690 kg. With a 30 kg load a safety factor of 23 is present. It can be seen that the load-bearing part of the SPS should be able to handle a load of 30 kg.

Another area of the SPS that is of interest is the arms, since the two propellers will create torque about the COM. An analysis of the tongue connecting the inner arm segment to the cone is done to see if the design can handle the torque. The tongue is the part of the arm with the least material and is located $l_{tongue} = 0.1850\ m$ from the centre of the cone. With the lengths to the centre of the thrust producing propellers seen in Figure 4.2, the arm from the thrust to the tongue can be calculated. The thrust is known from Appendix F.6, where the maximum thrust generated by the motors is found in both directions, since the thrust is not bidirectional. Thrust is found in the positive direction as $T_p = 13.89\ N$, and as $T_n = 5.25\ N$ in the negative direction. This has an impact since the motors are mounted with the propellers producing maximum thrust in different directions. So to get the maximum moment, the outer motor is set to deliver the maximum thrust while the inner motor delivers the lower reverse thrust. From this, the moment can be found by Equation (4.3).

$$M = (l_1 - l_{tongue}) \cdot T_n + (l_2 - l_{tongue}) + T_p = 4.13\ Nm \qquad (4.3)$$

The tongue can then be seen to be subject to a torque from the thrust of 4.13 Nm. The tongue is part of the 3D-printed structure, which was printed with a 25% infill and wall thickness of 1 mm. To simplify calculations, the tongue is assumed hollow, consisting of a box with $t = 1\ mm$ thick walls. The area moment of inertia can therefore be estimated by: $I_x = \frac{b \cdot h^3}{12} - \frac{(b-t) \cdot (h-t)^3}{12}$

with a height of $h = 10\ mm$ and width of $b = 149\ mm$ the area moment of inertia is given as $3.42 \cdot 10^{-9}\ m^4$. From Equation (4.2), the maximum stress can be calculated as 6.03 MPa. The PLA that the tongue is printed of has an ultimate strength of 52 MPa (3deksperten.dk 2025), resulting in a factor of safety of 8.62. In reality, it is likely larger due to infill, and additionally, four M3 bolts also pass through the segments adjacent to the tongue. From this, it can be determined that the arms can withstand the expected forces from the propellers.

Weight/payload capacity

The SPS system's payload capacity is intended to be maximised. Because the drone is limited to a 30 kg payload, this is achieved by making the SPS system as light as possible. The maximum allowed weight of the SPS system is 5 kg, resulting in a 25 kg payload capacity. The prototype's weight is measured to be 4.556 kg, resulting in a payload capacity of 25.444 kg. The prototype has not been weight optimised, and contains plenty of non-essential mass which can be removed in future designs.

Compatible with DJI FlyCart30

Airflight uses the DJI FlyCart 30 with modified legs increasing the drone's height by 300 mm and includes a rectangular landing frame of 1000 x 1000 mm as highlighted by the yellow part in Figure 4.3. The SPS prototype needs to fit within the landing legs, which as indicated by Figure 4.2, the length of the SPS system is 1090 mm, exceeding the width of the modified landing legs. If great control of attitude and enough torque about Ψ is achieved, it is possible to orient the SPS system at an angle where the SPS system fits within the legs. The required interval of Ψ is calculated by equation (4.4), and the interval is given by equation (4.5). The angle interval is slightly smaller as the arms are thick, but it can fit if the SPS system can be oriented within the 43° interval.

$$\left(\frac{500\ mm}{cos(\Psi)} > 545\ mm\right) \wedge \left(\frac{500\ mm}{sin(\Psi)} > 545\ mm\right) \tag{4.4}$$

$$23.45° \leq \Psi \leq 66.55° \tag{4.5}$$

In a final design, the arms would not be 3D-printed segments. Because they are printed in segments, they are larger than necessary. However, they could be shrunk by about 100 mm, making the system fit within the legs at any orientation.

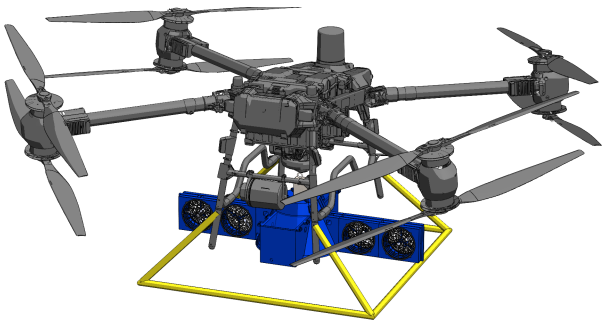


Figure 4.3. Sketch of the modified landing legs on the DJI FlyCart30 where the SPS system exceeds the maximum dimensions.

Frontal drag force

The dimensions of the SPS system have been set, and the maximum possible torque produced

by the aerodynamic forces can be decided. The drag coefficients for $C_{D.0}$ and $C_{D.90}$ are experimentally determined to fit the model in Section 5.3. The coefficients are shown in Table 5.4 and given as $C_{D.0} = 1.617$ and $C_{D.90} = 0.413$.

Section 2.2 gives the approach for calculating the forces and moments. The SPS prototype is assumed to be a solid rectangle with the length of $L = 1090\ mm$ with height and thickness equal to the height and thickness of the projected area of the the arm segments and electronics box of $h = 160\ mm$ and $t = 520\ mm$ respectively, as seen in Figure 4.2. The same assumptions were used when the drag coefficients were determined experimentally.

The drag force can then be calculated by Equation (2.1). The areas are given as $A_{p.90} = L \cdot h = 174400\ mm^2$ and $A_{p.0} = t \cdot h = 83200\ mm^2$. The wind speed is set to be $v_{wind} = 10\frac{m}{s}$ as specified by Table 4.2. The drag forces for each plane are calculated in Equation (4.6).

$$\begin{aligned} F_{D.0} &= \frac{1}{2} \cdot C_{D.0} \cdot \rho \cdot A_{p.0} \cdot v_{wind}^2 = 2.222\ N \\ F_{D.90} &= \frac{1}{2} \cdot C_{D.90} \cdot \rho \cdot A_{p.90} \cdot v_{wind}^2 = 18.232\ N \end{aligned} \tag{4.6}$$

The SPS system should also be able to handle a payload. The drag of the payload is estimated using the regression analysis described in Section 2.2. The drag coefficient of the payload is made from the minimum requirements from Table 4.2. The drag coefficient is calculated to be $C_{D.payload=1.95}$ and the area is calculated as $A_{p.payload} = 500\ mm \cdot 450\ mm$. The drag force on the payload is calculated in Equation (4.7), and the maximum drag force is calculated in Equation (4.8) using $F_{D.90}$ as it has the highest drag force.

$$F_{D.payload} = \frac{1}{2} \cdot C_{D.payload} \cdot \rho \cdot A_{p.payload} \cdot v_{wind}^2 = 28.291\ N \tag{4.7}$$

$$F_{D.max} = F_{D.90} + F_{D.payload} = 46.523\ N \tag{4.8}$$

The prototype design always has two motors pointing towards the wind stream and two pointing in the opposite direction. According to Appendix F.6, the maximum thrust provided by a single motor in the positive direction is $T_p = 13.98\ N$ and in the negative direction it is $T_n = 5.25\ N$. The Maximum thrust the SPS system can provide can be calculated by Equation (4.9). In Section A.4, graphs of the aerodynamic coefficients and forces are shown for a more detailed view.

$$T_{max} = 2 \cdot T_p + 2 \cdot T_n = 38.46\ N \tag{4.9}$$

With the calculated forces, it can be assumed that the SPS can dampen any expected disturbance without a payload. When the largest possible payload is attached, the system does not have the required thrust to dampen the system with a constant 10 $\frac{m}{s}$ wind. The use of bidirectional propellers or larger motors with the combination of larger propellers can solve this issue, but is investigated further in Chapter 9.

Max drag torque

Calculating the drag torque is the same as in Section 2.2. The SPS system and payload drag torque are computed under the same assumption used for the frontal drag force.

The calculations align the x-axis with the x-axis from Figure 4.2, and the direction of the wind also aligns with the x-axis. From the calculations, the maximum torque produced by the aerodynamic forces on the SPS system is 0.979 Nm at 57.27°, and for the payload it is 3.86 Nm at 78.18°. Adding the two torques together, the combined torque produced by the aerodynamic forces is calculated to be 4.373 Nm at 69.09°. In Section A.4, a further description and figures showing the drag and lift forces together with the produced torques between 0 and 90°.

The torque the SPS system can produce is calculated as the force produced by the propellers multiplied by the arm from the COM to the motor. The distances are given in Figure 4.2 where $l_1 = 272.5\ mm$ and $l_2 = 447.5\ mm$. The two central propellers on the SPS system point opposite the outer propellers. Equation (4.10) calculates the maximum torque produced by the SPS system.

$$\tau_{T.max} = T_p \cdot (l_1 + l_2) + T_n \cdot (l_1 + l_2) = 13.85\ Nm \tag{4.10}$$

The SPS system is designed to produce 3.17 times the maximum torque the aerodynamic forces produce at a wind speed of 10 $\frac{m}{s}$. Therefore, the design is expected to have great yaw position control and enough thrust to orient in a given position at lower wind speeds.

Modelling 5

In this chapter, the modelling of the SPS system will be derived and explained. The SPS model consists of four BLDC motor models and a mechanical pendulum model with added orientation and aerodynamic force input. Each model will have its limits, assumptions, and issues described throughout the chapter. These models are also validated against the real system for non-linear and linear model representations.

Figure 5.1 shows an open-loop representation of the SPS system model. This model consists of four BLDC motor models that describe the dynamics of the BLDC motors. When they rotate, they generate a certain amount of thrust given by the velocity of the propellers. The thrust affects the pendulum system and the orientation of the SPS by creating moments about the wire attachment point or the COM of the SPS. The model is also affected by external aerodynamic forces, creating drag and lift forces depending on the amplitude of the wind speed.

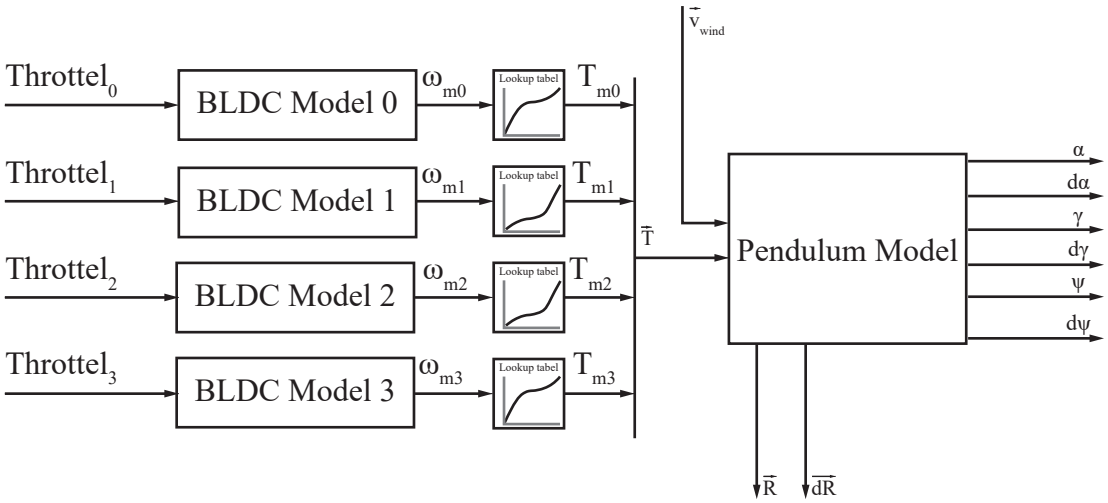


Figure 5.1. Block diagram of the open-loop SPS system model

The BLDC models are described in Section 5.1, the thrust generation block is described in Section 5.2, and the pendulum model is defined in Section 5.3.

5.1 BLDC motor model

The motor used in the SPS is a Tmotor F40pro IV Kv2400 (TMOTOR 2024). This motor is a BLDC and should therefore be modelled as one. A block diagram of the BLDC model can be seen in Figure 5.2. The model consists of some voltage and torque equations for BLDCs. These

equations can be found in Subsection 5.1.1. The trapezoidal back-EMF generator and the BLDC commutation are explained in Subsection 5.1.2. The model is set up such that a throttle is given as input, which results in a motor velocity as output.

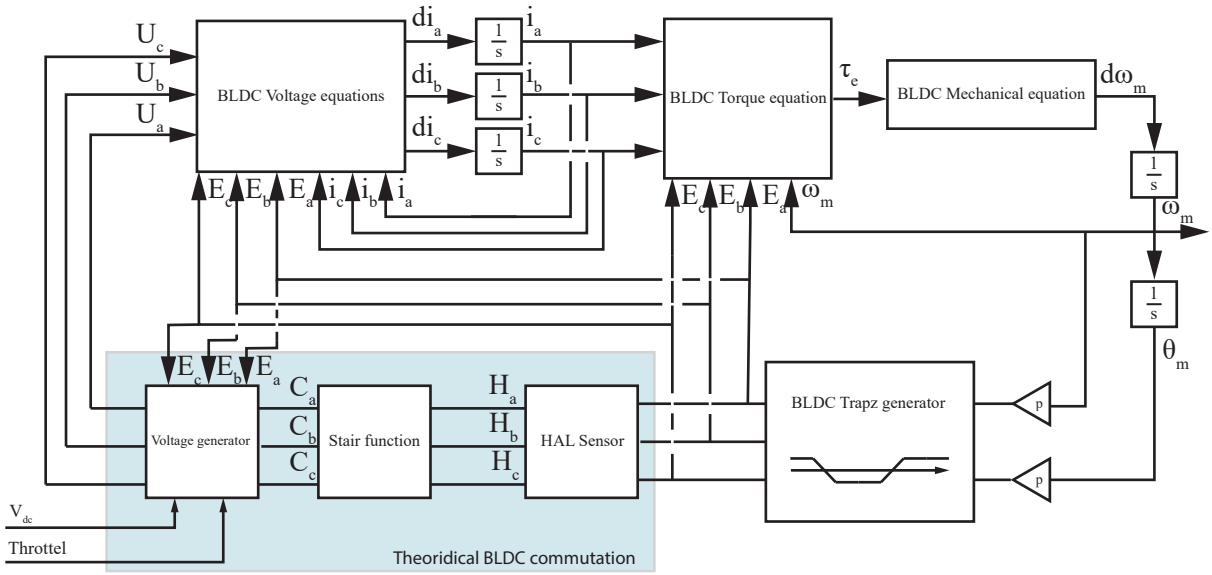


Figure 5.2. Block diagram of the BLDC model

Experiments have estimated the electrical and mechanical parameters of this model. The electrical parameters of the motor have been estimated in Appendix Chapter F. A complete experimental procedure for different parameters can be found in each appendix section. Table 5.1 contains the estimated parameters measured in the appendix.

Parameter	Value	Unit
Winding type	Δ (delta wound)	–
Resistance (R_s)	0.0796	Ω
Back-EMF constant (K_e)	$5.32e^{-4}$	$\frac{V \cdot s}{rad}$
Inductance (L_s)	$2.79e^{-6}$	H
Rotor inertia (J_r)	$1.6e^{-6}$	$\frac{\text{kg}}{\text{m}^2}$
Propellor inertia (J_p)	$7.04e^{-6}$	$\frac{\text{kg}}{\text{m}^2}$
Viscous friction (C_v)	$3.374e^{-6}$	$\text{Nm} \cdot \text{s}$
Coloumb friction (C_c)	$5.955e^{-4}$	Nm
Propellor resistive torque constant (C_p)	$3.18e^{-8}$	$\text{Nm} \cdot \text{s}^2$

Table 5.1. Electrical and mechanical parameters of the Tmotor F40pro IV Kv2400 BLDC motor

Model assumptions and limitations

The delta wound BLDC model has the following assumptions:

- Ideal trapezoidal back-EMF
- Lossless

- Neglects the Electronic Speed Controllers (ESC) influence - assumed linear relation between input throttle and output voltage

Implementing the torque equation required a non-zero initial condition for ω_m, as otherwise a division by zero occurs in Equation (5.4). The model can therefore also not handle when $\omega_m = 0$ during simulation.

5.1.1 BLDC equations

The equations for the BLDC motor are the same for both wye and delta wound motors. The difference comes from the shape of the back-EMF and the applied phase voltages. The BLDC motor has three phases, each requiring an equation to describe the currents. The BLDC voltage equations are given by Equation (5.1) (Lee, Song, et al. 2014, p. 2201).

$$\begin{bmatrix} U_a \\ U_b \\ U_c \end{bmatrix} = R_s \cdot \begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} + L_s \cdot \frac{d}{dt} \begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} + \begin{bmatrix} E_a \\ E_b \\ E_c \end{bmatrix} \tag{5.1}$$

$U_{a,b,c}$ is phase voltages, $i_{a,b,c}$ is phase currents, $E_{a.b.c}$ is back-EMF for each phase, R_s is phase resistance, and L_s is phase inductance. The equations are solved for the derivative of i, $\frac{di_{a,b.c}}{dt}$, as shown in equation (5.2).

$$\frac{di_{a,b,c}}{dt} = \frac{1}{L_s} \left(U_{a,b,c} - R_s \cdot i_{a,b,c} - E_{a,b,c} \right) \tag{5.2}$$

The output power of the BLDC can be calculated by equation (5.3).

$$P_{out} = i_a \cdot E_a + i_b \cdot E_b + i_c \cdot E_c \tag{5.3}$$

The motor output torque is then the relation between the output power and the mechanical output angular velocity as given by Equation (5.4)

$$\tau_e = \frac{P_{out}}{\omega_m} \tag{5.4}$$

Newton's second law is applied to calculate the angular velocity of the BLDC, where the frictions and resistive torque from the propeller are subtracted from the motor torque. The mechanical model of the BLDC is as shown in Equation (5.5).

$$\frac{d}{dt}\omega_m \cdot J_{tot} = \tau_e - \underbrace{C_p \cdot \omega_m^2}_{\text{drag}} - \underbrace{C_v \cdot \omega_m}_{\text{viscous}} - \underbrace{C_c \cdot sign(\omega_m)}_{\text{Coloumb}} \tag{5.5}$$

5.1.2 Delta wound back-EMF & applied phase voltages

In Appendix section F.1, it was determined that the BLDC was a delta wound BLDC. The delta wound BLDC back-EMF is unique as it is trapezoidal and has a 60° electrical flat top and 120° electrical slope of $\frac{3}{\pi}$. The back-EMF for each phase can be calculated by Equation (5.6), where θ_e is the electrical angle between $0 - 2\pi$ and ω_e is the electrical angular speed. The back-EMF for each phase is phase shifted by $\frac{2\pi}{3}$, resulting in the plot of the trapezoidal back-EMF on unit form in Figure 5.3.

$$E = \omega_e \cdot K_e \cdot \begin{cases} \frac{3 \cdot \theta_e}{\pi} & 0 \leq \theta_e < \frac{\pi}{3} \\ 1 & \frac{\pi}{3} \leq \theta_e < \frac{2\pi}{3} \\ 3 - \frac{3 \cdot \theta_e}{\pi} & \frac{2\pi}{3} \leq \theta_e < 2\pi \\ 3 - \frac{3 \cdot \theta_e}{\pi} & 2\pi \leq \theta_e < \frac{4\pi}{3} \\ -1 & \frac{4\pi}{3} \leq \theta_e < \frac{5\pi}{4} \\ -6 + \frac{3 \cdot \theta_e}{\pi} & \frac{5\pi}{4} \leq \theta_e < 2\pi \end{cases} \tag{5.6}$$

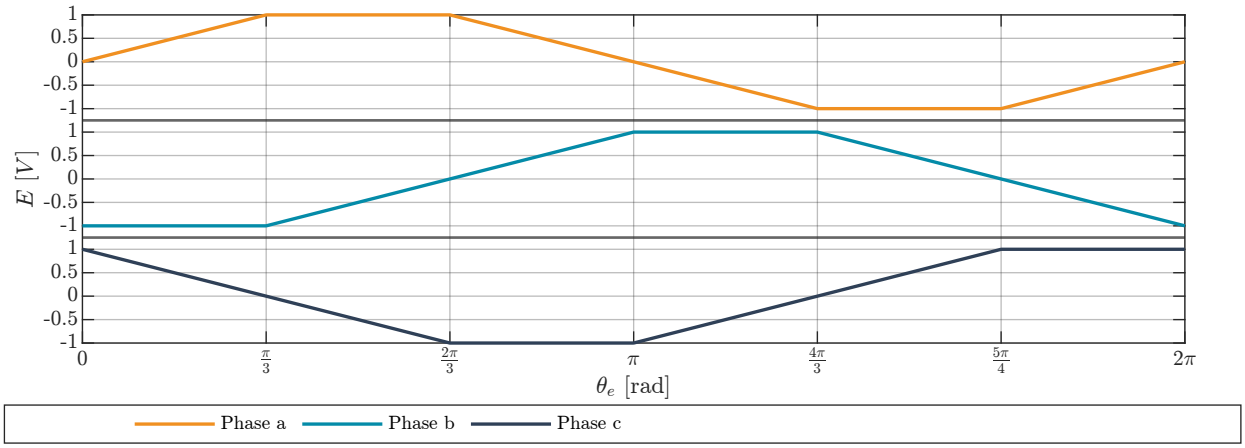


Figure 5.3. Unit trapezoidal back-EMF for each phase

The applied voltages are required to match the back-EMF to produce currents that are in phase, such that torque will be produced by Equation (5.3) and (5.4). This is done by hall sensors, which output a signal when the back-EMF of the phase is positive. Combining the outputs of all three phases results in six unique modes, each corresponding to a set of active switches on the full-bridge ESC. Because the BLDC is delta wound, there will always be a voltage and a current in the phase, which has been analysed in Appendix section D.1. The result is an input voltage following a stair function stepping between $U_{DC}, \frac{1}{2}U_{DC}, -\frac{1}{2}U_{DC}, -U_{DC}$. According to Equation (5.2), there will be a change in current when the applied voltage U and the back-EMF E do not have the same slope, e.g., when U is constant and E has a slope, resulting in a change in the current. U and E must have the same shape to produce the correct current shapes, further elaborated in Appendix section D.2. The applied phase voltages will look as shown in Figure 5.4, to produce the expected currents with the correct shapes.

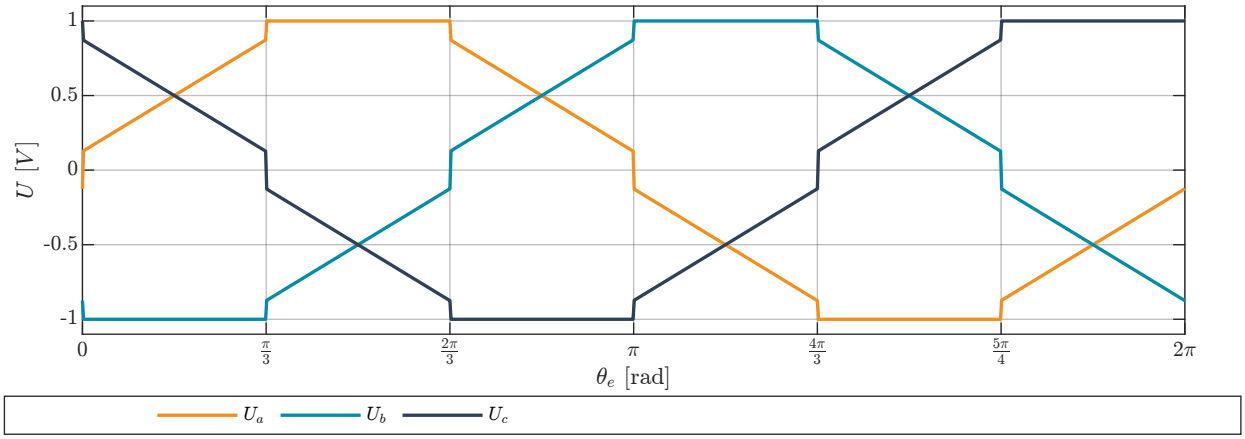


Figure 5.4. Applied simulation voltages

The phase voltages shown in Figure 5.4 are calculated by equation (5.7) and (5.8).

$$DU = U_{bat} \cdot throttle - \max(|E_a|, |E_b|, |E_c|) \tag{5.7}$$

$$U_{a,b,c} = DU \cdot C_{a,b,c} + E_{a,b,c} \tag{5.8}$$

Equation (5.7) calculated the available voltage DU, which is the difference between the produced back-EMF peak and the input voltage from the battery U_{bat} multiplied by the throttle percentage. The available voltage DU is then multiplied by a unit stair waveform $C_{a,b,c}$, corresponding to the shape of the phase voltages in a delta wound BLDC motor. Finally, the back-EMF is added to obtain the desired voltage waveform shape in Equation (5.8), as seen in Figure 5.4.

5.1.3 Model verdification

The previous equations were implemented in Simulink with the parameters from Table 5.1. A step and random input tests were compared with the model to validate it. More details about the test and data from more motor states can be in Appendix Section D.3. The input throttle from the test was input into the simulation, and the results can be seen in Figure 5.5.

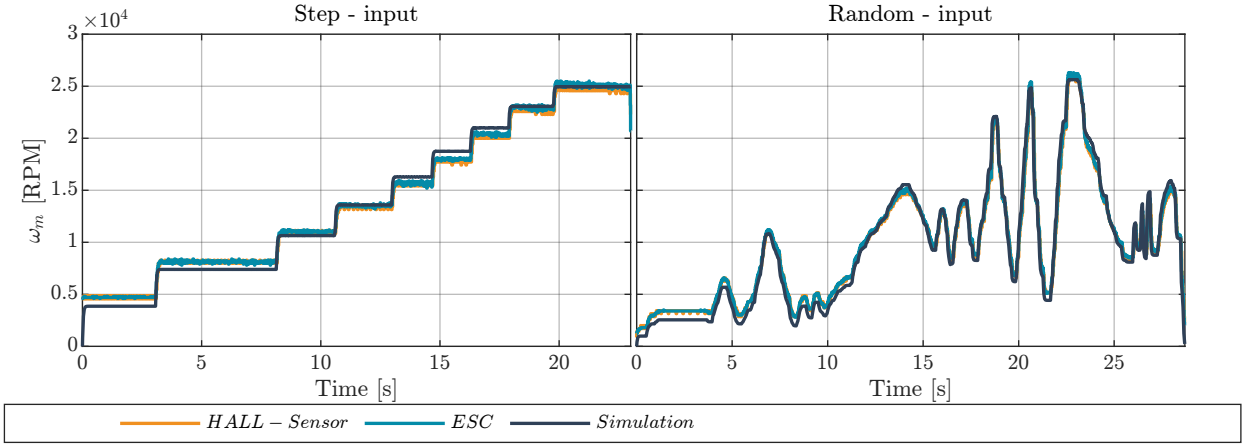


Figure 5.5. Step and random input tests with responding measurement and simulation results

Figure 5.5 compares hall sensors, ESC measurements, and BLDC motor simulation results with propellers. The step input is done in increments of 10% throttle from 10% to 90%. It can be seen in the step input test that the simulation does not predict the speed correctly at every throttle level. Plotting the speed at the throttle percentage as seen in Figure 5.6, it can be seen that the simulation follows a second-order polynomial as expected with a linear voltage throttle, e.g., $U_{bat} \cdot throttle$. The second-order polynomial is given by Equation (5.9), where each term corresponds to a friction.

$$f(\omega_m) = \underbrace{\omega_m^2 \cdot P_1}_{drag} + \underbrace{\omega_m \cdot P_2}_{viscous} + \underbrace{P_3}_{Coloumb} \tag{5.9}$$

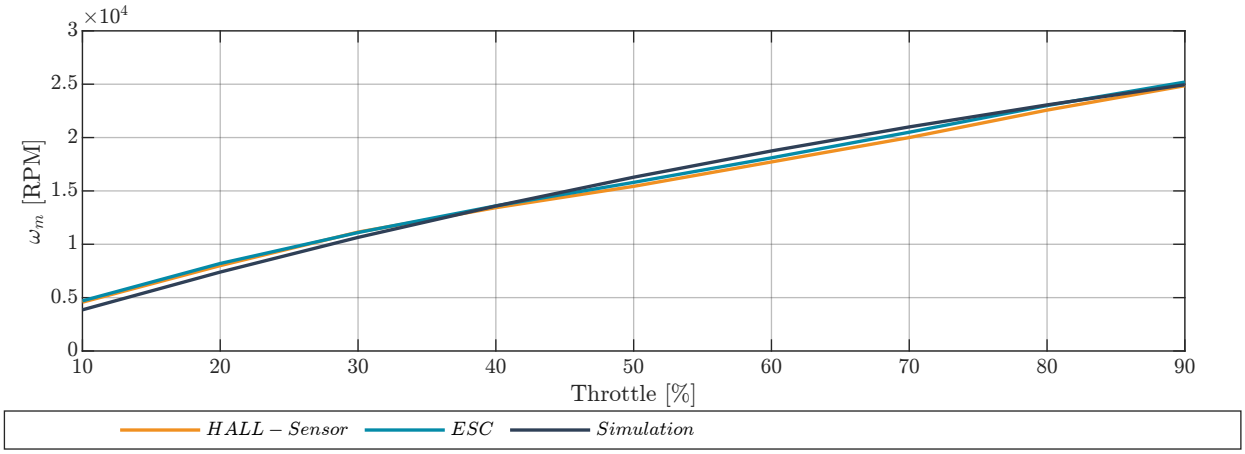


Figure 5.6. Comparison between measured motor speed and simulated speed from 0% to 90% throttle

The measurements from the ESC and hall sensors are linear with a slope change at 30% throttle. The linearisation of the output is expected to be due to the ESC being designed for FPV racing drones, where a linear relation between the speed and the throttle is desirable. The simulation underpredicts the speeds up until $\approx 35\%$ and overpredicts the speeds up until $80-90\%$ throttle. The most significant difference between simulation and measurements depends on the measurement tool, as the most considerable inconsistency for ESC measurement is at 10% throttle. In contrast, the most significant error for the hall sensor is at 60% throttle. Table 5.2 shows the most and least significant errors for the ESC and hall sensors compared to the simulation.

Throttle	ESC	Hall sensor	Unit
	Largest error		
10%	843	714	RPM
60%	645	1030	RPM
	Smallest error		
40%	13	159	RPM
90%	248	94	RPM

Table 5.2. The largest and smallest error between measurements and the simulation at the throttle percentage.

This discrepancy between measurements and simulation is expected to be negligible, as a robust controller can compensate. The random input test in Figure 5.5 shows that the same errors were transferred, but the dynamic responses fit well.

5.1.4 Linear model

A linear model is needed for control theory. The Park and Clarke transformations are used to linearise the motor model, which can convert the three-phase brushless motor model to an equivalent "brushed" motor representation (Ung et al. 2015). The Park and Clarke transformations can be seen in Equations (5.10) and (5.11).

Park:

$$P = \begin{bmatrix} cos(\theta_e) & sin(\theta_e) \\ -sin(\theta_e) & cos(\theta_e) \end{bmatrix} \tag{5.10}$$

Clarke:

$$C = \sqrt{\frac{2}{3}} \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \tag{5.11}$$

The Clark transformation takes a 3-phase system and transforms it into a stationary $\alpha\beta$ frame based on the electric position of the motor θ_e. The Park transformation transforms the stationary frame into a rotating dq frame. The dq frame can represent the alternating voltages and currents of the 3-phase system as constant voltages and currents in the d and q axes.

Applying the Park and Clarke transformation, the electrical BLDC model can go from a 3-phase BLDC model to an equivalent "brushed" motor representation. To do this, it must be assumed that the generated torque is ideal. This means that the back-EMF and currents need to be in phase, and are sinusoidal with 120 deg phase shift between the three phases. This model will therefore lose the torque ripples seen in the non-linear model as shown in Appendix Figure D.5. The three-phase currents can be described using a constant i^ϕ that defines the amplitude of the currents. With the Park and Clarke transformation, it is possible to calculate the i^d and i^q currents, which are given by Equation (5.12).

$$\begin{bmatrix} i^d \\ i^q \end{bmatrix} = P \cdot C \cdot \bar{i}^\phi = \sqrt{\frac{2}{3}} \begin{bmatrix} cos(\theta_e) & cos(\theta_e - \frac{2}{3}\pi) & cos(\theta_e + \frac{2}{3}\pi) \\ -sin(\theta_e) & -sin(\theta_e - \frac{2}{3}\pi) & -sin(\theta_e + \frac{2}{3}\pi) \end{bmatrix} \cdot \begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} = \sqrt{\frac{2}{3}} \begin{bmatrix} 0 \\ \frac{3}{2} i^\phi \end{bmatrix} \tag{5.12}$$

Through the Park and Clarke transformation, it can be seen that $i^q = 0$ and $i^q = \sqrt{\frac{2}{3}} \cdot \frac{3}{2} \cdot i^\phi = \sqrt{\frac{3}{2}} \cdot i^\phi$. The $\frac{3}{2}$ multiplied onto i^ϕ is a consequence of reducing the number of axes from three to two. Because the back-EMF is assumed to be sinusoidal, it can also be transformed by the Park and Clarke transformation. The back-EMF constant can be converted to the dq frame by $K_m^q = \sqrt{\frac{2}{3}} \cdot \frac{3}{2} \cdot K_m = \sqrt{\frac{3}{2}} \cdot K_m$, where $K_m = p \cdot K_e$ and p being the number of pole pairs.

Applying the Park and Clarke transformation to the electrical circuit model, the dq voltages rely on i^ϕ and K_m through Equation (5.13).

$$\begin{bmatrix} U^d \\ U^q \end{bmatrix} = \begin{bmatrix} 0 \\ U^q \end{bmatrix} = R_s \cdot \begin{bmatrix} 0 \\ \sqrt{\frac{3}{2}} \cdot i^\phi \end{bmatrix} + L_s \cdot \frac{d}{dt} \begin{bmatrix} 0 \\ \sqrt{\frac{3}{2}} \cdot i^\phi \end{bmatrix} + \omega_m \cdot \begin{bmatrix} 0 \\ \sqrt{\frac{3}{2}} \cdot K_m \end{bmatrix} \tag{5.13}$$

The BLDC circuit model can be reduced to a single equivalent model, where U^q is the input voltage. U^q can be calculated by transforming the line-line voltage U_{ll} through the Park and

Clarke transformation too, which results in Equation (5.14).

$$U^q = \sqrt{\frac{3}{2}} \cdot U_{ll} \tag{5.14}$$

The electrical linear model can then be represented by Equation (5.15).

$$U^q = R_s \cdot i^d + L_s \cdot \frac{di^q}{dt} + K_m^q \cdot \omega_m \tag{5.15}$$

The mechanical model of the BLDC motor is given by Equation (5.5), where the torque produced is given by Equation (5.3) and (5.4). The equations can be passed through the Park and Clarke transformation, resulting in Equation (5.16) to describe the torque.

$$\begin{aligned} P_{out}^q &= i^q \cdot K_m^q \cdot \omega_m \\ \tau_e^q &= \frac{P_{out}^q}{\omega_m} = i^q \cdot K_m^q \end{aligned} \tag{5.16}$$

Using Equation (5.16) to describe the torque τ_e in Equation (5.5) results in Equation (5.17).

$$\frac{d}{dt}\omega_m \cdot J_{tot} = K_m^q \cdot i^q - C_p \cdot \omega_m^2 - C_v \cdot \omega_m - C_c \cdot sign(\omega_m) \tag{5.17}$$

The torques produced by drag and Coulomb friction are non-linear. The mechanical model is linearised through the first-order Taylor series approximation. This is done by $\frac{d\dot{\omega}_m}{d\omega_m}\Big|_{\omega_m 0} \cdot \Delta\omega_m$ where ω_{m0} is a linearisation points for the drag torque. Equation (5.18) gives the linearised mechanical model.

$$\begin{aligned} \Delta\dot{\omega}_m \cdot J_{tot} &= K^q \cdot i^q - B_{lin} \cdot \Delta\omega_m \\ B_{lin} &= 2 \cdot C_p \cdot \omega_{m0} + C_v \end{aligned} \tag{5.18}$$

The linearisation point ω_{m0} specifies the dynamic response of the linear model. An analysis of how ω_{m0} was chosen is described in Appendix Section D.4. The linearisation point was selected because it is expected that the motor will operate the most between $50 - 90\%$ throttle, where a linearisation point at 75% throttle provides a reasonable estimate and dynamic response at this interval, as can be seen in Figure 5.7. The linearisation point is given as $\omega_{m0} = 16298RPM$

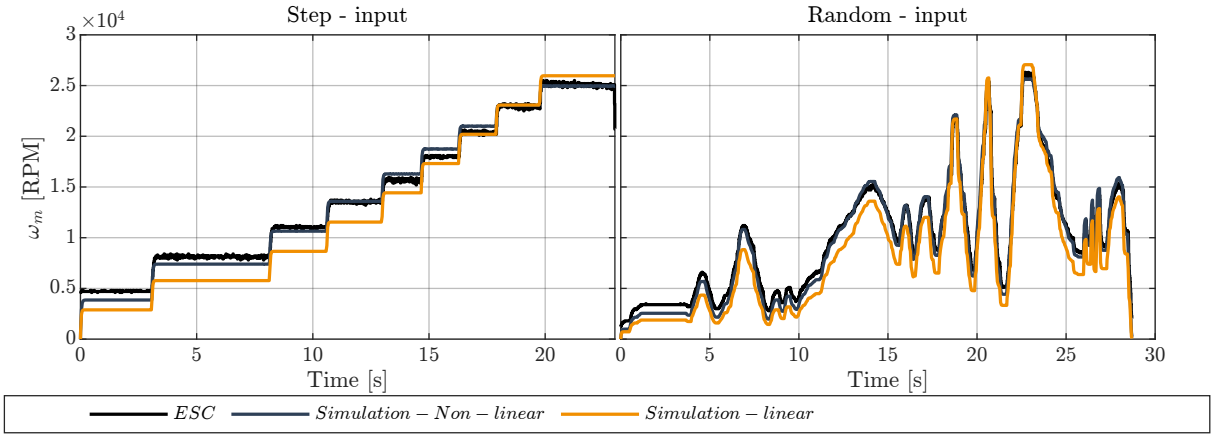


Figure 5.7. ESC measurement and non-linear BLDC model compared to the linear model at linearisation point $\omega_{m0} = 16298RPM$

5.1.5 Transfer function

A controller is developed for the BLDC motor in Chapter 6. The models must be transformed into the Laplace domain as a prerequisite for using classical control theory tools like the Bode diagram. Converting from the Laplace domain to the frequency domain can be done by substituting "s" with "$\omega \cdot j$", where j is the imaginary component. The Laplace transformation is done by replacing the time derivatives with "s" and the second-order time derivative with "s^2". This is done for the electrical circuit model from Equation (5.15) and the mechanical model from Equation (5.18). The Laplace transformations are given by Equation (5.19).

$$\begin{aligned} \mathcal{L}\{U^q\} &= U^q(s) = R_s \cdot i^q(s) + L_s \cdot i^q(s) \cdot s + K_m^q \cdot \omega_m(s) \\ \mathcal{L}\{\Delta\dot{\omega}_m\} &= \Delta\omega_m(s) \cdot J_{tot} \cdot s = K_m^q \cdot i^q(s) - B_{lin} \cdot \Delta\omega_m(s) \end{aligned} \tag{5.19}$$

The block diagram for the equivalent BLDC motor can be seen in Figure 5.8, showing that the back-EMF is a closed loop in the transfer function.

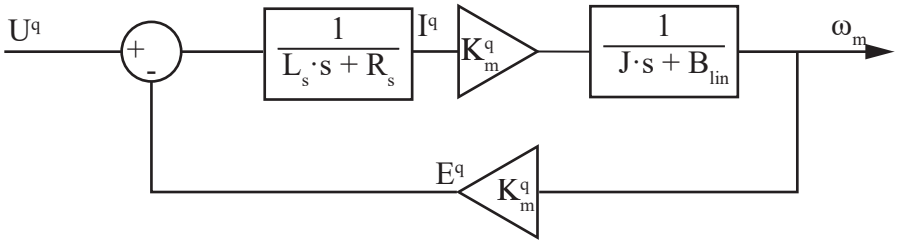


Figure 5.8. Block diagram of the equivalent BLDC motor

The electrical equation is solved for i^q as the back-EMF loop needs to be closed by implementing the mechanical equation, and $\Delta\omega_m$ is substituted with ω_m in the mechanical equation and is solved for $\frac{\omega_m}{i^q}$, which is done in Equation (5.20).

$$\begin{aligned} i^q(s) &= \frac{1}{R_s + L_s \cdot s} \cdot (U^q - K^q \cdot \omega_m(s)) \\ \frac{\omega_m(s)}{i^q(s)} &= \frac{K_m^q}{J_{tot} \cdot s + B_{lin}} \end{aligned} \tag{5.20}$$

Combining the two equations from Equation (5.20), the transfer function describing the velocity based on the voltage input is obtained. The transfer function for the BLDC G_{BLDC} is given by Equation (5.21) and will be used in Chapter 6 for the development of the BLDC controller.

$$G_{BLDC}(s) = \frac{\omega_m(s)}{U^q(s)} = \frac{K_m^q}{J_{tot} \cdot L_s \cdot s^2 + (B_{lin} \cdot L_s + J_{tot} \cdot R_s) \cdot s + B_{lin} \cdot R_s + (K_m^q)^2} \tag{5.21}$$

5.2 Thrust model

A thrust model has been proposed in Section 2.3 for an ideal propeller where there typically is a $10 - 15\%$ error in the power calculations. The proposed model is given by Equations (2.13) and (2.14), the equations have been combined into Equation (5.22).

$$P_T = T(v_{wind} + \left(-\frac{v_{wind}}{2} + \sqrt{\left(\frac{v_{wind}}{2}\right)^2 + \frac{T}{2\rho A_{pdisk}}} \right) \tag{5.22}$$

Assuming $v_{wind} = 0$ as data gathered in Appendix Section F.6 are measured in a laboratory environment with negligible airflow, the equation can be further simplified and solved for thrust T in Equation (5.23).

$$P_T = T\sqrt{\frac{T}{2\rho A_{pdisk}}} \rightarrow T = \left(2\rho A_{disk} P^2\right)^{\frac{1}{4}} \tag{5.23}$$

The thrust model is for an ideal propeller without losses. The power P_T can be assumed to be the input power from the BLDC $P_T = \tau_e \cdot \omega_m$ obtained from Equation (5.4). Substituting τ_e for $\tau_{prop} = C_p \cdot \omega_m^2$ assumes that all torque produced is used for thrust production. The combined equation is given by Equation (5.24).

$$T = \left(2\rho A_{pdisk}(C_p \omega_m^3)^2\right)^{\frac{1}{4}} \tag{5.24}$$

The comparison between the measured thrust graph and the thrust model can be seen in Figure 5.9. Because the power P_T is squared, the model cannot produce a negative thrust. The "Thrust model minus" line in the figure is calculated using logic, where the thrust is negative when ω_m is negative.

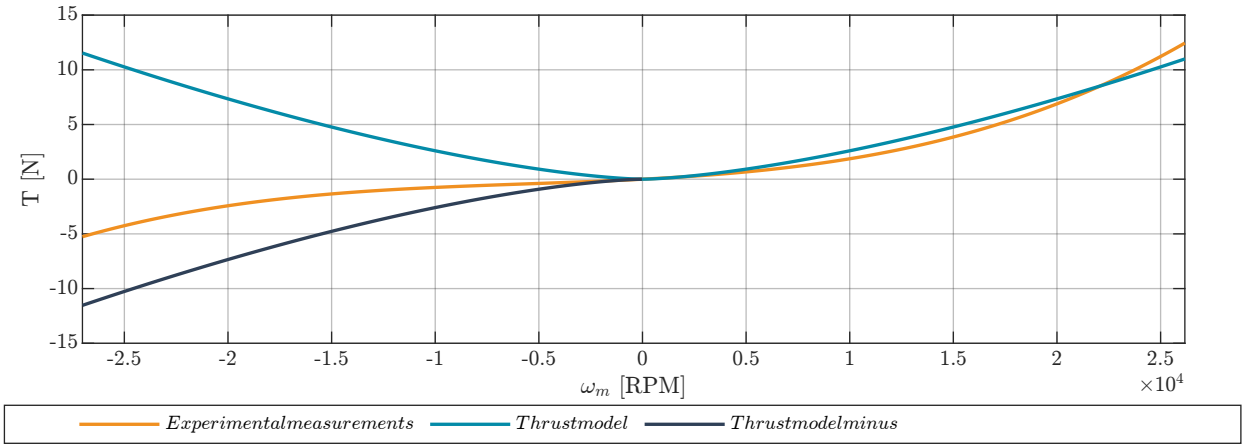


Figure 5.9. Comparison of thrust model vs experimental data.

The thrust model gives a decent estimate for the measured thrust in the positive thrust direction, where it shifts from overestimating to underestimating at $\approx 22100RPM$. Because the thrust is not symmetrical in both thrust directions, the thrust model overestimates the produced thrust in the negative thrust direction. The model can be fitted through A_{disk}, and an empirical power factor κ can be added to compensate for the discrepancy (Kumar 2021). The lack of voltage and current sensing for power calculation on the ESC complicates the implementation. Because of this, it is decided to implement lookup tables that directly translate motor velocity ω_m to thrust T.

A lookup table for the two thrust directions is implemented for the experimental measurements. Each lookup table consists of 350 points. The thrust in Newton is the input and is divided by the resolution of the table. The resolution is calculated by Equation (5.25)-

$$tab_{res} = \frac{T_{t.max}}{350} \tag{5.25}$$

$T_{t.max}$ is the maximum thrust of the table. Respectively $T_p = 13.98N$ and $T_n = 5.25N$ for the positive and negative thrust direction. The entrance to the lookup table is calculated by rounding the input in Newton divided by tab_{res} to the nearest integer. The lookup table for translating thrust T to motor velocity ω_m is plotted in Figure 5.10. A lookup table translating motor velocity ω_m to thrust T is also used in Chapter 6. This lookup table has the same line as the experimental measurements from Figure 5.9.

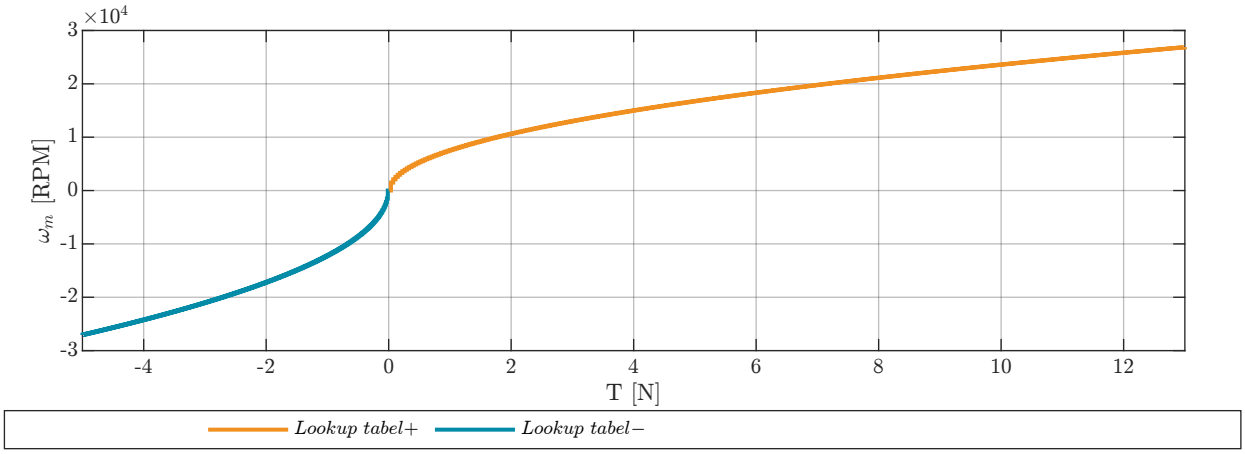


Figure 5.10. Lookup table as implemented in C for the positive thrust direction

5.3 Pendulum model

In this section, the pendulum model will be explored and elaborated on. Figure 5.11 shows an overview of the SPS model. The pendulum model consists of two dynamic models, the mechanical pendulum and the mechanical yaw model. The two models depend on the drag force, which uses the relative velocity between the air speed and the pendulum speed to calculate the drag forces. The parameters for the pendulum model are given in Table 5.3.

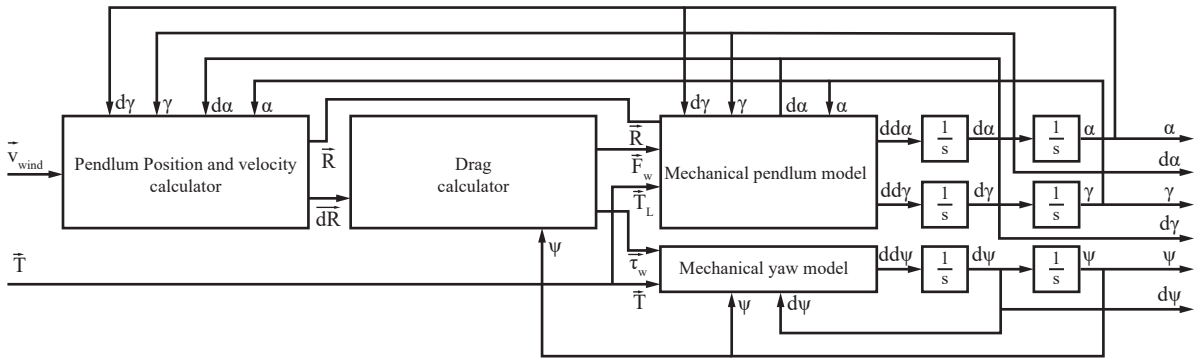


Figure 5.11. Pendulum model block diagram

The mechanical pendulum model and mechanical yaw model rely on Euler angle rotations, where Equation (5.26) gives the rotation matrices about a given X-, Y-, and Z-axis with an arbitrary angle χ.

Parameter	Value	Unit
Wire length (L_w)	1.9685	m
SPS weight (m)	4.556	kg
Drag coefficient ($C_{D.90}$)	1.616	–
Drag coefficient ($C_{D.0}$)	0.413	–
Viscous friction swing ($C_{v.\alpha}$)	0.494	$Nm \cdot s$
Viscous friction Yaw ($C_{v.\Psi}$)	0.11	$Nm \cdot s$
Mass moment of inertia Yaw (J_Ψ)	1.22	$kg \cdot m^2$

Table 5.3. Mechanical parameters of the pendulum model for the SPS test setup

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\chi_x) & -sin(\chi_x) \\ 0 & sin(\chi_x) & cos(\chi_x) \end{bmatrix} \quad R_y = \begin{bmatrix} cos(\chi_y) & 0 & sin(\chi_y) \\ 0 & 1 & 0 \\ -sin(\chi_y) & 0 & cos(\chi_y) \end{bmatrix} \quad (5.26)$$

$$R_z = \begin{bmatrix} cos(\chi_z) & -sin(\chi_z) & 0 \\ sin(\chi_z) & cos(\chi_z) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

5.3.1 Mechanical pendulum model

A spherical pendulum model can be used to describe the SPS system's moments in 3D space. Figure 5.12 shows a sketch of the representation of the pendulum model. This model representation assumes the wire is massless and the SPS mass to be a point mass, this way, two angles can describe the SPS position, the swing angle α and swing plane γ. The spherical pendulum model can be understood in a 3D coordinate system as a 2D pendulum that swings in a planar motion given by the swing angle α. The planar motion can be rotated in the 3D coordinate system by rotating the plane about the z-axis with the swing plane angle γ.

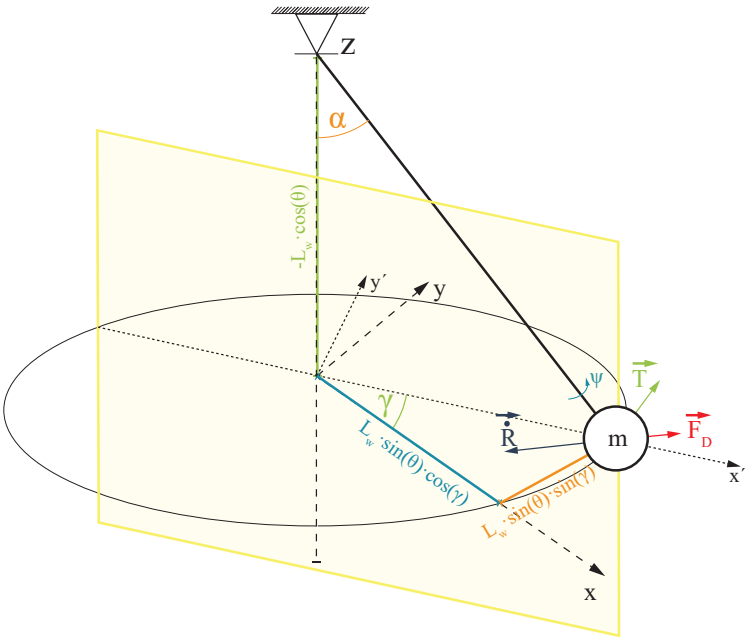


Figure 5.12. Spherical pendulum model and applied forces

Model assumptions and limitations

The spherical pendulum model has the following assumptions:

- Massless wire
- Rigid wire
- The SPS is always perpendicular to the wire
- SPS is a point mass
- The wire is supported by a hinge which only allows for 3D rotation - no movement in the x, y, and z-axis
- The drag coefficient is simplified to a 2D scenario by assuming α to be small
- $\ddot{\gamma}$ has a fixed inertia for simulation stability

The potential double pendulum effects are removed by assuming the wire to be massless and rigid and the SPS to be perpendicular to the wire. This assumption is expected to be valid because the SPS mass is much larger than the wire mass. The wire inertia is smaller than the SPS and will, therefore, follow and will not induce any moments. The reductions in DOFs will remove some dynamics that might be possible to see in the real system, but are expected to have a negligible influence.

Assuming a hinge below the drone supports the wire, no forces from the drone's movement will be transferred into the SPS system. This does not perfectly correlate to a real-world scenario where the DJI FlyCart30 has a small amount of movement as stated in Table 2.4. According to the table, the mass of the drone is 65 kg and the total payload mass with the SPS system is 30 kg. The influence of the payload on the drone cannot be neglected, but the drone is not available for this report, so the actual influence is unknown. This decision is discussed in Chapter 9.

Optimally, α is small, resulting in a stabilised system with minimal oscillations. Because of this, it can be assumed that the drag coefficient can be simplified to a 2D scenario.

5.3.2 Pendulum Lagrange mechanics equations

The pendulum mechanics will be derived through Lagrangian mechanics, where the Lagrangian is given by equation (5.27).

$$\mathscr{L} = T_{kin} - V_{pot} \tag{5.27}$$

T_{kin} is the kinetic energy and V_{pot} is the potential energy. Given by the model sketch in Figure 5.12, the model describes the mass by m at the end of the wire with length L_w, and the coordinates of the mass are described by α and γ.

As the system is in 3D, it can be described through vectors, where a ZY Euler transformation gives the mass coordinates. The coordinates are then given by Equation (5.28).

$$R = [x, y, z]^T = R_z(\gamma) \cdot R_y(\alpha) \cdot [0, 0, -L_w]^T \tag{5.28}$$

Simplifying Equation (5.28) yields the XYZ coordinates given by Equation (5.29), and differentiating with respect to time results in the velocities of the mass and are given by Equation (5.30).

$$\begin{aligned} x &= L_w \cdot sin(\theta) \cdot cos(\gamma) \\ y &= L_w \cdot sin(\theta) \cdot sin(\gamma) \\ z &= -L_w \cdot cos(\theta) \end{aligned} \quad (5.29) \qquad \begin{aligned} \dot{x} &= L_w \cdot (\dot{\alpha} \cdot cos(\alpha) \cdot cos(\gamma) - \dot{\gamma} \cdot sin(\alpha) \cdot sin(\gamma)) \\ \dot{y} &= L_w \cdot (\dot{\alpha} \cdot cos(\alpha) \cdot sin(\gamma) + \dot{\gamma} \cdot sin(\alpha) \cdot cos(\gamma)) \\ \dot{z} &= L_w \cdot \dot{\alpha} \cdot sin(\alpha) \end{aligned} \quad (5.30)$$

The Lagrangian can now be derived by calculating the kinetic energy by Equation (5.31) and the potential energy by Equation (5.32).

$$T_{kin} = \frac{1}{2} \cdot m \cdot v^2 \quad (5.31) \qquad V_{pot} = m \cdot g \cdot h \quad (5.32)$$

The velocity of the pendulum is calculated as the size of the vector $[\dot{x}, \dot{y}, \dot{z}]$ using the Pythagorean equation as $v = \sqrt{\dot{x}^2 + \dot{y}^2 + \dot{z}^2}$, and inserting into Equation (5.31) yeilds equation (5.33). The potential energy is given by Equation (5.32) where h corresponds to z. Equation (5.34) can then describe the potential energy.

$$T_{kin} = \frac{1}{2} \cdot m \cdot \left(\dot{\gamma}^2 \cdot sin(\alpha)^2 + \dot{\alpha}^2\right) \cdot L_w^2 \quad (5.33) \qquad V_{pot} = -m \cdot g \cdot L_w \cdot cos(\alpha) \quad (5.34)$$

The Lagrangian from Equation (5.27) can then be calculated. Applying Euler-Lagrange's equation in Equation (5.35), the moments can be calculated as a force equilibrium in generalised coordinates.

$$\frac{d}{dt}\frac{\partial}{\partial \dot{q_k}}\mathscr{L} - \frac{\partial}{\partial q_k}\mathscr{L} = M_{q_k} \quad (5.35)$$

Equation (5.35) results in two moment equations, which can be solved for angular acceleration. The angular acceleration for α is given by Equation (5.36), and the angular acceleration for γ is given by Equation (5.37). The derivation of the angular acceleration equations from the Euler-Lagrange equation is described in Appendix Section E.1.

$$\ddot{\alpha} = \frac{m \cdot L_w \cdot sin(\alpha) \cdot (L_w \cdot \dot{\gamma}^2 \cdot cos(\alpha) - g) - M_\alpha}{m \cdot L_w^2} \quad (5.36)$$

$$\ddot{\gamma} = \frac{-2 \cdot m \cdot L_w^2 \cdot \dot{\gamma} \cdot \dot{\alpha} \cdot sin(\alpha) \cdot cos(\alpha) + M_\gamma}{m \cdot L_w^2 \cdot sin(\alpha)^2} \quad (5.37)$$

To verify that these equations behave like a theoretical pendulum, a simulation has been run comparing the period of a theoretical pendulum with the simulation. Here, a pendulum with a mass of 30 kg and a wire length of 10 m is used. It was found that the model's period differs by 39 ms from the theoretical period of a pendulum with these parameters. The model is therefore expected to function correctly. More details about the verification are in Appendix Section E.3.

In both Equation (5.36) and (5.37), the denominator is the mass moment of inertia about the corresponding axes. For the angular acceleration $\ddot{\gamma}$, the inertia depends on α. When α is small, the inertia becomes small, resulting in large accelerations for $\ddot{\gamma}$. Additionally, if $\alpha = 0$, the model breaks because a division by zero occurs. To ensure that this does not happen, it was chosen to implement a permanent mass moment of inertia for $\ddot{\gamma}$ has been impelmentet were $m \cdot L_w^2 \cdot sin(\alpha)^2$ is set to I_γ witch is given a value of 12 $kg \cdot m^2$. In addition, a viscous friction $C_{v,\ddot{\gamma}} = 10\ Nms$

was added. This was only done in the simulation, as linear swings are the primary focus to be investigated, and it improved simulation stability. The influence of this decision is that the rotation of the swing plane does not depend on the angle α and therefore does not rotate faster when the α angle is small.

5.3.3 Mechanical yaw model

The spherical pendulum model assumes the SPS system to be a point mass and, therefore, does not account for the orientation or moments about the wire. In Chapter 4, a design with four propellers in a line was chosen as the best design, and the Mechanical yaw model has to represent that configuration. A sketch of the system can be seen in Figure 5.13.

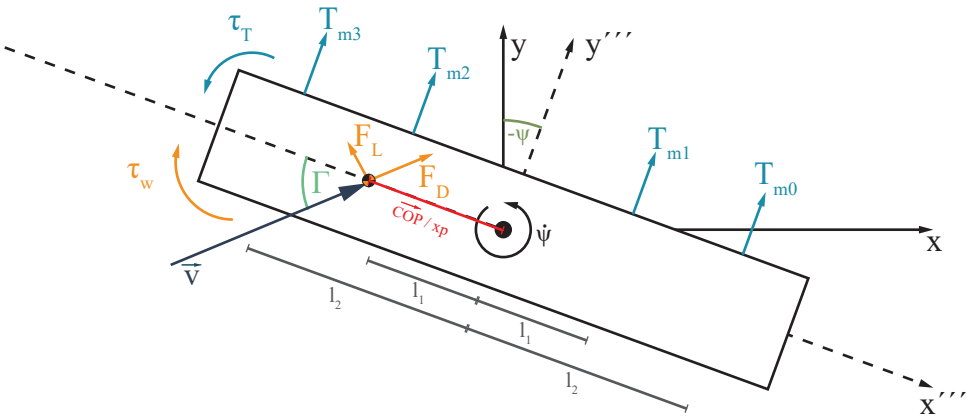


Figure 5.13. The pendulum box with the relative wind speed, drag force and thrust vectors and their corresponding moments

The yaw model is affected by aerodynamic forces, which are calculated according to Section 2.2. The yaw model can produce thrust along the local y-axis, specified as y''' when rotated about γ, α, and Ψ. Because the drag coefficient is assumed to be 2D in the xy-plane, and the wire is rigid and perpendicular to the SPS, only one rotation can occur, which is about the wire with the angle Ψ. It is therefore possible to describe the angular acceleration by Newton's second law, as seen in Equation (5.38).

$$\ddot{\Psi} = \frac{\tau_w + \tau_T - C_{v.\Psi} \cdot \dot{\Psi}}{J_\Psi} \tag{5.38}$$

τ_w is the sum of the torque contributed by drag and lift and is calculated by Equation (5.39). τ_T is the torque produced by the propellers and is controllable, and is calculated by Equation (5.40). $\vec{F}_D$ is the drag force and $\vec{F}_L$ is the lift force. l_1 and l_2 are the lengths from COM to the centre of the motor location as specified by the design from Figure 4.2. $C_{v.\Psi}$ is an estimate of the drag torques and viscous friction from the pivot hinge combined, when rotating with $\dot{\Psi}$.

$$\tau_w = C\vec{O}P \times (\vec{F}_D + \vec{F}_L) \tag{5.39}$$

$$\tau_T = T_{m0} \cdot l_2 + T_{m1} \cdot l_1 - T_{m2} \cdot l_1 - T_{m3} \cdot l_2 \tag{5.40}$$

An in-depth explanation of the force calculations and $C\vec{O}P$ can be found in Appendix Section E.2.

5.3.4 Pendlum Moments

The aerodynamic forces acting on the yaw model also act on the pendulum, inducing oscillations in α and deflecting the oscillation by γ. The drag coefficient is assumed to be 2D, where the relative wind velocity $\vec{v}_{rel}$ is the difference of the wind speed $\vec{v}_w$ and the velocity of the pendulum mass $\vec{R}$. $\vec{v}_{rel}$ is then used to calculate aerodynamic forces $\vec{F_w} = \vec{F_D} + \vec{F_L}$. The thrust produced by the motors $\vec{T}_{local}$ needs to be oriented by Equation (5.41), from the yaw model into the mechanical pendulum model $\vec{T}$, which is done by Equation (5.42). $\vec{T}_{local}$ is defined by the model and design as $\vec{T}_{local} = [0,\ T_{m0} + T_{m1} + T_{m2} + T_{m3},\ 0]$.

$$R_{ZYZ} = R_z(\gamma) \cdot R_y(\alpha)^T \cdot R_z(\Psi) \tag{5.41}$$

$$\vec{T} = R_{ZYZ} \cdot \vec{T}_{local} \tag{5.42}$$

The total force acting on the pendulum can be represented as a vector $\vec{F_{sum}}$ and is calculated by Equation (5.43), where $\vec{F_w}$ is the aerodynamic forces from drag and $\vec{T}$ is the sum of the thrust by the propellors.

$$\vec{F_{sum}} = \vec{F_w} - \vec{T} \tag{5.43}$$

The total moment contributed by F_{sum} is calculated by taking the cross product with $\vec{R}$, as shown in Equation (5.44).

$$\vec{M_{total}} = \vec{R} \times \vec{F_{sum}} \tag{5.44}$$

The Lagrange Equations (5.36) and (5.37) require an input moment M_α and M_γ. By calculating the dot product of $\vec{M_{total}}$ and a unit vector along the z-axis $\vec{\lambda_z}$, M_γ is calculated. The same can be done for M_α, where the unit vector $\vec{\lambda_{y'}}$ is perpendicular to the swing plane along the rotated y'-axis, and therefore needs to be rotated with $R_z(\gamma)$. The calculation of the unit vectors is given by Equation (5.45) (B. D. W. and H. W. 2025). A sketch of unit vectors and the corresponding pendulum system with moments is shown in Figure 5.14.

$$\begin{aligned} \vec{\lambda_{y'}} &= R_z(\gamma) \cdot [0, 1, 0]^T \\ \vec{\lambda_z} &= [0, 0, 1]^T \end{aligned} \tag{5.45}$$

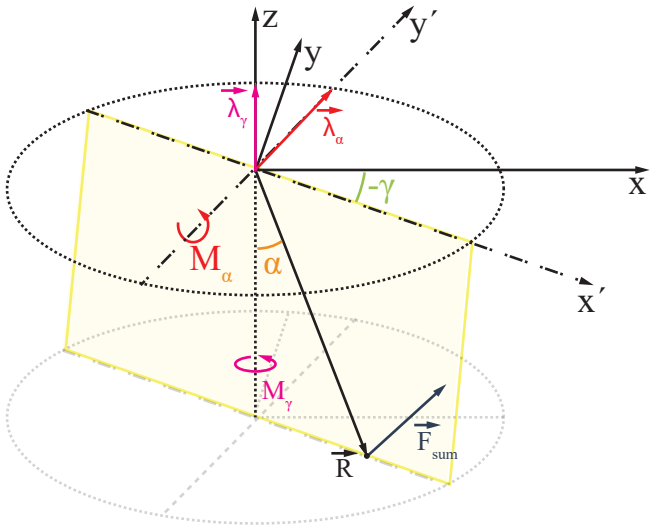


Figure 5.14. Unit vectors used for calculating moments for Lagrange equations

M_α and M_γ are calculated as shown in Equation (5.46), by the use of the dot product.

$$\begin{aligned} M_\alpha &= \vec{\lambda}_{y'} \bullet \vec{M}_{total} \\ M_\gamma &= \vec{\lambda}_z \bullet \vec{M}_{total} \end{aligned} \tag{5.46}$$

5.3.5 Fitting and validation of SPS model

To ensure that the presented SPS model fits the physical system, the drag coefficient, the viscous friction, and the mass moment of inertia of SPS are estimated through tests. The SPS is rotated and swung at $\Psi = 0°$ and $\Psi = 90°$ to estimate the drag coefficients. The attitude of the SPS system is measured using the AHRS system that will be further described in Section 7.4. To ensure consistency, the SPS system is raised to a swing angle of $\approx 20°$ before it is let go. The system then swings as shown in Figure 5.15, and measurements are made for both roll and pitch swings.

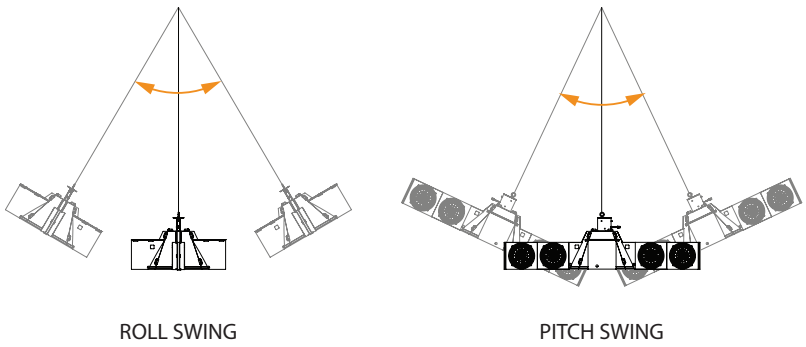


Figure 5.15. Swing directions used in the tests

When testing, it is optimal that Ψ is constant to only measure the wanted swing direction. The system is simplified to be 3 DOF, and none are controlled; Ψ will deviate from its original position to the nearest aerodynamically stable position. A sketch of the rough position of the aerodynamically stable point is shown in Figure 5.16. The rotation towards the aerodynamically stable point makes the measurements unusable, as they are no longer pure roll or pitch swings. To counteract the deviation, Ψ is kept constant manually during testing. The influence of the manual force input is minimal and will influence the results a lot less than letting Ψ deviate, although Ψ reached a maximum deviation of $\Psi \approx 17°$ during testing.

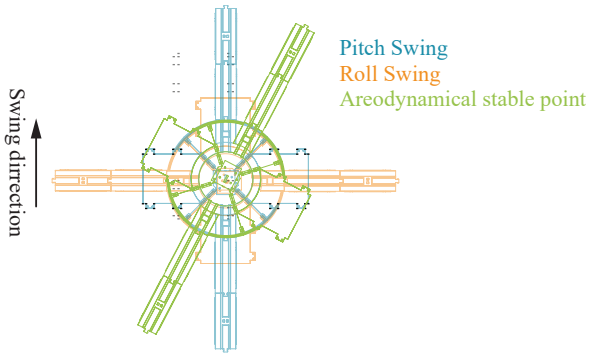


Figure 5.16. The aerodynamically stable point as observed through testing

Data were collected from the roll and pitch swing tests to create Figure 5.17. The fitted model simulated in Simulink is also shown. The "Parameter estimator" in Simulink was used to estimate the parameters, and the results are shown in Table 5.4. $C_{D.90}$ and $C_{v.\alpha}$ was estimated by the roll swing test data, and $C_{D.0}$ was estimated by the pitch swing data.

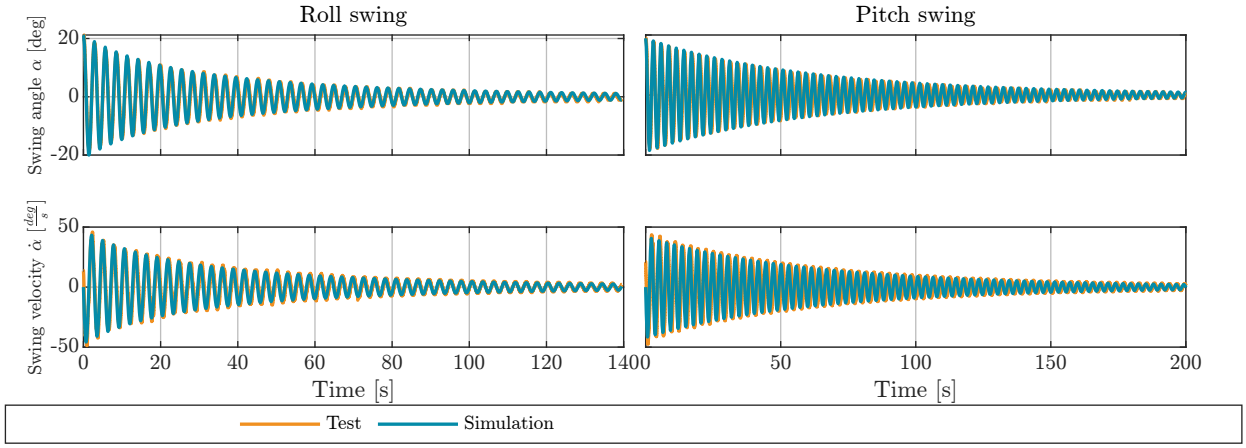


Figure 5.17. Swing angle over time for SPS system in test and simulation

$C_{D.90}$	1.616
$C_{D.0}$	0.413
$C_{v.\alpha}$	$0.494 Nm \cdot s$

Table 5.4. Estimated parameters for roll and pitch swing tests

It can be seen in Figure 5.17 that the model fits the measurements well over time. The roll and pitch swing simulation matches the measurements with almost identical maximum and minimum amplitudes for both angle and angular velocity.

More importantly, it is necessary to match the period of the measurements with the simulation. The Matlab function "findpeaks" was used to find the maximum at each oscillation. The data used for the parameter determination was also used to determine the period, where the angles and peaks are shown in Figure 5.18. The timestamps for each peak were compared for an average period and frequency.

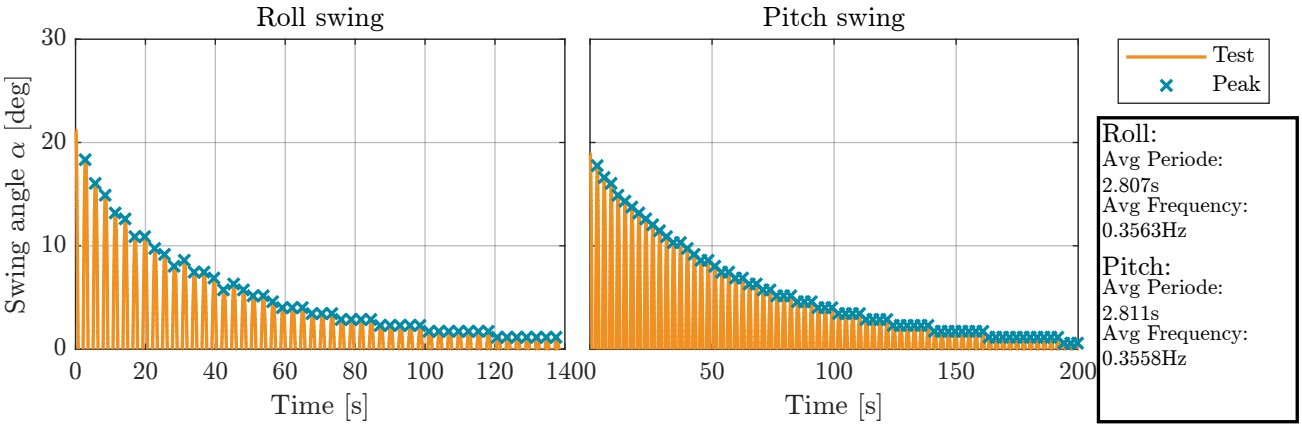


Figure 5.18. Peak detection on the swing angle over time to determine the period of the swings in roll and pitch

The process was repeated for the simulation data, where the results between the theoretical, test, simulation, and linear model simulation periods are compared. The linear model is compared with and without enabled drag forces, described in Section 5.3.6. The results are shown in Table 5.5. The theoretical period is calculated by $2 \cdot \pi \cdot \sqrt{\frac{L_w}{g}}$ (sciencefacts.net 2023).

	Roll Periode (T_f)	Pitch Periode (T_f)	Roll Frequency (f)	Pitch Frequency (f)
Theoretical	$2.813s$	$2.813s$	$0.3555Hz$	$0.3555Hz$
Test	$2.807s$	$2.811s$	$0.3563Hz$	$0.3558Hz$
Simulation	$2.816s$	$2.816s$	$0.3551Hz$	$0.3552Hz$
Simulation Linear	$2.816s$	$2.816s$	$0.3550Hz$	$0.3550Hz$
Simulation Linear w.d	$2.816s$	$2.816s$	$0.3550Hz$	$0.3550Hz$

Table 5.5. Period and frequency of the theoretical, measured, and simulated pendulum.

Table 5.5 shows that the theoretical period lies between the measured and simulated period, where the simulated period is $0.003s$ higher, and the test period is $0.006s$ lower at the most. The deviation is small, and the difference of $0.009s$ between the simulated and test period is within an acceptable margin of error.

Estimation of yaw model parameters

Lastly, the parameters for yaw are estimated. For yaw, the mass moment of inertia J_Ψ and the resistive torque produced by the aerodynamic forces during rotation and the friction between the metal plates on the swivel hook. The drag coefficient and swivel hook friction are simplified to a combined viscous friction $C_{v.\Psi}$.

Three measurements will be made to estimate the parameters where the motor m_0 will rotate at three different speeds, resulting in different torques. This will result in an angular acceleration from which J_Ψ and $C_{v.\Psi}$ can be estimated by utilising Equation (5.38). A sketch of the test can be seen in Figure 5.19, and the motor speeds, thrusts, and torques can be seen in Table 5.6, where the torque arm to the propeller is $0.4475m$.

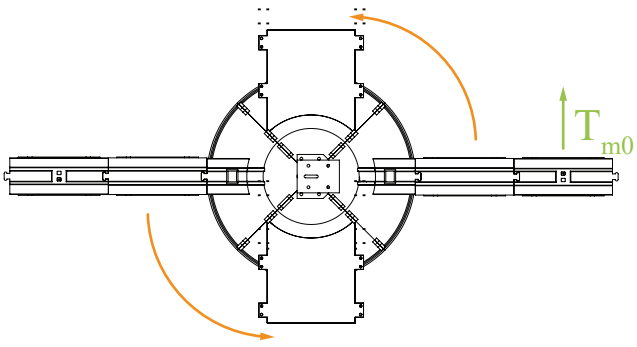


Figure 5.19. Yaw response test

	Values	Units
ω_{m0}	$[4775, 5730, 9550]$	RPM
T_{m0}	$[0.376, 0.553, 1.647]$	N
τ_T	$[0.168, 0.248, 0.737]$	Nm

Table 5.6. Table of motor velocity and corresponding thrust and produced torque.

The motor velocities are set as a step input for the motor controller, resulting in a short rise time before a steady state thrust is achieved, which is further specified in Section 6.1 where the motor controller is derived. To ensure consistency between test and simulation, the test velocities are input into the simulation, from where the thrust is calculated via the same lookup table used on the Nucleo board. This is described in Section 5.2.

The results of the test and corresponding fitted simulation can be seen in Figure 5.20, where the estimated mass moment of inertia J_Ψ and viscous friction $C_{v.\Psi}$ are shown in Table 5.7.

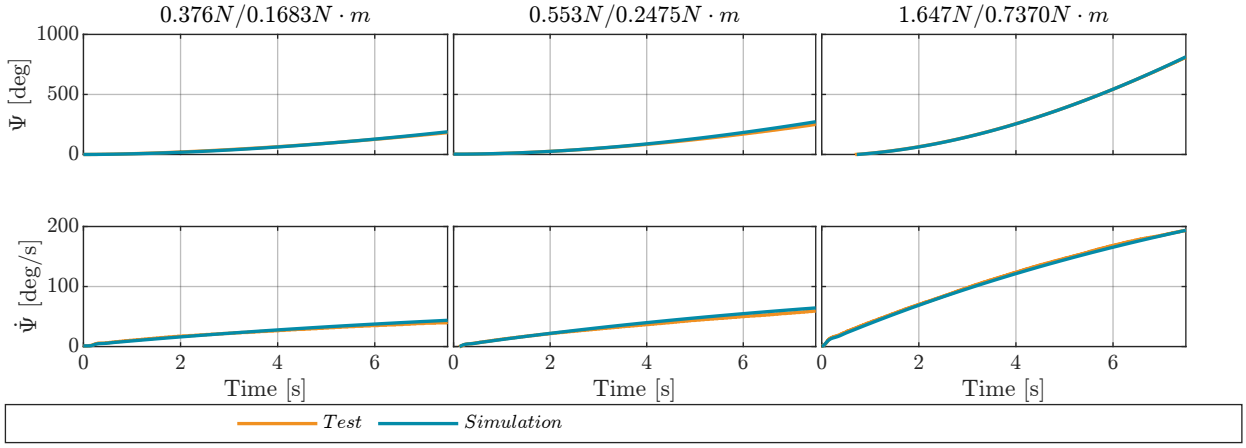


Figure 5.20. Yaw test and simulation response

J_Ψ	$1.22kg \cdot m^2$
$C_{v.\Psi}$	$0.11Nm \cdot s$

Table 5.7. Estimated parameters for yaw rotation test

From Figure 5.20, it can be seen that the simulation with the estimated parameters matches the test response well in both angular velocity $\dot{\Psi}$ and position Ψ. It can then be concluded that the mechanical pendulum and the Mechanical yaw model represent the physical system.

5.3.6 Linear pendulum model

The SPS model is required to be linear if linear control theory is to be applied. As the SPS model shown has three states to be controlled and four inputs in the form of each propeller's thrust, applying Multi-Input Multi-Output (MIMO) control theory is advantageous. Section 6.2 describes this derivation and tuning. The equations required to be linearised are Equation (5.36) and (5.37) for the mechanical pendulum dynamics and Equation (5.38) for the mechanical yaw model dynamics.

The only non-linear input in Equation (5.38) is the torques produced by drag τ_w. τ_w is described by Equation (5.39), which is a cross product between the $C\vec{O}P$ and the force vectors from drag and lift, and is dependent on the direction and amplitude of the wind vector. τ_w can therefore, in one situation, contribute to $\ddot{\Psi}$ and, in another situation, counteract $\ddot{\Psi}$, and it cannot be predicted which one it will be, along with how strong the torque is. τ_w is therefore removed through the linearisation resulting in the linearised $\ddot{\Psi}$ in Equation (5.47).

$$\ddot{\Psi} = \frac{(T_{m0} \cdot l_2 + T_{m1} \cdot l_1 - T_{m2} \cdot l_1 - T_{m3} \cdot l_2) - C_{v.\Psi} \cdot \dot{\Psi}}{J_\Psi} \tag{5.47}$$

The pendulum moments were derived in Section 5.3.4 where the rotation matrix R_{ZYZ} gives the relation between the local and global thrust vector. It is desired to linearise R_{ZYZ} for Ψ where α is dampened the most, which is at $\Psi = \pm 90°$. When $\Psi = \pm 90°$, the $\vec{T}_{local}$ is rotated from the y-axis into the swing plane and all the thrust is thereby used for dampening α. The wire vector is given by Equation (5.28), and the global thrust vector $\vec{T}$ is given by Equation (5.42). When $\Psi = \pm 90°$, the equations are given by Equations (5.48).

$$\vec{R} = \begin{bmatrix} L_w \cdot cos(\gamma) \cdot sin(\alpha) \\ L_w \cdot som(\alpha) \cdot sin(\gamma) \\ -L_w \cdot cos(\alpha) \end{bmatrix} \qquad \vec{T} = \pm 1 \cdot \begin{bmatrix} \frac{1}{2} \cdot \sqrt{2} \cdot cos(\gamma) \\ \frac{1}{2} \cdot \sqrt{2} \cdot sin(\gamma) \\ \frac{1}{2} \cdot \sqrt{2} \end{bmatrix} \tag{5.48}$$

For the purpose of linearisation $\vec{F}_D = 0$ such $\vec{F}_{sum} = -\vec{T}$. The total moment contribution $\vec{M}_{total}$ of $\vec{F}_{sum}$ is calculated with the cross product on $\vec{R}$ in Equation (5.44). The cross product result when $\Psi = \pm 90°$ is calculated in Equations (5.49).

$$\begin{aligned} \vec{M}_{total.\pm 90} = \vec{R} \times \vec{F}_{sum} = \begin{bmatrix} \vec{R}_y \cdot \vec{F}_{sum.z} - \vec{R}_z \cdot \vec{F}_{sum.y} \\ \vec{R}_z \cdot \vec{F}_{sum.x} - \vec{R}_x \cdot \vec{F}_{sum.z} \\ \vec{R}_x \cdot \vec{F}_{sum.y} - \vec{R}_y \cdot \vec{F}_{sum.x} \end{bmatrix} = \\ \pm 1 \cdot \begin{bmatrix} -\frac{1}{2} \left(\sqrt{2} \cdot \vec{F}_{sum.y} \cdot L_w \cdot cos(\alpha) \cdot sin(\gamma) - \sqrt{2} \cdot \vec{F}_{sum.y} \cdot L_w \cdot sin(\alpha) \cdot sin(\gamma) \right) \\ \frac{1}{2} \left(\sqrt{2} \cdot \vec{F}_{sum.y} \cdot L_w \cdot cos(\alpha) \cdot cos(\gamma) - \sqrt{2} \cdot \vec{F}_{sum.y} \cdot L_w \cdot cos(\gamma) \cdot sin(\alpha) \right) \\ 0 \end{bmatrix} \end{aligned} \tag{5.49}$$

The moments about M_α and M_γ are calculated with the dot product in Equation (5.46). The results of the dot products can be seen in Equation (5.50).

$$\begin{aligned} M_{\alpha.\pm 90} &= \vec{\lambda}_{y'} \bullet \vec{M}_{total} \neq 0 \\ M_{\gamma.\pm 90} &= \vec{\lambda}_z \bullet \vec{M}_{total} = 0 \end{aligned} \tag{5.50}$$

When $\Psi = 90°$, $M_{\alpha.\pm 90}$ is nonzero and $M_{\gamma.\pm 90}$ is zero. The system can therefore not control the angle γ, so an alternative solution is necessary. M_α is nonzero and is dependent on both α and γ. Fortunately, the model assumptions made for the system assume the SPS is always perpendicular to the wire, which can be tested with the dot product between $\vec{R}$ and $\vec{M}_{total}$ in Equation (5.51).

$$\text{⊾} = \vec{R} \bullet \vec{M}_{total} = 0 \tag{5.51}$$

Because ⊾ is zero, the two vectors are perpendicular, which results in M_α being calculated as the sum of the thrust times the length of the wire vector $\vec{R}$ which is given by L_w resulting in Equation (5.52).

$$M_\alpha = L_w \cdot \sum_{i=0}^{3} T_{m.i} \tag{5.52}$$

The pendulum dynamics described by Equation (5.36) and (5.37) are linearised by a first-order Taylor series approximation. The first-order Taylor series approximation structure is shown in Equation (5.53) for a given function $f(x)$ at a linearisation point x_0 (Iqbal 2025).

$$P(x) = f(x_0) + \frac{df(x_0)}{dx_0} \cdot (x - x_0) \tag{5.53}$$

The Taylor series approximation is transformed to a delta variable solution, where the change in output is described. Equation (5.54) defines the delta variable Taylor series approximation.

$$P(x) - f(x_0) = \frac{df(x_0)}{dx_0} \cdot (x - x_0) \rightarrow \Delta f(x) = \frac{df(x_0)}{dx_0} \cdot \Delta x \tag{5.54}$$

The derivative $\frac{df(x_0)}{dx_0}$ will become a constant of the function, where the linearisation values will change the dynamics of the linear model. The delta variable Taylor series approximation is applied to each variable in Equation (5.36) and (5.37), which are $\alpha, \dot{\alpha}, \dot{\gamma}$. Equation (5.55) gives the first-order Taylor series approximation.

$$\begin{aligned}
\Delta\ddot{\alpha} &= \left.\frac{\partial\ddot{\alpha}}{\partial\alpha}\right|_{\alpha_0,\dot{\alpha}_0,\dot{\gamma}_0} \cdot \Delta\alpha + \left.\frac{\partial\ddot{\alpha}}{\partial\dot{\alpha}}\right|_{\alpha_0,\dot{\alpha}_0,\dot{\gamma}_0} \cdot \Delta\dot{\alpha} + \left.\frac{\partial\ddot{\alpha}}{\partial\dot{\gamma}}\right|_{\alpha_0,\dot{\alpha}_0,\dot{\gamma}_0} \cdot \Delta\dot{\gamma} \\
\Delta\ddot{\gamma} &= \left.\frac{\partial\ddot{\gamma}}{\partial\alpha}\right|_{\alpha_0,\dot{\alpha}_0,\dot{\gamma}_0} \cdot \Delta\alpha + \left.\frac{\partial\ddot{\gamma}}{\partial\dot{\alpha}}\right|_{\alpha_0,\dot{\alpha}_0,\dot{\gamma}_0} \cdot \Delta\dot{\alpha} + \left.\frac{\partial\ddot{\gamma}}{\partial\dot{\gamma}}\right|_{\alpha_0,\dot{\alpha}_0,\dot{\gamma}_0} \cdot \Delta\dot{\gamma}
\end{aligned} \tag{5.55}$$

The following terms give the partial derivatives at an unknown linearisation point:

$$\left.\frac{\partial\ddot{\alpha}}{\partial\alpha}\right|_{\alpha_0,\dot{\alpha}_0,\dot{\gamma}_0} = (cos(\alpha_0)^2 - sin(\alpha_0)^2) \cdot \dot{\gamma}_0^2 - \frac{cos(\alpha_0) \cdot g}{L} \quad \left.\frac{\partial\ddot{\alpha}}{\partial\dot{\alpha}}\right|_{\alpha_0,\dot{\alpha}_0,\dot{\gamma}_0} = -\frac{C_{v.S}}{m \cdot L^2} \quad \left.\frac{\partial\ddot{\alpha}}{\partial\dot{\gamma}}\right|_{\alpha_0,\dot{\alpha}_0,\dot{\gamma}_0} = 2 \cdot sin(\alpha_0) \cdot cos(\alpha_0)$$

$$\left.\frac{\partial\ddot{\gamma}}{\partial\alpha}\right|_{\alpha_0,\dot{\alpha}_0,\dot{\gamma}_0} = \frac{2 \cdot \dot{\gamma}_0 \cdot \dot{\alpha}_0 \cdot cos(\alpha_0)^2}{sin(\alpha_0)^2} + 2 \cdot \dot{\gamma}_0 \cdot \dot{\alpha}_0 \quad \left.\frac{\partial\ddot{\gamma}}{\partial\dot{\alpha}}\right|_{\alpha_0,\dot{\alpha}_0,\dot{\gamma}_0} = -\frac{2 \cdot \dot{\gamma}_0 \cdot cos(\alpha_0)}{sin(\alpha_0)} \quad \left.\frac{\partial\ddot{\gamma}}{\partial\dot{\gamma}}\right|_{\alpha_0,\dot{\alpha}_0,\dot{\gamma}_0} = -\frac{2 \cdot \dot{\alpha}_0 \cdot cos(\alpha_0)}{sin(\alpha_0)}$$

Next, the linearisation points are to be determined. This is done by comparing the results of the non-linear model with the linearised model at a given linearisation point and comparing the change in accelerations. The chosen linearisation points result in the acceleration changes shown in Figure 5.21, plotting the non-linear model and linear model results with the linearisation points in the top row. The bottom row shows the linear model error.

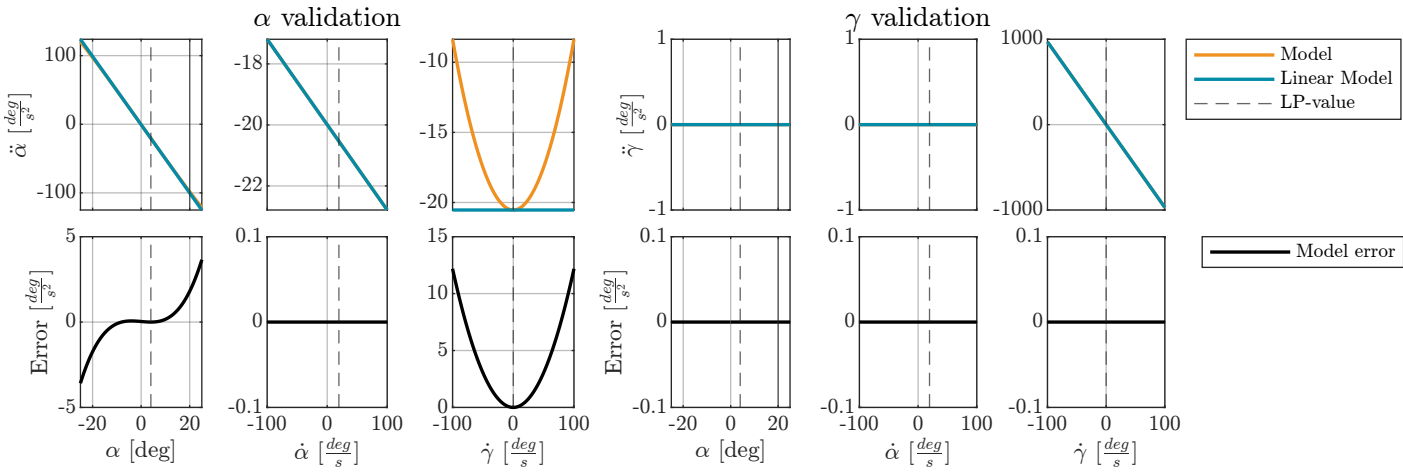


Figure 5.21. Comparison of model states at the linearisation point chosen.

The linearisation points are given in Equation (5.56) and the corresponding Taylor series linearisation constants are given in Table 5.8. $\dot{\gamma}_0$ is defined to be $0\frac{rad}{s}$ because of a coupling between $\Delta\ddot{\gamma}$ and $\dot{\alpha}$ where if any velocity of $\dot{\alpha}$ in $\frac{\partial\ddot{\gamma}}{\partial\alpha}\Big|_{\alpha_0,\dot{\alpha}_0,\dot{\gamma}_0}$ would result in an acceleration in $\Delta\ddot{\gamma}$ and does not correspond with the real system dynamics. The consequence of $\dot{\gamma}_0 = 0\frac{rad}{s}$ is expected to be errors in the linear model during spiral movements of the pendulum. The linearisation value for $\dot{\alpha}$ is chosen easily as the derivative is a constant in $\Delta\ddot{\alpha}$. The wanted value for α_0 decides the linearisation point for α. α is wanted to be small, but α_0 will be non-zero to preserve the swing dynamics of the model.

$$x_0 = \begin{bmatrix} \alpha_0 \\ \dot{\alpha}_0 \\ \dot{\gamma}_0 \end{bmatrix} = \begin{bmatrix} 0.07rad \\ 0.34rad/s \\ 0rad/s \end{bmatrix} \tag{5.56}$$

linearisation constant	Value	Unit
$K_{T\alpha} = \frac{\partial\ddot{\alpha}}{\partial\alpha}\Big\|_{x_0}$	-4.9764	rad/s^2
$K_{T\dot{\alpha}} = \frac{\partial\ddot{\alpha}}{\partial\dot{\alpha}}\Big\|_{x_0}$	-0.0280	rad/s
$K_{T\dot{\gamma}} = \frac{\partial\ddot{\alpha}}{\partial\dot{\gamma}}\Big\|_{x_0}$	0	rad/s
$K_{G\alpha} = \frac{\partial\ddot{\gamma}}{\partial\alpha}\Big\|_{x_0}$	0	rad/s^2
$K_{G\dot{\alpha}} = \frac{\partial\ddot{\gamma}}{\partial\dot{\alpha}}\Big\|_{x_0}$	0	rad/s
$K_{G\dot{\gamma}} = \frac{\partial\ddot{\gamma}}{\partial\dot{\gamma}}\Big\|_{x_0}$	-9.6984	rad/s

Table 5.8. Linearisation point for the Taylor series approximation constants from Equation (5.56)

The linearised model is compared to the non-linear model and test measurements. All models have the same initial conditions as used in Section 5.3.5. The linear model is simulated with and without the influence of aerodynamic forces. The aerodynamic forces are added as a non-linear disturbance where the wind velocity is 0, so they are calculated from the velocity of the SPS system given by the models. The aerodynamic forces are added to the linearised system by adding Equation (5.57) and (5.58) to (5.55) respectively.

$$\frac{M_\alpha}{m \cdot L^2} \tag{5.57}$$

$$\frac{M_\gamma}{m \cdot L^2 \cdot sin(\alpha)^2} \tag{5.58}$$

The results are shown in Figure 5.22 for the roll swing and Figure 5.23 for the pitch swing. The linear roll swing falls slower than simulation and test results, where a maximum error of $\approx 5.22°$ and $\approx 11.5\ °/s$ is obtained. Adding the aerodynamic forces lowers the errors to $\approx 1.44°$ and $\approx 4.1\ °/s$. Referring back to Table 5.5, it can be seen that the linear model matches the periods of the simulation and test.

From Figure 5.22 and 5.23 and Table 5.5, it is concluded that the linear model gives reasonable estimates for velocities and positions. The linear model is deemed sufficient for use in a controller design, although the models have limitations regarding control. This will be elaborated in the MIMO control design in Section 6.2.

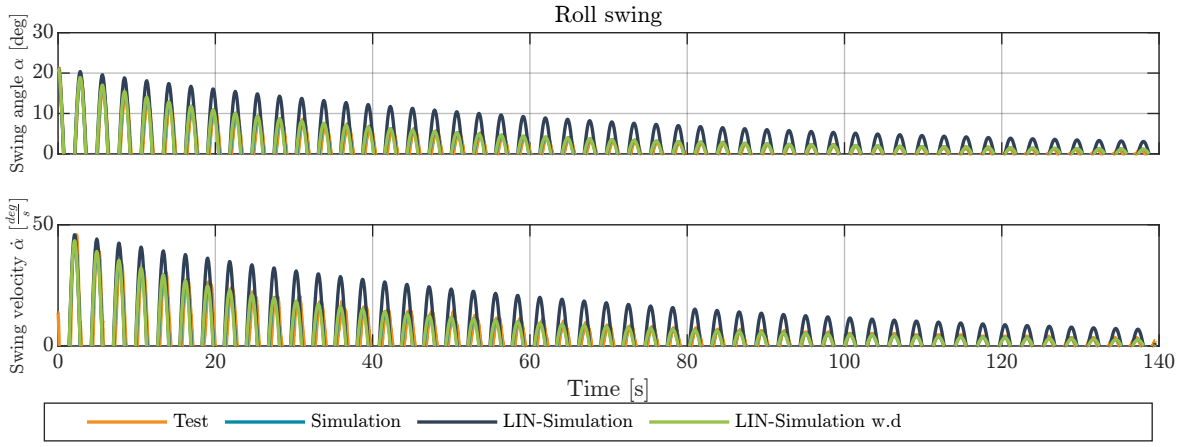


Figure 5.22. Results of Roll swing with Simulation results (w.d = with disterbance)

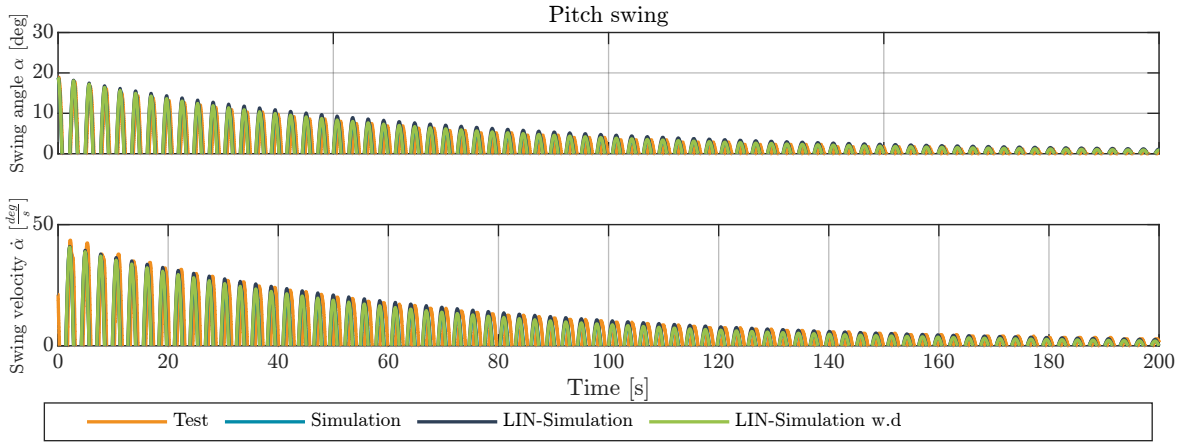


Figure 5.23. Results of Pitch swing with Simulation results (w.d = with disterbance)

Controllers 6

This chapter will develop the controllers for the BLDC motors and the pendulum system, both described in Chapter 5. The main objective is to control the attitude of the pendulum system. To do this, thrust is needed, and therefore, the BLDC motors must also be controlled. Therefore, this chapter will design and tune controllers for the BLDC motors and the pendulum system. For each system, the structure and controller will be described and simulated in continuous and discrete time before being implemented in the MCU.

In Figure 6.1, the control algorithm implemented in the SPS system is sketched. The blue box contains the MIMO control system for the pendulum, which controls the attitude by requesting a thrust T_{m-ref}. The MIMO system control structure is LQI with anti-integral wind-up and is described in Section 6.2. The Single-Input Single-Output (SISO) control system requires a velocity reference ω_{m-ref} as defined by the transfer function in Section 5.1.5. A lookup table transfers the requested thrusts T_{m-ref} to ω_{m-ref}. The lookup table is implemented because of the asymmetry and non-linearity in the thrust curve as described in Section 2.3. Figure 6.1 shows an additional lookup table after the SISO controllers that transfers the velocities back into thrust in the Simulink implementation. This lookup table is only implemented in simulation and is not required on the MCU. The thrust outputs are then given to the pendulum model, which reacts and provides feedback to the MIMO controller.

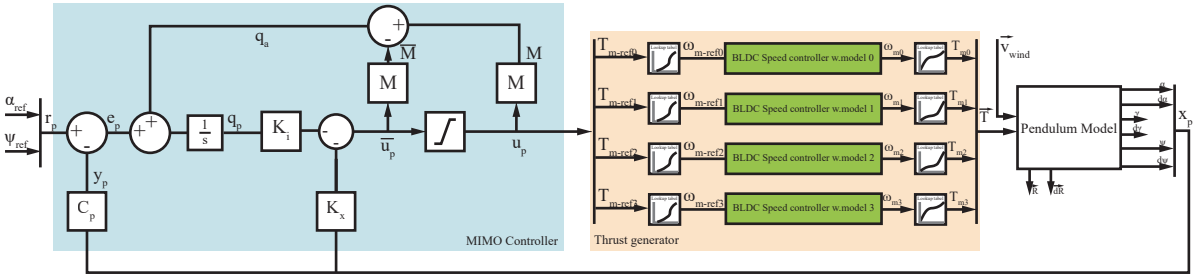


Figure 6.1. Structure of full SPS control algorithm. As implemented in Simulink

6.1 BLDC motor controller

The BLDC motor controller block is shown in the thrust generator box in Figure 6.1, where Figure 6.2 shows what is inside the block. The BLDC motor controller receives a reference velocity ω_{m-ref} from the thrust-velocity lookup table. $\omega_{m.error}$ is calculated as the error between ω_{m-ref} and ω_m. The error is passed to the SISO controller C_{BLDC}, which outputs a voltage U^q, which is then given to the BLDC motor. In the simulation, the orange box shown in Figure 6.2 contains the BLDC transfer function G_{BLDC} as derived in Section 5.1.5. G_{BLDC} transfers the

voltage U^q to a motor velocity ω_m, which is then passed through a velocity-thrust lookup table. G_{BLDC} and the velocity-thrust lookup table are not implemented on the MCU.

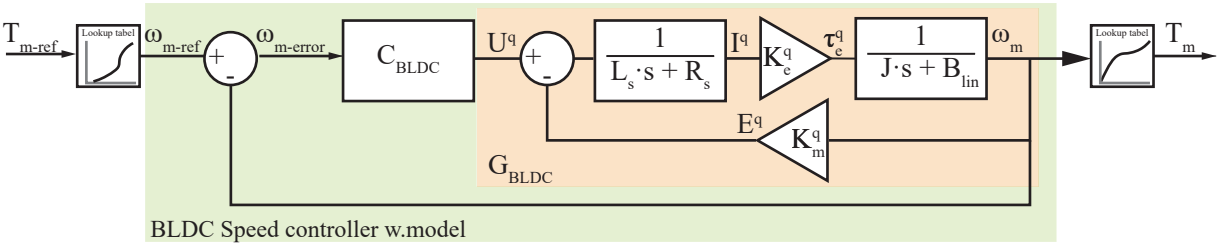


Figure 6.2. Block diagram of BLDC control structure

For controller design, loop-shaping is applied, where the hardware specifies the limits of the tuning process. For proper control, it is required to have ≈ 10 measurements during the rise time (Bech 2022). In Section 7.3, it is specified that the sample time for the BLDC motors is given at $8.5ms$. To have 10 measurements on the MCU, the rise time of the BLDC controller should be at least $85ms$.

As classical control theory is used, it is possible to take an offset in the Bode plot of the transfer function. The bandwidth can be determined from the Bode plot where the magnitude crosses $-3dB$. The bandwidth indicates the speed of the system, where frequencies below are disturbances and frequencies above are noise. The gain then indicates how well the controller can reject disturbances and noise. The phase plot indicates how well the system is dampened, where a phase of $-90°$ indicates the system is critically damped. A phase of $> -90°$ overdamps the system, and a phase of $< -90°$ underdamps the system. A controller with the desired criteria can be tuned from the Bode plot. The Bode plot for G_{BLDC} is shown in Figure 6.3.

The Bode plot indicates steady-state error at lower frequencies because G_{BLDC} does not have a slope below $10^1 \frac{rad}{s}$. Therefore, a Proportional-Integral (PI) controller is designed as a controller and is shown in Equation (6.1).

$$C_{BLDC}(s) = k \cdot \frac{1}{s} \tag{6.1}$$

In Figure 6.4, the step response of G_{BLDC} is shown, and the rise time is calculated to be $0.25\ ms$. At this speed, it is not possible to have the required 10 samples during the rise time. To make the system slower, the controller gain k is tuned to lower the bandwidth of the system, such that 10 measurements during the rise time on the MCU are possible. The gain is tuned to be $k = 0.085$. The free integrator $\frac{1}{s}$ removes steady state error by adding a slope of $-20dB$ per decade.

Equation (6.2) gives the open-loop representation of the controller and plant. The open-loop transfer function specifies the expected dynamics of the system when the controller is added.

$$OL = C_{BLDC}(s) \cdot G_{BLDC}(s) \tag{6.2}$$

The Bode plot of the plant G_{BLDC}, the controller C_{BLDC}, and open-loop OL are shown in Figure 6.3. The Bode plot of the controller C_{BLDC} is just a slope of $-20dB$ per decade with a lower gain margin, resulting in OL having a slope of $-20dB$ at frequencies up to $\approx 50\frac{rad}{s}$ and a slope of $-40dB$ per decade for frequencies above. The controller gain k lowers the bandwidth from $\approx 5 \cdot 10^3 rad/s$ to $\approx 15 rad/s$. The phase margin is 67.68°, making the OL system slightly underdamped.

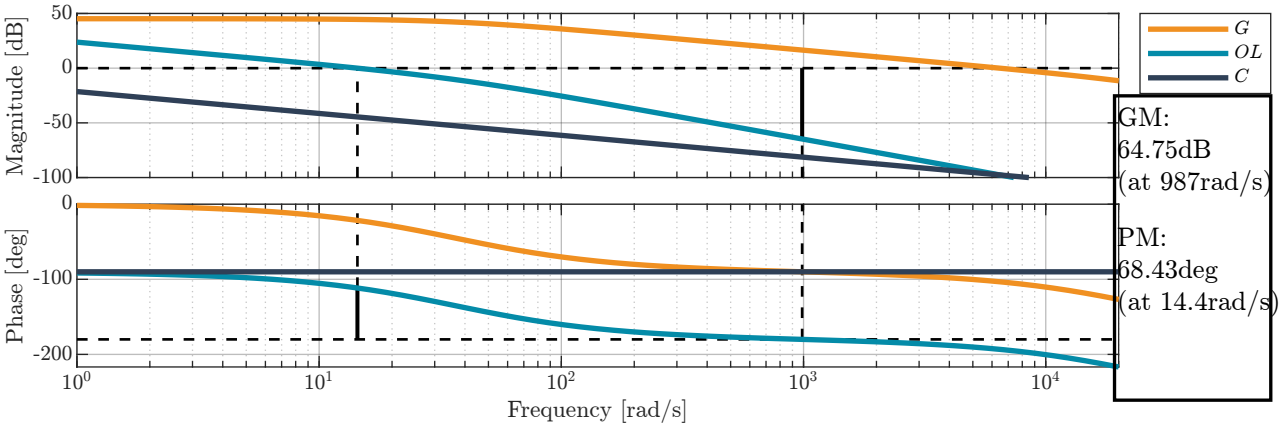


Figure 6.3. Bode diagram of the BLDC system, BLDC system with controller and the controller

The step response of the BLDC system is also interesting to investigate, as it is given by the closed-loop dynamics CL. Through block reduction of the block diagram in Figure 6.2, the closed-loop dynamics are given by Equation (6.3).

$$CL = \frac{OL}{1 + OL} \tag{6.3}$$

The step response of the closed-loop dynamics of the plant G_{BLDC} ($CL : G$) and the closed-loop of OL $CL : OL$ are both shown in Figure 6.4. The figure shows that the rise time increases from $0.25ms$ to $98.9ms$. The settling time is $233ms$ with an overshoot of 2.332%. The closed-loop dynamic of the controller is deemed acceptable, and no further tuning is done.

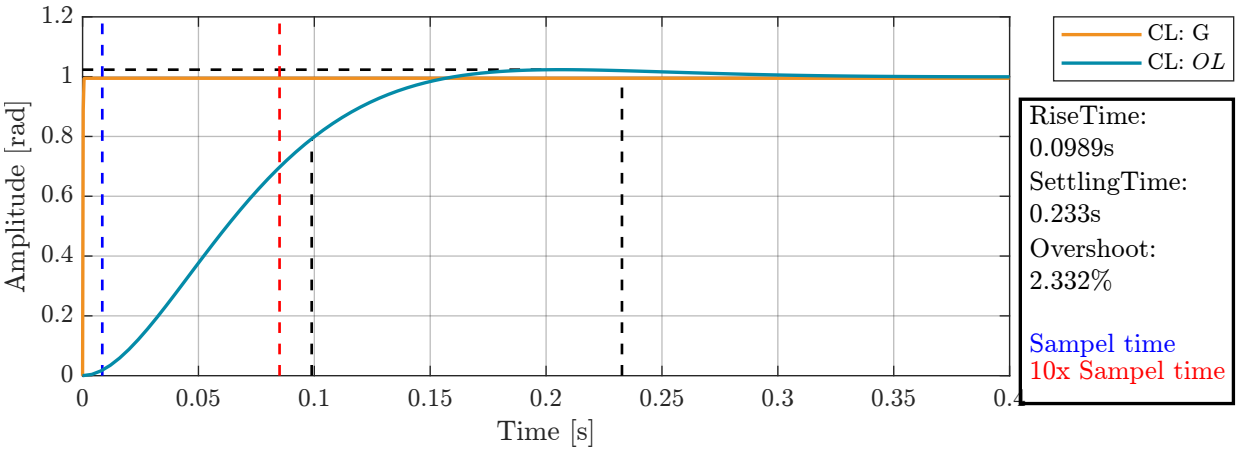


Figure 6.4. Step response of the system dynamics described by Equation (6.3)

The controller is applied to the BLDC model from Section 5.1 and tested in a Simulink simulation. Through the simulation, it is concluded that the controller is sufficient for BLDC control. This is further elaborated in Appendix G.

6.1.1 BLDC controller discretisation

The controller is discretised using the "c2d" Matlab function set to bilinear method with a sample time of $8.5ms$. The discretised controller is shown in Equation (6.4).

$$u_n = 0.00036125 \cdot e_n + 0.00036125 \cdot e_{n-1} + u_{n-1} \tag{6.4}$$

The controller is compared to the continuous-time variant in Simulink. Through testing of the controller, the discretised controller is shown to match the continuous time controller with minor deviations and is therefore implemented on the MCU. The controller comparison and discretised implementation are shown in Appendix G.

6.1.2 BLDC Controller test

A sinusoidal input is given as a reference for testing the MCU controller. Two different sinus inputs are given, one with only a positive velocity reference and one with positive and negative velocity references.

For the first test, a sinus wave with an amplitude of 500 $\frac{rad}{s}$ ($4.77e^3$ RPM) with an offset of 1250 $\frac{rad}{s}$ ($1.19e^4$ RPM) is given as the reference at frequencies 0.4 Hz, 1.2 Hz, 2.4 Hz. The results are shown in Figure 6.5.

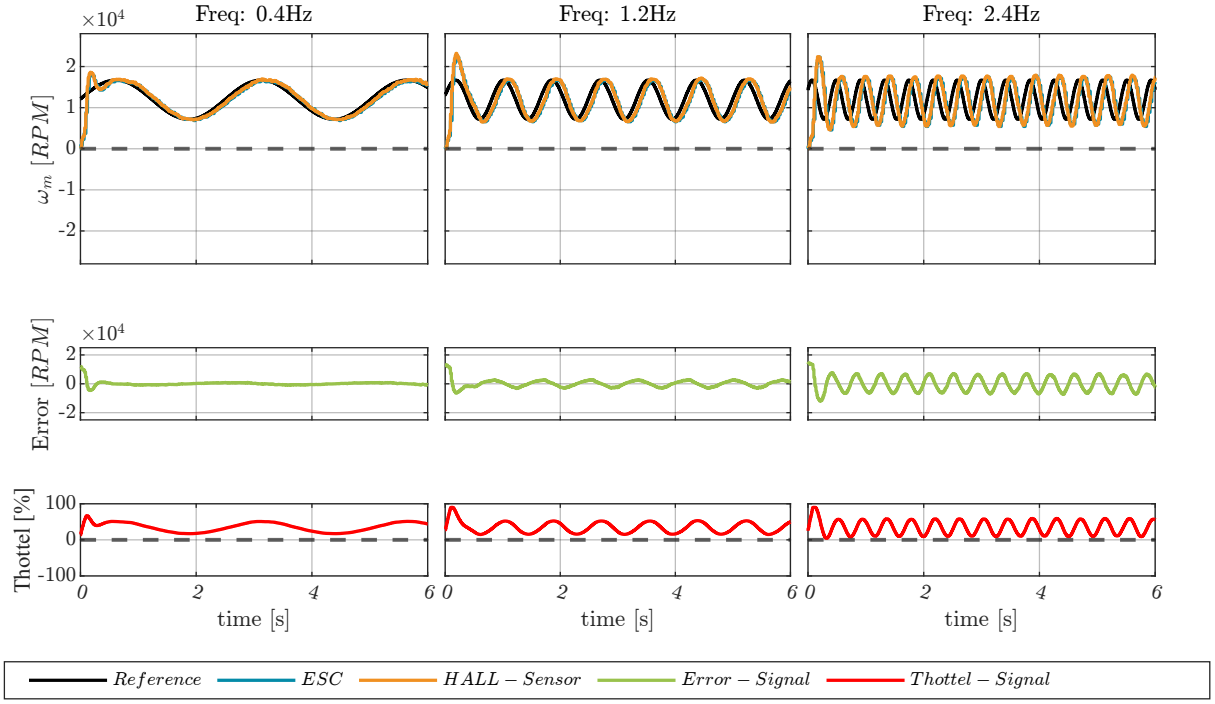


Figure 6.5. Controller test at frequencys: 0.4 Hz, 1.2 Hz, 2.4 Hz

For the 0.4 Hz test, the system follows, while only being 67 ms out of phase. The maximum amplitude of reference velocity is $1.671e^4$ RPM, where the maximum measured amplitude is $1.680e^4$ RPM measured $5-7$ samples later than the reference velocity. Increasing the frequency to 1.2 Hz increases the phase shift to 83 ms, while the maximum measured velocity is $1.690e^4$ RPM. At 2.4 Hz, the phase shift is increased by a small amount again to $89ms$ while the overshoot becomes more visible at an amplitude of $1.750e^4$ RPM.

The second test is a sinus wave with an amplitude of $2500\frac{rad}{s}$ ($2.39e^4$ RPM) and an offset of $0\frac{rad}{s}$. Because of the hardware limitation, when the motor needs to change rotation direction, the motor must be at zero velocity to do so. Changing the rotation direction takes ≈ 0.5 s, which greatly limits the frequency of the sine wave that the controller can follow. This test is then run at frequencies of 0.24 Hz, 0.3 Hz, 0.4 Hz and the results are shown in Figure 6.6.

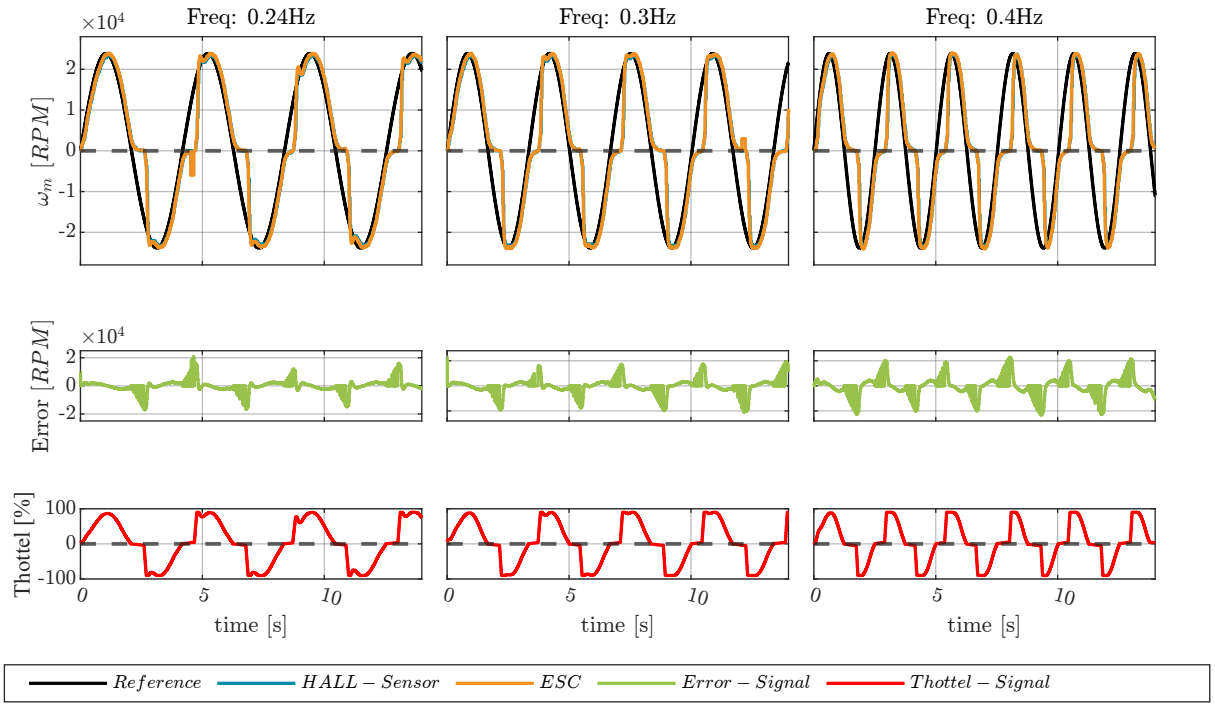


Figure 6.6. Controller test at frequencys: 0.24 Hz, 0.3 Hz, 0.4 Hz with a zero corssing event

The amplitude of the reference velocity is $2.387e^4$ RPM, where the measured amplitude for all the tests is $\approx 2.39e^4$ RPM. The phase shift for each test is, respectively, $[104, 126, 169]$ ms.

To conclude, the BLDC controller can follow frequencies without zero crossing as expected by the Bode plot shown in Figure 6.3. Due to the hardware's limits, changing the rotation direction on the BLDC requires the motor to be at zero velocity. To obtain zero crossing, the motor pauses for ≈ 0.5 s, meaning the motor will be at a standstill for frequencies above 2 Hz. The zero-crossing limitation is an issue when quick directional changes are required. But the frequency of the pendulum period shown in Table 5.5 is ≈ 0.36 Hz, given the motors plenty of time to change direction. For controlling Ψ, it is potentially a greater issue when minor corrections are to be made quickly. The impact is observed during testing in Chapter 8, and will be discussed further in Chapter 9.

6.2 Pendulum and yaw controller

A MIMO controller is desired to control the SPS pendulum and orientation. This is because the four thrusters specified in the design in Chapter 4 are the inputs for controlling α, γ, and Ψ as described in Section 5.3. This specifies a system with four motor inputs and three angle outputs, where a MIMO system is ideal for controlling this type of system.

The plant is presented in state-space form for tuning a MIMO controller. The state-space form is given by Equation (6.5), where x_p is the state vector consisting of the angle rates described by the models, u_p is the input vector consisting of the motor thrust, y is the output vector, A_p is the state transition matrix, B_p is the input matrix, and C_p is the output matrix.

$$\begin{aligned} \dot{x}_p &= A_p \cdot x_p + B_p \cdot u_p \\ y_p &= C_p \cdot x_p \end{aligned} \tag{6.5}$$

As shown by Equation (6.5), the state space representation is a matrix vector system, and the equations must be linear. The linear model for the mechanical pendulum and mechanical YAW model that was derived in Section 5.3.6 and is given by Equation (5.55) and (5.47). Representing the equations in state space form, where the states are given by $x_p = [\Delta\alpha, \Delta\dot{\alpha}, \Delta\gamma, \Delta\dot{\gamma}, \Psi, \dot{\Psi}]^T$. For attitude control, it is desired to control the positions, which is done by expanding A_p so that $\Delta\alpha$, $\Delta\gamma$, and $\Delta\Psi$ are included. Due to the linearisation and the chosen linearisation points given in Table 5.8 of the mechanical pendulum model, the linkages between u_p and $\Delta\dot{\gamma}$ are removed. Therefore, there is no direct way of applying an input to $\Delta\dot{\gamma}$, and row 4 in matrix B_p is therefore given as zeros. Rows 2 and 6 in B_p translate the thrust forces from u_p to rotational accelerations in $\Delta\ddot{\alpha}$ and $\ddot{\Psi}$. The state transition matrix A_p is given by Equation (6.6), the input matrix B_p is given by Equation (6.7), and the output matrix C_p is given by Equation (6.8). In c_p, the outputs for the control have been set to the positions $\Delta\alpha$ and Ψ.

$$A_p = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ \frac{\partial\ddot{\alpha}}{\partial\alpha}\Big|_{x_{p0}} & \frac{\partial\ddot{\alpha}}{\partial\dot{\alpha}}\Big|_{x_{p0}} & 0 & \frac{\partial\ddot{\alpha}}{\partial\dot{\gamma}}\Big|_{x_{p0}} & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ \frac{\partial\ddot{\gamma}}{\partial\alpha}\Big|_{x_{p0}} & \frac{\partial\ddot{\gamma}}{\partial\dot{\alpha}}\Big|_{x_{p0}} & 0 & \frac{\partial\ddot{\gamma}}{\partial\dot{\gamma}}\Big|_{x_{p0}} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & \frac{C_{v,\Psi}}{I_\Psi} \end{bmatrix} \tag{6.6}$$

$$B_p = \begin{bmatrix} 0 & 0 & 0 & 0 \\ \frac{1}{m \cdot L_w} & \frac{1}{m \cdot L_w} & \frac{1}{m \cdot L_w} & \frac{1}{m \cdot L_w} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{L_2}{I_\Psi} & \frac{L_1}{I_\Psi} & -\frac{L_1}{I_\Psi} & -\frac{L_2}{I_\Psi} \end{bmatrix} \tag{6.7}$$

$$C_p = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{6.8}$$

The state $\Delta\dot{\gamma}$ is removed because of the lack of input in B_p. This reduces the state-space representation to a 2D pendulum with a yaw model. The state vector is thereby reduced to $x_p = [\Delta\alpha, \Delta\dot{\alpha}, \Psi, \dot{\Psi}]^T$. Section 7.5 describes the implementation of the MISO system. Furthermore, as mentioned in Section 5.3.6, the linearisation of the thrust rotation matrix $R_z(\Psi)$ implies that all the thrust produced is used to generate a torque for M_α. This is only the case

when Ψ is rotated by 90° from γ. Limit switches are therefore to be defined, so that the MIMO controller will not dampen α when Ψ deviates too far from γ. The limit switches are described in Section 7.6.1. The reduced state transition matrix A_p is given by Equation (6.9), the reduced input matrix B_p is given by Equation (6.10), and the reduced output matrix is given by Equation (6.11).

$$A_p = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{\partial \ddot{\alpha}}{\partial \alpha}\big|_{x_{p0}} & \frac{\partial \ddot{\alpha}}{\partial \dot{\alpha}}\big|_{x_{p0}} & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & \frac{C_{v.Y}}{I_\Psi} \end{bmatrix} \quad (6.9) \qquad B_p = \begin{bmatrix} 0 & 0 & 0 & 0 \\ \frac{1}{m \cdot L^2} & \frac{1}{m \cdot L^2} & \frac{1}{m \cdot L^2} & \frac{1}{m \cdot L^2} \\ 0 & 0 & 0 & 0 \\ \frac{L_2}{I_\Psi} & \frac{L_1}{I_\Psi} & -\frac{L_1}{I_\Psi} & -\frac{L_2}{I_\Psi} \end{bmatrix} \quad (6.10)$$

$$C_p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad (6.11)$$

6.2.1 System analysis

The controllability and observability of the system are checked to determine the feasibility of designing a controller for this system. If the system is controllable, this means that a set of external inputs to the system, given by the control variables, can change the states of the system from any initial state to any final state over time. The controllability of a system is evaluated by checking if Equation (6.12) has full rank. Full rank is obtained when "rank(R)" matches the system's order.

$$R = \begin{bmatrix} B & AB & A^2B & ... & A^{n-1}B \end{bmatrix} \quad (6.12)$$

The controllability matrix of the system given by Equation (6.9) and (6.10) has a rank of 4, and the system order is also given by 4. This means that the system should be fully controllable.

The system is also checked for observability, which describes whether all the states can be described from the selected output vector. The observability is evaluated by checking if Equation (6.13) has full rank.

$$\mathcal{O} = \begin{bmatrix} C & CA & CA^2 & ... & CA^{n-1} \end{bmatrix}^T \quad (6.13)$$

The system's observability matrix also has a rank of 4, so it is fully observable.

Before deciding how to control the system, it is analysed. The system can be transformed into an open-loop transfer function matrix with Equation (6.14). $H(s)$ is a $2x4$ transfer function matrix.

$$H(s) = C_p \cdot (s \cdot I - A_p)^{-1} \cdot B_p \quad (6.14)$$

Singular Value Decomposition (SVD) can be done with the transfer function matrix, which will give a gain spectrum depending on the inputs. The maximum and minimum gains can be plotted

at different frequencies, resulting in a gain spectrum for $H(s)$, resembling the gain plot for a SISO controller. The SVD plot for $H(s)$ is shown in Figure 6.7.

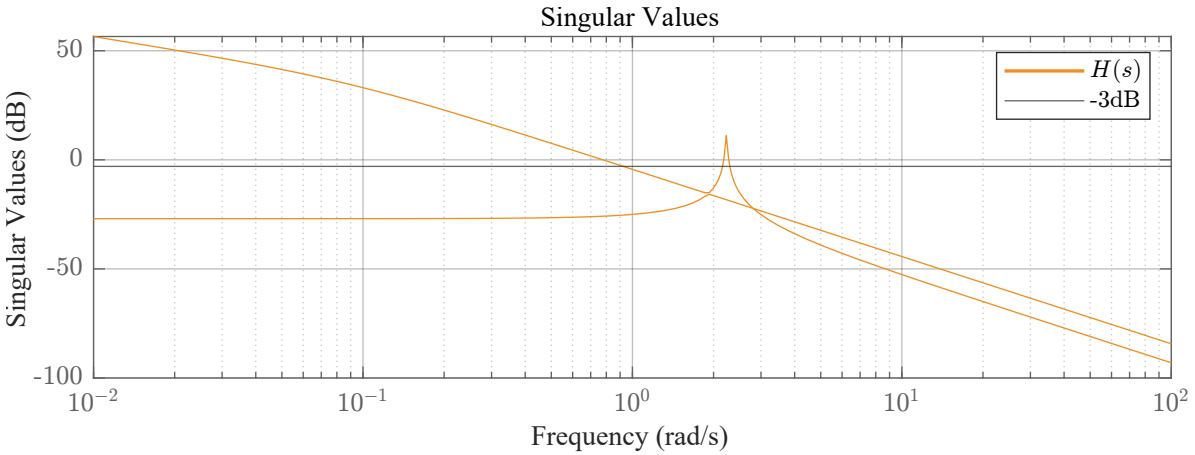


Figure 6.7. SVD plot of $H(s)$

By analysing Figure 6.7, it can be seen that at frequencies below 10^0 $\frac{rad}{s}$ the minimum gain is far below -3 dB, suggesting some input disturbances will not be suppressed appropriately. There is also a significant difference between the minimum and maximum gains below 10^0 $\frac{rad}{s}$, suggesting strong directionality, e.g., the inputs affect one output more than the other. The gain breaks off to suppress noise as wanted at frequencies above 10^0 $\frac{rad}{s}$. The difference between the minimum and maximum gain also becomes closer, suggesting balanced noise rejection. The system's bandwidth is estimated where the maximum gain crosses -3 dB at 0.925 $\frac{rad}{s}$. This is slightly more than ten times slower than the controller for the BLDC motor. Increasing the bandwidth of the results in a system where the SISO control cascade loop cannot be ignored, and its dynamics must be included during the tuning process of the MIMO controller.

On Figure 6.7 a resonant frequency at $2.23 \cdot 10^0$ $\frac{rad}{s}$ can be observed. This is expected to be the eigenfrequency of the pendulum. The eigenfrequency also matches the swing frequencies observed during the testing of ≈ 0.35 $Hz = 2.2$ $\frac{rad}{s}$ and is shown in Table 5.5 (S and I. 2005, pp. 71–81).

6.2.2 Linear Quadratic Integral augmentation

It is chosen to design a controller with Linear Quadratic Integral control (LQI), also known as Linear Quadratic Regulator (LQR), with integrator action for tuning the controller. The State space equations from Equation (6.5) and state vector x_p are augmented to contain an integral state x_I. The size of y gives the size of x_I. C_p specifies y to have a height of two, x_I therefore also has a height of two. The LQI augmented state space representation is given by Equation (6.15). r is the system reference vector. Also, the integral state vectors x_I components will be referred to as q_α and q_ψ when not written in vector form. The reference r_p consists of the controlled state references, which are α_{ref} and Ψ_{ref}.

$$\begin{aligned}\underbrace{\begin{bmatrix}\dot{x}_p\\ \dot{x}_I\end{bmatrix}}_{\dot{x}_{aug}} &= \underbrace{\begin{bmatrix}A_p & 0\\ -C_p & 0\end{bmatrix}}_{A_{aug}}\underbrace{\begin{bmatrix}x_p\\ x_I\end{bmatrix}}_{x_{aug}} + \underbrace{\begin{bmatrix}B_p\\ 0\end{bmatrix}}_{B_{aug}} u_p + \begin{bmatrix}0\\ I\end{bmatrix} r_p\\ y_{aug} &= \underbrace{\begin{bmatrix}C_p & 0\end{bmatrix}}_{C_{aug}}\underbrace{\begin{bmatrix}x_p\\ x_I\end{bmatrix}}_{x_{aug}}\end{aligned} \tag{6.15}$$

When tuning the controller with LQI, control gains are obtained, where the control law is given by Equation (6.16). K_x is the state gains, and K_I is the integral gains (A. 2010).

$$u_p = -\begin{bmatrix}K_x & K_I\end{bmatrix}\begin{bmatrix}x_p\\ x_I\end{bmatrix} \tag{6.16}$$

The controller gains K_x and K_I, which are computed through LQR and augmented to contain the integrator states, thereby forming an LQI system. LQR and LQI both take an offset in minimising the infinite horizon cost function seen in Equation (6.17). J is the cost-to-go and is wished to be minimal for the given Q and R penalty matrices.

$$J = \int_0^\infty x_p^T Q x_p + u_p^T R u_p \, dt \tag{6.17}$$

Through derivation of Equation (6.17), the algebraic Ricatti equation can be derived, as shown in Equation (6.18).

$$A_p^T P + P A_p - P B_p R^{-1} B_p^T P + Q = 0 \tag{6.18}$$

Equation (6.18) is solved for P, which is quadratic, resulting in the solution being non-trivial. But it is known that the solution has a single positive definite solution, which will provide the control gains by Equation (6.19) (Tedrake 2023). The control gains will provide the optimal input through Equation (6.16) that gives the minimal solution for Equation (6.17).

$$K = -R^{-1} B_p^T P \tag{6.19}$$

K is a gain matrix of the form $K = \begin{bmatrix}K_x & K_I\end{bmatrix}$ or as shown by Equation (6.20) (Triantafyllou 2003), where the Matlab "LQI" function is used to compute the controller gains based on the specified Q and R matrices.

$$K = \begin{bmatrix}K_x & K_I\end{bmatrix} = \begin{bmatrix}K_{x11} & K_{x12} & K_{x13} & K_{x14} & K_{I11} & K_{I12}\\ K_{x21} & K_{x22} & K_{x23} & K_{x24} & K_{I21} & K_{I22}\\ K_{x31} & K_{x32} & K_{x33} & K_{x34} & K_{I31} & K_{I32}\\ K_{x41} & K_{x42} & K_{x43} & K_{x44} & K_{I41} & K_{I42}\end{bmatrix} \tag{6.20}$$

A block diagram for the state space system where the LQI controller has been implemented can be seen in Figure 6.8.

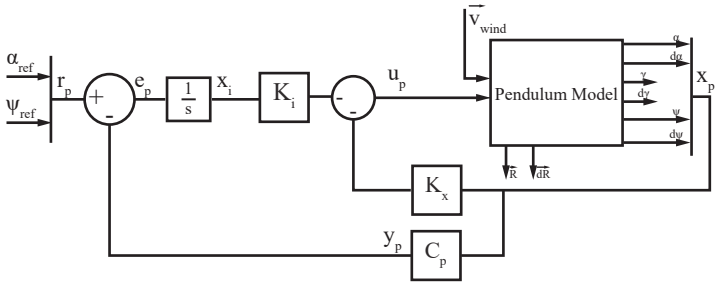


Figure 6.8. Block diagram of the state space system where the LQI controller is implemented

6.2.3 Anti-integral wind-up

The controller has two integrator states q_α and q_ψ in x_I. The integral states, accumulates the error and goes towards $\pm\infty$. The discrete implementation of integral calculations is shown in Equation (6.21). t_s is the sample time, r_x is the reference for state x, and α and Ψ are the angle measurements.

$$\begin{aligned} q_{\alpha.n+1} &= q_{\alpha.n} + (r_{\alpha.n} - \alpha_n) \cdot t_s \\ q_{\Psi.n+1} &= q_{\Psi.n} + (r_{\Psi.n} - \Psi_n) \cdot t_s \end{aligned} \tag{6.21}$$

The SPS system has a limit on how much thrust the motors can produce, and because it is a MIMO system, the limit does not necessarily correspond to the limit in thrust and torque specified in 4.3.1. Integral wind-up can occur if the controller asks for a thrust or torque that the SPS system cannot deliver. The error will build up in the integrator states because the system does not behave as the controller expects. The output can therefore get stuck, and the system does not react according to the given reference, but first when the integrator's buildup has been removed. Integrator wind-up is time-dependent, where the longer the error occurs, the larger the buildup.

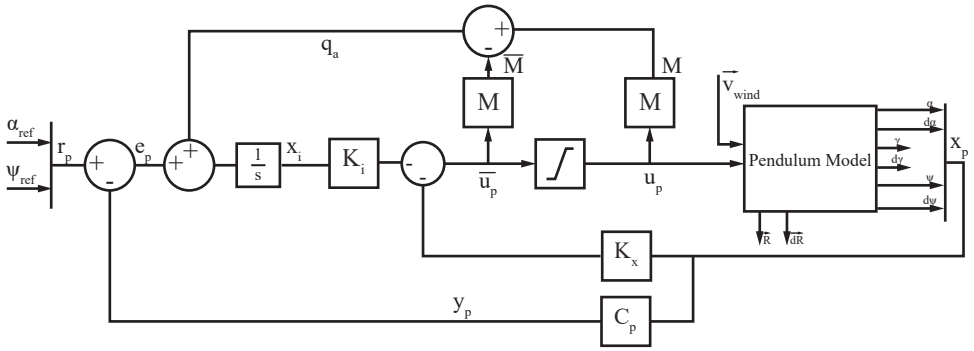


Figure 6.9. Block diagram of anti-integral wind-up implementation in SPS system

Integrator wind-up can be combated in multiple ways, one is to overdimension the system such that the limits of the system are never reached. The best solution is to tune the controller so that integral overflow does not occur, but this is not guaranteed. An anti-integral wind-up system can be implemented, which is shown in Figure 6.9. The saturation block limits the thrust output according to Table 6.1, which depends on the direction of the propeller. $\bar{u}$ is the unsaturated controller output, and u is the saturated output. Both $\bar{u}$ and u are passed to the M matrix,

which translates the thrust wanted from the motors to torques, M_α and M_Ψ. The M matrix is given by Equation (6.22).

$$M = \begin{bmatrix} L_w & L_w & L_w & L_w \\ l_2 & l_1 & l_1 & l_2 \end{bmatrix} \tag{6.22}$$

By translating the outputs of u to system torques, the number of variables is reduced from four to two, corresponding to the number of integrator states. The issue with the implementation of the anti-integral wind-up is when $\bar{u_p}$ exceeds u_p, because of the difference in thrust limits as shown in Table 6.1. $\bar{u_p}$ continues to rise evenly, where u_p is saturated in only some signals. This means that $\bar{u_p}$ rises faster than u_p, resulting in an anti-integral signal q_a because $\bar{u_p}$ becomes larger than the total available torque. q_a starts to suppress x_I and $\bar{u_p}$ stops increasing, effectively limiting the integral wind-up and u_p before using all available thrust.

Saturation limits	u0	u1	u2	u3
max	5.25	13.98	13.98	5.25
min	-13.98	-5.25	-5.25	-13.98

Table 6.1. Table over thrust saturation limits

To combat this, logic is implemented, such that the full available thrust is used. The logic only enables q_a when the total thrust saturation is reached. The pseudo code for the implementation can be seen below:

```
Mu = M * u;
Mubar = M * ubar;
if ((u0 - ubar0 < 0) || (u1 - ubar1 > 0) || (u2 - ubar2 > 0) || (u3 - ubar3 < 0)){
    qa = Mu - Mubar;
} else{
    qa = 0;
}
```

$q_a = [q_{a\alpha}\ q_{a\Psi}]^T$, and the anti-integrator wind-up can then be implemented in Equation (6.21) for the integral states, resulting in Equation (6.23).

$$\begin{aligned} q_{\alpha.n+1} &= q_{\alpha.n} + ((r_{\alpha.n} - \alpha_n) + q_{a\alpha.n}) \cdot t_s \\ q_{\Psi.n+1} &= q_{\Psi.n} + ((r_{\Psi.n} - \Psi_n) + q_{a\Psi.n}) \cdot t_s \end{aligned} \tag{6.23}$$

According to simulations, when no payload is included, the SPS system has enough thrust and will never reach saturation under normal circumstances. Two tests will be conducted to test the anti-integral wind-up implementation. The first test aims to show the anti-integrator working on the α state, where three different references with increasing values are given. The thrust saturations are reduced to $\pm 2N$ and $\pm 4N$ for this test. The second test aims to show the anti-integrator working on the Ψ state, where three different references are given. The thrust saturations are reduced to $\pm 0.015N$ and $\pm 0.008N$ for this tests. A controller tuned in Section 6.2.4 is used for the tests, and the result of the system with and without anti-integral wind-up can be seen in Figure 6.10.

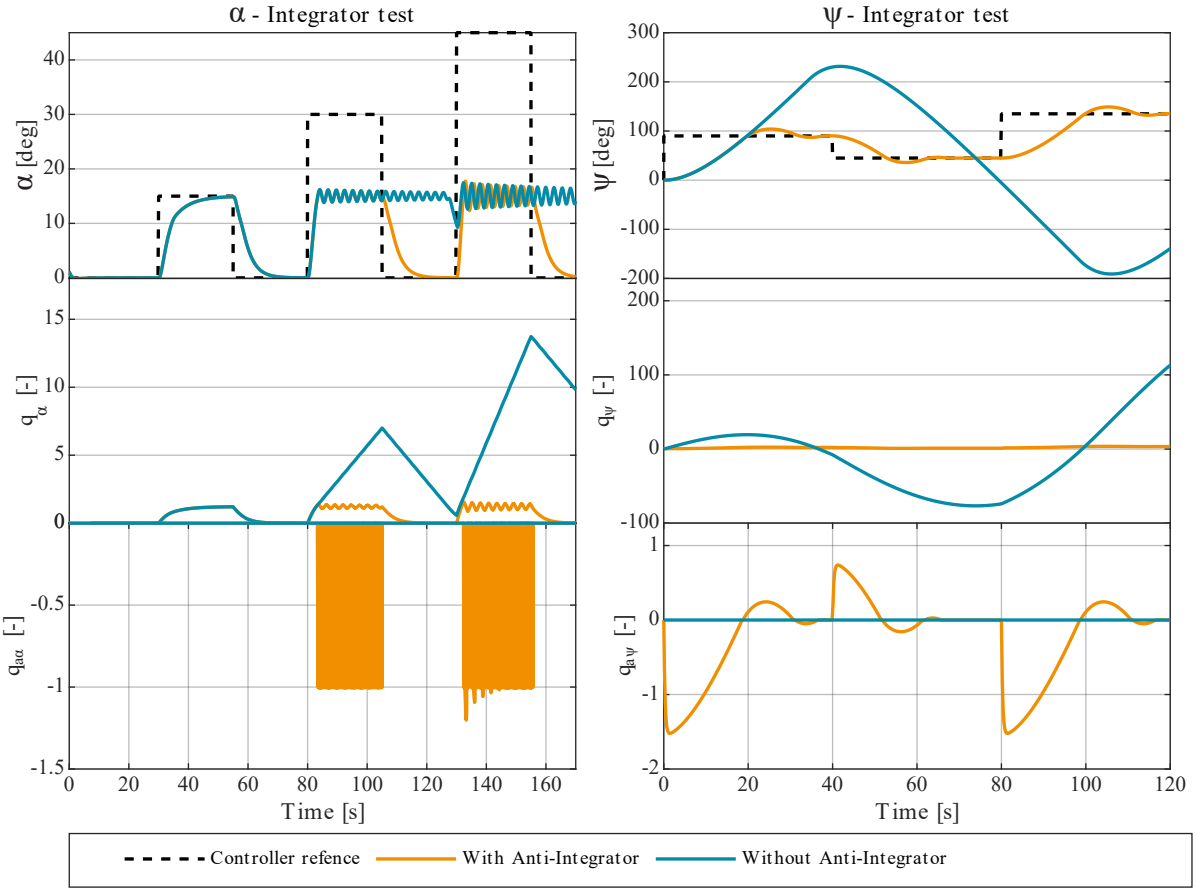


Figure 6.10. α and Ψ tests with and without anti-integrator windup.

For the α test, the SPS system has enough thrust to reach the first reference, and no integrator wind-up occurs. The system does not have enough thrust for the second and third reference steps. The integrator q_α continues to rise during the reference, and when the reference is changed to zero, the integrator starts to wind down, first allowing the system to reduce thrust right before the third step and therefore following the reference poorly. When the anti-integrator wind-up is implemented, the system reacts as soon as the reference is changed, resulting in better reference tracking.

In the Ψ test, the system believes it is much faster than it is, and the integrator builds too quickly. When the system without anti-integrator reaches the reference, the buildup value in the integrator starts to decrease, but the system continues to increase Ψ. This effect amplifies itself when the reference changes and cannot be reached. When the anti-integrator wind-up is implemented, there is only a small overshoot, and the system can track the reference according to the tuning.

6.2.4 LQI tuning

To tune the controller with LQI, the values of the penalty matrices are adjusted. The Q matrix, known as the state cost matrix, specifies a weighted penalty applied to state errors. The Q matrix is a diagonal matrix specifying a weighted penalty for each state. The higher the penalty, the

heavier the controller will be tuned to eliminate errors in that specific state. The integrator states x_I also has a penalty, which is adjusted to how much the controller responds to the integrator.

$$Q = \begin{bmatrix} Q_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & Q_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & Q_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & Q_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & Q_5 & 0 \\ 0 & 0 & 0 & 0 & 0 & Q_6 \end{bmatrix} \qquad R = \begin{bmatrix} R_1 & 0 & 0 & 0 \\ 0 & R_2 & 0 & 0 \\ 0 & 0 & R_3 & 0 \\ 0 & 0 & 0 & R_4 \end{bmatrix} \tag{6.24}$$

The R matrix, known as the input cost matrix, specifies a weighted penalty for each input u. R is also a diagonal matrix, specifying a penalty for each input. The R matrix adjusts how much the system's actuators u are used. It can limit how much or how little the motors are used. Q and R are given by Equation (6.24).

To tune a controller for the SPS system, appropriate values must be determined for the Q and R matrices. Initial values for Q and R that resulted in a stable controller in the simulation were used as a starting point. The simulation included the motor control and dynamics to ensure the system behaves as expected. The controller was then implemented on the test setup, where the response was measured. Through an iterative process, the values for Q and R were adjusted individually to obtain a desired response.

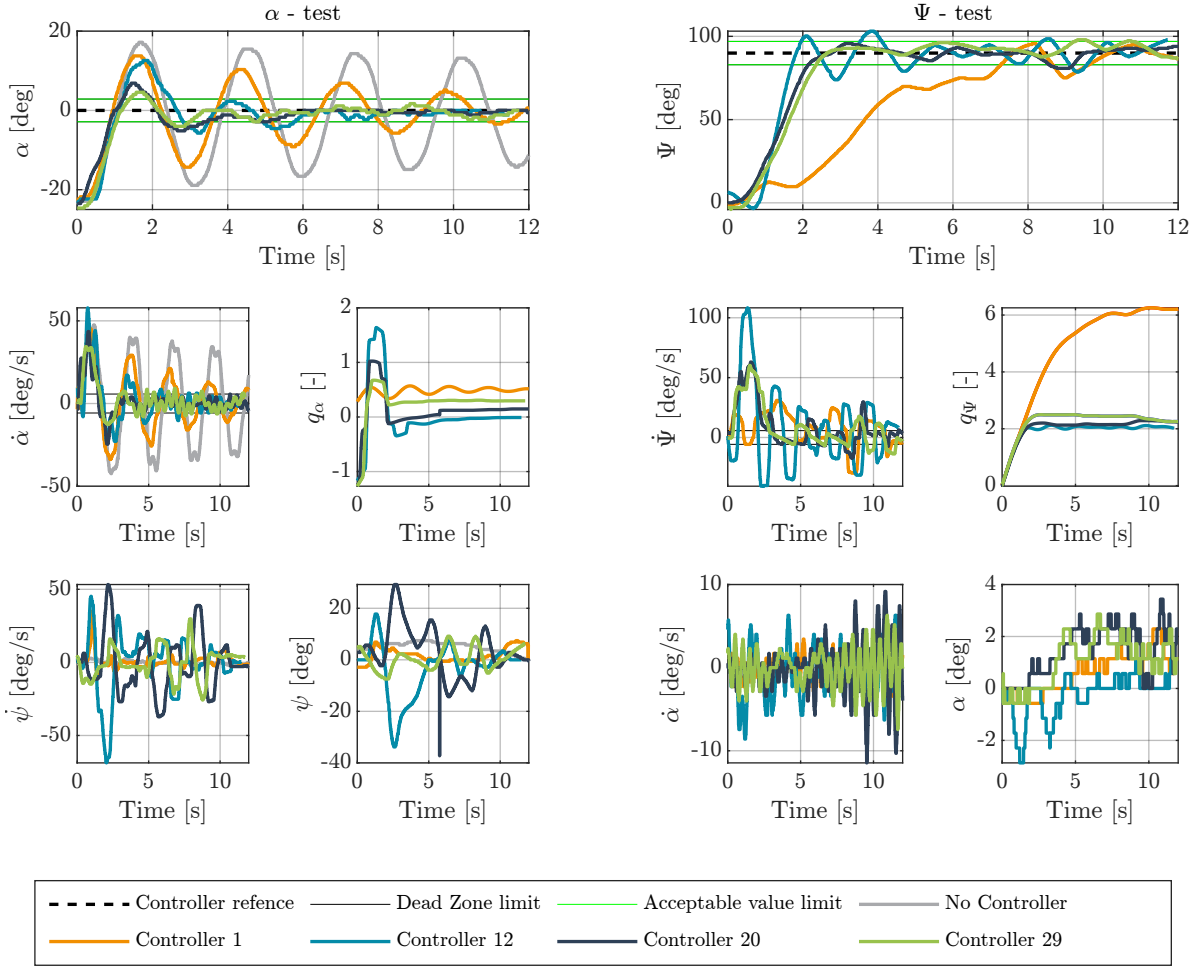


Figure 6.11. . Graphs of System response to $\alpha-test$ & $\Psi-test$ for different controllers from the iterative progress. (Video of test of Controller 29: `https://youtu.be/B_KVQvnpddo?si=WtJZa5tiqaADOzUF`)

Each controller was tested in simulation and on the test setup. The simulation result was used to ensure the changes in Q and R result in a desired change before being implemented on the test setup. Because the simulation does not account for the motor stop at zero crossing for $\approx 0.5\ s$, so the test setup results were used to evaluate the controllers. A deadzone is implemented on α and Ψ as it was observed that the system could not obtain a stable position at the reference due to measurement noise and the motor zero crossing problem. The dead zones are described in Section 7.6.1. For evaluating the controllers, it is assumed that the system has obtained the reference value once it enters the dead zone.

The obtained controller is tested in two separate tests, one dampening α from an initial angle of $\approx 23°$ to $0°$, and one for rotating in Ψ from $0°$ to $90°$. The system response with no controller, versus controller 1, 12, 20, and the final controller, controller 29, are shown in Figure 6.11.

The controller is evaluated on the time it takes from the starting conditions at $t = 0$ until the system is within the acceptable value range. The target time for the controller is the total time for α dampening plus the Ψ $90°$ turn to be equal to or less than the "time until stability" from the engineering specifications described in Appendix B.2. The system's overshoot and stability within the acceptable range are also evaluated. The measured times are specified in Table 6.2

	No CON	CON 1	CON 12	CON 20	CON 29
α	101 s	14 s	6s	5 s	3.5 s
Ψ	NAN	7.6 s	5.1 s	2.6 s	2.7 s
Total	NAN	21.6 s	11.1 s	7.6 s	6.2 s

Table 6.2. Controller test results

Controller 29 is deemed sufficient, and the Q and R matrix penalty values used for calculating the controller gains are given in Table 6.3. High penalties have been applied for both the α and $\dot{\alpha}$ states of 35. This was done for a faster response to the error in the α state. q_α is set to 5, which makes the controller ask for more thrust when an error is present. The penalties for Ψ, $\dot{\Psi}$, and q_Ψ are lower as it keeps the yaw rotation slow enough to find a stable position in the dead zone, but also such when the motors have changed directions, the step input after zero-crossing is not so high that the current limits of the ESC is triggered. The low penalties are still high enough for fast reactions, as seen in Figure 6.11.

All inputs are given the same penalties in the R matrix of 0.03125 as it kept the requested thrust in the $\approx \pm 10\ N$ range when reacting to larger errors in the α state. The gain matrix K values calculated by Equation (6.20) are then given by Table 6.4.

penalty parameters	value	discription
Q_1	35	penalty for α
Q_2	35	penalty for $\dot{\alpha}$
Q_3	0.015	penalty for Ψ
Q_4	0.055	penalty for Ψ
Q_5	5	penalty for q_α
Q_6	0.75	penalty for q_Ψ
R_1	0.03125	penalty for using u_0
R_2	0.03125	penalty for using u_1
R_3	0.03125	penalty for using u_2
R_4	0.03125	penalty for using u_3

Table 6.3. Controller penalty values for solving the Riccati equation

Calculating the singular values of the MIMO system allowed for the estimation of its disturbance, noise rejection, and bandwidth. The system has now been modified by LQI tuning, where the gain matrix K changes the system response. K contains K_x and K_I where K_x reacts on all the states in x_p and K_I reacts on $x_p \cdot -C_p$ as shown by Equation (6.15). Therefore, the gain matrix K requires feedback to apply its control, where the SVD plot plots the open-loop response. Assuming K applies on the open-loop system, the loop transfer function L_{LQI} is obtained from the augmented LQI state space and shown in Equation (6.25) (S and I. 2005, pp. 67–70).

$$L_{LQI}(s) = \left(C_{aug} \cdot (s \cdot I - A_{aug})^{-1} \cdot B_{aug}\right) \cdot K \tag{6.25}$$

The SVD plot of $L_{LQI}(s)$ is compared to $H(s)$ in Figure 6.12, where it can be seen that the bandwidth of the system has been increased with the use of LQI to be between 2.9 $\frac{rad}{s}$ to 4.5 $\frac{rad}{s}$. The gain margin at lower frequencies has been increased, so the minimum gain is above $-3\ dB$, and low-frequency disturbances can be suppressed. The resonance peak at 2.23 $\frac{rad}{s}$ has not been moved, implying the pendulum will swing with the same frequency but will be slowed down faster.

Controller gain	value
K_{x11}	14.1181
K_{x21}	14.1181
K_{x31}	14.1181
K_{x41}	14.1181
K_{x12}	18.4659
K_{x22}	18.4659
K_{x32}	18.4659
K_{x42}	18.4659
K_{x13}	4.2447
K_{x23}	2.5848
K_{x33}	-2.5848
K_{x43}	-4.2447
K_{x14}	2.9256
K_{x24}	1.7815
K_{x34}	-1.7815
K_{x44}	-2.9256
K_{I11}	-6.3246
K_{I21}	-6.3246
K_{I31}	-6.3246
K_{I41}	-6.3246
K_{I12}	-2.9587
K_{I22}	-1.8017
K_{I32}	1.8017
K_{I42}	2.9587

Table 6.4. Controller gain values from solving the Riccati equation

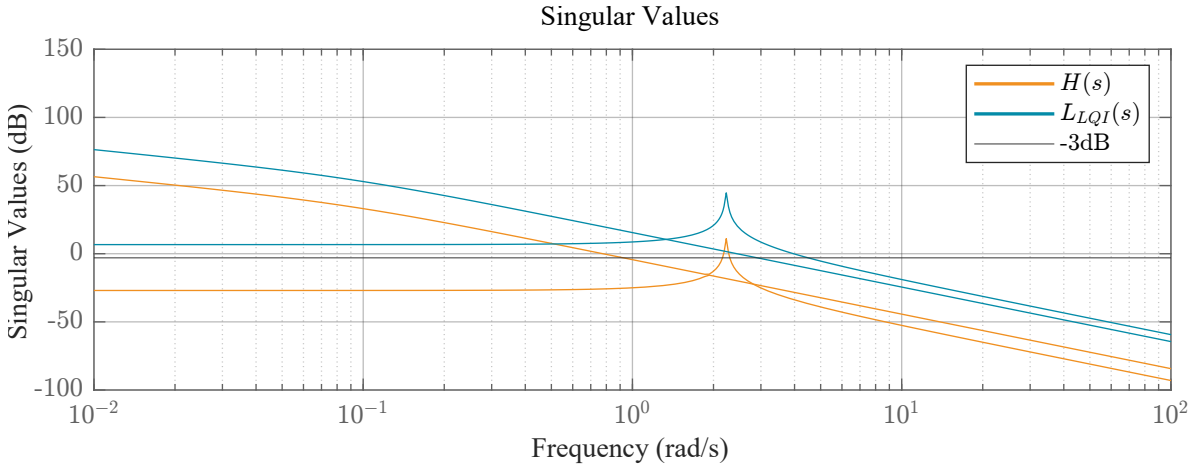


Figure 6.12. SVD plot with controller inputs

Implementation 7

This chapter will introduce the hardware used to design the SPS system. It will also describe how this hardware has been wired together and the communication protocols used. The chapter will also describe the two programs used to control the SPS. This includes describing how the raw sensor data is used to find the SPS's attitude. The chapter will also discuss how the MIMO system has been implemented on the microcontroller unit (MCU).

7.1 Hardware

From Chapter 4, it was determined that the SPS should consist of four BLDC motors to control the system. The motors need an Electronic Speed Controller (ESC) to apply the correct voltages, and an MCU will provide the signals. An Arduino determines attitude using sensor fusion between a gyroscope, accelerometer, and magnetometer. A battery will provide the required power for the setup, where a voltage regulator will reduce the voltage from the battery to an acceptable range for the MCU and Arduino. All the Hardware used on the SPS can be seen in Table 7.1.

Component	**Description**	**Amount**
F40PRO IV KV2400	BLDC motor	4
Mini F60A 3-8S 4in1 ESC	ESC for BLDC	1
Nucleo STM32f446RE	MCU used for control	1
Flipsky Anti Spark Switch 200A	Anti Spark Switch to turn on and off power to the system	1
Gens ace G-Tech Hardcase 6750mAh 14.8V 60C 4S1P Lipo Battery	Battery to power the motors and microcontrollers	1
Arduino Nano 33 IoT	Microcontroller with accelerometer and gyroscope.	1
LSM6DS3	Gyroscope and accelerometer chip located on the Arduino Nano 33 IoT	1
HMC5883L	Magnetometer to determine heading	1
L7805CV	5V Voltage regulator for lowering the 14.8-16.2V to 5V for power delivery to microcontroller	1
SS460P	Hall sensor latch to verify the motor velocity from the ESC	1

Table 7.1. Hardware used for the SPS system

7.2 Wireing & digital comunication

Figure 7.1 shows a wiring schematic of the SPS system. The Nucleo board is the main MCU that computes the control signals sent to the ESC for motor control, from the attitude estimations given by the Arduino Nano. The Nucleo boards communicate with all the components using different digital communication protocols.

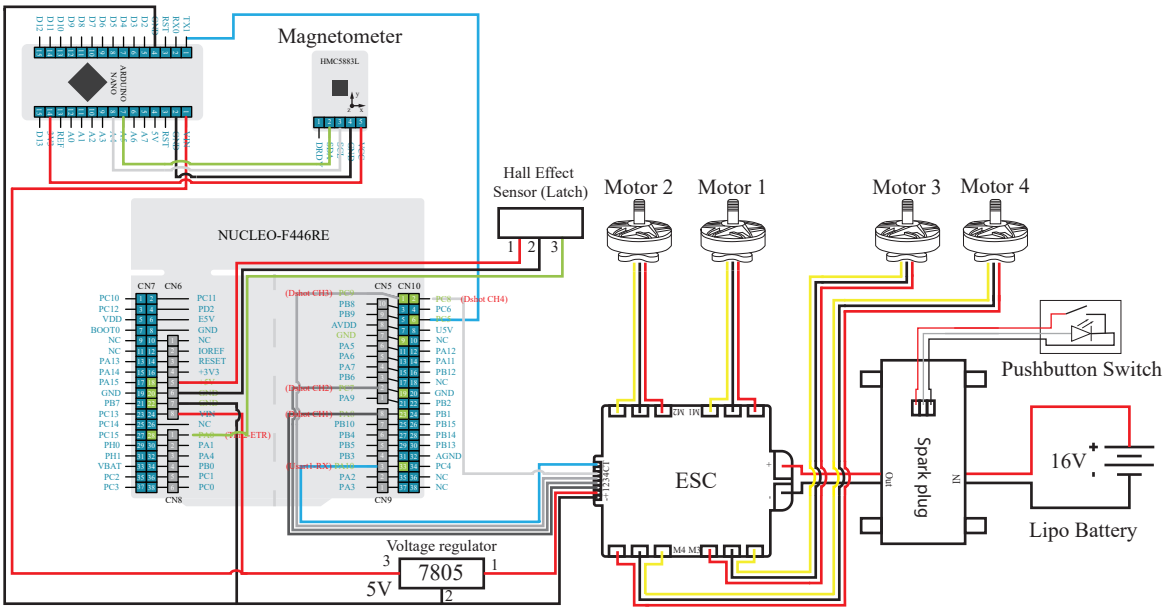


Figure 7.1. Wiring schematic of how the electronic components are connected in the SPS system

Nucleo to ESC

Communication from the Nucleo to the ESC is done using a protocol called Dshot, which is often used in FPV racing drones. DShot is a digital version of Pulse-Width-Modulation (PWM), which offers greater control and is less susceptible to noise. DShot is supported in 150, 300, 600, and 1200 kHz per bit, where faster frequencies offer faster motor speed updates at the cost of more noise signals. For this report, DShot-300 offers adequate speed and is chosen. The DShot protocol expects a 16-bit signal where the first 11 bits from the left send a number between 0 and 2048. The next bit requests telemetry for the specific motor on the telemetry line. The last 4 bits are used as a checksum, ensuring more accurate data transfers. An arbitrary DShot signal can be seen in Figure 7.2. DShot values between 0 and 47 are commands where more information can be found in Appendix H. Values between 48 and 2048 correspond to throttle percentages between 0 and 100% (Landa 2021).

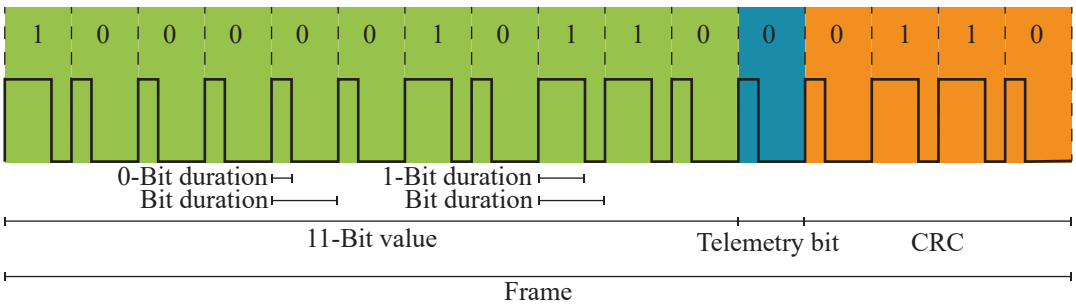


Figure 7.2. DShot frame of 11-bit value of 1046 where telemetry is set to false

ESC to Nucleo

The telemetry data from the ESC is sent to the Nucleo board via asynchronous serial communication. A single wire connects the ESC from the TX pin to the Nucleo RX pin. The data is sent in packages consisting of frames of 8 bits, where a start and stop bit are used to mark the start and end of a frame, a sketch of this is shown in Figure 7.3. The communication works by sending a high signal for a high bit and a low signal for a low bit (Siebeneicher 2024). The used ESC applies a telemetry protocol called KISS, where one transmission will have 10 frames of 8-bit bytes sent with a baudrate of 115200. The frames contain one frame for temperature, two for battery voltage, two for current, two for power consumption, two for eRPM, and one for CRC (KISS 2025). The "F60Amini 4in1" only supports temperature, battery voltage, eRPM and CRC. So only these bytes will contain data.

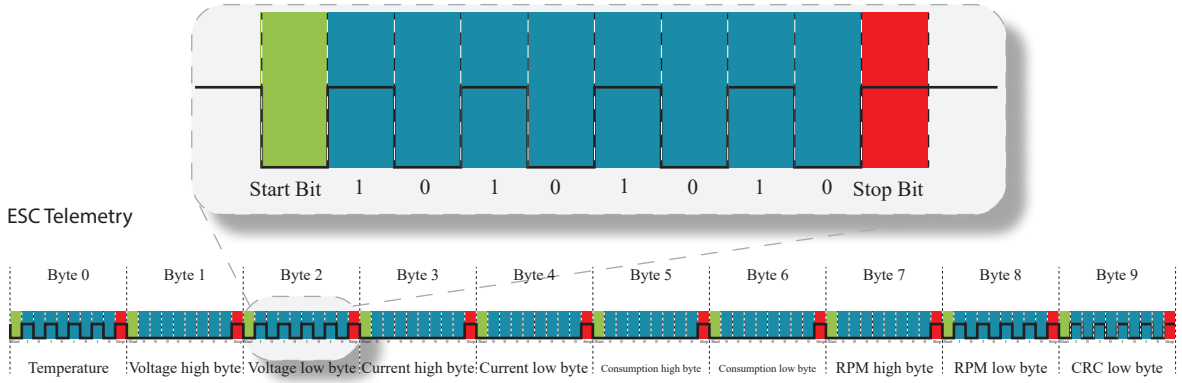


Figure 7.3. The serial communication from the ESC

Arduino to Nucleo

The Arduino also sends the attitude data via serial communication. A custom attitude package has been implemented, consisting of 14 8-bit frames. The first six frames are the roll, pitch, and yaw attitude data, where the subsequent six frames are the roll, pitch, and yaw rotational rates. The following two frames are unused command bytes, where the last two frames are a checksum, which is the sum of the other bytes. The data is sent at a baud rate of 115200.

Sensors to Arduino

The sensors are connected to the Arduino via an Inter-Integrated Circuit (I^2C), where both sensors are slaves, and connected to the same I^2C bus. I^2C is a two-wire communication protocol where multiple masters and slaves can be connected to the same bus. The SDA wire (serial data) transfers data between units on the bus, where the SCL wire (serial clock) synchronises the data flow. A sketch of how multiple slaves are connected to a master is shown in Figure 7.4.

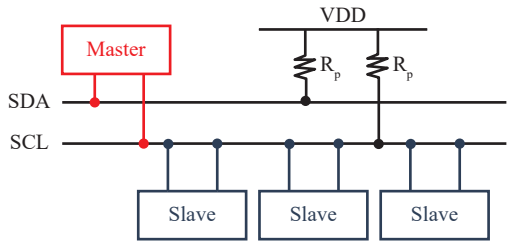


Figure 7.4. The master device connected to slave peripherals

The SDA and SCL wires are pulled up with resistors, where a low result in a 1 bit, and leaving it high leaves a 0 bit. Data can be placed on SDA when SCL is low and sampled when SCL is high. The I^2C communication starts by pulling SDA low, which puts all peripherals on notice. The first frame consists of a 7 bit address specified for the requested peripheral, followed by a read (1)/write (0) bit. The receiving peripheral acknowledges (ACK) or does not (NACK) the request. Acknowledgement has to be sent before the ninth clock pulse, or it is inferred that the data was not received. After the address frame has been acknowledged, the 8-bit data frames are sent, followed by an ACK. The number of data frames is specified in the code for the specific peripheral, but can be an arbitrary number (Sparkfun 2025) (STMicroelectronics 2024).

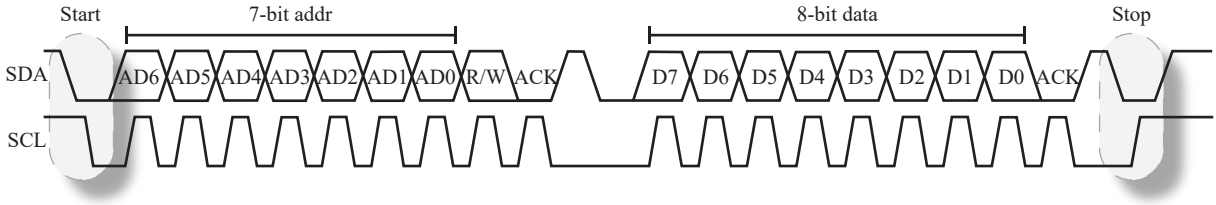


Figure 7.5. I^2C frame for SDA and SCL

Power line

The battery is a $4S$, which means four lipo cells of 4.2 V in series and provides an ideal voltage of 14.8 V but can be charged safely up to 16.6 V. The battery is connected to the ESC through a spark plug, where power can be switched on and off mechanically. The ESC provides power to the BLDCs according to the communication described in Appendix Section D.1. The ESC outputs a voltage equal to the input voltage over the + and − pins. The Necleo board and Arduino are connected to a voltage regulator, which lowers the ESC output voltage to 5 V.

The implemented code consists of C code on the Nucleo board and C++ code on the Arduino. All the code can be seen in Appendix K. The Nucleo board is the main control board, where the MIMO and BLDC control signals are calculated and sent to the ESC. It received motor telemetry from the ESC and attitude data from the Arduino. The Arduino reads the data from the gyroscope, accelerometer, and magnetometer, then calculates the attitude and heading. The data is then sent to the Nucleo board. The specific code for the Nucleo board and Arduino will be described in more detail in the following. Information about the Nucleo board, which is used in the programming and wiring process, can be found in the Nucleo STM32f446RE datasheets (ST.com 2020) (ST.com 2021). The Arduino Nano 33 IoT datasheet has also been used in the programming and wiring process (Arduino.com 2022). A dashboard has been developed using "STM32CubeMonitor", where the system states can be altered. The monitor also logs data, where a live view of the motor velocity and the attitude with corresponding references are plotted. The dashboard is shown in Appendix K.

7.3 Program

7.3.1 Nucleo board

The Nucleo board code is timer interrupt-based, except for the initialisation. An interrupt-based code structure is chosen because different chunks of the code must be executed at a specific frequency. These frequencies do not match, and multiple timers are therefore necessary for optimal data use. The code is not executed in a particular order as the interrupts are synchronised, but data flows between the different timer interrupts, where the newest value is used. The timer frequencies are shown in Table 7.2, where the data flow and necessary control loop frequencies give the chosen frequencies. Timers 1 and 8 are used for DShot communication, timer 4 is used for the BLDC motor control loop, timer 5 updates the DShot package and which motors' telemetry is requested from, and timer 6 is the MIMO control loop.

Timer	Prescaler	Autoreload	Systemclock	Theoretical frequency	Measured frequency
TIM1	7	20	APB2 PER	267 kHz (3.733 us)	-
TIM4	141	5387	APB1 TIM	117.63 Hz (8.5 ms)	119.47 Hz (8.37 ms)
TIM5	17	5385	APB1 TIM	928.33 Hz (1.07ms)	943.4 Hz (1.06 ms)
TIM6	15	56249	APB1 TIM	100 Hz (9.99ms)	103.73 Hz (9.64 ms)
TIM8	7	20	APB2 PER	267 kHz (3.733 us)	-
UART1	-	-	-	-	471.7 Hz (2.12 ms)
UART3	-	-	-	-	98.03 HZ (10.2ms)

Table 7.2. PSC and ARR values with corresponding system clocks and frequencies

The Nucleo board runs at its maximum system clock frequency of 180 MHz. The timers and peripherals do not run directly from this timer, but rather from the advanced peripheral bus (APB). Two APBs exist on the Nucleo board. Timer 1 and 8 and their associated peripherals are on APB2, and the rest of the timers and their peripherals are on APB1 (ST.com 2021). The APBs can run the timers and peripherals at different frequencies. APB1 runs the timers at 90 MHz and the peripherals at 45 MHz. In APB2, the timers run at 180 MHz and the peripherals at 90 MHz. Each timer can have a further specified frequency through prescaler (PSC) and autoreload (ARR). The PSC can be viewed as a scale for how slowly the clock is run, while ARR is the resolution of the timer or peripheral. The PSC and ARR values must be chosen for a timer to run at a desired frequency, which can be done with Equation (7.1) (Magdy 2025).

$$T_{out} = \frac{(ARR + 1) \cdot (PSC + 1)}{F_{CLK}} \tag{7.1}$$

All the values for PSC and ARR used for the timers and peripherals are shown in Table 7.2, with the corresponding frequencies. The measured frequencies of the UART interrupts are also shown.

Program structure

The C code consists of a main loop, three timer interrupts, from timers 4, 5, and 6. the over all structure of these can be seen in Figure 7.6 Additionally, there are two UART interrupts when data is received from either the ESC or Arduino and the structure of these can be seen in Figure

7.7. Initially, the program goes through the main loop, which consists of an initialising phase where the variables, timers, UART, peripherals and DMA are initialised before the main loop is entered. The main loop is empty because all the code exists in the interrupts, and is run until it is aborted.

In the interrupt on timer 6, the MIMO control is calculated, using the latest available data from the Arduino received by UART3. The frequency of the EKF calculated on the Arduino activates UART3, which is at $\approx 100\ Hz$, therefore, timer 6 is specified to run at a frequency of $\approx 100\ Hz$ to match. With this data, the MIMO control is calculated, and the output thrusts are determined and made available to timer 5.

Timer 4 takes the motor velocity reference from timer 5, with this and the latest telemetri data from UART1, the velocity error is calculated and is passed through the time BLDC motor control described in Section 6.1. The motor throttle percentage is calculated from the motor controller output by Equation (7.2) and made available for timer 5. Timer 4 runs at a frequency of $\approx 120\ Hz$ since UART1 runs at $\approx 470\ Hz$. UART1 must run four times to update all four motor velocities before timer 4 is triggered. This results in new data for all motors, which then can be used in timer 4 to update the motor controllers.

$$throttle = \frac{u_n}{16.2} \cdot 100 \tag{7.2}$$

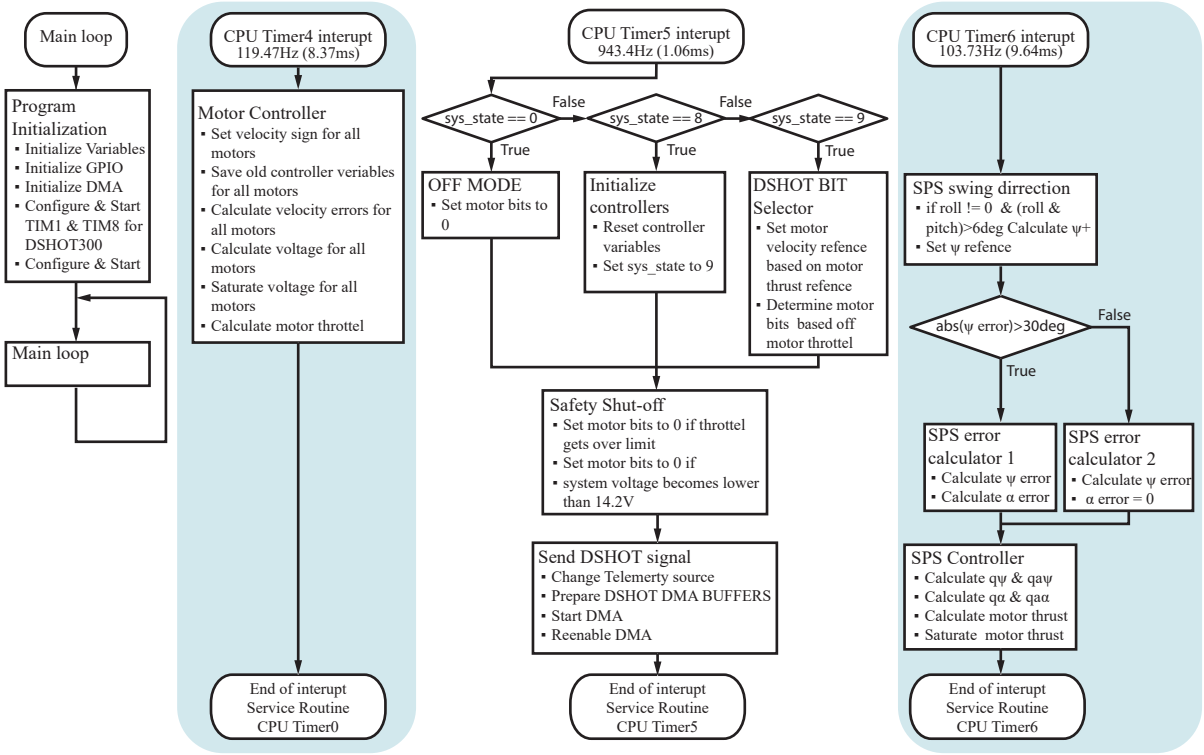


Figure 7.6. Program flowchart Part 1

Timer 5 is the main body of the program and is where the SPS system mode is determined based on sys_state“. sys_state” is set with the use of the dashbord. When the “sys_state” is 9, the SPS system is active, and the latest thrust data from timer 6 is read and passed through thrust-velocity look-up tables from Section 5.2 and the motor velocity reference is made available for

timer 4. After this, the throttle values from Timer 4 are read, which must be translated to the byte value needed for the DShot package. This is done with the function "Motor_bit_selector", which runs for each motor. The function determines the 11-bit DShot value between 48 and 2048 from the motor throttle percentage. The function also determines the motor spin direction and changes it accordingly.

The 11-bit Dshot value is calculated from the throttle percentage with the linear equation $byte(x) = 48 + x \cdot \frac{2000}{100}$. The 2000 is the bit value range for the throttle and the 100 is value the throttle percentage is spread over, which for a percentage is 100. The motor spin direction is changed using commands between 1 and 48. If the throttle and motor spin direction are opposite, the motor spin direction must be changed. This is done by sending a DShot command to change the motor's rotation direction. This is done by setting the 11-bit Dshot value to 7 and sending it eight times in a row for a positive spin direction. And for a negative spin direction, the 11-bit Dshot value is set to 8. For the ESC to accept the commands, the velocity of the motor must be 0 RPM. To change the spin direction, the motor is stopped, and the direction is commanded. This turns the motor off for ≈ 0.5 s. After the command has been issued, it is registered in the spin direction tracking array.

Timer 5 then goes through a safety shut-off that sets the "sys_state" to 0 to stop the SPS system if the thrust goes out of range or the voltage on the battery becomes too low. Because the ESC only allows reading telemetry for one motor at a time, the telemetry source is updated for the next motor in line (0-1-2-3). The four 11-bit motor values and the telemetry request are placed in two arrays with an entrance for each motor. These arrays are then passed to the "UPDATE_DSHOT" function. This function creates four DShot packets from the DShot bit values and telemetry arrays, then sends the DShot packets on Timer 1 and 8's PWM channels.

PWM was used to create the DShot signals on timer 1 and timer 8. DShot300 has a bit rate of $f_{DShot} = 300$ kHz (3.33 μs). The PWM clock is set to run at a frequency of $20 \cdot f_{DShot}$ with an AAR of 20. This is done to have 20 PWM steps cycled through for each bit in the frame. To determine if a bit is 0 or 1, the signal is set high for a percentage of the bit duration. To send a high bit, the PWM signal is too high for $\approx 70\%$. And to send a low bit, the signal is on high for $\approx 35\%$ of the duration time of the bit. This can be used to create the signal. This means the PWM can have its counter compare value (CCR) set to 7 to send a low bit or 14 to send a high bit, since this would result in the PWM being on for 35% when CCR is 7 and 70% when CCR is 14.

To generate a signal with PWM, the CCR value has to be changed very quickly between bits to send the correct signal. Therefore, Direct Memory Access (DMA) is used. Since DMA allows the MCU to place a buffer in memory that a peripheral can read directly from without using the CPU when the DMA is set to memory-to-peripheral mode. This results in values going to the peripheral that can be updated quickly. In this case, a DMA buffer of CCR values is created to change the CCR value for every bit, thereby generating the correct signal. To generate the 16-bit signal used in a Dshot signal, a DMA buffer of 16 is necessary, though a DMA buffer of 18 was used to create spacing between frames. When a DMA buffer has been read, a callback turns off the DMA. This means that every time a buffer has been made, the DMA needs to be

activated again. This also ensures that only the specified number of frames is sent(Embedded 2025).

The DMA buffer is created by first making a packet by taking the 11-bit value, adding a telemetry bit, and the calculated checksum, resulting in a 16-bit number. The calculation of the CRC is based on the calculation given by (Landa 2021). This number is then converted to a buffer where the CCR values are set on the basis of the bits of the 16-bit number. The code manipulating the bits for generating the packet that DShot needs to send can be seen below, and Appendix H shows how a packet is placed in a DMA buffer. This implementation of DSHOT is based on a GIT repository (Seok 2021).

```
//Add telemetry bit to packet;
packet = (value << 1)|(Telemerty ? 1 : 0);

// calculate checksum
crc = (packet ^(packet >> 4) ^ (packet >> 8)) & 0x0f;

// Attach checksum to packet
packet = (packet << 4) | CRC;
```

In the implementation of Dshot300 on the Nucleo, timers 1 and 8 are set to have a prescaler of 7, which gives a frequency of $267kHz$, which is the closest frequency to the $300kHz$ that it should run at. But even though the frequency is not entirely correct, the ESC accepts this signal since Dshot can handle a margin of error. It can then be concluded that this implementation works. Even though one timer supports four PWM channels, both timers 1 and 8 are used. This was because of an issue where if one timer were used for all four PWM signals, nothing would be outputted. If three PWM signals were on time 8 has outputs and timer 1 has a single output everything works as expected.

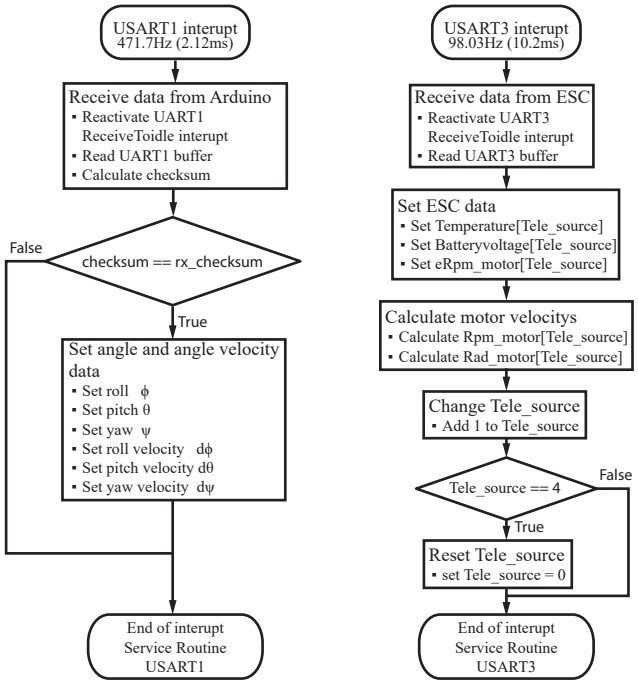


Figure 7.7. Program flowchart Part 2

When telemetry for the ESC or Arduino is received, the UART1 or UART3 interrupt function is activated. Both start by reactivating the "ReciveToIdle_IT" to look for new telemetry data. When data is received, the UART buffers are read and flushed. As mentioned for UART3 in Section 7.2, a simple checksum is created by adding all the values sent together. This value is recalculated and compared on the Nucleo. If the checksum matches, the attitude measurements and rates are updated. On UART1, the telemetry data from the ESC is received and stored in an array for one motor at a time. As mentioned in Section 7.2, the received data consists of temperature, battery voltage, and motor velocity in eRPM. The motor velocities are then calculated in $\frac{rad}{s}$. When this is done, the UART1 telemetry source is changed to the next motor in the order (0-1-2-3). If the telemetry source becomes 4 it resets to 0.

7.3.2 Aduino

The Arduino code starts by initialising variables, UART, and peripherals. After this, the main loop is entered, which runs infinitely until the code is aborted. In the main loop, the code is run at a constant frequency of 100 Hz, which is achieved by an if statement. For each cycle of the main loop, the system time is read, and if $Ts_{current} - Ts_{prev} > 0.01\ ms$ is true, the if statement is entered, where a block diagram is shown in Figure 7.8. Otherwise, the main loop continues until the if statement becomes true. $Ts_{current}$ is the current system time, and Ts_{prev} was the system time last time the if statement was true.

The gyroscope, accelerometer, and magnetometer data are read with I^2C communication. All the data is passed through a low-pass filter to remove the sensors' high-frequency noise. The filtered measurements are then given to the Extended Kalman Filter (EKF), which, via sensor fusion, estimates the Euler angles roll, pitch, and yaw. Additionally, the Euler rates are calculated. The implemented EKF is described later in Section 7.4. The Euler rates are checked for overflow, as this will result in errors in the attitude estimation. The Euler angles and rates are multiplied by 100 and converted int16_t integers, giving the control system measurements two decimal places of accuracy. A string with a checksum is created from the Euler angles and rates. The string also contains an unused command byte, for potential wireless communication, which is further discussed in Chapter 9. The checksum is the sum of all the elements in the string added together. The string is then sent via serial to the Nucleo board with the Arduino "Serial1" function. The serial baud rate is configured to 115200 baud.

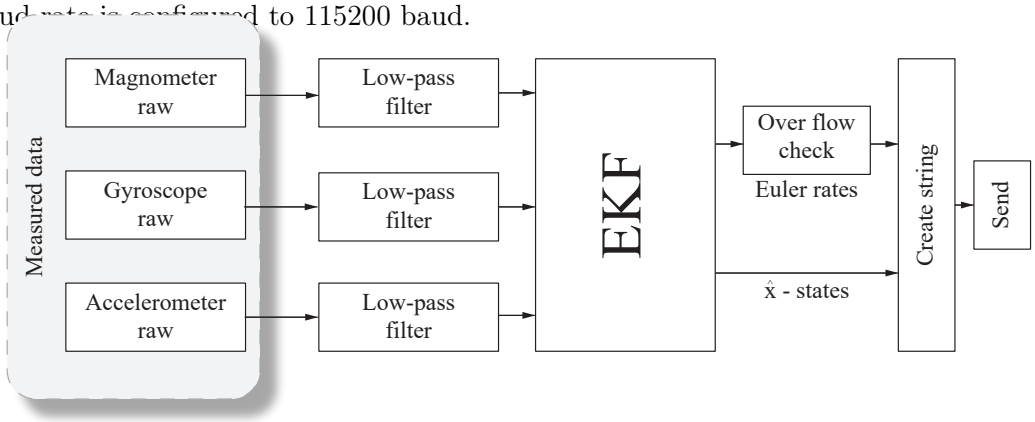


Figure 7.8. Code structure of main loop implemented on the Arduino NANO 33 IoT

7.4 AHRS

The design chosen in Chapter 4 requires Ψ-control to orient itself to the correct heading, so that the thrusters point in the swing direction. The controller designed in Section 6.2, requires feedback for α and Ψ. The two angles cannot be directly estimated from the implemented sensors, as they give measurements along their three axes. Instead, the sensors allow for constructing an Attitude and Heading Reference System (AHRS), which can provide the attitude and heading of the system in Euler angles (vectornav.com 2025). The AHRS is a non-linear sensor fusion implementation, requiring a non-linear filter such as the Extended Kalman filter (EKF). The EKF works mostly like explained in Section 2.6, but accounts for the non-linearities by introducing the process and observation model, which can be non-linear. The EKF linearises the non-linear models at first-order accuracy using Jacobians for each state in the model (Wondosen et al. 2021, p. 6).

For the construction of the AHRS, the sensor axis must be aligned with the same reference. In aeronautics, the reference is usually the Earth, and the reference frame East-North-Up (ENU) has been chosen as it resembles the model frame. In ENU, East corresponds to the positive x-axis, North is the positive y-axis, and up is the positive z-axis. The AHRS will provide the angles $[roll, pitch, yaw]^T = [\phi, \theta, \psi]^T$, as Euler rotations which correspond to the rotations in each axis. Roll describes the rotation about the x-axis, pitch the y-axis, and yaw the z-axis of the SPS system (AHRS 2025).

The ARHS model is a simplification of the gyroscope, accelerometer, and magnetometer models, which are described in the Appendix by Equation (I.2), (I.1), and (I.3), which describe what each sensor measures. The AHRS cannot capture the full scope of the models, and they are therefore simplified. The accelerometer model is reduced to the gravitational acceleration, the gyroscope is reduced to angular rates, and the magnetometer is reduced to magnetic field measurements. Therefore, the ARHS only accounts for the noise through the EKF, and elements such as bias must be compensated for during sensor calibration.

The sensor settings are defined in the initialisation of the Arduino code, where the control registers are set to the desired settings. A more detailed process of how the control registers are configured, and the process of sensor axis alignment and calibration can be found in Appendix I. The sensors are configured to have a "full scale" and "output rate" according to Table 7.3. The full scale values for the gyroscope and accelerometer are specified through testing as values that are close to expected accelerations and rotational rates but with a buffer so that overflow will not occur. The full scale of the magnetometer is configured on the lowest setting, as it is sufficient to capture all magnetic fields the system is exposed to. The output rates are configured as the lowest rate but higher than the Arduino's cycle time to compute the EKF, which is 100 Hz.

Sensor	Full scale	Output rate
Accelerometer	4 g	104 Hz
Gyroscope	1000 dps	104 Hz
Magnetometer	2 G	200 Hz

Table 7.3. Sensor register configuration.

AHRS kinematic model

The gyroscope, accelerometer, and magnetometer sensors measure in the body reference frame, where the gyroscope body rates are noted as $[p, q, r]^T$. The accelerometer measures the gravity vector depending on the orientation of the body, and the same is valid for the magnetometer, which is calibrated to measure the Earth's magnetic field in the body frame. The AHRS kinematic model transforms the body rates into Euler rates in the ENU frame. The rotation sequence is given by Euler angles and is described as $R_{ENU} = R_x(\phi) \cdot R_y(\theta) \cdot R_z(\Psi)$. The rotation matrices are given by Equation (5.26) (AHRS 2025).

R_{ENU} transforms a vector in the ENU frame to the body frame. Equation (7.3) describes how the Euler rates $[\dot{\phi}, \dot{\theta}, \dot{\Psi}]$ relate to the body rate measurements $\vec{\omega}_{gyro}$ of the gyroscope through the transformation order of R_{ENU} (Titterton and Weston 2004, p. 42).

$$\vec{\omega}_{gyro} = R_x(\phi) \cdot \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + R_x(\phi) \cdot R_y(\theta) \cdot \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + R_x(\phi) \cdot R_y(\theta) \cdot R_y(\Psi) \cdot \begin{bmatrix} 0 \\ 0 \\ \dot{\Psi} \end{bmatrix} \tag{7.3}$$

Equation (7.3) is given on matrix form by Equation (7.4) where $\vec{\omega}_{gyro} = [p, q, r]^T$.

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & sin(\theta) \\ 0 & cos(\phi) & -cos(\theta) \cdot sin(\phi) \\ 0 & sin(\phi) & cos(\theta) \cdot cos(\phi) \end{bmatrix} \cdot \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\Psi} \end{bmatrix} \tag{7.4}$$

The AHRS kinematic model transforms the body rates of ω_{gyro} into the Euler rates, which can be done with a matrix inversion on Equation (7.4), resulting in Equation (7.5). The Equation is simplified by $tan(x) = \frac{sin(x)}{cos(x)}$. The AHRS kinematic model is used in the prediction step of the EKF to transform the measured gyroscope rates into Euler rates.

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\Psi} \end{bmatrix} = \begin{bmatrix} 1 & tan(\theta) \cdot sin(\phi) & -cos(\phi) \cdot tan(\theta) \\ 0 & cos(\phi) & sin(\phi) \\ 0 & \frac{-sin(\phi)}{cos(\theta)} & \frac{cos(\phi)}{cos(\theta)} \end{bmatrix} \cdot \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{7.5}$$

Gyroscope prediction step

The AHRS state vector x_n is given as $x_n = [\phi, \theta, \Psi]$ where n is the iteration of the EKF loop. At initialisation, the state vector is defined as $x_0 = [0, 0, 0]^T$ for simplicity. Any inconsistencies in the attitude estimation for x_0 will be corrected in the first iterations of the EKF.

The implementation of the Kalman filter in Section 2.6 closely represents the structure of the EKF, where the primary difference is the introduction of non-linear functions and the iteration-based linearisation via Jacobians. The state vector x_n is predicted by integrating the gyroscope measurements in the prediction step. This is done by multiplying Equation (7.5) by the cycle time t_s, yielding the input function f in Equation (7.6), which is dependent on the gyroscope measurements, the previous estimated state vector, and the cycle time.

$$f(\vec{\omega}_{gyro}, x_{n-1}, t_s) = t_s \cdot \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\Psi} \end{bmatrix} \tag{7.6}$$

The covariance can then be predicted by Equation (7.7). where F is the jacobian of f with respect to the state vector x_n and Q_v is the process noise. At initialisation $P_0 = Q_v$.

$$P_n^- = F_n \cdot x_n^- \cdot F_n^T + Q_v \tag{7.7}$$

The states are also predicted in Equation (7.8), where the gyroscope integration is added to the previous estimated state vector.

$$x_n^- = x_{n-1} + f(\vec{\omega}_{gyro}, x_{n-1}, t_s) \tag{7.8}$$

Accelerometer update step

After the prediction step, the states can be updated. This is done in two stages, where the state vector and covariance matrix are updated with the accelerometer and then the magnetometer data. This reduces computational complexity by decreasing the matrices' size and thus decreasing computation time (P. Wang, Z. Wang, and Zhu 2017).

The SPS system and the gravity vector are sketched in Figure 7.9. Depending on the attitude of the SPS system, the body measurements from the accelerometer y_{acc} are different from the gravity vector in ENU. In ENU, the Earth's gravity is measured and represented as a vector pointing towards the Earth's centre. The gravity vector is represented in ENU as $\vec{a}_{ENU} = [0, 0, -g]^T$, where g is the gravitational constant of $g = 9.82\ \frac{m}{s^2}$.

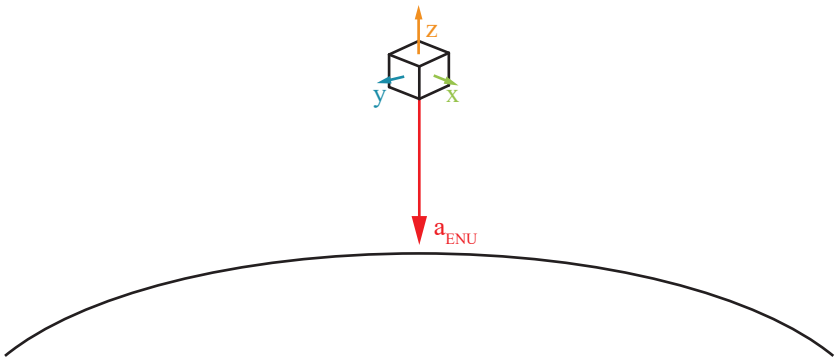


Figure 7.9. Sketch of the attitude of the SPS system, its body axis, and the ENU gravity vector

A non-linear output function for the accelerometer can be created by rotating $\vec{a}_{ENU}$ by R_{ENU} into the body frame, from where it can be compared to the measured accelerations y_{acc} to determine the correctness of the state prediction. h_{acc} is given by Equation (7.9).

$$h_{acc.n} = R_{ENU.n} \cdot a_{ENU} = \begin{bmatrix} -g \cdot sin(\theta) \\ g \cdot cos(\theta) \cdot sin(\phi) \\ -g \cdot cos(\theta) \cdot cos(\phi) \end{bmatrix} \tag{7.9}$$

The Kalman gain and the covariance are calculated via Equation (7.10) and (7.11), respectively. They are computed via matrix multiplication, where linearisation of the output function h_{acc} is required. h_{acc} is linearised by calculating the Jacobian with respect to the state vector, which

becomes H_{acc}. The Jacobian is first-order accurate of the Taylor series expansion, and can therefore only predict small changes in the state vector. This sets a requirement for how much the states can change per cycle.

$$K_{acc.n} = \frac{P_n^- \cdot H_{acc.n}^T}{H_{acc.n} \cdot P_n^- \cdot H_{acc.n}^T + R_{acc}} \tag{7.10}$$

$$P_n^{--} = (I - K_{acc.n} \cdot H_{acc.n}) P_n^- \tag{7.11}$$

R_{acc} is the observation noise matrix of the accelerometer, and I is the identity matrix.

The state vector can then be updated by comparing the predicted body measurement of the accelerometer h_{acc} with the actual body measurements from the accelerometer y_{acc}. The Kalman gain K_{acc} then determines how much the state vector should be updated.

$$x_n^{--} = x_n^- + K_{acc.n} \cdot (y_{acc.n} - h_{acc.n}) \tag{7.12}$$

Due to the structure of h_{acc}, because a_{ENU} is one-dimensional, the Kalman gain will only have zeros in the row corresponding to yaw (Ψ). The accelerometer can therefore only update roll (ϕ) and pitch (θ).

Magnetometer update step

Like the accelerometer, the magnetometer reference vector can be transformed back to the body measurements via R_{ENU}. The magnetic reference vector $B_{ENU} = [B_{x0}, B_{c.y0}, B_{c.z0}]^T$ in the ENU reference frame has the x component pointing east and the y component pointing north if it wanted to have zero heading north. This is not required for the SPS system, and B_{ENU} is therefore defined during the initialisation, in whatever heading the system has at the given point. The limitation for this approach is that it is required that $\phi \approx 0$ and $\theta \approx 0$ when B_{ENU} is defined.

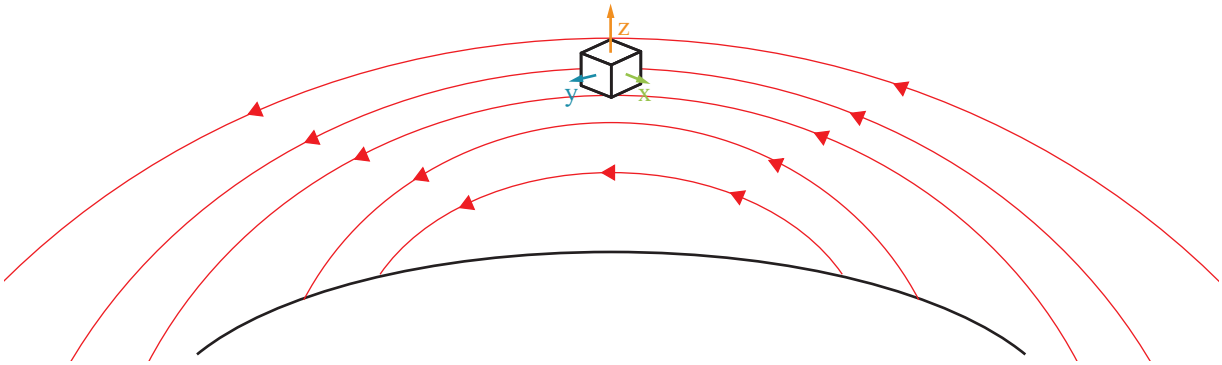


Figure 7.10. Sketch of the attitude of the SPS system, its body axis, and the ENU magnetic field vector (Not to scale)

The non-linear output function for the magnetometer is given by Equation (7.13), where B_{ENU} is rotated from ENU back into the body frame via R_{ENU}.

$$h_{mag.n} = R_{ENU.n} \cdot B_{ENU} = \tag{7.13}$$

$$\begin{bmatrix} B_{z0} \cdot sin(\theta) + B_{x0} \cdot cos(\theta) \cdot cos(\Psi) - B_{y0} \cdot cos(\theta) \cdot sin(\Psi) \\ B_{x0} \cdot (cos(\phi) \cdot sin(\Psi) + cos(\Psi) \cdot sin(\theta) \cdot sin(\phi)) + B_{y0} \cdot (cos(\phi) \cdot cos(\Psi) - sin(\theta) \cdot sin(\phi) \cdot sin(\Psi)) - B_{z0} \cdot cos(\theta) \cdot sin(\phi) \\ B_{x0} \cdot (sin(\phi) \cdot sin(\Psi) - cos(\phi) \cdot cos(\Psi) \cdot sin(\theta)) + B_{y0} \cdot (cos(\Psi) \cdot sin(\phi) + cos(\phi) \cdot sin(\theta) \cdot sin(\Psi)) + B_{z0} \cdot cos(\theta) \cdot cos(\phi) \end{bmatrix}$$

The same process is repeated for the output function h_{mag} as was done for h_{acc} to calculate the Kalman gain for the magnetometer and update the covariance matrix. The Kalman gain and covariance matrix update are given by Equation (7.14) and (7.15), where H_{mag} is the Jacobian of h_{mag} with respect to the state vector x_n.

$$K_{mag.n} = \frac{P_n^{--} \cdot H_{mag.n}^T}{H_{mag.n} \cdot P_n^{--} \cdot H_{mag.n}^T + R_{mag}} \tag{7.14}$$

$$P_n = (I - K_{mag.n} \cdot H_{mag.n}) P_n^{--} \tag{7.15}$$

R_{mag} is the observation noise matrix of the magnetometer. The state vector x_n can be updated again with the magnetometer measurement $y_{mag} = \vec{B}_c$ and compared to the output model h_{mag}'s prediction. Because B_{ENU} has components in all axes, the Kalmal gain K_{mag} will have a value in every element of the matrix and can therefore be used to update Ψ but also further improve the ϕ and θ estimates. The state vector is updated for the magnetometer measurements in Equation (7.16), which is then used in Equation (7.8) for the next iteration.

$$x_n = x_n^{--} + K_{mag.n} \cdot (y_{mag.n} - h_{mag.n}) \tag{7.16}$$

The block diagram of the EKF is shown in Figure 7.11. The blue boxes indicate a sensor input, the yellow box is the input function, and the dark blue boxes are the output functions. The grey boxes are the process and observation noise and are constants, green is the covariance matrix calculations, magenta is the Kalman gain calculation, and orange is the state vector calculations. The black J block is the calculation of the Jacobian matrix.

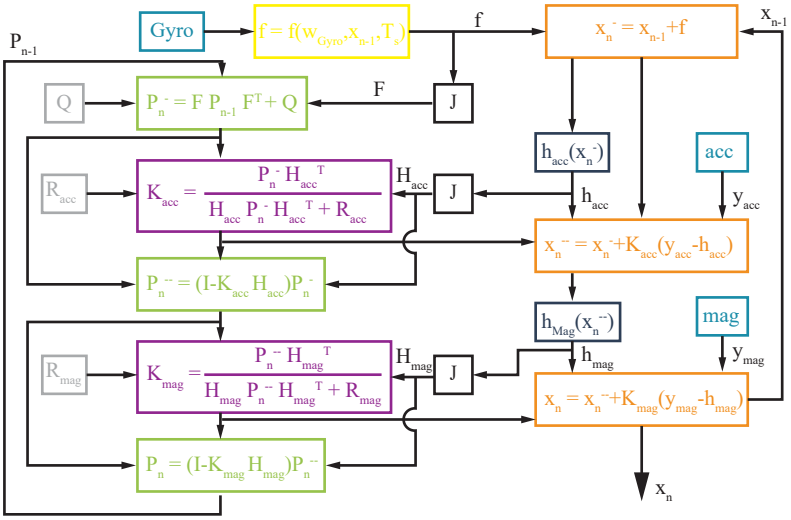


Figure 7.11. Block diagram of the EKF

Process and observation noise

The process noise Q_v and the observation noise for the accelerometer R_{acc} and magnetometer R_{mag} need to be defined to predict the states correctly. The Q_v and R matrices are based on

knowledge of the system and statistics of the measurements, where the noise is assumed to be discrete, zero-mean Gaussian. In Section 7.3.2 it was described that the sensor measurements are passed through a low-pass filter before they are used in the EKF. Because the EKF gets the filtered measurements, the noise matrices will be based on the filtered measurements. Each axis for each sensor is measured when the system is at a standstill to capture only the noise of the sensors. The variance of the measured data is calculated and will be the diagonal elements of the noise matrices and are shown in Table 7.4 (Titterton and Weston 2004, pp. 511–516).

	x-axis	y-axis	z-axus	Unit
Gyro	$9.35e^{-8}$	$2.32e^{-7}$	$1.10e^{-7}$	$\left(\frac{m}{s}\right)^2$
Acc	$2.30e^{-5}$	$2.40e^{-5}$	$2.20e^{-5}$	$\left(\frac{m}{s^2}\right)^2$
Mag	$3.25e^{-2}$	$2.50e^{-2}$	$3.99e^{-2}$	$(mT)^2$

Table 7.4. Variance for each axis of the sensors

The matrices are finally defined by Equation (7.17). Q is multiplied by the desired cycle time of $T_{s.0} = 0.01s$ as discretisation influences the process noise.

$$Q_v = \begin{bmatrix} 9.35e^{-8} \cdot T_{s.0} & 0 & 0 \\ 0 & 2.32e^{-7} \cdot T_{s.0} & 0 \\ 0 & 0 & 1.10e^{-7} \cdot T_{s.0} \end{bmatrix} \quad R_{acc} = \begin{bmatrix} 2.30e^{-5} & 0 & 0 \\ 0 & 2.40e^{-5} & 0 \\ 0 & 0 & 2.20e^{-5} \end{bmatrix} \tag{7.17}$$

$$R_{mag} = \begin{bmatrix} 3.25e^{-2} & 0 & 0 \\ 0 & 2.50e^{-2} & 0 \\ 0 & 0 & 3.99e^{-2} \end{bmatrix}$$

7.5 MIMO implementation

The implemented MIMO control on the MCU differs from the theoretical perfect implementation due to the available data. The measured data does not produce swing angle α or swing plane angle γ. It has to be extrapolated from the roll and pitch measurements. Furthermore, it is also necessary to introduce a dead zone for the SPS to become steady when the error is small. Also, measurement noise will introduce a constant error, which cannot be corrected optimally due to the motor's zero-crossing turn-off.

7.5.1 Swing detection

From Section 5.3.6, the mechanical pendulum model was linearised, and it was determined that $\Psi = \gamma \pm 90°$ for all the thrust to have maximum effect in reducing the swing angle α. The AHRS does not directly provide α or γ, and they therefore need to be determined from the available data. From the AHRS roll (ϕ), pitch (θ), and yaw (Ψ) measurements are available, where it can be seen that if $\theta = 0$, then $\phi = \alpha$. It is then desired that with a given set of measurements $[\phi,\ \theta,\ \Psi]$, to determine how much Ψ has to be rotated, in order for $\phi = \alpha$. The rotation is given by $R_z(\Psi_+)$, where Ψ_+ is how much Ψ has to change. By solving Equation (7.18), Ψ_+ can be calculated.

$$\begin{bmatrix} \alpha \\ 0 \end{bmatrix} = R_z \cdot \begin{bmatrix} \phi \\ \theta \end{bmatrix} = \begin{bmatrix} cos(\Psi_+) & -sin(\Psi_+) \\ sin(\Psi_+) & cos(\Psi_+) \end{bmatrix} \cdot \begin{bmatrix} \phi \\ \theta \end{bmatrix} \tag{7.18}$$

Ψ_+ is calculated by solving the $0 = sin(\Psi_+) \cdot \phi + cos(\Psi_+) \cdot \theta$, and is given by Equation (7.19).

$$\Psi_+ = -atan\left(\frac{\theta}{\phi}\right) \tag{7.19}$$

A validation of the approach can be seen in Figure 7.12, where different roll and pitch angles have been inserted into Equation (7.19). The approach suffers from division by 0 when $\phi = 0$. When division by 0 occurs, $\Psi_+ = 90°$. From Figure 7.12, it can be seen that the Ψ_+ values will rotate the SPS system in the shortest distance to obtain $\phi = \alpha$.

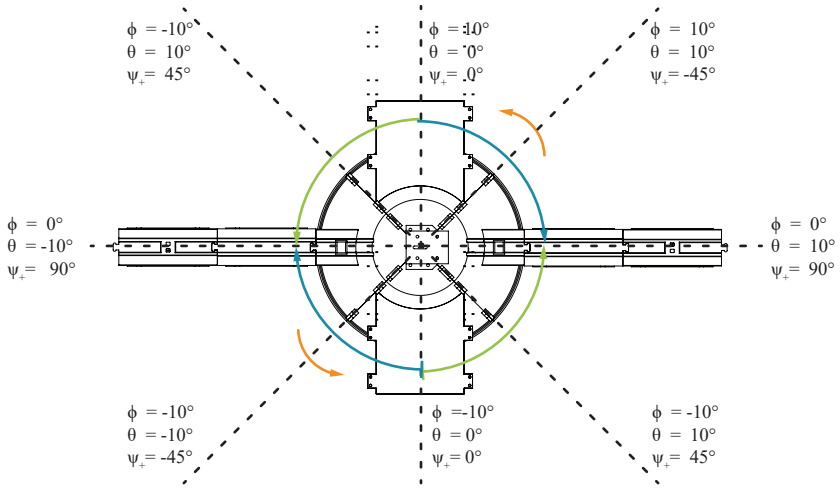


Figure 7.12. The calculated Ψ_+ rotations for different ϕ and θ values. The orange arrows indicate the positive rotation direction. The blue and green arrows indicate the rotation direction in the corresponding quadrant

The Ψ_{ref} reference for the SPS is then calculated by Equation (7.20), where Ψ_+ is added to the current Ψ measurement. This results in the $\Psi_{ref} = \gamma \pm 90°$.

$$\Psi_{ref} = \Psi + \Psi_+ \tag{7.20}$$

A problem with this approach is that when both ϕ and θ are small, the measurement noise influences the calculation of Ψ_+, which leads to dramatic and unwanted changes in Ψ_{ref}. Therefore, a reference update limit is implemented, such that Ψ_{ref} is not updated when either ϕ or θ is less than 6° and has a velocity under $15°/s$.

When α is large, the reference is more stable. Still, sometimes a small double pendulum effect can be noted, which disturbs Ψ_{ref}. The assumption of the SPS being perpendicular to the wire is not entirely true. To get a more stable Ψ_{ref}, a low-pass filter has been added to reduce the double pendulum effects on Ψ_{ref}. Equation (7.21) gives the transfer function for the low-pass filter, where the time constant τ is set to 0.025 to suppress frequencies above 40 Hz.

$$lowpass(s) = \frac{1}{1 + \tau \cdot s} \tag{7.21}$$

The implemented discrete low-pass filter is given by Equation (7.22). The swing detection approach is implemented in timer 6 on the Nucleo board, with a sample rate of $\approx 100\ Hz$.

$$y_0 = 0.166 \cdot u_0 + 0.166 \cdot u_1 + 0.666 \cdot y_1 \tag{7.22}$$

7.6 α thrust zone

Because of the Ψ rotation, thrust can be in directions which does not contribute to dampening α. This is accounted for by calculating Ψ_{ref} as a reference such for control $\alpha = \phi$. If $\alpha = \phi$ is used for control, it has to face the swing direction for $\alpha = \phi$ to be allowed. This is not always possible as an Ψ_{error} can be present. Therefore, a "α thrust zone" is implemented, such that when Ψ_{error} becomes larger than a specified threshold, the MIMO control should not try to dampend α.

To check if the SPS is facing the correct dirrection the yaw error is found by $\Psi_{error} = \Psi_{ref} - \Psi$. If the absolute yaw error is greater than 30° it is wanted to disable the α dampening of MIMO control system. The if condition is given by $abs(\Psi_{error}) \geq 30°$, where Equation (7.23), shows how the α angle control is disableded in the MIMO control. This is done by setting all states acting on the α state to 0.

$$\begin{aligned} \alpha_{error} &= 0 \\ q_\alpha &= 0 \\ x_n &= [0,\ 0,\ \Psi,\ \dot{\Psi}]^T \end{aligned} \tag{7.23}$$

When $abs(\Psi_{error}) < 30°$ it is assumed most of the thrust is in the swing direction and can be used to dampend α. α is then set to $\alpha = \phi$, where the α_{error} is calculated as $\alpha_{error} = \alpha_{ref} - \phi$.

7.6.1 Dead zone

During the tuning process and testing of Ψ, it was observed that the system had difficulties finding a stable position at the specified reference point. The primary reason the this is suspected to be because of the motor zero crossing issue, where it has a dead period of $\approx 0.5\ s$ when changing spin direction. This makes it difficult to make quick small adjustment, and thereby stop at the reference. This is an expected issue, discovered during the BLDC controller test in Section 6.1.2.

Because of the system structure of the MIMO system, it is not possible to turn off the controller output u for α or Ψ. The dead zone is only active when ϕ, θ, and Ψ are within the dead zone limits. To enter the dead zone, four conditions have to be met. The absolute values of ϕ_{error} and θ_{error} both have be be less than α_{dead}. At the same time $\dot{\phi}$ and $\dot{\theta}$ has to be less than $\dot{\alpha}_{dead}$. For Ψ the Ψ_{error} has to be less than Ψ_{dead} and $\dot{\Psi}$ is less than $\dot{\Psi}_{dead}$. The dead zone values are given in Table 7.5. If all four criteria are met, u and $\bar{u}$ in the MIMO system are both set to 0. Resulting in no thrust output, while not changing the integrator states.

	Dead zone values
α_{dead}	2.86°
$\dot{\alpha}_{dead}$	$5.73°/s$
Ψ_{dead}	7°
$\dot{\Psi}_{dead}$	$5.73°/s$

Table 7.5. Dead zone values for the states.

Test of system 8

This chapter aims to show and describe the primary test results of the test setup and simulation. The purpose of the test is to verify the design and control system. The tests of the physical system will demonstrate that the design can dampen swings when released from an initial position, representing a large position disturbance, such as a sudden burst of wind or a drone movement. Additionally, a large fan capable of producing wind speeds above 10 $\frac{m}{s}$ is used to simulate wind conditions. Lastly, additional mass is placed as a payload to validate the system's response when carrying loads. The same tests will be simulated to compare the model with the real world. Furthermore, the system can be tested in situations beyond the limitations of the test setup, such as the lowering procedure.

8.1 Test setup

The SPS test setup is sketched in Figure 8.1. The SPS system is hanging from the ceiling at the centre of the circle. The wire length is $L_w = 1.78\ m$. The circle is defined according to the global swing plane angle γ, where a fan is located at 0° blowing towards the SPS. Tests will be conducted at two swing plane angles, 180° and 270°. At each swing plane angle, the system is dropped in either a roll swing or a pitch swing as described in Figure 5.15. The SPS system is dropped from an initial angle to represent a disturbance at different wind speeds, allowing for a comparison of the wind's influence on the SPS system and its ability to control attitude.

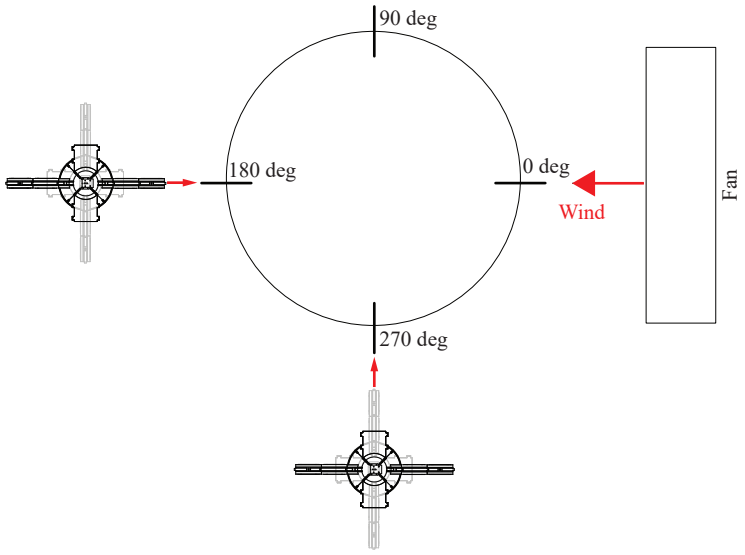


Figure 8.1. Sketch of the test setup where the location of the fan and the two starting positions can be seen

8.2 Data processing

The SPS system attitude results are presented in a polar plot, corresponding to the circle shown in Figure 8.1. To display and analyse the data from the SPS, which consists of the measured altitude, given by the SPS's roll ϕ, pitch θ, and yaw Ψ has been used to estimate the swing angle α and swing plane angle γ. This aims to make the data more tangible and similar to the model data given in Section 5.3.1.

α is a combination of ϕ and θ. Rotating ϕ and θ by $-\Psi$ as shown in Equation (8.1), yields ϕ_{rot} and θ_{rot}. ϕ_{rot} and θ_{rot} is a global representation of the swing components.

$$\begin{bmatrix} \phi_{rot} \\ \theta_{rot} \end{bmatrix} = \begin{bmatrix} cos(\Psi) & -sin(\Psi) \\ sin(\Psi) & cos(\Psi) \end{bmatrix}^T \begin{bmatrix} \phi \\ \theta \end{bmatrix} \tag{8.1}$$

From ϕ_{rot} and θ_{rot} the swing angle α is estimated according to Equation (8.2). When Pythagoras is used, only absolute angles are calculated.

$$\alpha = \sqrt{\phi_{rot}^2 + \theta_{rot}^2} \tag{8.2}$$

γ can also be estimated with the use of atan2, which finds the angle of two components of a 2D plane. γ is estimated in Equation (8.3).

$$\gamma = atan2(\theta_{rot}, \phi_{rot}) \tag{8.3}$$

As a result of using atan2, the γ angle changes by 180° when the SPS passes the origin. This is not present in the model from Section 5.3.1, where it instead would flip the sign of α. In the test results plots in Section 8.3, the plots of the α angles are altered so that when γ jumps by 180°, the sign of the α angle is flipped. This is a visual change intended to enhance the representation of the swing angle. γ is not altered.

Stabilisation condition

It is also necessary to have a condition for when the SPS system is considered stable. Here, it was decided that the SPS is viewed as stable when the swing angle α is less than the dead zone. The dead zone is used as a stability condition, as the movement of the SPS is minimal when it remains within the dead zone. To check when the condition is met, a "for" loop is used to check when the rest of the swing angle α dataset is less than the dead zone for the remaining part of the dataset.

8.3 SPS tests results

This section will show and analyse the test results of the SPS system under different conditions and test scenarios. The tests conducted will be compared to the pendulum system without control, as a reference. The SPS system is tested with a roll and pitch start at global swing plane angles of 180° and 270° at windspeeds of 0, 5 7.5 10 $\frac{m}{s}$.

SPS - system with no load

To start testing the SPS system. It was decided to test the system's response when a pitch disturbance is present on the SPS system, with no external wind affecting the system. The same test with a roll disturbance was used in the tuning process of the MIMO controller in Section 6.2, where it was determined that it could stabilise from a roll disturbance of $\approx 23°$ to the dead zone in $\approx 3.5s$.

For the pitch tests, the SPS must first correct its yaw angle to align with the swing plane angle, before α can be dampened. The results of the pitch swing with and without a controller are shown in Figure 8.2.

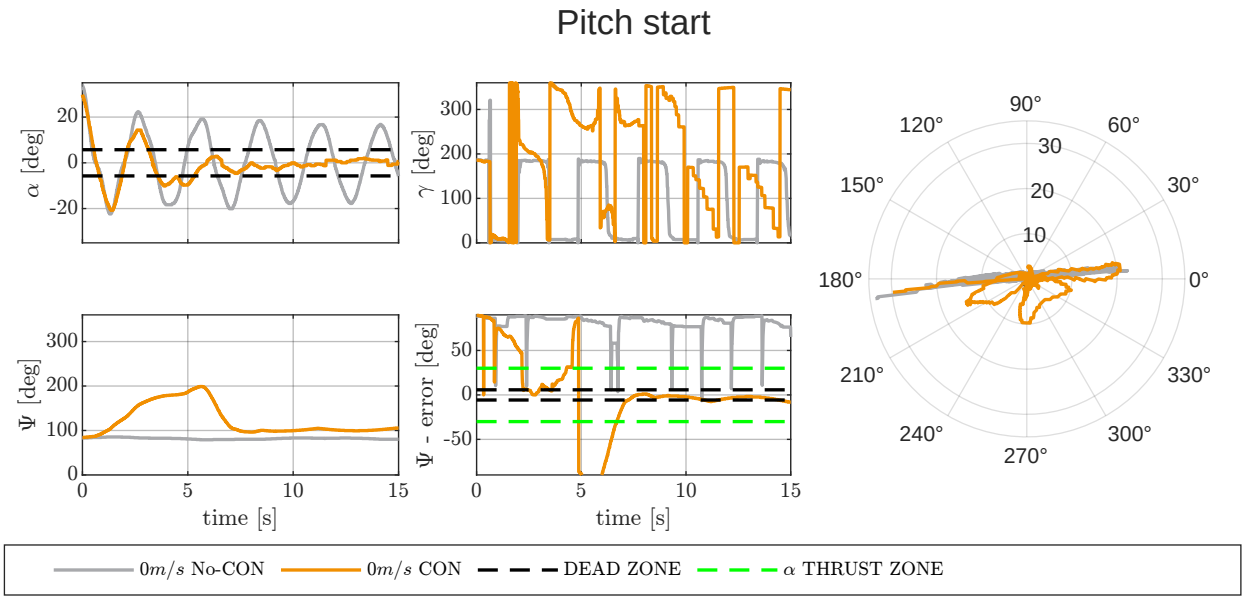


Figure 8.2. Plot of SPS attitude measurements during pitch disturbance test with no wind (Video: `https://www.youtube.com/watch?v=XNPwJqQ5Oo4&t=1s`)

In the Pitch start test with no wind, it can be seen that yaw starts rotating with $\approx 90°$ and resulting in a final yaw angle of $\Psi \approx 180°$ during the first ≈ 3 s. Whereafter the SPS enters the "α thrust zone" and the SPS with controller starts deviating from the one without in α. α is then fully dampened after ≈ 2 s, resulting in a time until stability of 5.22 s from an initiall disturbance of $\theta = 33.8°$. It can be noted that at $t \approx 5$ s a Ψ_{ref} disturbance occurs, and Ψ rotates to the new reference, even though stability was already obtained. This was supposed to be suppressed by the update limit described in Section 7.5.1, however, it is due to measurement errors, which have been removed from the plots.

Tests with wind

It can be seen that the system works fine with both initial disturbances in roll and pitch swings. Therefore, it was decided to add an external wind to the system, where the system is tested at the following wind speeds $[5, 7.5, 10]$ m/s. To ensure the fan was applying the expected wind speeds, they were measured using a hot-wire anemometer, specifically the SwemaAir 30. The test results of the "Pitch start - with wind - 180°" are shown in Figure 8.3. The naming scheme of the test implies the swing is started at 180° on the global swing plane, and in a pitch swing.

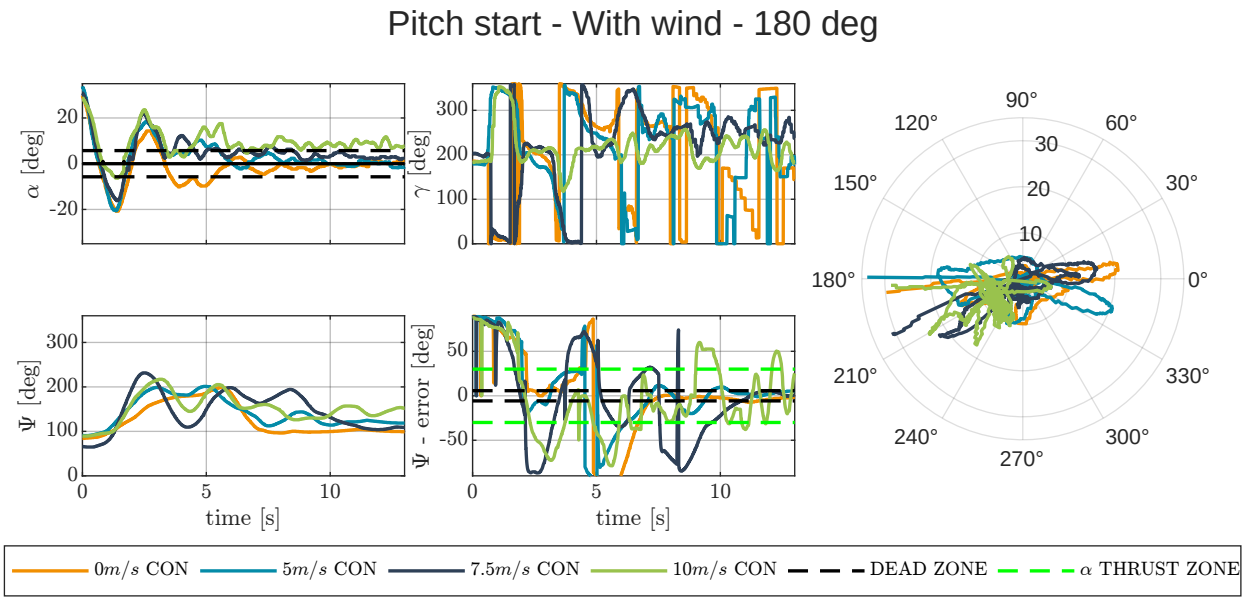


Figure 8.3. Plot of SPS attitude measurements doing pitch disturbance test with wind starting at swing angle 180° (Video: `https://www.youtube.com/watch?v=HM-9akl5OlA`)

In Figure 8.3, it can be observed that the yaw starts by rotating with $\approx 90°$ for all the tests, whereafter it can start to dampen the swing angle. The time until stability is given by 5.33 s for the 5 $\frac{m}{s}$ test, 5.53 s for the 7.5 $\frac{m}{s}$, and in the 10 $\frac{m}{s}$ test the SPS system never reaches stability. In Table 8.1, a summary of the until stability, and the initial α angles are shown. The 10 $\frac{m}{s}$ can keep a steady Ψ within the α thrust zone, and can therefore never produce enough thrust to reach $\alpha_{ref} = 0°$.

In Figure 8.4, the test is replicated for a roll swing.

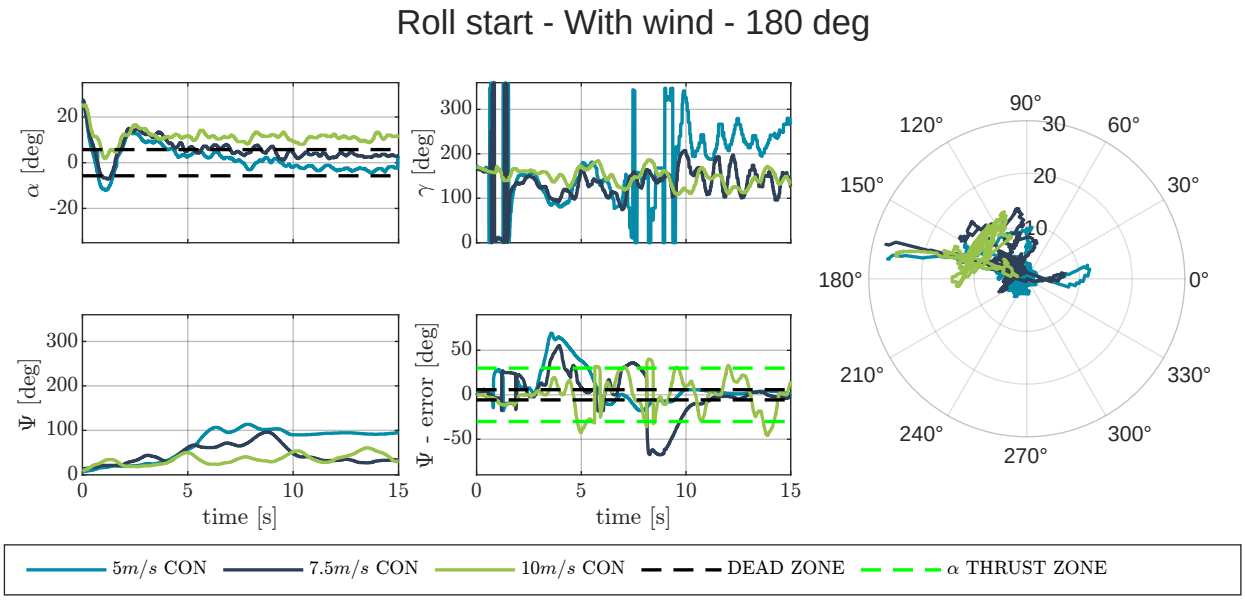


Figure 8.4. Plot of SPS attitude measurements doing roll disturbance test with wind starting at swing angle 180° (Video of test: `https://www.youtube.com/watch?v=GQh0L-o4U1k`)

In Figure 8.4, it can be observed that the Ψ is not required to rotate to enter the α thrust zone, which results in generally faster times until stability. The time until stability is given by 4.34 s for the 5 $\frac{m}{s}$ test, 7 s for the 7.5 $\frac{m}{s}$, and in the 10 $\frac{m}{s}$ test the SPS system never reaches stability.

In Table 8.1, a summary of the until stability, and the initial α angles are shown. The 7.5 $\frac{m}{s}$ test is slower to reach stability because of noise in Ψ_{ref}, and the α angle is hovering on the edge of the dead zone for a long period.

SPS - system with load

The SPS is designed to carry a payload of up to 25.44 kg, but due to laboratory restrictions, only a 10 kg load was permitted. A mounting cone was designed to dock inside the docking cone of the SPS, allowing it to lock into the locking mechanism and securely fasten the payload to the SPS. It was also designed to maintain the aerodynamic properties of the SPS by keeping the added mass internal. The mounting cone can be seen in Figure 8.5. A metal block was added, resulting in the payload having a total weight of 10.75 kg, and increasing the SPS system mass to a total weight of 15.3 kg. This is a 235% increase in the total mass of the system.

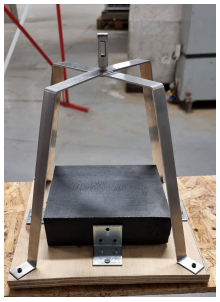


Figure 8.5. The mounting cone with weight

The SPS system was then tested again without wind, incorporating the added weight of the payload. The plots of the attitude during these tests can be seen in Figure 8.6 for the pitch swing test and in Figure 8.7 for the roll swing test.

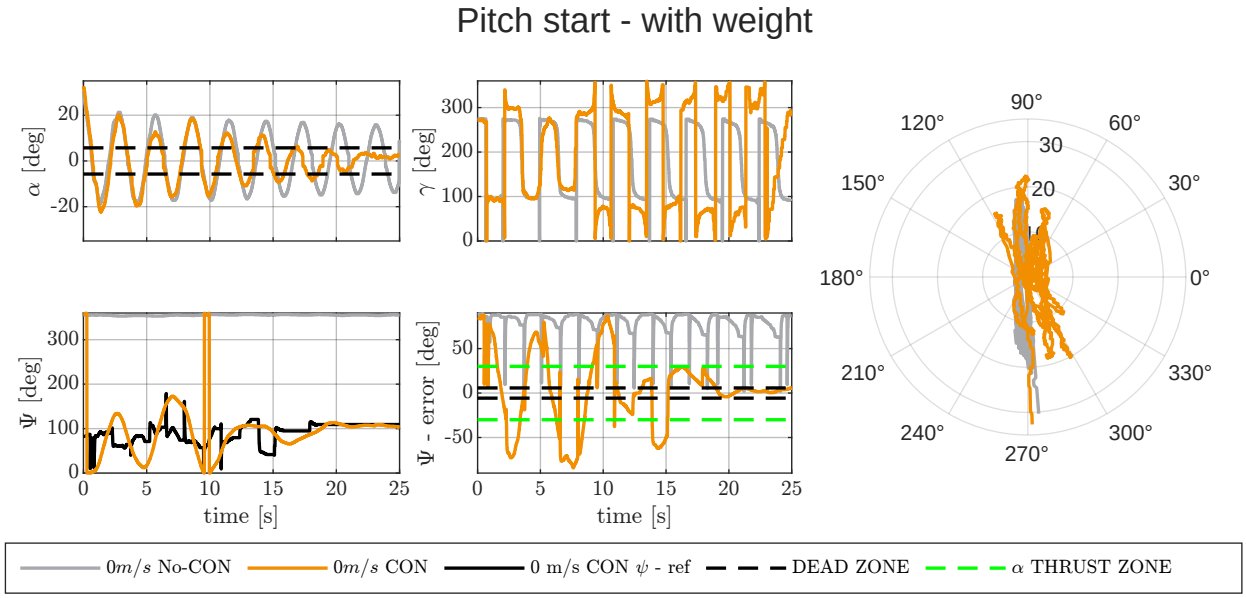


Figure 8.6. Plot of SPS system with load attitude measurements doing pitch disturbance test with no wind. (Video: `https://www.youtube.com/watch?v=DN5OmTJ5CbE`)

In Figure 8.6, the results of adding the load to a pitch swing can be seen. First, the duration for the system to naturally stabilise was tested. For an initial pitch of 30.36°, it took 198.9s to stabilise; this represents a 130% increase in stabilisation time. When the controller is activated, it is evident that the yaw control begins relatively poorly, starting with overshooting, then oscillating between 0° and 90° before finding the correct swing orientation after $\approx$ 13 s. It can also be observed that α starts decreasing quickly after yaw is facing the proper direction. It therefore results in a stabilisation time of 18.53s for when a pitch swing of 32.65° is applied to the system.

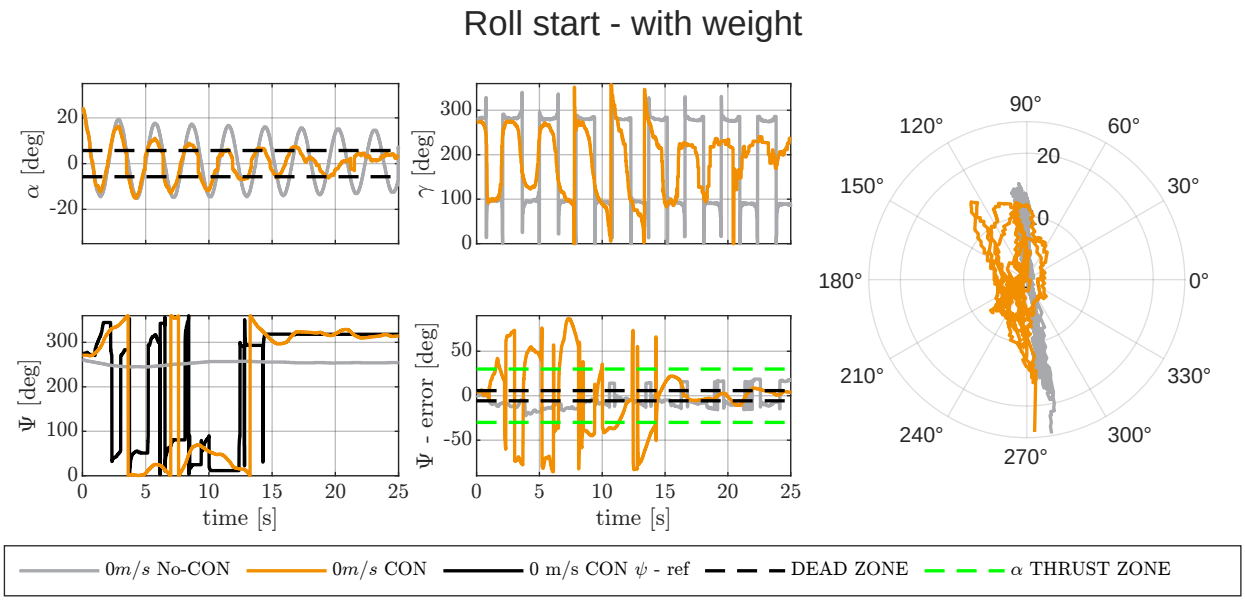


Figure 8.7. Plot of SPS system with load attitude measurements during roll disturbance test with no wind (Video: `https://www.youtube.com/watch?v=Cl-IqmKBkyo`)

For the "roll start - with weight" test, it can also be observed that the control system similarly struggles to maintain the yaw angle. But for this test, it is not because the controller overshoots due to the added mass, but rather the calculation of Ψ_{ref} does not calculate the correct heading. It is suspected that this issue occurs because the payload mass is not balanced, and thereby introduces a permanent error in Ψ_{ref} calculations. It is first after 15 s where the system obtains the desired yaw angle, because Ψ_{ref} is no longer updated. Even though this happens, the system is still capable of stabilising in 11.27 s from an initial swing angle of 24.12°. The natural system starts at an initial swing angle of 24.63°, and takes 178.42 before stabilising.

Tests with wind & weight

The SPS system with a load was also tested under the influence of wind. The pitch swing test with load and wind starting at a swing angle of 180° can be seen in Figure 8.8, and the roll swing test with the same conditions can be seen in Figure 8.10.

In Figure 8.8, the results of the pitch swing test with load and wind are shown. It can be seen that Ψ_{ref} is much more stable compared to when no wind was applied. This is expected to be because the wind pushes the SPS, which results in the payload mass offset having a lesser influence. The tests still exhibit a yaw overshoot, but it appears that the wind helps orient the SPS in the correct heading. This results in the system being in the α thrust zone earlier, and thereby reduces the swing in α more. For the 5 $\frac{m}{s}$ test the time until stability was 14.99 s, the 7.5 $\frac{m}{s}$ test took 12.2 s, and the 10 $\frac{m}{s}$ took 13.92s. The initial swing angles are given in Table 8.3.

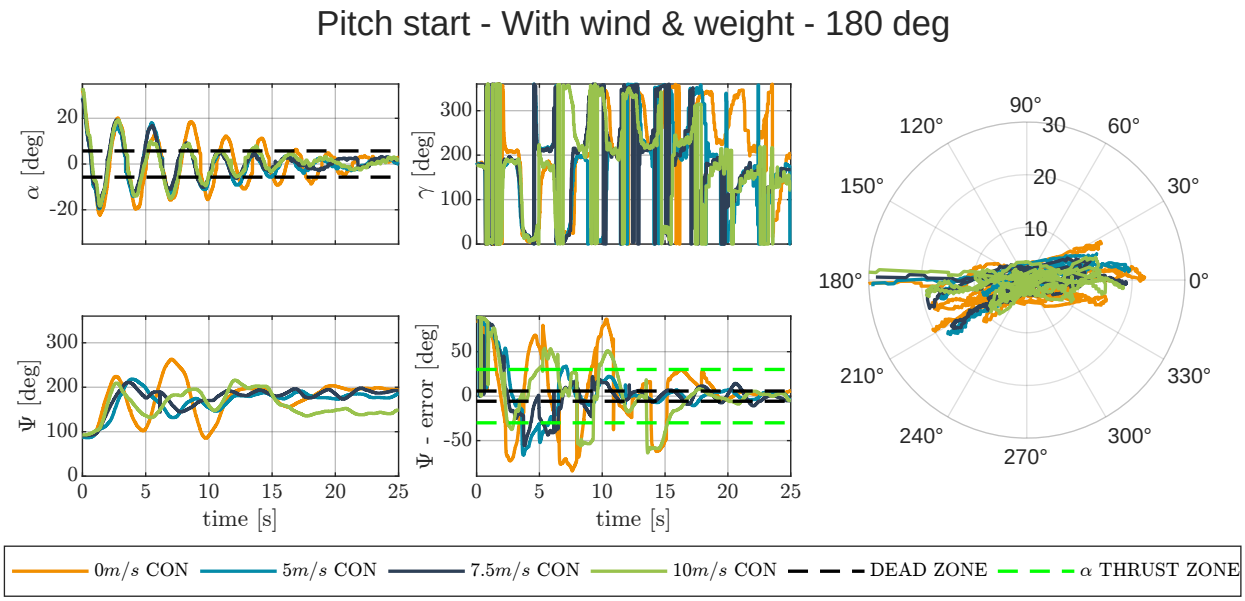


Figure 8.8. Graphs of SPS System response to angle disturbance with load & wind (Video: `https://www.youtube.com/watch?v=3sGXjpYzzYg`)

The motor velocities and references are shown in Figure 8.9 for the 10 $\frac{m}{s}$ with and without the added payload mass.

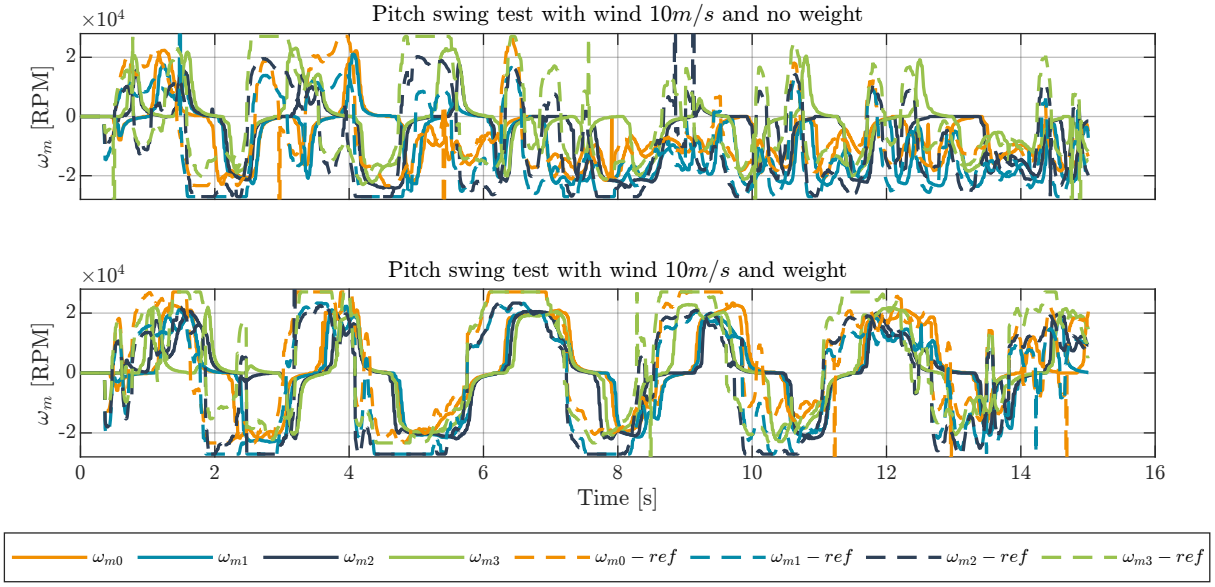


Figure 8.9. The motor velocities and references in the pitch swing tests with wind speeds of 10 $\frac{m}{s}$

From the previous tests, it can be seen that when a wind speed of 10 $\frac{m}{s}$ is applied, the system does not reach the dead zone, but when the payload mass is added, it can now. This is because of the force equilibrium between gravity and aerodynamics. When the SPS mass is increased, gravity now has a greater influence, and because the drag force has not increased, it is easier to bring the system into the dead zone. The extra mass also increases the inertia of the SPS, which results in the overshoots in yaw, but it also makes the wind have less of an influence as the yaw rotation system becomes slower. Because the system has become slower, the motor references do not change as often, which means that they are on for a more extended period of time with

fewer zero-crossing turn-off events as seen in Figure 8.9. This allows the motors to follow the references, which in return also improves the system stability at higher wind speeds.

The results of the roll swing test with wind and weight are shown in Figure 8.10. The results look similar to the pitch swing with wind and weight shown in Figure 8.8, although the tests at 7.5 $\frac{m}{s}$ and 10 $\frac{m}{s}$ have a more noise Ψ_{ref}. All the tests reach stability, where 5 $\frac{m}{s}$ takes 15.5 s, 7.5 $\frac{m}{s}$ takes 18.86 s, and 10 $\frac{m}{s}$ takes 15.77 s. The results further prove that the wind speed does not directly correlate to the time until stability, as the 5 $\frac{m}{s}$ and 10 $\frac{m}{s}$ take approximately the same amount of time. The time until stability is mostly specified from how fast $\Psi = \gamma \pm 90°$ is obtained. The initial swing angles are given by Table 8.3.

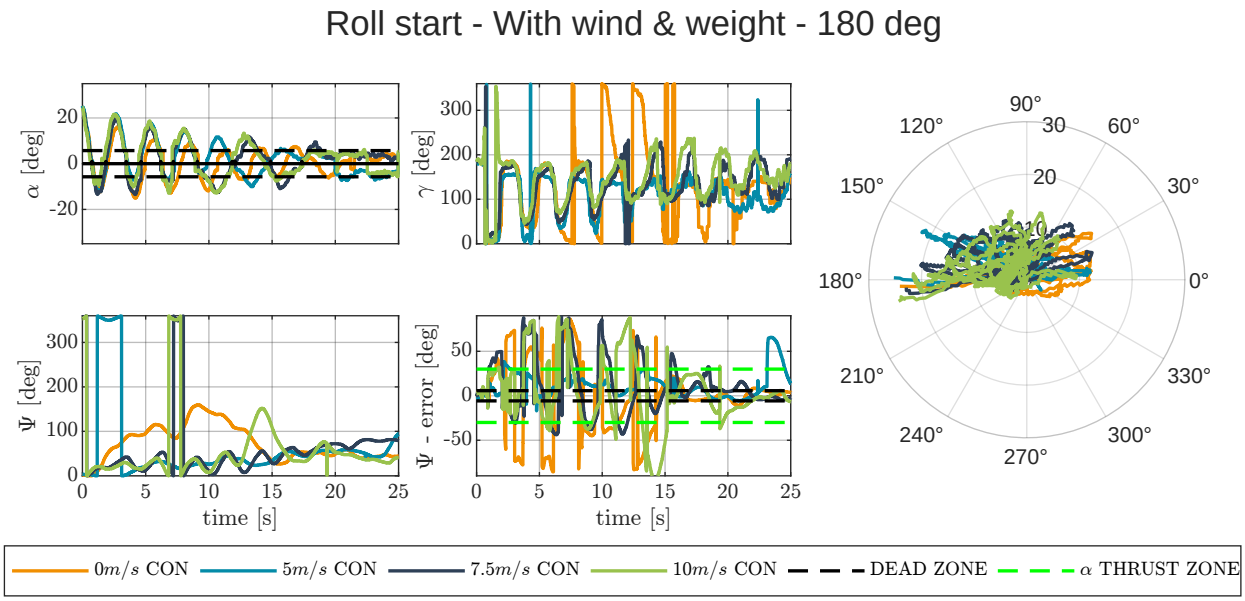


Figure 8.10. Graphs of SPS System response to angle disturbance with load & wind (Video: `https://www.youtube.com/watch?v=vQxtA40eZ8M`)

All the shown tests have also been done from the global swing plane angle of 270°, and are shown in Appendix J. This is done to investigate how the system behaves when it is affected by a wind force orthogonal to the swing plane angle. In this case, the system needs to dampen the existing swing angle and the angle produced by the wind force.

Experiment No Load	α_0	γ_0	t_{DZ}	t_{diff}
		Pitch start 180°		
NOCON	33.80°	187.44°	86.41 s	–
0 $\frac{m}{s}$	29.79°	175.72°	5.33 s	81.08 s
5 $\frac{m}{s}$	33.8°	179.42°	5.53 s	80.88 s
7.5 $\frac{m}{s}$	30.94°	203°	6.38 s	80.03 s
10 $\frac{m}{s}$	28.67°	182.86°	*nan*	*nan*
		Roll start 180°		
NOCON	26.92°	180.52°	70.69 s	–
0 $\frac{m}{s}$	23.7°	180.2°	3.52 s	67.17 s
5 $\frac{m}{s}$	26.93°	171.90°	4.34 s	66.35 s
7.5 $\frac{m}{s}$	27.5°	166.82°	7.01 s	63.68 s
10 $\frac{m}{s}$	24.63°	170.25°	*nan*	*nan*

Table 8.1

Experiment No Load	α_0	γ_0	t_{DZ}	t_{diff}
		Pitch start 270°		
NOCON	33.80°	187.44°	86.41 s	–
0 $\frac{m}{s}$	29.79°	295.72°	5.33 s	81.08 s
5 $\frac{m}{s}$	29.79°	280.84°	8.42 s	77.99 s
7.5 $\frac{m}{s}$	29.79°	280.84°	8.26 s	78.15 s
10 $\frac{m}{s}$	30.94°	283.09°	*nan*	*nan*
		Roll start 270°		
NOCON	26.92°	180.52°	70.69 s	–
0 $\frac{m}{s}$	23.7°	270.2°	3.52 s	67.17 s
5 $\frac{m}{s}$	28.67°	274°	3.93 s	66.76 s
7.5 $\frac{m}{s}$	28.09°	272°	3.01 s	67.68 s
10 $\frac{m}{s}$	28.07°	284.89°	*nan*	*nan*

Table 8.2

All the test results are summarised in tables. Tables 8.1 to 8.4 provide the no-load results for, respectively, 180° and 270° global swing plane angles. Tables 8.2 and 8.3 provide the results for

the tests with load for, respectively, 180° and 270° global swing plane angles. t_{DZ} is the time until the system is stable, which is defined as the time until the dead zone is entered for the rest of the test. t_{diff} is the difference between the t_{DZ} times for each test and the NOCON test. t_{diff} indicates the improvement when the controller is added.

Experiment Load	α_0	γ_0	t_{DZ}	t_{diff}
		Pitch start 180°		
NOCON	30.36°	184.58°	198.94 s	–
0 $\frac{m}{s}$	32.65°	181.71°	18.53 s	180.41 s
5 $\frac{m}{s}$	29.22°	182.29°	14.99 s	183.95 s
7.5 $\frac{m}{s}$	28.64°	178.85°	12.32 s	186.62 s
10 $\frac{m}{s}$	33.23°	177.13°	13.92 s	185.02 s
		Roll start 180°		
NOCON	24.63°	189.16°	178.42 s	–
0 $\frac{m}{s}$	24.12°	182.93°	11.27 s	167.15 s
5 $\frac{m}{s}$	25.23°	179.16°	15.5 s	162.92 s
7.5 $\frac{m}{s}$	22.92°	183.72°	18.86 s	159.56 s
10 $\frac{m}{s}$	24.66°	189.53°	15.77 s	162.65 s

Table 8.3

Experiment Load	α_0	γ_0	t_{DZ}	t_{diff}
		Pitch start 270°		
NOCON	30.36°	274.58°	198.94s	–
0 m/s	32.65°	271.71°	18.53s	180.41s
5 m/s	28.64°	271.71°	8.27s	190.67s
7.5 m/s	28.07°	269.42°	15.5s	183.44s
10 m/s	33.80°	272.11°	14.01s	184.93s
		Roll start 270°		
NOCON	24.63°	279.16°	178.42s	–
0 m/s	24.12°	272.93°	11.4s	167.02s
5 m/s	23.55°	278.19°	12.78s	165.64s
7.5 m/s	24.66°	273.23°	11.54s	166.88s
10 m/s	23.51°	273.23°	11.23s	167.19s

Table 8.4

Test - conclution

The test results for the SPS system without payload for both swing types are stabilised in $\approx 3 - 8.5$ s. This is within the desired criteria for time until stability, specified in Chapter 4 of 15 s. When the 10 kg payload is added, the system mass is increased by 235%, the time until stability is increased to $\approx 11 - 18$ s. This is on the edge of the design criteria. If the full weight of the SPS system of 30 kg is reached, it would increase the mass of the system by 560%, which probably will make the SPS system even slower. Therefore, it is not expected to meet the design criteria.

The results of the 10 $\frac{m}{s}$ tests without payload did not reach stability within the dead zone. When the payload mass was added, the system was now able to achieve stability in $\approx 11.5 - 15$ s. It is presumed that the increase in mass changes the equilibrium between the gravitational and aerodynamic forces. This results in smaller swing angle deviation from the wind, and the SPS can overcome these. Increasing the mass to 30 kg would further improve this relation, although aerodynamic forces from the payload are not accounted for. The yaw control would likely require tuning for higher payload masses to minimise the overshoot. As long as the dry mass of the system is slightly increased or the drag forces are reduced, the system should be able to operate in wind speeds of 10 $\frac{m}{s}$.

The roll tests were expected to reach the dead zone faster than the pitch tests, as they could dampen from the start of the test. This is not what is evident from the test results, which show they are closer to having identical performance. The lack of performance for the roll tests is expected to be due to the method used for calculating Ψ_{ref}, which is described in Section 7.5.1. The method performed as expected when initially tested on the SPS system in steady-state conditions. During the tests, the method finds the expected Ψ_{ref}, but it exhibits numerous inconsistencies and noise, which the system reacts to. This results in the system having difficulties reaching the true $\Psi_{ref} = \gamma \pm 90°$. Oscillations between the true Ψ_{ref} and references generated by the noise, increasing the time until stability.

Figure 8.9 shows the motor velocities and references for the tests of pitch swings at wind speeds of 10 $\frac{m}{s}$, with and without payload mass. From the figure, it can be seen that if a lot of zero-crossing events occur, the motors have difficulties following the references and are turned off for a significant amount of time. Then the payload mass is added, the system slows down, and the motors can better follow their references, although there are still problems due to zero-crossing events.

8.4 Simulation results

This section presents the results of the simulated system, which aims to implement as many features of the real system as possible. The simulation features the motor zero crossing shut-off implemented, along with dead zones and disabling α when the *psi*-error is too large. The EKF has also been implemented, but model constraints do not allow for calculating ψ_{ref} the same way as the EKF calculates the Euler angles from the global reference frame, resulting in roll measurements directly corresponding to coordinates along the y-axis and the pitch measurements along the x-axis. Changes in yaw, therefore, do not change the values of roll or pitch. This limiting factor is that the rotation of the reference vectors has to be calculated using an XYZ rotation, as the EKF also does. The inputs for the XYZ rotation are extracted from the ZYZ rotation, which the model relies on, by the Matlab function "rotm2eul(ZYZ, "XYZ")", which converts the ZYZ rotation matrix to the corresponding Euler angles in the XYZ frame. From Section 8.3, where the SPS system was tested, it was concluded that one of the weaknesses of the system is the method for calculating Ψ_{ref}. The simulation results will therefore extract the available data already present in the simulation to demonstrate how the system behaves with an ideal, calculated Ψ_{ref}. The fact that the swing plane angles γ are known in the simulation is used for calculating Ψ_{ref}. This can be seen in Equation (8.4).

$$\Psi_{ref} = \gamma + \frac{\pi}{2} \tag{8.4}$$

All the following simulation results were obtained using a discrete C-function block in Simulink. The "pitch start - 180deg" test will be replicated in simulation for comparison at 0, 5,7.5, 10 $\frac{m}{s}$ and with and without an additional 10 kg payload. The two simulations aim to replicate the results shown in Figure 8.3.

In Figure 8.11, the results for the simulated pitch swing at 180° are shown. The initial swing angle is set to $\alpha = -30°$. For the simulations 0 $\frac{m}{s}$ and 5 $\frac{m}{s}$, the aerodynamic forces are low enough that the pendulum makes a full swing before the system is stabilised. The 7.5 $\frac{m}{s}$ simulation find an equilibrium just outside the dead zone, which the system then removes when the $\Psi - error$ is entirely within the α thrust zone. For the 10 $\frac{m}{s}$ simulation, an equilibrium is found almost immediately, where oscillations in yaw prevent the system from suppressing the swing angle. The oscillations are self-induced as the thrust changes the swing plane γ, and thereby Ψ_{ref}. Simulations 0 $\frac{m}{s}$ to 7.5 $\frac{m}{s}$ reaches stability in ≈ 5.1 s, and the 10 $\frac{m}{s}$ reaches stability in 13.4 s. From Table 8.1 and 8.2 the pitch tests took $\approx 5-8$ s to reach stability for the tests up to 7.5 $\frac{m}{s}$. From this, it can be seen that the simulation and test results are fairly similar to each other.

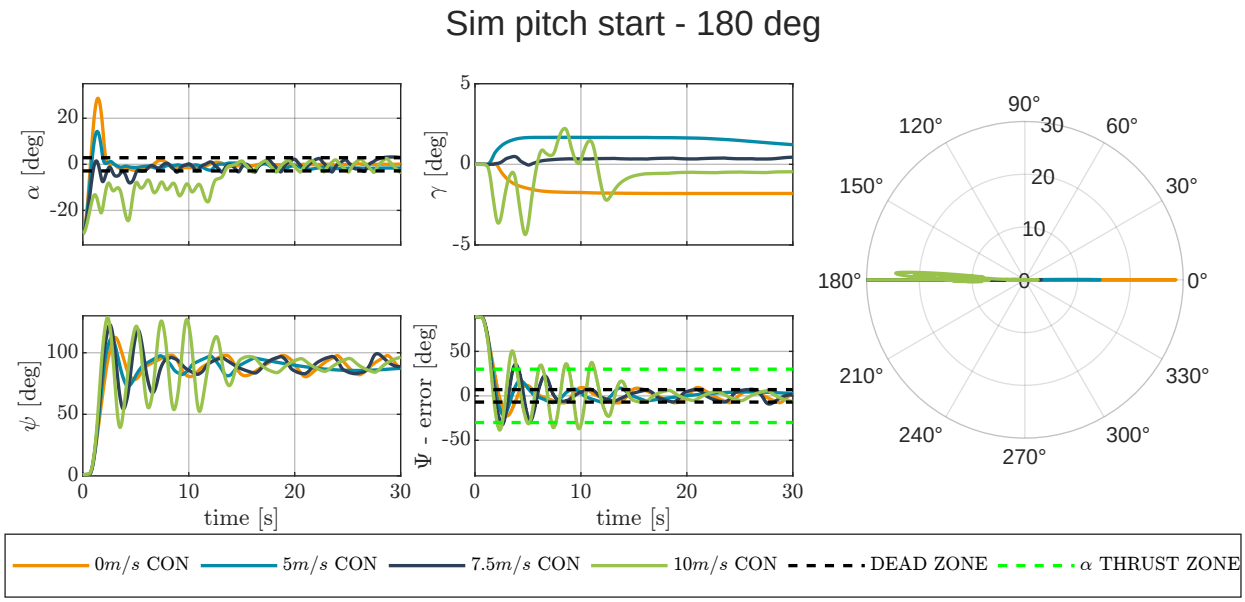


Figure 8.11. Graphs of the simulated SPS system response to an angle disturbance and wind

The following simulation, shown in Figure 8.12, adds an additional mass $m_{extra} = 10.755kg$ to the previous "pitch start - 180deg" test to replicate the results shown in Figure 8.3. When adding the payload to the system, the inertia of the payload has to be added to the SPS. The extra inertia is calculated in Equation (8.5) and is added to J_{SPS} just like the extra mass is added to m_{SPS}. The dimensions of the added payload are measured to be $w = 0.180\ m$ and $d = 0.158\ m$.

$$J_{extra.10} = \frac{1}{12} \cdot m_{extra} \cdot (w^2 + d^2) = 0.0514kg \cdot m^2 \tag{8.5}$$

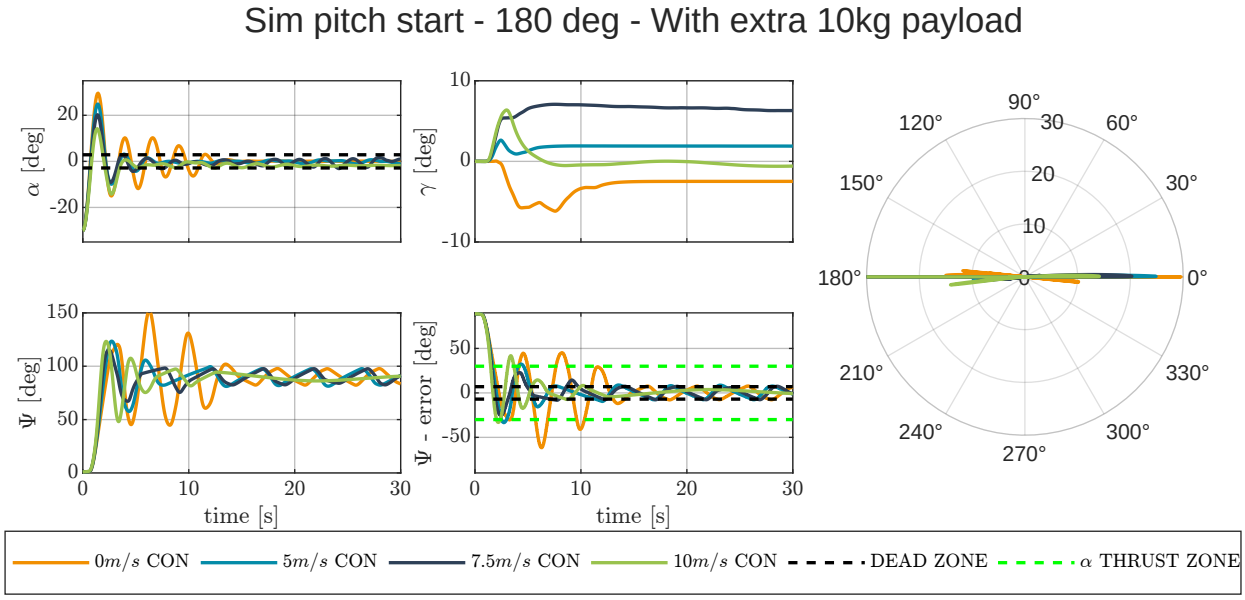


Figure 8.12. Graphs of the simulated SPS system response to an angle disturbance and wind with an extra 10 kg payload

The total inertia of the system is only increased by 4.2% and should therefore not have a significant influence on the control of yaw. The extra mass added to the pendulum adds potential

energy to the system, which is required to be removed by the thrusters. Because the control system is tuned for the system without mass, the energy removed will be removed at the same rate as without any payload, which will increase the time until stability. For simulations at all wind velocities, a full swing can now be observed, where the simulations with wind speeds from 5 $\frac{m}{s}$ to 10 $\frac{m}{s}$, has a time until stability of $\approx 5.3\ s$. For the 0 $\frac{m}{s}$ simulation, oscillations in yaw increase the damping time to 13 s, which matches the previous tests on the system. This also proves that oscillations in yaw increase the time until stability, indicating the importance of Ψ_{ref}.

8.4.1 Extrapolating results

The previous test replicates the results of the SPS system closely enough to extrapolate further results that are not possible to test on the physical system with the available test setup. The simulation results shown in Figure 8.13 increase the wire length to 15 m, and reduce the initial angle to $\alpha = -5°$. The dead zone limits are also decreased to $\alpha_{dead} = 0.5°$ and $\dot{\alpha}_{dead} = 1\ °/s$. The results for wind speed at 0 enter the dead zone once the $\psi - error$ is within the α thrust zone. For the results with wind speeds at 5, 7.5, and 10 $\frac{m}{s}$ all increase in the α angle initially due to the aerodynamic forces, and once yaw is within the α thrust zone, the α angle is dampened. The simulation with 10 $\frac{m}{s}$ continues to have oscillations in yaw, and is therefore not in the α thrust zone for long enough to decrease the α angle. Due to the increased wire length, the moment arms have been drastically increased compared to the previous simulations, which for the 10 $\frac{m}{s}$ simulations results in instabilities in yaw. It can also be seen that γ changes constantly, which results in constant changes for Ψ_{ref}.

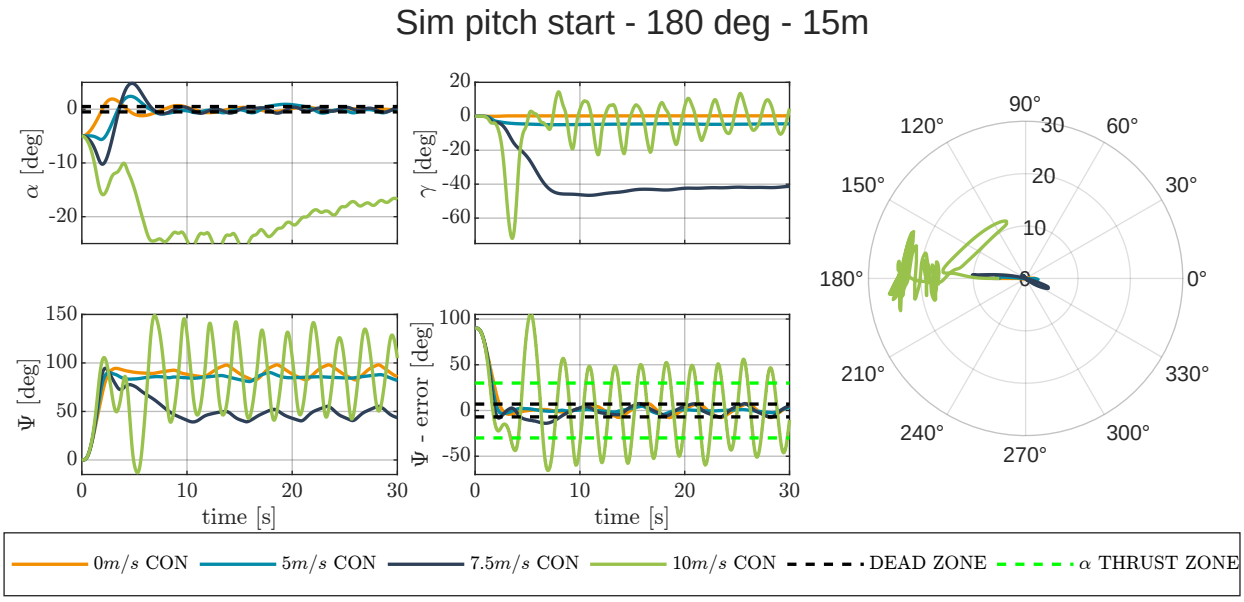


Figure 8.13. Graphs of the simulated SPS system response to an angle disturbance and wind with a wire length of 15 m

Figure 8.14 adds the maximum amount of payload of 25.5 kg to the SPS at the full length of 15 m. The extra mass and inertia are added in the same way as in the "Sim pitch start - 180 deg - With extra 10kg payload" simulation. The inertia of the payload is calculated as a square plate

with width and depth of $w = d = 0.5\ m$, which is the maximum payload size of the SPS system. The inertia is calculated in Equation (8.6), and increases the inertia of the SPS system with 87%. The results of Figure 8.14 show that all the simulations reach stability, it just takes $\approx 60\ s$, which is far beyond the expected settling time. However, the extra mass does not seem to have a significant impact on the stability of the yaw control. From the tests results in Section 8.3, this is the expected bevahior of the system, when the system mass is increases and not proper tuned for it.

$$J_{extra.25} = \frac{1}{12} \cdot m_{extra} \cdot (w^2 + d^2) = 1.0625kg \cdot m^2 \tag{8.6}$$

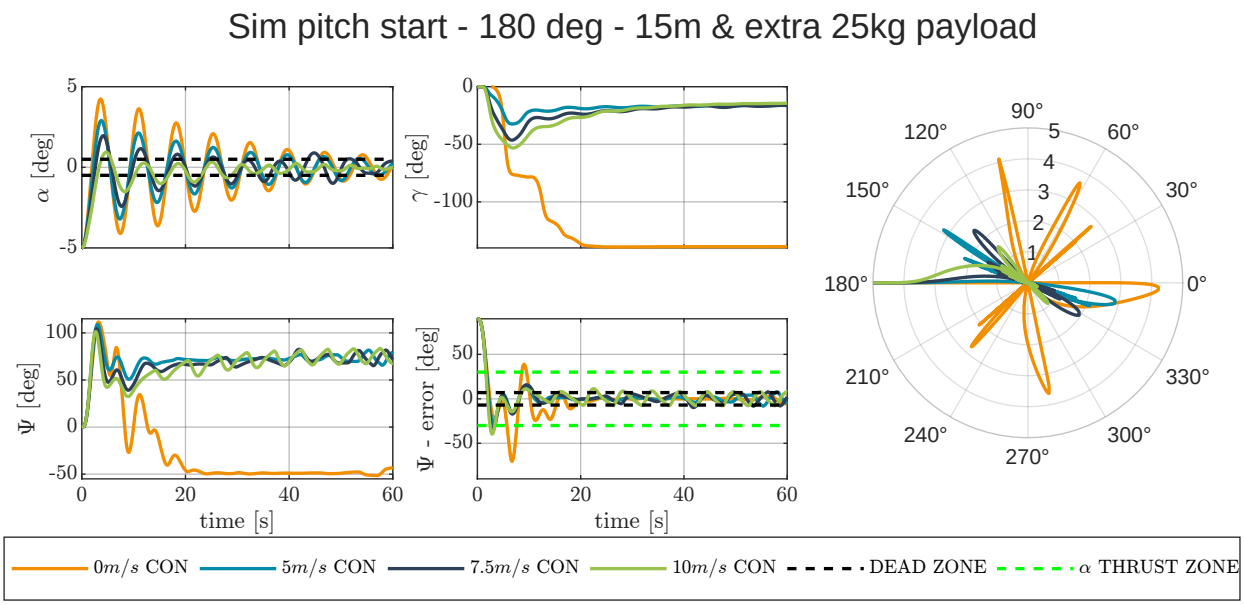


Figure 8.14. Graphs of the simulated SPS system response to an angle disturbance and wind with a wire length of 15 m and a 25 kg payload

In Figure 8.15, the simulated motor velocity output and references for each of the motors are shown. It can be seen that the added mass slows the system further down, and the motors are capable of following their references. The first seconds of each simulations is used for stabilising yaw, which can be seen by the motors running opposite directions. Later all the motors runs in the direction indication the system is being dampned. At the higher wind speeds the systes starts to only push in one direction, to compensate for the wind forces, but the system is still capable of controlling yaw, without having the motors run in each direction in these situations. This is done by altering the thrusts between the motors on each side of the SPS, such a torque is applied.

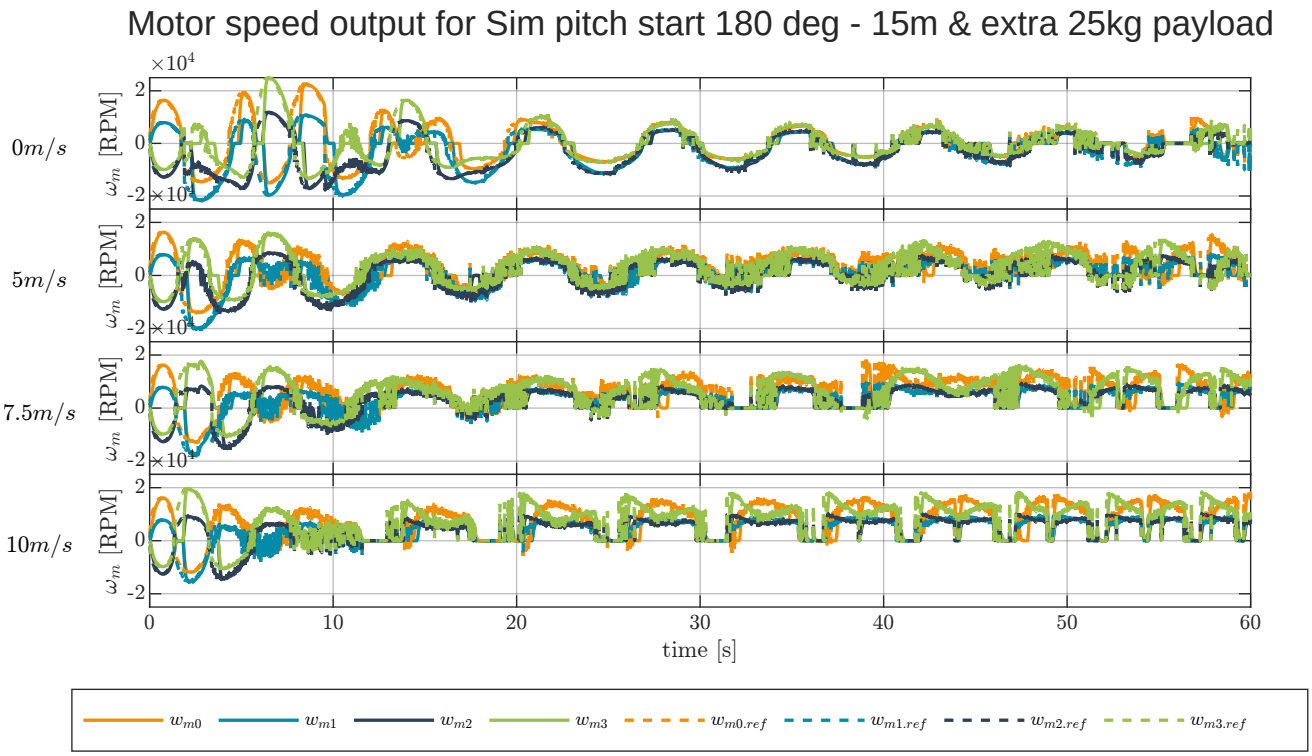


Figure 8.15. Graphs of the simulated SPS system thrust provided by the motors during the "Sim pitch start - 180 deg - With extra 10 kg payload" simulation

The simulations shown in Figure 8.16 increases the wire from 1 m to 15 m at a rate of 0.5 m/s and will reach the max length at $t = 28$ s. This simulation aims to simulate lowering the system during deployment. As this is a critical procedure, the system must be able to control for actual use. The initial angle of the simulation is $\alpha = 0°$, and will therefore show how well the system controls itself from deployment until the full length has been reached. The simulation is conducted without any payload, as previous tests show greater stability when a payload is present. If the SPS system can handle the procedure without any payload, it is expected to be able to handle it with a payload, according to the previous tests and simulations. The simulation is therefore the closest to an actual work scenario for the SPS system, as the system will not turn on with an initial angle as shown in the previous simulations. For the 0 $\frac{m}{s}$ simulation, only a rotation for Ψ occurs due to the method used for calculating Ψ_{ref}, but as there are no external forces, and nothing is required to happen. For the simulations with wind speeds, the SPS system is pushed by the wind, increasing α while Ψ rotates into the α thrust zone. Once α can be dampened, it quickly enters the zone where it stays for the remainder of the simulation.

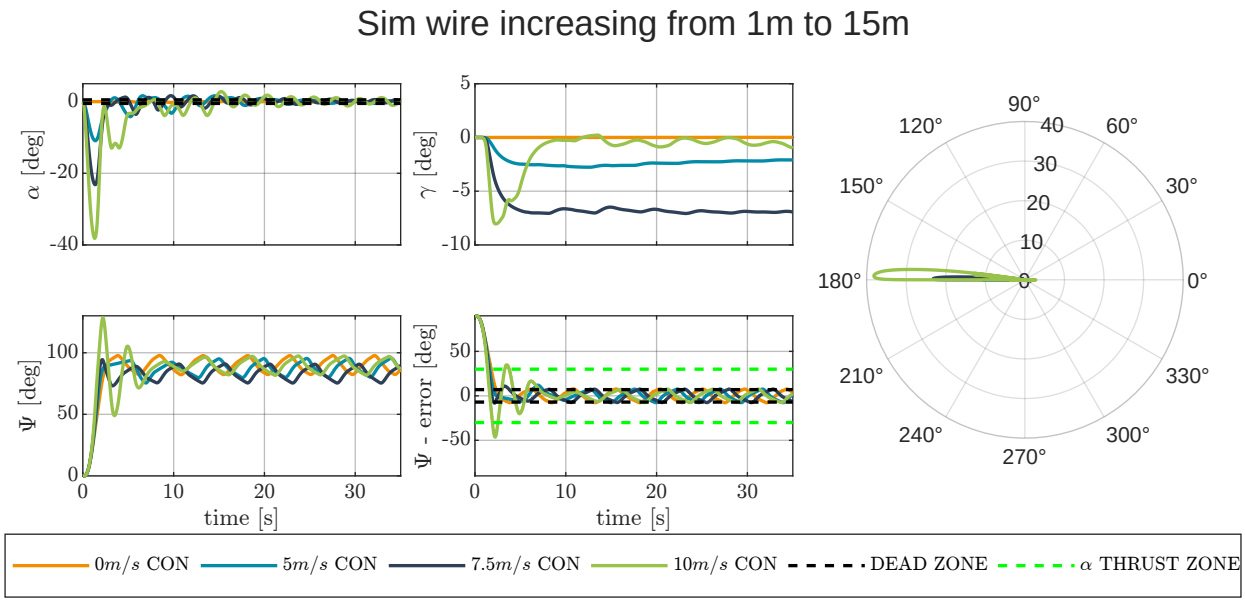


Figure 8.16. Graphs of the simulated SPS system response during the lowering from the drone from 1 m to 15 m

8.4.2 Simulation conclusion

From Figure 8.11 and 8.12 it can be concluded that the simulations replicate the real system under the same circumstances. Although the simulation is slightly faster at dampening, which is expected to be due to an accurate Ψ_{ref} which does have inconsistancies. Although oscillations in γ will restults in oscillations in Ψ_{ref}. Therefore the simulation give an adequete representations of the real system, and it is concluded that it can be used to extrapolate results for a modified system.

When extrapolating results in scenarios where the wire length in increased to 15 m the moment on the pendulum is larger, and the aerodynamic forces has an greater impact. The system is still capable of dampening oscillations, and find the correct yaw angle. The simulations also indicate that adding the full ≈ 25 kg payload, slows down the system greatly, but the system is still capable of dampening the oscillations for all wind speeds. It is not expected that in a real scenerio a swing angle of 5° will be reached, so the long time until stability should not ocour. The SPS system will dampen the system before this happens. When lowerring the SPS from the drone, it is desired that the SPS system can be oriented in the correct heading, before being lowered, to minimise the initial swing, as seen in Figure 8.16.

Discussion 9

This chapter will present certain assumptions, choices, and simplifications made in the project and discuss their impact on the final results. Moreover, it will be discussed whether the SPS system meets the requirements set in Chapter 4.

9.1 Design

In Chapter 4, it was chosen to use Concept 11 instead of Concept 12 as the basis for the design of the SPS system, since only 4 BLDC motors were available for the project. If the SPS system were to have been based on Concept 12, two additional BLDC motors would have been mounted perpendicular to the current motors. This is expected to provide greater control over the SPS system as thrust is produced in all directions, making the system less dependent on the orientation of Ψ. If Concept 12 were used, the control system would need to be modified accordingly. Concept 11 is a more cost-effective solution, while still offering many of the strengths of Concept 12.

The design created from Concept 11 in this report has some flaws, especially regarding the aerodynamic properties. The orientation of the SPS must point in a direction which is relatively unstable aerodynamically, as shown in Figure 5.16. The aerodynamic point is further toward a pitch swing, and therefore, active compensation is required constantly. A design that accounts for aerodynamic properties and incorporates utilities would provide a more stable design, which is expected to improve results. The design of Concept 11 utilises 3D-printed segments of PLA, allowing for further mass reduction optimisation and thereby increasing payload capacity.

Through the tests conducted on the system in Chapter 8, it can be seen that the torque provided in yaw is more than enough. Decreasing the length of the arms would still provide sufficient thrust while reducing mass, space, and drag. The calculation of the provided and necessary torque in yaw was calculated in Section 4.3.1, where the system offers ≈ 3.2 times the required torque.

The evaluation of the design of Concept 11 is shown in Table 9.1. The achieved results, marked in green, symbolise whether the engineering specification has been achieved. If it is marked in red, the engineering specification has not been fulfilled. The engineering specifications marked with yellow are fulfilled, however, further clarifications are required and can be seen below the table.

#	Engineering specification	Min	Max	Acheved results
1	SPS weight	–	5 kg	4.56 kg
2	Payload capacity	25 kg	–	25.44 kg
3	Wind speed	10 $\frac{m}{s}$	–	≈ 10 $\frac{m}{s}$
4	Max swing distance	–	1 m pr. 15 m	0.75 m
5	Landing pad size	–	2 x 2 m	–
6	SPS size	350/350/200 mm	754/472/385 mm	1090/520/370 mm
7	Payload size	450/450/500 mm	–	–
8	Measured attitude accuracy	–	1°	–
9	Control attitude accuracy	–	1°	–
10	Torque about Ψ	8.26 Nm	–	13.85 Nm
11	Exposed hazardous parts	–	0	0
12	Ease of access	0 min	10 min	> 10 min
13	Time between service	120 min	–	–
14	Number of wear parts	–	6	4
15	Number of sensors	1	10	4
16	Time for swapping battery	–	2 min	< 2 min
17	Max wire torsion	–	0 Nm	0 Nm
18	Automatic docking	–	Manual oversight	–
19	Propulsion efficiency	–	–	–
20	Number of motors	4	8	4
21	Operating time	18 min	–	–
22	Time until stability	–	15 s	$\approx 10 - 15$ s

Table 9.1. Engineering specification with boundary values and achieved results

Regarding the engineering specification *"wind speed"*, it was observed through testing in Chapter 8 that the system could dampen the system within the dead zone at 10 $\frac{m}{s}$ only when a payload was added. The engineering specification *"SPS size"* was defined for the DJI FlyCart 30 drone without modified legs. As mentioned in Section 4.3.1, the SPS design can fit within a specific yaw range.

For the engineering specification *"max swing distance"*, the requirement is met, as the SPS system reached the dead zone on the test setup. The dead zone is defined as $\alpha_{dead} = 2.86°$, resulting in a max swing distance of 0.75 m pr 15 m wire. This has not been tested on a full-length wire and cannot be completely verified. However, the simulations also indicate fulfilment of the requirement, where it can reach the dead zone of $\alpha_{dead} = 0.5°$ corresponding to a horizontal displacement of 0.13 m.

The *"time until stability"* engineering requirements are set to be $\approx 10 - 15$ s, where it is observed in testing that there are pretty large fluctuations from test to test. It is also challenging to replicate the tests under the same conditions. This means that there is a degree of randomness in the initial conditions of the tests. The engineering specification *"measured attitude accuracy"* is challenging to determine with the available test equipment. The EKF has a static and dynamic accuracy. The static accuracy is only made for ϕ because the θ is calculated in the same way, so similar results are expected. The expected angles and the estimated angles by the EKF are shown in Table 9.2, where it can be seen that the accuracy is within the requirement of 1°. The

dynamic accuracy requires an extensive setup to measure the angles during movement of the SPS, for comparison with the EKF. This has not been possible, and the dynamic accuracy is therefore unknown. For the Ψ, it was not feasible to conduct an accurate accuracy test, but a rotation of $\approx 90°$ results in a $\approx 90°$ rotation in the EKF.

	ϕ		
Expected angle	5°	10°	15°
EKF	5.3858°	10.4851°	15.6417°

Table 9.2. Static measurement result comparison for validation.

9.2 Model

In Section 5.3.1, the mechanical pendulum model assumes that the drone is stationary. This is assumed because the DJI FlyCart 30 specifies its hovering accuracy at $\pm 10\ cm$, which over 15 m is negligible. The drone's weight is 65 kg and the SPS with payload is 30 kg, and contains $\approx 66\%$ of the total mass. It could therefore be expected that there is a coupling between the drone and pendulum, which is not accounted for in the model. The DJI FlyCart 30 has unknown dynamic characteristics, and therefore, it is not possible to determine how the drone reacts to pendulum disturbances. The drone could be implemented by adding the drone as a mass, with its own set of generalised coordinates. If the influence of the pendulum on the drone is to be understood, the drone can be modelled as a mass-spring-damper system. The spring and damper coefficients will be estimates of the drone dynamics. If only the drone's influence on the pendulum is desired, the drone can be added as a mass, where the generalised coordinates are changed to simulate drone movement.

The implemented mechanical pendulum model has the generalised coordinates α and γ, which are independent. Frictions are added to $\ddot{\gamma}$, as the equation can be generalised to a rotating mass with an inertia that is changed based on the α angle. This means that at small α angles, the forces applied result in astronomically large angular velocities, which are not representative of the real system. The dynamic inertia is removed, and frictions are added for a better fit with the real SPS system.

During testing in Chapter 8, small double pendulum dynamics were observed. This occurred if the SPS was released when the wire and SPS were not perpendicular or in certain instances when wind was applied. These effects were minor and only affected the Ψ_{ref} calculations. When the payload mass was added, the SPS-payload system became a double pendulum system in itself. The complete pendulum can therefore be described as a triple pendulum. The SPS-payload double pendulum system increased the double pendulum dynamics of the SPS, but the system can mostly still be described as a single pendulum. The SPS payload double pendulum dynamics can be reduced by improving the connection between the SPS and the payload.

The motor model assumes a constant viscous propeller friction C_p. The coefficient represents the aerodynamic load torque from the propeller on the motor at a given speed ω_m. C_p is calculated when the motor runs in the positive direction. Still, a different C_p may be present when the

motor rotates in the opposite direction, where the aerodynamic properties of the propeller are different.

9.3 Control

In Section 6.1 and Chapter 8 it can be seen that when the motor velocity crosses zero, the motor stops for $\approx 0.5\ \ s$ to change the motor direction as explained in Chapter 7. These results show that the motors cannot follow references where zero crossings are given at frequencies above $\approx 2\ Hz$. This means small adjustments cannot be made consistently, and a dead zone was therefore necessary to find a stable position. For future design, an ESC with the capabilities of bidirectional motor velocities is desired, such commands are not needed to control motor direction.

The MIMO controller provides a thrust reference, and the motor requires a velocity reference. A thrust-velocity look-up table translates between the references. The thrust-velocity relation is highly non-linear and cannot be effectively linearised. The look-up table is therefore necessary to include the non-linearities in the thrust-velocity relationship.

The swing detection method calculates the swing plane angle from the combination of roll and pitch. This makes the method incredibly sensitive to measurement noise and disturbances from double pendulum effects. A low-pass filter and update restrictions have been implemented to combat the sensitivity of the method. From Chapter 8, the efforts have not been sufficient, and an entirely different method is required. The frequency of the low-pass filter has not been lowered, as this would increase the phase, and the method would become too slow. Increasing the update limit results in no updates at lower angles, and requires the swing angle to exceed the limit before the Ψ_{ref} can be found.

The MIMO controller was tuned by implementing the motor dynamics, but with no payload mass. The controller is not as robust as desired when a payload is added, but it provides sufficient control. Multiple controllers for different load scenarios could be tuned, where gain scheduling could be used to pick the best controller gains for the applied payload mass. A load cell could be implemented on the SPS system to estimate the added mass and inertia, and thereby adjust the controller gain appropriately.

In Section 5.2, the velocity-thrust relation is determined. The relationship has been implemented into look-up tables used for the SISO control in Section 6.1. It is assumed that the thrust-velocity look-up table will provide the SISO controller with the ω_{m-ref}, which provides the thrust T_m specified by the thrust reference T_{m-ref}. This has never been explicitly tested, but it is assumed to be true.

The motor controller slows down the motor dynamics significantly in order to obtain 10 measurements during the rise time. The motor control could be made open-loop, where the relation between thrust and throttle would be used. This approach does not account for the change in the thrust-throttle relation when the battery voltage drops. It can therefore not be ensured that the correct thrust is applied, but a robust MIMO controller could compensate.

It is possible that decoupling the MIMO system dynamics could provide improved control. For decoupling, an RGA analysis is required to determine coupling dynamics, but this requires the system matrices to be square. For the system described in Section 6.2, the B-matrix must be square which can be done by either making the inputs in $u_{aug} = [M_\alpha,\ M_\Psi]^T$ or $u_{aug} = [M_l,\ T_r]^T$. M_α is the swing angle moment, and M_Ψ is the yaw angle moment. T_l and T_r are the left end and right side thrust, respectively. The thrust is then equally distributed between the two motors of each side.

The mechanical pendulum-yaw system has measurements for roll, pitch, and yaw. If Concept 12 were to be used for design, it would be possible to implement a control structure for each of the angles, similarly to how it is done in a quadcopter, where a PID controller is made for roll, pitch, and yaw. The outputs of the three PID controllers are then added together to find the controller signals (Romero, Pozo, and Rosales 2014). A similar approach could be implemented if thrust were available in multiple directions.

Conclusion 10

The purpose of this report has been to design and develop a single wire stabilised payload system in cooperation with Airflight, for implementation on a DJI FlyCart 30. To do so, it is necessary to design a system which meets Airfligts' design specification. The system needs to be modelled so that a controller can be developed, tuned, and implemented. The model is also used to verify design specifications that cannot be ascertained from the test setup.

According to Airflights' design specifications, a single-wire, stabilised payload system has been designed to fit the DJI FlyCart 30 drone and incorporates Airflights' payload mounting system. The design incorporates four BLDC motors in a line, faced in opposite directions for symmetric thrust and torque characteristics. The design, therefore, must face in the swing direction to dampen the system. The design meets the weight criteria by weighing less than 5 kg, but weight optimisation is possible to improve payload capacity. The design incorporates undesired aerodynamic characteristics, which complicates the system's control.

The SPS system model consists of four BLDC motor models, a 3D mechanical pendulum model, and a mechanical yaw model for system orientation. The BLDC motor parameters have been identified for the BLDC motor model and have been verified to fit the real BLDC motor. The BLDC model has been linearised through a Park and Clark transformation to a DC equivalent model, and verified to provide sufficient results. The 3D mechanical pendulum is described through a ZY-rotation, leading to a model described through Lagrangian mechanics, where the generalised coordinates are given as the swing plane angle γ and swing angle α. The mechanical yaw model has been added to the pendulum model as an additional parameter Ψ describing the system orientation. The yaw model provides the thrust direction and is also used to determine the aerodynamic properties of the SPS system. The pendulum and yaw model has been fitted in a roll $\Psi = 90°$ and pitch $\Psi = 0°$ swing, where it is observed that the models provide good estimates of pendulum frequencies and amplitudes over time. The pendulum has been linearised, where the linearisation constants still provide good forecasts of pendulum frequencies and amplitudes over time.

The SPS system incorporates a control algorithm where a MIMO controller provides thrust references T_{m-ref}, where a SISO controller rotates at the required speed to produce the requested thrust. The relation between thrust and motor speed is non-linear and is accounted for by a look-up table. Linear Quadratic Integral augments the MIMO system for LQI control. α and Ψ are controlled with the use of the four BLDC motors. Through LQI optimisation, gains have been tuned for the system with no load to achieve the desired dynamics. The thrust refer-

ences from the MIMO controller are passed to a look-up table to determine the BLDC velocity references ω_{m-ref}. The SISO controller can follow the references well, but when a zero-crossing event occurs, the motors stop for $\approx 0.5\ s$, making bidirectional motor references difficult to follow.

Through testing, it is shown that the system with and without a payload mass of 10 kg stabilises faster than the natural system without any controller. The system stabilises in $\approx 10 - 15\ s$ compared to $70 - 180\ s$ for the natural system without any controller. The limiting factor for stabilisation time is correctly identifying $\Psi_{ref} = \gamma \pm 90°$. The approach employed to ascertain Ψ_{ref} is prone to susceptibility from noise and external disturbances. It is determined that the model provides sufficiently accurate results for extrapolation, which could not be tested on the test setup. From this, it was also determined that the SPS system will work with a wire length of 15 m and with a full payload mass of $\approx 25\ kg$.

In conclusion, the provided design fulfils the specified requirements from Airflight. Limitations remain, and further development is still necessary to achieve a commercially viable product that can be effectively used in real-world scenarios.

10.1 Future work

PCB and addtional features

In the current implementation, the SPS system remains connected to a computer via a USB cable. If the system is to be mounted on a drone, a wireless connection is required to send commands to the system. The Arduino Nano 33 IoT features Wi-Fi and Bluetooth capabilities, which can be utilised on an unused command byte that has been configured for this purpose. In a commercially viable product, a custom PCB would be preferable to simplify assembly and remove wires. On the PCB, a microcontroller, sensors, wireless communication, and possibly motor control could be combined. The motor controller should allow for bidirectional motor velocities without turning off the motors at zero-crossing events. The locking mechanism features an electronic input for opening the lock, which, when implemented, would enable automated pick-up and drop-off of the payload.

Alternative aproche to find Ψ_{ref}

A better method for finding Ψ_{ref} is required. A proposed method is to utilise an optical flow sensor, which can uniquely identify objects and track their movement between frames (Wright n.d.). This allows for determining the movement direction, which can be used to identify the swing plane, thereby defining an appropriate Ψ_{ref}. An optical flow sensor is typically a camera, where software is used to determine the direction of pixel flow. A limitation of the camera is visibility, as fog can render the system useless. Instead, LiDAR could be used with a similar approach to track movement direction, where visibility is less of an issue, but requires more computational power. The flow sensor could be mounted on the SPS system, tracking its movement relative to the ground, or on the drone, tracking the movement of the SPS system.

Bibliography

3deksperten.dk (2025). *3DE Premium - PLA*. Seen the 03-03-2025. URL: https://3deksperten.dk/products/pirate-black-3de-premium-pla-1-75mm.

A., Bemporad (2010). *Integral action in state feedback control*. Seen the 26/05/2025. URL: https://cse.lab.imtlucca.it/~bemporad/teaching/ac/pdf/08-integral-action.pdf.

AHRS (2024). *Complementary Filter*. Seen the 11/09/2024. URL: https://ahrs.readthedocs.io/en/latest/filters/complementary.html.

– (2025). *Extended Kalman Filter*. Seen the 17/03/2025. URL: https://ahrs.readthedocs.io/en/latest/filters/ekf.html.

Airflight (2023). *The Flying Crane Revolution*. Seen the 23/09/2024. URL: https://www.airflight.io/.

Arduino.com (Feb. 2022). *Arduino Nano 33 IoT*. Seen the 27/05/2025. URL: https://akademi.robolinkmarket.com/Data/EditorFiles/PDF/arduino-iot-datasheet.pdf.

ASHISH, P.R and G VINCENT (2014). "SENSORLESS CONTROL OF BLDC MOTOR USING THIRD HARMONIC BACK EMF". In: *International Journal of Industrial Electronics and Electrical Engineering* 2. ISSN: 2347-6982. URL: https://iraj.in/journal/journal_file/journal_pdf/11-61-140308337535-38.pdf.

Bech, Michael M. (2022). *A practical introduction to discrete-time control systems*. Aalborg University.

Benson, Tom (n.d.). *The Drag Equation*. Seen the 17/09/2024. URL: https://www.grc.nasa.gov/www/k-12/VirtualAero/BottleRocket/airplane/drageq.html.

Bobek, Viktor (2013). "PMSM Electrical Parameters Measurement". In: *Freescale Semiconductor* 1, p. 16.

Cao, Huiliang (2023). "Silicon Based MEMS Gyroscope Structure and Working Principle". eng. In: *Dual-Mass Linear Vibration Silicon-Based MEMS Gyroscope*. Singapore: Springer Singapore Pte. Limited, pp. 21–47. ISBN: 9789811992469.

CERTEX.dk (2025). *Rustfri stålwire ROPETEX S19*. Seen the 27-02-2025. URL: https://www.certex.dk/lofteudstyr/stalwire-og-tilbehor/stalwire/rustfri-stalwire/rustfri-stalwire-ropetex-s19-p895391?categoryId=490600#.

Chaplin C.R. Rebel G., Ridge I.M.L. (2003). "Tension-torsion fatigue effects in wire ropes". In: *OIPEEC*. URL: http://www.bgisl.com/RRR_OE_Twist_article_OIPEEC_1999.pdf.

Connectivity, TE (2017). *Choosing the right type of accelerometer*. Seen the 30-09-2024. URL: https://www.mouser.com/pdfdocs/choosing-the-right-accelerometer-white-paper.pdf.

DJI (n.d.). *DJI FlyCart 30 specs*. Seen the 5/9/2024. URL: https://www.dji.com/dk/flycart-30/specs.

Doornik, Gerrit van (2023). *Cranes cost 50% of wind turbine maintenance expenses. Here's how to reduce that*. Seen the 27-09-2024. URL: https://www.windpowerengineering.com/cranes-cost-50-of-wind-turbine-maintenance-expenses-heres-how-to-reduce-that/.

Dr. Arar, Steve (Dec. 2021). *Introduction to Capacitive Accelerometers: Measuring Acceleration with Capacitive Sensing*. Seen the 9/9/2024. URL: https://www.allaboutcircuits.com/technical-articles/introduction-to-capacitive-accelerometer-measure-acceleration-capacitive-sensing/.

Embedded, Wala (2025). *What is Direct Memory Access (DMA)?* Seen the 27/05/2025. URL: https://embeddedwala.com/Blogs/embeddedsystem/what-is-dma.

Emetor (2024). *Brushless DC machines*. Seen the 17-12-2024. URL: https://www.emetor.com/glossary/brushless-dc-machine/.

F. Landis Markley, John L. Crassidis (2019). *Fundamentals of Spacecraft Attitude Determination and Control*. Springer, p. 486. ISBN: 978-1-4939-0801-1.

Ghanipoor, Farhad, Mojtaba Hashemi, and Hassan Salarieh (2020). "Toward Calibration of Low-Precision MEMS IMU Using a Nonlinear Model and TUKF". In: *IEEE Sensors Journal* 20.8, pp. 4131–4138. DOI: 10.1109/JSEN.2019.2963538.

GWEC (Mar. 2024). *GWEC | Global Wind Report 2024*. Seen the 3/9/2024. URL: https://gwec.net/wp-content/uploads/2024/04/GWR-2024_digital-version_final-1.pdf.

Iqbal, Kamran (May 2025). *Linearization of Nonlinear Models*. Seen the 23/05/2025. URL: https://eng.libretexts.org/Bookshelves/Industrial_and_Systems_Engineering/Introduction_to_Control_Systems_(Iqbal)/01%5C%3A_Mathematical_Models_of_Physical_Systems/1.07%5C%3A_Linearization_of_Nonlinear_Models.

Jensen, Anna Lyhne and Jakob Hærvig (2014). "Extending the Existing Modelling Framework for Non-Spherical Particles to Include Flat Plates in Free Fall - An Experimental and Numerical Investigation of the Unsteady Aerodynamics of Flat Plates". In: p. 154. DOI: https://vbn.aau.dk/ws/portalfiles/portal/214692689/masterThesis.pdf.

Kim, Sang-Hoon (2017). *Electric Motor Control DC, AC, and BLDC Motors*. 428th ed. Elsevier Inc., p. 747. ISBN: 978-0-12-812138-2.

KISS, ESC (2025). *KISS ESC 32-bit series onewire telemetry protocol*. Seen the 27/05/2025. URL: http://ultraesc.de/downloads/KISS_telemetry_protocol.pdf.

Kumar, Sumeet (2021). *Momentum Theory*. Seen the 24/09/2024. URL: https://kumar-sumeet.github.io/HeliAeroNotes/Lectures/1_MT/MT.html.

Landa, Chris (2021). *DSHOT - the missing Handbook*. Seen the 15-01-2025. URL: https://brushlesswhoop.com/dshot-and-bidirectional-dshot/.

Lee, Tae-Yong, Myung-Ki Seo, et al. (June 2015). "Design and Torque Ripple Analysis of Brushless DC Motor According to Delta Winding Connection". In: *Journal of Magnetics* 20, pp. 166–175. ISSN: 2233-6656. DOI: 10.4283/JMAG.2015.20.2.166.

Lee, Tae-Yong, Jun-Young Song, et al. (Nov. 2014). "Phase Advance Control to Reduce Torque Ripple of Brush-less DC Motor According to Winding Connection, Wye and Delta". In: *Journal of Electrical Engineering and Technology* 9, pp. 2201–2208. ISSN: 2093-7423. DOI: 10.5370/JEET.2014.9.6.2201.

Magdy, Khaled (2025). *STM32 Timer Interrupt HAL Example – Timer Mode LAB*. Seen the 27/05/2025. URL: https://deepbluembedded.com/stm32-timer-interrupt-hal-example-timer-mode-lab/.

Mandø, Matthias and Lasse Rosendahl (2010). "On the motion of non-spherical particles at high Reynolds number". In: *Powder Technology* 202.1, pp. 1–13. ISSN: 0032-5910. DOI: https://doi.org/10.1016/j.powtec.2010.05.001. URL: https://www.sciencedirect.com/science/article/pii/S0032591010002391.

Mathworks (Mar. 2017). *An Optimal State Estimator | Understanding Kalman Filters, Part 3*. Seen the 21/05/2025. URL: https://se.mathworks.com/videos/understanding-kalman-filters-part-3-optimal-state-estimator--1490710645421.html.

MPrograms (2023). *QMC5883L Compass Arduino Library*. Seen the 12-02-2025. URL: https://github.com/mprograms/QMC5883LCompass.

Munsen, Bruce R. et al. (2013). *Fundamentals Of Fluid Mechanics*. 7th ed. John Wiley & Sons, Inc., p. 747. ISBN: 978-1-118-11613-5.

NASA (2021). *Welcome to the Beginner's Guide to Propulsion*. Seen the 30-09-2024. URL: https://www.grc.nasa.gov/WWW/k-12/airplane/bgp.html#:~:text=Different%20propulsion%20systems%20generate%20thrust%20in%20slightly%20different.

– (2024). *Air Mass/Density*. Seen the 25/09/2024. URL: https://www.earthdata.nasa.gov/topics/atmosphere/atmospheric-pressure/air-mass-density.

Omega (n.d.). *How to Measure Acceleration*. Seen the 25/09/2024. URL: https://www.omega.com/en-us/resources/accelerometers.

Ozyagcilar, Talat (Nov. 2015). "Calibrating an eCompass in the Presence of Hard- and Soft-Iron Interference". In: *Freescale Semiconductor*, pp. 1–18. URL: https://www.nxp.com/docs/en/application-note/AN4246.pdf.

Passaro, Vittorio M. N. et al. (2017). "Gyroscope Technology and Applications: A Review in the Industrial Perspective". In: *Sensors* 17.10. ISSN: 1424-8220. DOI: 10.3390/s17102284. URL: https://www.mdpi.com/1424-8220/17/10/2284.

Ren, Zhengru et al. (2021). "Offshore wind turbine operations and maintenance: A state-of-the-art review". In: *Renewable and Sustainable Energy Reviews* 144, p. 110886. ISSN: 1364-0321. DOI: https://doi.org/10.1016/j.rser.2021.110886. URL: https://www.sciencedirect.com/science/article/pii/S1364032121001805.

Romero, Luis E., David F. Pozo, and Jorge A. Rosales (2014). "Quadcopter stabilization by using PID controllers". In: *I+D+ingeniería 2014*.

S, Skogestad. and Postlethwaite I. (2005). *MULTIVARIABLE FEEDBACK CONTROLL Analysis and Design*. John Wiley & Sons, Ltd. Chap. 3.

Scherr, Chris et al. (2014). *Underwater Navigation System*. Seen the 20-01-2025. URL: https://ece.montana.edu/seniordesign/archive/SP14/UnderwaterNavigation/kalman_filter.html.

sciencefacts.net (Sept. 2023). *Simple Pendulum*. Seen the 23/05/2025. URL: https://www.sciencefacts.net/simple-pendulum.html#:~:text=Students%5C%20are%5C%20often%5C%20asked%5C%20to%5C%20evaluate%5C%20the%5C%20value,the%5C%20string%5C%2C%5C%20i.e.%5C%2C%5C%20T%5C%20%5C%3D%5C%20mg%5C%20cos%5C%20%5C%CE%5C%B8..

Seok, Eunhye (2021). *stm32$_h$al$_d$shot*. Seen the 26/05/2025. URL: https://github.com/mokhwasomssi/stm32_hal_dshot.

Siebeneicher, H. (Sept. 2024). *Universal Asynchronous Receiver-Transmitter (UART)*. Seen the 27/05/2025. URL: https://docs.arduino.cc/learn/communication/uart/.

Sparkfun (2025). *I2C*. Seen the 19/03/2025. URL: https://learn.sparkfun.com/tutorials/i2c/all.

Sreevidya, P.V. et al. (2018). "Development of two axes magnetometer for navigation applications". In: *Journal of Magnetism and Magnetic Materials* 448. International Conference on Magnetic Materials and Applications – ICMAGMA2017, pp. 298–302. ISSN: 0304-8853. DOI: https://doi.org/10.1016/j.jmmm.2017.08.064. URL: https://www.sciencedirect.com/science/article/pii/S0304885317304754.

ST.com (Mar. 2020). *UM1724 User manual*. Seen the 27/05/2025. URL: https://www.st.com/en/evaluation-tools/nucleo-f072rb.html#documentation.

– (Mar. 2021). *RM0390 Reference manual*. Seen the 27/05/2025. URL: https://www.st.com/en/microcontrollers-microprocessors/stm32f446/documentation.html.

STMicroelectronics (Sept. 2024). *Getting started with I2C*. Seen the 19/03/2025. URL: https://wiki.st.com/stm32mcu/wiki/Getting_started_with_I2C.

Tedrake, Russ (2023). *Underactuated Robotics. Algorithms for Walking, Running, Swimming, Flying, and Manipulation*. MIT. Chap. 8. URL: https://underactuated.csail.mit.edu.

Tillisch, Per (2024). *Arduino$_L$SM6DS3Library for Arduino*. Seen the 12-02-2025. URL: https://github.com/arduino-libraries/Arduino_LSM6DS3?tab=readme-ov-file#arduino_lsm6ds3-library-for-arduino.

Titterton, D. H. and J. L. Weston (2004). *Strapdown inertial navigation technology*. eng. 2. ed. Progress in Astronautics & Aeronautics S. ; No. 207. Reston: American Institute of Aeronautics and Astronautics. ISBN: 1563476932.

TMOTOR (2024). *F40PRO IV Fpv Racing Drone Motor 4-6S KV1950/KV2400*. Seen the 14/10/2024. URL: https://store.tmotor.com/product/f40pro-4-fpv-motor.html.

tomstewart89 (2024). *Basic Linear Algebra*. Seen the 12-02-2025. URL: https://github.com/tomstewart89/BasicLinearAlgebra.

Triantafyllou Michael S.; Hover, Franz S. (2003). *MANEUVERING AND CONTROL OF MARINE VEHICLES*. MIT. Chap. 19. URL: https://ocw.mit.edu/courses/2-154-maneuvering-and-control-of-surface-and-underwater-vehicles-13-49-fall-2004/2609f8b1bdebe4566891080afe007f51_1349_notes.pdf.

Ullman, David G. (2018). *The Mechanical Design Process*. 6th ed. David G. Ullman. ISBN: 978-0-9993578-0-4.

Ung, Hee Lee et al. (Mar. 2015). "How to Model Brushless Electric Motors for the Design of Lightweight Robotic Systems". In: *JOURNAL OF LATEX CLASS FILES VOL. 14, NO. 8,* URL: https://www.bing.com/ck/a?!&&p=d7d0a38b1048ebdae992a0f9f106fdcaa31251ed98f6aa4692d7a200ptn=3&ver=2&hsh=4&fclid=24aa6467-4b8a-696c-29e6-76904af6684f&psq=BLDC+motor+model+dq&u=a1aHR0cHM6Ly9hcnhpdi5vcmcvcGRmLzIzMTAuMDAwODA&ntb=1.

vectornav.com (2025). *Attitude and Heading Reference System (AHRS)*. Seen the 13-03-2025. URL: https://www.vectornav.com/resources/inertial-navigation-primer/theory-of-operation/theory-ahrs.

W., Baker D. and Haynes W. (May 2025). *Engineering Statics: Open and Interactive*. Seen the 23/05/2025. URL: https://engineeringstatics.org/Chapter_04-moments-in-three-dimensions.html.

Wang, Peinan, Zhiqiang Wang, and Qing Zhu (Mar. 2017). "Attitude Estimation Based on Modified Two-stage EKF". In: *Proceedings of the 2017 2nd International Conference on Modelling, Simulation and Applied Mathematics (MSAM2017)*. Atlantis Press, pp. 134–138. ISBN: 978-94-6252-324-1. DOI: 10.2991/msam-17.2017.30. URL: https://doi.org/10.2991/msam-17.2017.30.

Wondosen, Assefinew et al. (2021). "Improved Attitude and Heading Accuracy with Double Quaternion Parameters Estimation and Magnetic Disturbance Rejection". In: *Sensors* 21.16. ISSN: 1424-8220. DOI: 10.3390/s21165475. URL: https://www.mdpi.com/1424-8220/21/16/5475.

Wright, Stephen (n.d.). *Optical Flow Sensors*. Seen the 27/05/2025. URL: https://stevetheengineer.com/optical-flow/.

Appendices

Aero coefficents & Forces A

A.1 Regression of drag coefficent

The table from (Munsen et al. 2013, p. 529) used for the regression analysis is also shown in Table A.1

$\frac{h}{d}$	C_D
≤ 0.1	1.9
0.5	2.5
0.65	2.9
1	2.2
2	1.6
3	1.3

Table A.1. Table from (Munsen et al. 2013, p. 529) showing the ratio between h and d and the corresponding C_D.

Matlab's built-in curve-fitting tool was used for the regression analysis. The x-data is the ratio, and the y-data is C_D. The fit on the data was determined to be a two-term Gaussian curve. The curve-fitter tools calculated the R-squared value of the fit to be $R^2 = 1$. The fit is estimated to fit perfectly, but this is most likely not the case because of the lack of data points. However, it is deemed sufficient for this estimation.

A.2 The drag coefficients reliance on α

In Figure A.1, the drag coefficient and drag force can be seen under a constant $\Psi = 0°$ but with a change in α. The shape ratio for $C_{D.0}$ has been calculated as $\frac{l}{h} = 1.95$ and for $C_{D.90}$ has been calculated as $\frac{h}{l} = 0.51$. The drag coefficients are thereby calculated to be $C_{D.0} = 1.61$ and $C_{D.90} = 2.52$.

It can be seen that the drag coefficient C_D only changes by 0.11 for Rosendalhs and 0.23 with Presavento when changing α from 0° to 30°. Including the projected area, the change will only result in a force change of $26 - 30\ N$. It is therefore concluded that it is a valid assumption to assume the aerodynamic forces are calculated at $\alpha = 0°$, as the change in the drag coefficients is small. An angle of $\alpha = 30°$ will not occur under ordinary circumstances.

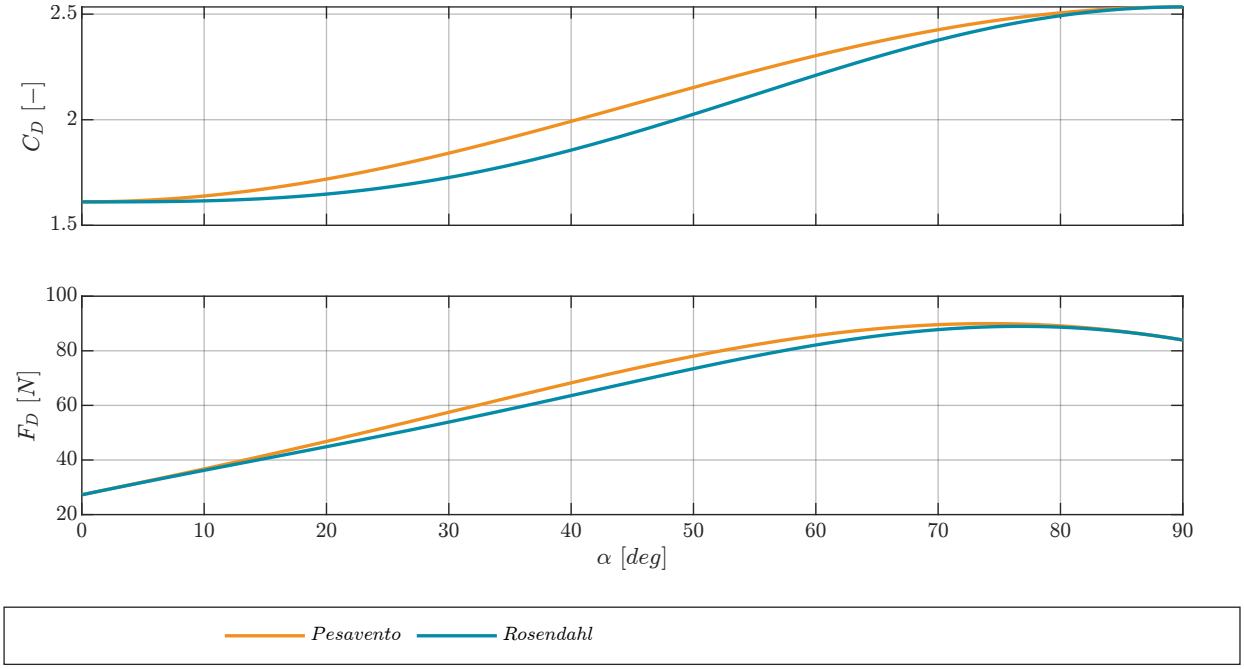


Figure A.1. The drag coefficient and drag force when $\Psi = 0°$ and the α angle is changed

A.3 Lift based on angle

Figure A.2 shows the lift coefficient and lift forces. Note that between 0-90 and 180-270 degrees, the sign of the force is negative, since the high pressure would be created on the top of the box and the low pressure on the bottom.

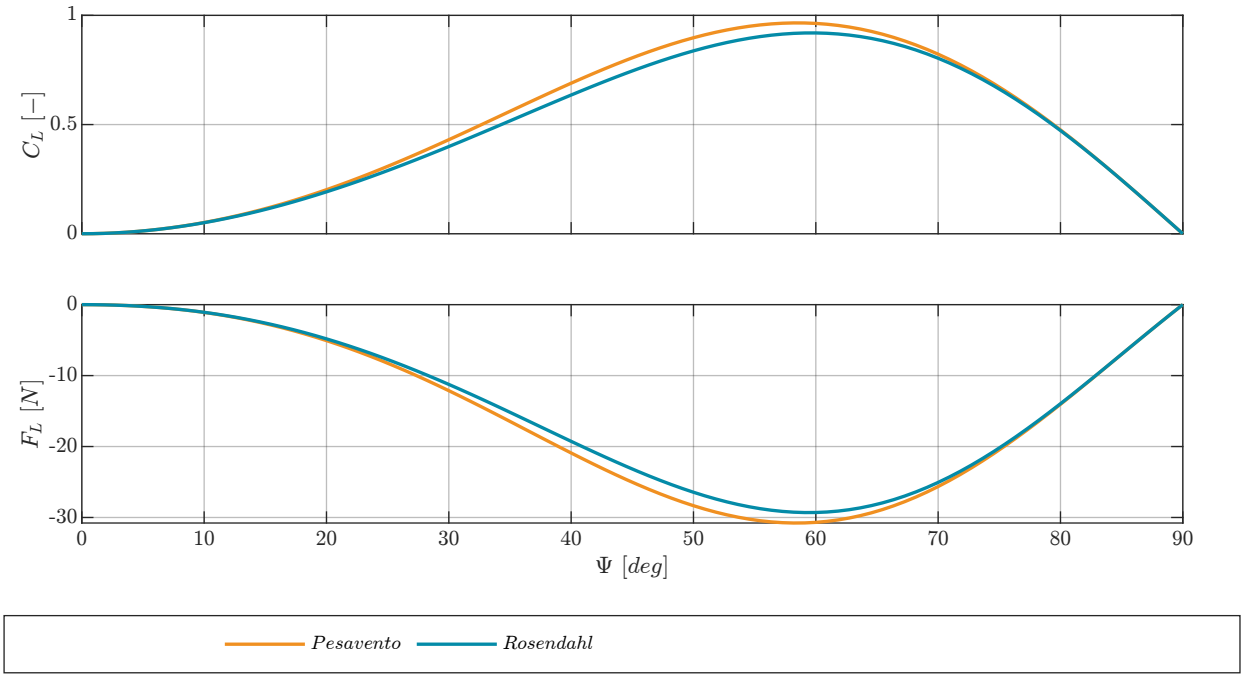


Figure A.2. Lift coefficient and force can be seen as a function of the angle of attack ψ

A.4 Recalculations for drag and lift

This section will recalculate the previous calculations for drag and lift for the prototype design using the fitted drag coefficient for $C_{D.0}$ and $C_{D.90}$. The drag coefficient for the payload is also estimated at the ratio $\frac{0.5}{0.45} = 1.11$, resulting in a $C_{D.0} = C_{D.90} = 1.95$. In Figure A.3, the drag coefficient and drag forces for the SPS system and the payload are plotted during a $0 - 90°$ rotation in Γ. The sum of the drag forces produced by the aerodynamic forces on the SPS system and payload is also plotted.

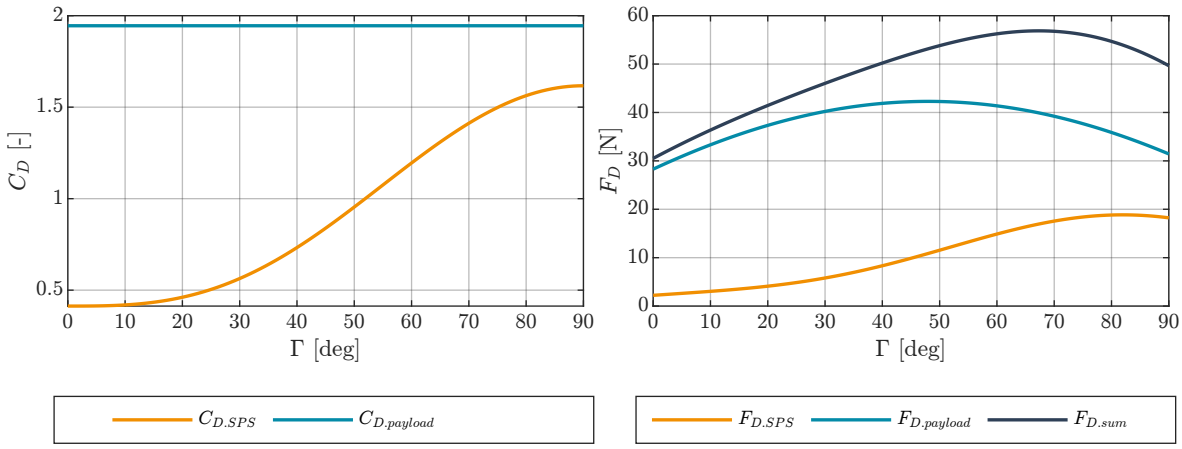


Figure A.3. Drag coefficient and drag force for the SPS system and payload during an $0 - 90°$ rotation in Γ.

Figure A.4 calculates the lift coefficients and lift force for the SPS system and payload during a $0 - 90°$ rotation in Γ. The sum of the lift forces produced by the aerodynamic forces is also plotted.

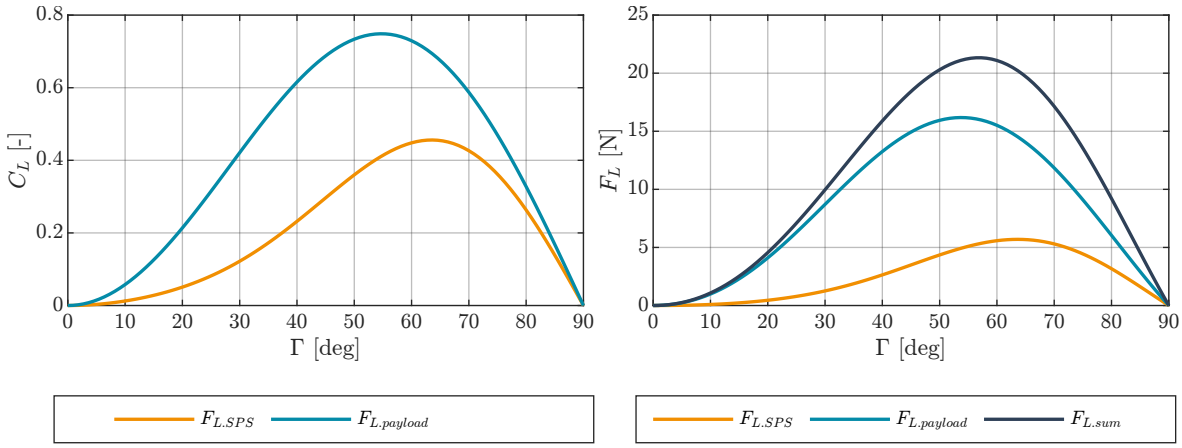


Figure A.4. Lift coefficient and lift force for the SPS system and payload during an $0 - 90°$ angle of attack.

Figure A.5 shows the torque produced by the drag and lift forces during a $0 - 90°$ rotation in Γ on the SPS system and the payload. The figure also shows the sum of aerodynamic torque on the SPS system and the payload.

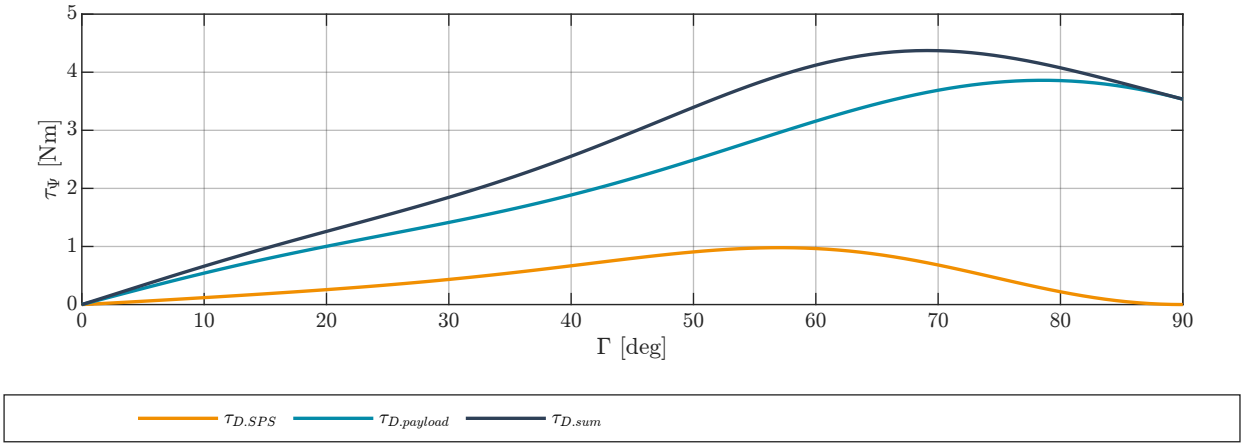


Figure A.5. Torques produced in the SPS system, payload, and the total sum during a $0-90^\circ$ rotation in Γ

Requeriments and Engineering specfications B

In the following appendix, the requirements and engineering specifications will be elaborated on, and the house of quality will be used to weight the engineering specifications, which are elaborated upon in the following appendix.

B.1 Requeriments

To design a stabilised payload system as mentioned in Chapter 1, the requirements for the system must be specified. These requirements are based on the manufacturer and the customer's requirements for the system. The system requirements are specified in Table 4.1 and are elaborated further in the following:

Damp pendlum osilations:
The SPS is exposed to external forces that can induce oscillations. Therefore, the SPS is required to dampen or even stop the oscillations caused by the external forces that affect the pendulum system.

Land on target:
The SPS is made for transporting payloads. The landing area to which the SPS needs to be able to deliver the payload can be limited, especially if the payload is to be delivered to the nacelle of a wind turbine. The drone is also not expected to have absolute position control, so the SPS is required to adjust the payload's position to some degree. The SPS should also be small enough to land on a wind turbine nacelle.

Messure attitude:
The SPS needs to measure its attitude to sense deflections caused by disturbances and control the position. The measured attitude can be used to analyse the SPS's position in relation to the drone. It also gives feedback for damping and position control.

Control position:
To fulfil the requirement *"land on target"*, it is required to be able to control the position of the SPS. The SPS, therefore, needs some level of control over its position with a certain amount of precision and accuracy.

Maintanence:
The SPS will be designed to operate outside, exposed to the environment, and transported between locations. The system should not contain unnecessary wear parts and should protect

electronics and wear parts from any environment to which it is exposed. Maintenance should be rare, but it should be possible at the location.

Minimal input:
The SPS should be as autonomous as possible so that it can be operated with minimal input and potentially by personnel with only introductory training.

Motor position:
On the SPS, the motors have to be positioned so that they provide enough thrust and momentum to remove oscillations and allow the system to land on target with the specified maximum wind speeds. The motors are positioned so that they provide efficient thrust, so a minimal number of motors can be used.

Compatebility:
To broaden the potential customer base for the SPS system, it has to be compatible with as many drone systems as possible. Airflight specifies that the prototype must be compatible with the DJI FlyCart 30, but compatibility with other drones has also been considered during the evaluation of this requirement.

Maximize Payload capacity:
The weight of the SPS has to be kept as low as possible since the SPS mass is part of the drone's total payload capacity. The lower the system weight, the higher the payload capacity.

Fit within DJI FlyCart 30 limitations:
The SPS should be designed so that the system fits the limitations of the DJI FlyCart 30. The specifications of the DJI FlyCart 30 can be found in Section 2.7.

Price:
The SPS's price is to be kept as low as possible so that either the manufacturer can earn more money per unit or the customer might get a cheaper product.

Safety:
The SPS should be safe to use so that possible dangers are minimised. This is done by closing off any moving parts and exposed electronics. But the danger of a 30 kg pendulum persists.

B.2 Engineering specfications

The engineering specifications are designed to satisfy the requirements from Appendix Section B.1. The following section describes the engineering specifications and the corresponding values given.

SPS weight:
The SPS system's weight is intended to be as low as possible so that more of the drone's available carrying capacity can be used on the payload. From Airflight, a maximum weight has been specified for the SPS system of 5 kg.

Payload capacity:
The payload capacity should be as large as possible. The carrying capacity of the DJI FlyCart

30 is 30 kg, so with a maximum weight of 5 kg given to the SPS, the minimum payload capacity is 25 kg.

Wind speed:
The SPS system should work within the DJI FlyCart 30's wind limitations given at 12 $\frac{m}{s}$. Airflight specifies the maximum wind speed as 10 $\frac{m}{s}$. Therefore, the system's total thrust must be large enough to counteract the drag forces.

Max swing distance:
From the drag forces of the wind, the SPS will be affected and will swing out as a pendulum. This swing is to be limited. Airflight allows it to swing to a maximum of 1 m horizontal pr 15 m vertical, or approximately 3.8°.

Landing pad size:
From the engineering specification of *"max swing distance"*, the maximum of 1 m horizontal pr 15 m vertical, the maximum landing pad size is 2 x 2 m when the SPS is lowered 15 m below the drone. For payload pickup, the landing area is reduced to the payload size, limiting the maximum wind speed allowed for such missions.

SPS size:
The volume available specified by DJI for the DJI Flycart 30 cargo case is 754/472/485 mm as described in Section 2.7. The drone will be configured with a hoist system and custom legs, increasing the available volume. The available volume is not uniform, allowing the SPS to exceed the constraints when oriented in specific orientations. As a reference, the engineering specification is set to 754/472/485 mm, but it can exceed the dimensions if suitable. Due to the design choice made by Airflight to use a docking cone, the minimum size of the SPS is the size of the docking cone specified at 350/350/200 mm.

Payload size:
The payload is placed under the SPS. Airflight specifies the payload dimensions to be 450/450/500 mm as a minimum, and as a maximum, it is required to fit within the drone limitations.

Measured attitude accuracy:
Better measured attitude accuracy is desired for better control and position estimates. As the desired maximum swing distance is approximately 3.8°, the measured attitude accuracy is specified as 1°. This specification is also attained by the accuracy of the sensors, where a more accurate measurement would require better sensors.

Control attitude accuracy:
When the control system is dampened, the attitude achieved will be obtained with an accuracy of 1°.

Torque about Ψ:
The purpose is to minimise the time to stability, possibly including orienting the SPS system to a specific orientation for optimal dampening. This can be obtained faster with a larger torque. In addition, the SPS system is also required to resist any torque from aerodynamic forces for optimal orientation control. Based on the preliminary calculations in Section 2.1, the required

torque is 8.26 Nm. This specification depends on the final design, as the aerodynamic coefficients are subject to change.

Exposed hazardous parts:
The propellers can rotate with an angular velocity of up to 28000 RPM, which is potentially dangerous to the operator. Large electrical currents are also present during operation, which could be a potential hazard. This is undesirable from the perspective of the customer and the manufacturer and, therefore, the maximum exposed hazardous parts are defined as 0.

Ease of access:
When the SPS needs to be maintained, assessing the components and wear parts for service and replacement should be easy. The number of bolts and steps required should be limited to a minimum while still avoiding exposing the operator and the system to unwanted hazards. Accessing and replacing a component should, at most, take approximately 10 min.

Time between service:
The time between services is specified under normal operation and is wanted to far between. The primary factor for the time between services is the mechanical components, such as the motors and the locking mechanism. An estimate for the minimum time between services is expected to be approximately 120 min of active operation.

Number of wear parts:
The number of parts that can wear down should be limited to increase the time between services. Wear parts are any moving parts, such as motors and locking mechanisms, that wear down over time. From the expected wear parts of the SPS system, the maximum allowed wear parts are specified as 6.

Number of sensors:
Sensors can be expensive, and a balance between the required measurements and price has to be attained. Measurements are required for each feedback loop, and because a feedback loop is necessary for dampening the system, a minimum of 1 sensor is required. If motor control is wanted, a single sensor for each motor is required, plus attitude measurements. A maximum of 10 sensors have been specified for the final design.

Time for swapping battery:
The time the SPS system is inoperable is wasted. Swappable batteries are preferred, as they allow for recharging batteries when SPS is in operation instead of waiting for the batteries to charge. Additionally, it is desired to allow for quick replacements of the battles, which should take less than 2 min.

Max wire torsion:
When the wire holding the box is spun around, torsion can build up in the wire, limiting the rotational freedom of the box. Since the wire might build up torsion, resulting in a moment in the wire. This might also result in another problem, since the wire is likely a steel wire rope, the wire may untwist if a torque is applied to it while it is under a tensile load (Chaplin C.R. 2003). To ensure that this does not happen, it has been desired that no torque should be applied to the rope. Therefore, the maximum wire torsion is set to 0 Nm.

Automatic docking:
For the operation specified in Chapter 1, it is desired that the SPS can pick up remote payloads. The SPS should be capable of automatically picking up the payload when lowered onto it. The maximum limit for this requirement is that it can perform this task under manual supervision, where the operator can start and stop the operation.

Propulsion efficiency:
The engineering specification for propulsion efficiency ensures that all propulsion is used effectively and is not placed in a direction where thrust is not used effectively in either damping or rotation.

Number of motors:
The most expensive parts of the SPS prototype are the motors and the ESC. An ESC is required for each set of four motors. To minimise the total price, the number of motors should be minimised. A minimum of 4 motors has been set to fully utilise a single ESC, and a maximum of 8 motors to limit the total price.

Operating time:
The DJI FlyChart 30 can fly for 18 minutes when fully loaded, according to Table 2.3, and the SPS system should not be the limiting factor for operational time. The minimum operating time is specified as $18min$.

Time until stability:
When the SPS is affected by external forces, it must stabilise within a specific time frame. The longer the time until stability, the longer the payload delivery time, thereby shortening the total operating time. The specified time until stability is given when the SPS is dropped at the maximum swing distance, where it must stabilise within the limit at any Ψ angle. The time until stability is specified as 15 s.

House of Quality

A House of Quality has been set up to determine the weight of the Engineering specifications. The requirements have been given points out of a hundred, depending on how important it is for the group. Two groups have been set up: *"Customer"* and *"Manufacture"*. The point distribution is given by distributing 100 points over the requirements from the perspective of the manufacturer and the customer. The influence of the manufacturer and the customer has also been weighted, where the customer has 60% of the impact on the weighted sum. The points for the manufacturer and the customer have been distributed through a discussion with Airflight.

The engineering specifications are then related to the requirements, where a 9 is given as a strong relation, 3 as a medium relation, 1 as a weak relation, and a black space is no relation. From this, a weighted prioritisation is set up where the sum of the relation is multiplied by the normalised weight applied to the relevant requirement. The House of Quality and the weighted prioritisation can be seen in Figure B.1.

#	Requirements:	Weight Customer	Weight Manufacture	Weight Sum	Weight normalized	Engenering Specifications:	SPS weight [Kg]	Payload capacity [Kg]	Wind speed [m/s]	Max swing distance [m]	Landing pad size [m x m]	SPS size [m x m x m]	Payload size [m x m x m]	Measured attitude accuracy [°]	Control attitude accuracy [°]	Torque about ŷ [Nm]	Exposed hazardous parts [No]	Ease of access [No of bolts]	Time between service [Min]	Number of wear parts [No]	Number of sensors [No]	Time for swapping battery [s]	Max wire torsion [Pa]	Automatic docking	Propulsion efficiency [%]	Number of motors [No]	Operating time [Min]	Time until stability [s]
		0,6	0,4			Desired size:	↓	↑	↑	↓	↓	↓	↑	↓	↓	↓	↑	↓	↓	↓	↓	↓	↓	↓	↑	↑	↑	↓
1	Damp pendlum osilations	11	8	9,8	0,098		3		9	3	9	9	9	9	9	3					1		1		3	9		9
2	Land on target	15	12	13,8	0,138		1		9	9	9	9	9	9	9	3					1		1		3	9		9
3	Messure attitude	4	8	5,6	0,056									9							9							
4	Control position	13	11	12,2	0,122		1	1	9	9	9	9	9	9	9	9							3	3	3	9		9
5	Maintanence	4	3	3,6	0,036		1										3	9	9	9	1	9	3			3		
6	Minimal input	8	6	7,2	0,072									3	9	9							1	9				
7	Motor position	0	6	2,4	0,024				9	9	3	3			9	9	9	1		3					9			
8	Compatebility	5	12	7,8	0,078		3	3	1	1		3	3			1									3		3	
9	Maximize Payload capacity	13	8	11	0,110		9	9	1			1					3										3	
10	Fit within DJI FlyCart30 limitations	10	10	10	0,100		9		3		1	9				3									3		9	3
11	Price	10	7	8,8	0,088		3	1	3			3	1	3	3	1	3	3	9	9	9	3	1	3	3	9	9	
12	Safety	7	9	7,8	0,078				3	3				1	3		9				3		1	3		1		
Sum:		100	100	100	1	Prioritization:	30	14	47	34	31	46	31	43	51	38	27	13	18	21	24	12	11	21	24	43	21	36
						Weighted Prioritization:	2,98	1,43	4,42	3,16	3,39	4,8	3,54	4,28	4,58	3,14	1,62	0,61	1,12	1,19	1,8	0,59	0,95	1,75	1,85	4,53	1,93	3,99

Figure B.1. House of Quality

To gather insight into the relationship between the different engineering specifications, the roof of the House of Quality is seen in B.2. The arrow visualises the relation between the engineering specifications when there is an increase in the numerical value of the column. The arrow indicates the numerical change in the engineering specifications in the row of the table. An upward arrow indicates an increase in the engineering specification in the row and does not indicate a positive or negative change influence on the specifications.

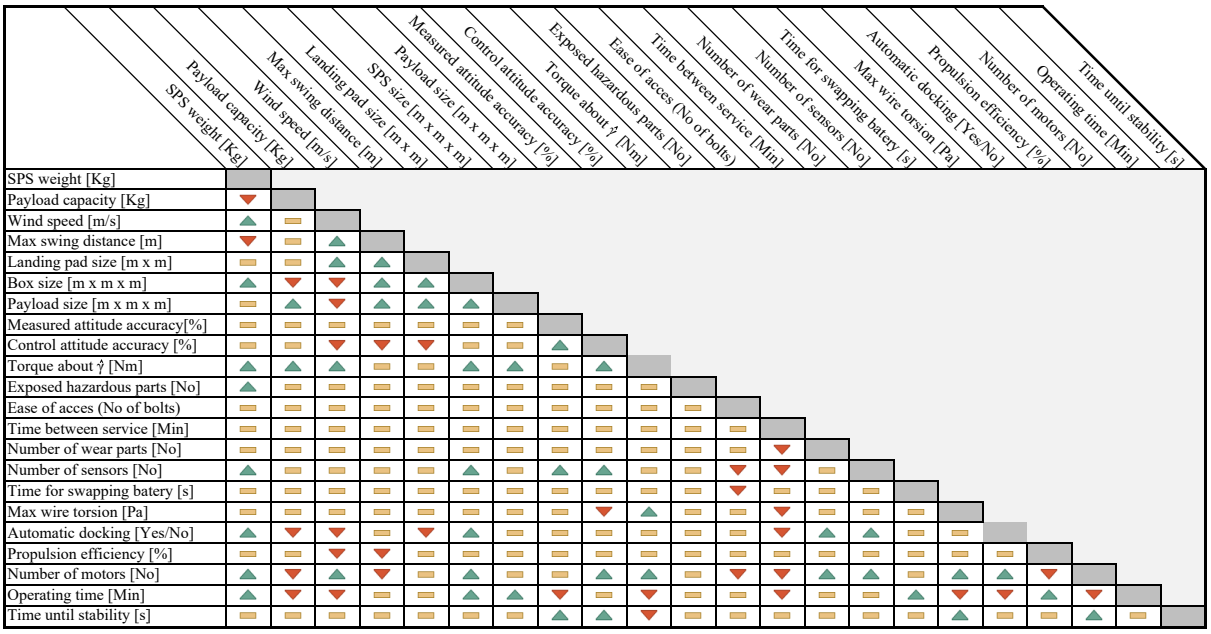


Figure B.2. House of Quality Roof

Morphological Analysis C

In the following Appendix, the morphological analysis that led to the choice of a design concept is explained.

The morphological analysis can be split into defining dimensions, creating solutions, combining, and evaluating (Ullman 2018, pp. 210–214). The dimensions are the different elements of the SPS system. When defining the dimensions, the SPS system is analysed to determine what elements the SPS system consists of. The dimensions of the SPS system are listed below. Dimensions such as safety, communication, and transportation could also be added. Because this report will design a prototype focusing on the control method, these dimensions do not impact the design.

Motor position
The position and number of motors impact the direction and amount of thrust the SPS system can produce. The position of the motors also affects the amount of torque that might be generated. The available propellers are directional, which means that it can generate more thrust in one direction than the other, as seen in Appendix Section F.6. Therefore, the motor's positioning significantly impacts the system's properties.

Payload coupling
The SPS system needs to pick up and deliver a payload. Therefore, to do so effectively, a coupling between the payload and the SPS system is required.

Hoist mount
The wire needs to be mounted on the SPS system so that the system can rotate about its own COM. At the same time, the mount also needs to ensure that the forces from the payload are transferred to the wire without large strains or deformations.

Hoist system
The SPS system is required to move up and down between the drone and the landing location, and a hoist system must be implemented. The hoist system should be compatible with as many drones as possible, so this dimension investigates the location of the hoist system.

XYZ attitude sensing
For control feedback, the SPS system needs to measure attitude. This can be done in different ways, resulting in different sensor options.

SPS to drone distance sensing
The SPS system should not crash into the drone, so a method is required to detect the distance between the SPS system and the drone.

SPS to ground distance sensing

The SPS system should not crash into the ground, so a method is required to detect the distance between the SPS system and the ground.

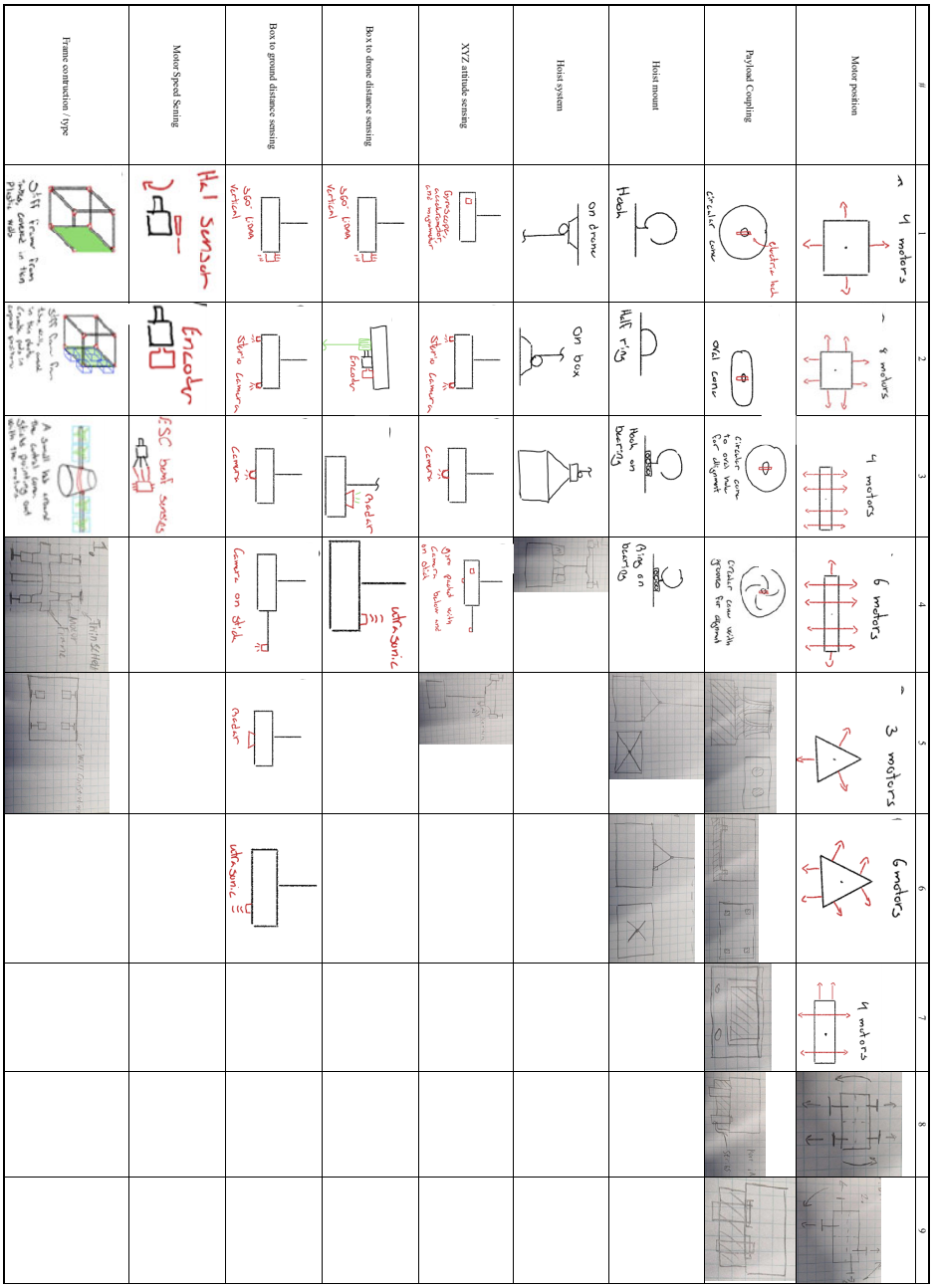
Motor speed sensing

Measuring the speed of the motors is necessary for the SPS system to provide the best possible thrust control.

Frame construction/type

The SPS system needs a frame to support the components and transfer the forces. Different frame types can be constructed, resulting in different assembly methods.

Different solutions can be created from the dimensions to achieve the objectives of the dimensions. The solutions are created through brainstorming, resulting in the solutions seen in Figure C.1.



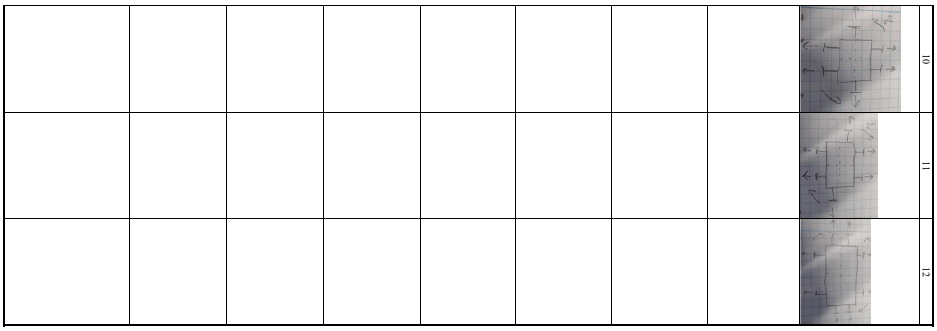


Figure C.1. Solutions for each dimension used for concept generation

One solution for each dimension is combined to create a concept. The concepts try to cover as many different solutions as possible for each dimension to get the broadest view of the solutions during the evaluation. Each concept has a corresponding 9-digit code corresponding to the chosen solutions for each dimension. The dimension order is given in the same order as they were named previously, and the numbers correspond to the ones seen in Figure C.1, e.g. $4-3-4-1-4-4-4-3-3$ has solution 4 for the motor position and solution 3 for payload coupling and so on. This process has been carried out for all dimensions except for the hoist system, where only solution 1 was chosen because the DJI FlyCart 30 already has a host system.

A total of 10 concepts are generated, and from this, a sketch of how the solutions might be put together is drawn. The drawings of the concepts can be seen in Figure C.3. Some of the solutions are similar to each other, where some are very long with the motors mounted on a stick-like extrusion, and others are more box-shaped with a single triangle concept. The motors are placed in different configurations, allowing for different thrust and torque applications. Sensors are not necessarily drawn, since sensor locations depend on the design and can be easily changed.

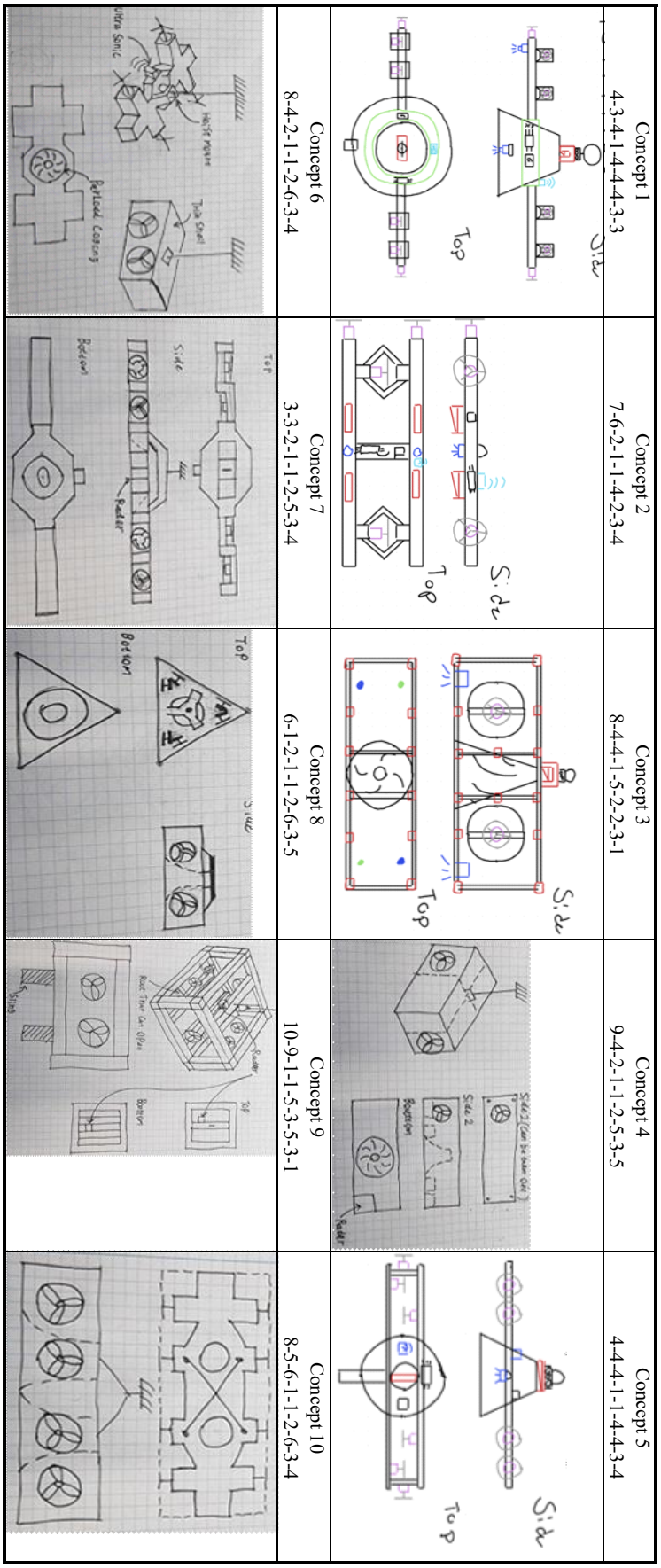


Figure C.2. The 10 concepts generated and used for evaluation

To evaluate the concepts, some of the engineering specifications are easier to evaluate if an estimate is calculated. Calculating an estimate of the thrust and torque characteristics can be used when evaluating *"wind speed"*,*"max swing distance"*, *"torque about* Ψ*"*, and *"time until stability"*.

The estimated characteristics were calculated to get an idea of the thrust and torque characteristics of the concepts. The motors can produce a thrust of 13.98 N in the positive direction and 5.23 N in the negative direction as calculated in Appendix Section F.6. A constant distance of 0.129 m equal to the propeller's diameter has been defined as the distance between the propellers for the torque calculations, but has also been used for estimating the SPS size.

For calculating thrust and torque estimates of the number of motors and their location, can be found in Table C.1. Thrust can be generated in the positive and negative x and y directions. The thrust a concept can provide in a given direction is calculated by Equation (C.1) where a_x refers to a_0 to a_3 and b_x refers to b_0 to b_3. Torque can be calculated by Equation (C.2). The inputs to the equations are given in the following format:

a_0		b0	
a_1		b_1	
a_2		b_2	
a_3		b_3	
c	d	f	e

$$T = a_x \cdot 13.98N + b_x \cdot 5.23N \tag{C.1}$$

$$M = (13.98N \cdot (2 \cdot c + d) + 5.23N \cdot (2 \cdot e + f)) \cdot 0.129m \tag{C.2}$$

	Concept 1		Concept 2		Concept 3		Concept 4		Concept 5		Concept 6		Concept 7		Concept 8		Concept 9		Concept 10	
Number of motors	6		4		4		4		6		4		4		6		6		8	
X+	1	1	2	0	0	0	1	1	1	1	0	0	0	0	$2 \cdot cos(\pi/3)$	$2 \cdot cos(\pi/3)$	1	1	0	0
X-	1	1	0	2	0	0	1	1	1	1	0	0	0	0	$2 \cdot cos(\pi/3)$	$2 \cdot cos(\pi/3)$	1	1	0	0
Y+	4	0	2	0	2	2	1	1	2	2	2	2	2	2	$4 \cdot sin(\pi/3)$	2	2	2	4	4
Y-	0	4	0	2	2	2	1	1	2	2	2	2	2	2	2	$4 \cdot sin(\pi/3)$	2	2	4	4
M	1 1 1 1		0 1.5 1.5 0		0 2 2 0		0 2 1 0		1 1 1 1		0 2 2 0		1 1 1 1		0 3	3 0	0 2 2 0		2 2 2 2	
Thrust x_+	19.24N		27.98N		0		19.24N		19.24N		0		0		19.24N		19.24N		0	
Thrust x_-	19.24N		10.51N		0		19.24N		19.24N		0		0		19.24N		19.24N		0	
Thrust y_+	55.95N		27.98N		38.48N		19.24N		38.48N		38.48N		38.48N		58.96N		38.48N		76.96N	
Thrust y_-	21.01N		10.51N		38.48N		19.24N		38.48N		38.48N		38.48N		46.17N		38.48N		76.96N	
Torque	7.48Nm		3.74Nm		4.99Nm		4.30Nm		7.48Nm		4.99Nm		7.48Nm		7.48Nm		4.99Nm		13.59Nm	

Table C.1. Thrust and torque estimations for the ten concepts

An expected median concept was chosen as a reference point to evaluate the concepts. The other concepts were then compared for each engineering specification and given points depending on how much better or worse they were compared to the median concept. A point of 1 would be a little better, and a point of 3 would be a much better, the same concept is applied for the negative points. As points do not correspond to anything measurable, a rule was to always start with 1 or -1 if the concept evaluated differed from the median. If a different concept provides a value between the median and the previously evaluated concept, its point would be changed to a 3 or -3. Figure C.3 shows the evaluation of all the concepts, where the sum is the sum of all the points given to a concept. The weighted sum is the sum of all the points given with regard to the weight of the engineering specifications.

The evaluation provides three concepts with weighted sums near each other and shares many solutions. These concepts are evaluated an extra time to generate the final concepts by combining the best characteristics. Concept 11 $(3-4-4-1-1-2-6-3-4)$ has four motors in the same direction, and concept 12 $(4-4-4-1-1-2-6-3-4)$ differs by having two additional motors to provide thrust perpendicular to the main for four motors. Although Concept 12 scores the highest, Concept 11 will be used for the design process. This is because this report is limited to four motors, and concept 12 is not feasible. In Chapter 9 it will be discussed if concept 12 is a better solution.

#	Requirement Specifications:	Weight	Concept 1	Concept 2	Concept 3	Concept 4	Concept 5	Concept 6	Concept 7	Concept 8	Concept 9	Concept 10		3-4-4-1-1-2-6-3-4	4-4-4-1-1-2-6-3-4
1	Box weight [Kg]	2,98	1	3	-1	-3	1	0	1	0	0	-3		1	0
2	Payload capacity [Kg]	1,43	1	-1	0	-1	1	0	1	0	-1	-1		1	1
3	Wind speed [m/s]	4,42	1	-1	0	-1	1	0	1	1	0	3		1	1
4	Max swing distance [m]	3,16	1	-1	0	-1	1	0	1	1	0	3		1	1
5	Landing pad size [m x m]	3,39	1	1	0	1	1	0	0	1	1	0		0	1
6	Box size [m x m x m]	4,80	-1	0	0	0	-1	0	0	-1	0	-1		0	-1
7	Payload size [m x m x m]	3,54	0	-1	0	0	0	0	0	0	-1	-1		0	0
8	Measured attitude accuracy[%]	4,28	0	0	-3	0	0	0	0	0	-3	0		0	0
9	Control attitude accuracy [%]	4,58	3	1	0	1	3	0	1	1	3	1		1	3
10	Torque about y' [Nm]	3,14	1	-1	0	0	1	0	1	1	0	3		1	1
11	Exposed hazardous parts [No]	1,62	-1	-3	0	0	-3	0	1	0	0	0		0	0
12	Ease of acces (No of bolts)	0,61	1	3	0	0	1	0	1	0	0	0		1	1
13	Time between service [Min]	1,12	-3	-3	0	-1	-3	0	0	-1	0	0		0	0
14	Number of wear parts [No]	1,19	-1	-1	0	0	-1	0	0	-1	-1	-1		0	-1
15	Number of sensors [No]	1,80	-1	0	-1	0	0	0	0	0	0	0		0	0
16	Time for swapping batery [s]	0,59	1	3	0	0	1	0	1	0	0	0		1	1
17	Max wire torsion	0,95	1	0	1	0	1	0	0	0	0	0		1	1
18	Automatic docking	1,75	-1	-3	0	0	0	0	-1	-1	-3	-1		0	0
19	Propulsion efficiency [%]	1,85	1	0	0	-1	0	0	0	-1	0	3		0	1
20	Number of motors [No]	4,53	-1	-1	0	0	-1	0	0	-1	-1	-3		0	-1
21	Operating time [Min]	1,93	0	-1	0	-1	-1	0	0	-1	-1	-1		0	0
22	Time until stability [s]	3,99	1	0	0	-1	1	0	0	1	1	0		0	1
	Sum:		5	-6	-4	-8	3	0	7	-1	-6	1		9	10
	Weigted sum:		21,24	-16,28	-16,68	-18,86	17,76	0,00	19,17	5,53	-9,58	5,15		21,866	26,774

Figure C.3. Evaluation table over the ten concepts together with the two new concepts

Electrical model D

In this Appendix, the electrical model of the BLDC is expanded upon together with a more detailed description of the model validation and determination of the linearisation point.

D.1 Sensorless control and applied phase voltage

The ESC used in this report applies sensorless control in which hall sensors are integrated into the ESC to measure the back-EMF generated by the BLDC. When the back-EMF is positive, the hall sensors output a signal which over 2π electrical degrees results in 6 different modes. This can be seen in Figure D.1. Each mode extends over 60° electrical and corresponds to a set of different switches that are opened on the ESC.

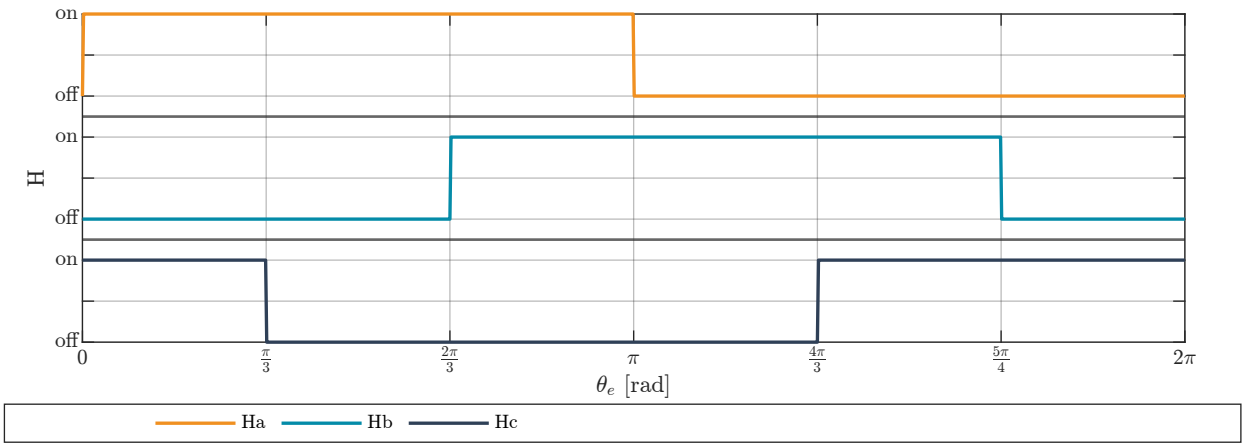


Figure D.1. Hall sensor signals corresponding to the generated back-EMF

Each set of active switches results in line voltages corresponding to Table D.1. Each mode and how the corresponding currents flow in the delta-configured BLDC can be seen in Figure D.2.

Mode 1 $(0 - \frac{\pi}{3})$	Mode 2 $(\frac{\pi}{3} - \frac{2\pi}{3})$	Mode 3 $(\frac{2\pi}{3} - \pi)$	Mode 4 $(\pi - \frac{4\pi}{3})$	Mode 5 $(\frac{4\pi}{3} - \frac{5\pi}{4})$	Mode 6 $(\frac{5\pi}{4} - 2\pi)$
$W+$	$U+$		$V+$		$W+$
$V-$		$W-$		$U-$	

Table D.1. The active switches result in the line voltages shown in the table

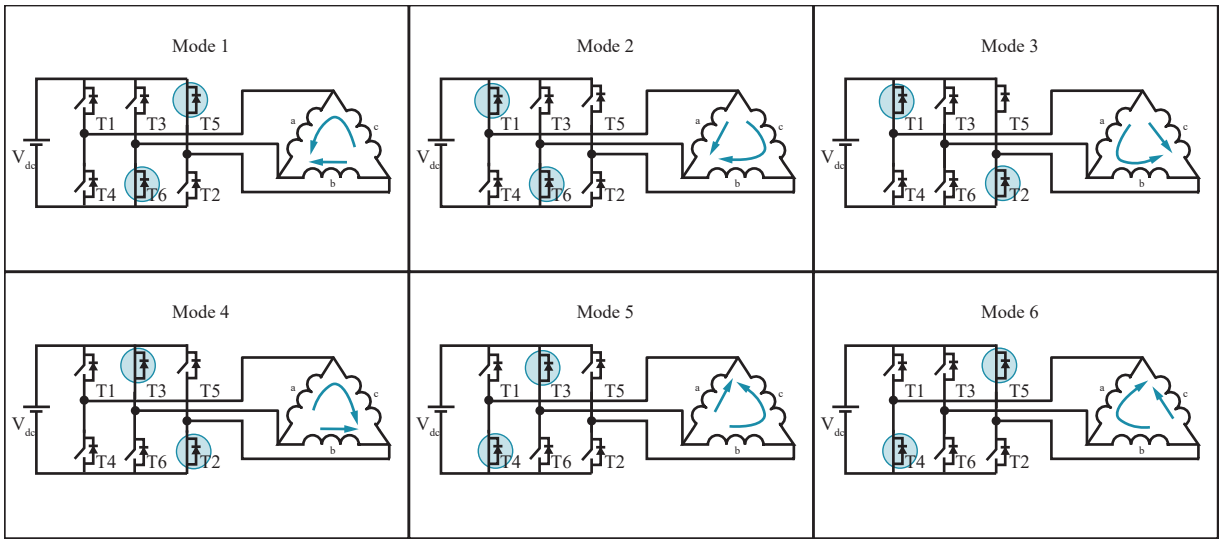


Figure D.2. Swichning sequence corresponding to Table D.1 and the corresponding currents flows for delta wound BLDC motor

Due to the BLDC being a delta wound, the phase voltages are never zero, but $\pm U_{DC}$ or $\pm\frac{1}{2}U_{DC}$. This results in the phase voltages following a stair form according to Figure D.3.

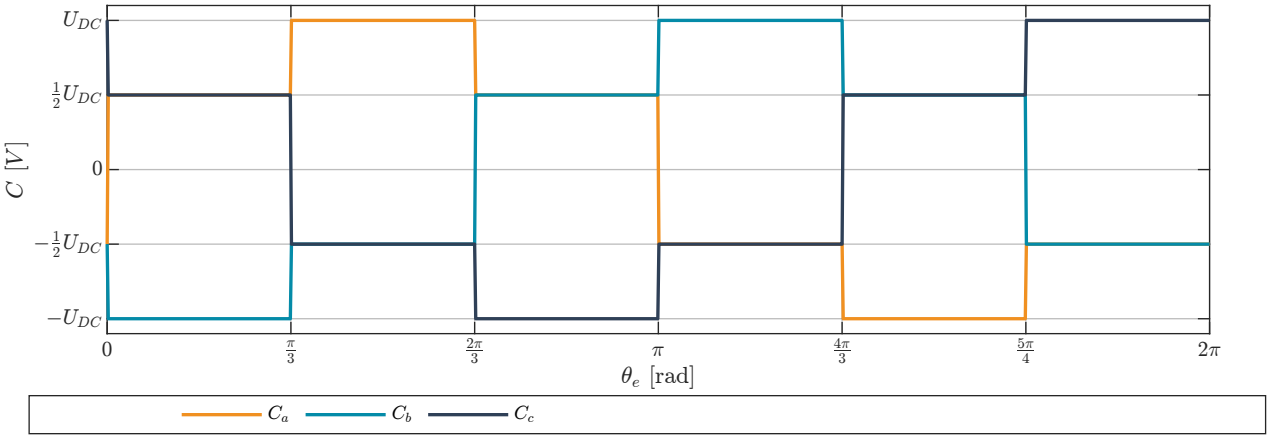


Figure D.3. The phase voltages applied by the ESC

D.2 Validation of the currents and torque ripples

When applying phase voltages with the shape shown in Figure 5.4, the output phase and line currents look as shown in Figure D.4, where a 10% throttle is applied. It can be observed that the phase currents at their maximum are $\frac{3}{2}$ that of the line currents, as theory suggests (Lee, Seo, et al. 2015, p. 169).

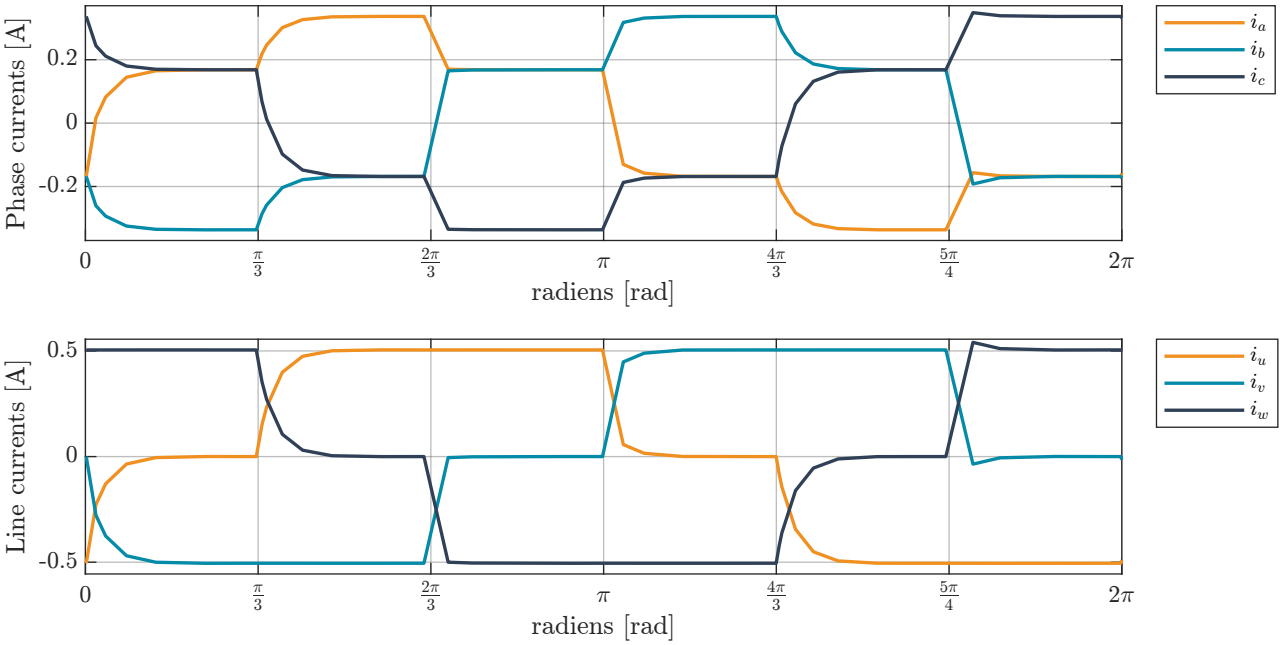


Figure D.4. Phase and line currents during 2π electrical degree

When the switching occurs, the current requires time to ramp up and down due to the inductance, which results in torque ripples, which for the same period as shown in Figure D.4 yields the torque seen in Figure D.5. Ideally, the torque ripple will be expected to drop downwards, but at $\frac{5\pi}{4}$, a small positive torque ripple is observed due to Simulink and the solver.

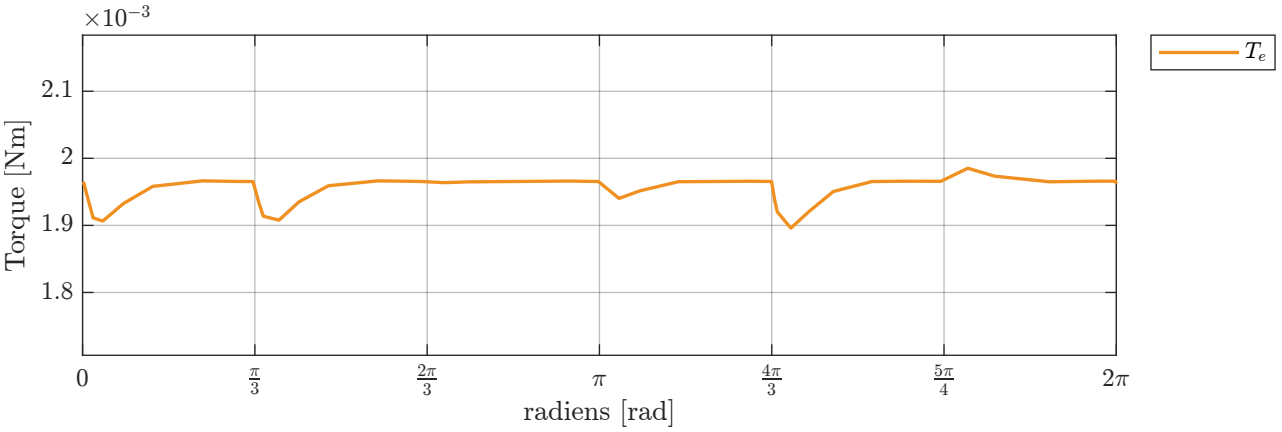


Figure D.5. Torque of BLDC during the same period as shown in Figure D.4. Visible torque ripples are visible

D.3 Electrical model validation

To validate that the electrical model matches the behaviour of the real system, two tests were conducted with the motors. The first test is a step test that adds the throttle from 0 to 98% in steps. The same is done in the simulation, where the percentage is taken as a percentage of the battery voltage of 16.2 V. In the second test, the throttle is randomly changed between 0 to 98% to observe the system's dynamics.

Figure D.6 shows the simulation's result. Here, the simulation is compared to the data measured when running the motors. In the figure, the moments affecting the motor are split up so that the electrical torque from the motor is given by τ_e. The torque from the viscous and column friction is provided by τ_f, and the torque from the wind affecting the propeller is given by τ_w.

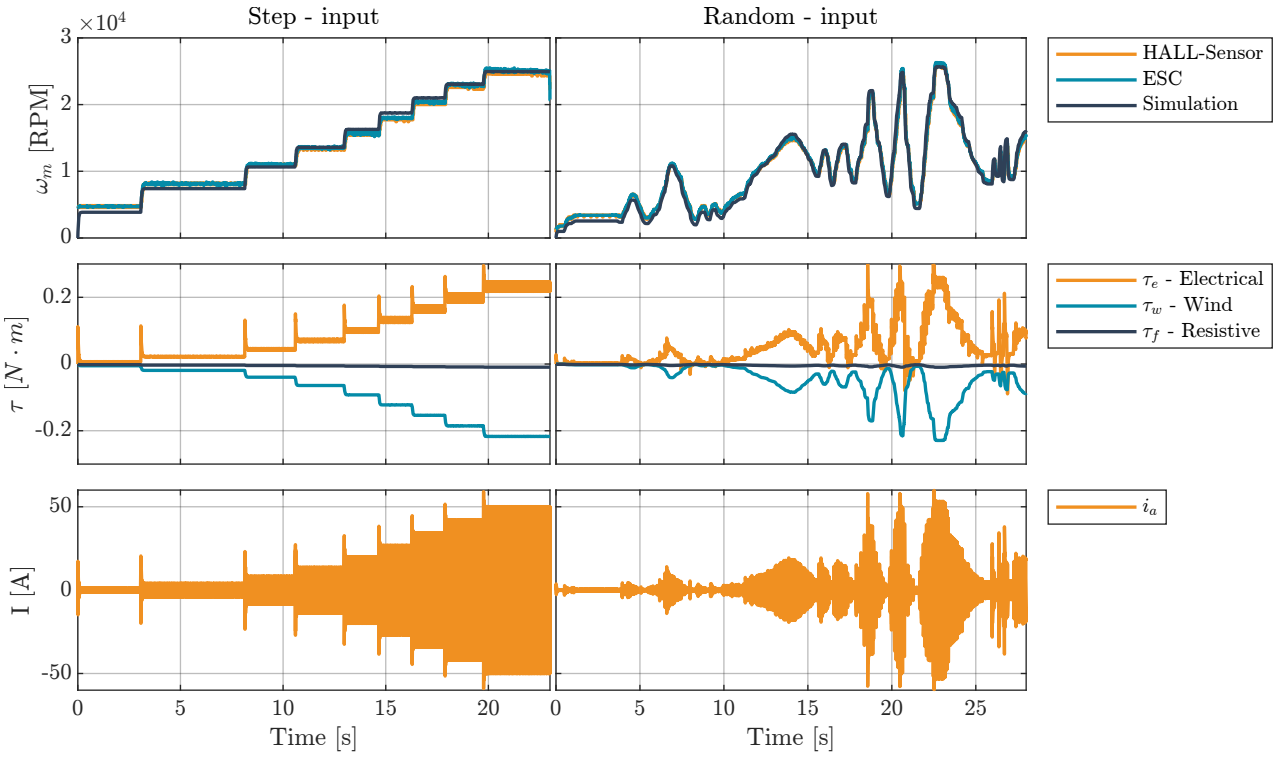


Figure D.6. BLDC - validation, with torques and a phase current i_a

Here, it can be seen that on the step test, the Simulink simulation achieves a rotational velocity, varying between ± 800 RPM from the measured data throughout the steps. The difference between the simulated and measured data is lowest around 40% throttle at time 13 s. In the random input test, it can also be observed that the rotational velocity is a bit off from the measured value, but it can also be seen that the dynamics of the simulation seem to match well. The most significant difference is at the lower speed at $\approx 10\%$ throttle.

The simulation shows that the motor is producing torque up to ≈ 0.23 Nm and that to produce this, the phase current has a max value of ≈ 50 A.

D.4 Linear BLDC model linearisation point analysis

The linearisation point ω_{m0} will determine the dynamic response of the linearised model. The step input and random input test from Section 5.1.3 are replicated for the linear model to find the best linearisation points. It is desired to linearise at a speed at which the motor will be during a significant part of its operation. Because of the exponential curve of the thrust, the motor is expected to run mostly between $50 - 90\%$ throttle. The linear model is simulated with the following linearisation points: $\omega_{m0} = \omega_{m.max} \cdot [0.5, 0.55, 0.57, 0.6, 0.63, 0.65, 0.70]$, where the best fit is chosen. $\omega_{m.max} = 26500RPM$.

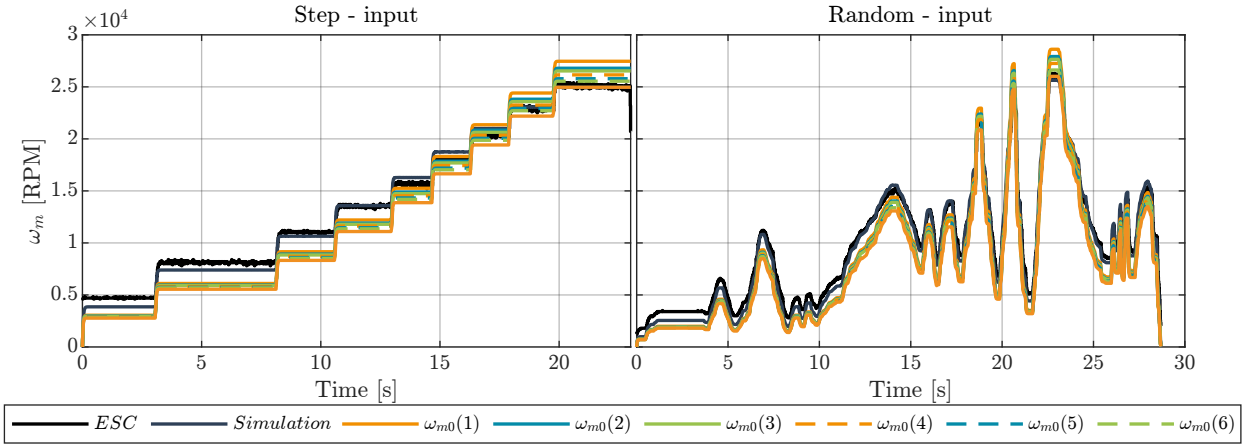


Figure D.7. ESC measurement and non-linear BLDC model compared to the linear model at different linearisation points

Because the BLDC is expected to mostly operate in the $50-90\%$ throttle region, the linearisation point chosen is the best fit for $70-80\%$ throttle. The best linearisation point for 70% throttle is at $\omega_{m0}(4)$ and for 80% throttle it is $\omega_{m0}(5)$. A linearisation point between the two will provide a good estimation at 75% throttle. The linearisation point is given by Equation (D.1).

$$\omega_{m0} = \omega_{m.max} \cdot 0.615 = 16298RPM \tag{D.1}$$

Mechanical model E

In this Appendix, some aspects of the mechanical model of the pendulum will be expanded upon. This includes some more steps to the Lagrange derivation and the determination of the aerodynamic forces affecting the pendulum model. In addition, a check of the model's behaviour can also be found.

E.1 Pendulum Lagrange mechanics derivation

The velocity of the global coordinates vector $\dot{\vec{R}}$ from Equation (5.30), can then be used to find the size of the velocity vector. This can be seen in Equation (E.1).

$$v^2 = \dot{x}^2 + \dot{y}^2 + \dot{z}^2 = \left(\dot{\gamma}^2 \cdot sin(\alpha)^2 + \dot{\alpha}^2\right) \cdot L^2 \tag{E.1}$$

The position and velocity of the pendulum can then be used to describe the kinetic and potential energy for a point mass. The Potential energy is set to 0 at $z = 0$, which is found at $\alpha = 90°$. This means that a negative value gives the potential energy when α is between 0 and 90°. The kinetic and potential energy equations can be found in Equation (E.2) and (E.3).

$$T_{kin} = 0.5 \cdot m \cdot v^2 = 0.5 \cdot m \cdot \left(\dot{\gamma}^2 \cdot sin(\alpha)^2 + \dot{\alpha}^2\right) \cdot L^2 \tag{E.2}$$

$$V_{pot} = m \cdot g \cdot z = -m \cdot g \cdot L \cdot cos(\alpha) \tag{E.3}$$

From this, the Lagrangian of the system can be found by subtracting the potential energy from the kinetic energy. The Lagrangian is given by Equation (E.4).

$$L = T_{kin} - V_{pot} = 0.5 \cdot m \cdot \left(\dot{\gamma}^2 \cdot sin(\alpha)^2 + \dot{\alpha}^2\right) \cdot L^2 + m \cdot g \cdot L \cdot cos(\alpha) \tag{E.4}$$

The force equation's can then be derived by use of the Euler–Lagrange equation $\frac{d}{dt}\frac{\partial}{\partial \dot{\phi}}L - \frac{\partial}{\partial \phi}L = 0$ for each of the generalised coordinates ϕ this results in equation (E.5) and (E.6), which describes the dynamics and couplings in the system.

$$\ddot{\alpha} \cdot m \cdot L^2 - m \cdot L^2 \cdot \dot{\gamma}^2 \cdot sin(\alpha) \cdot cos(\alpha) + m \cdot g \cdot L \cdot sin(\alpha) = M_\alpha \tag{E.5}$$

$$\ddot{\gamma} \cdot m \cdot L^2 \cdot sin(\alpha)^2 + 2\dot{\gamma} \cdot m \cdot L^2 \cdot sin(\alpha) \cdot cos(\alpha) \cdot \dot{\alpha} = M_\gamma \tag{E.6}$$

The moments M_α, M_γ can be used to input the forces affecting the pendulum. This includes the moments created by the wind forces and the torques the thrust produces. These two equations can be isolated for the angular accelerations as seen in Equation (5.36) and (5.37). To ensure that the pendulum works as expected, the model was checked in the Section E.3 to ensure that it behaves like a theoretical pendulum by comparing the period over which the pendulum oscillates.

E.2 Aerodynamic forces and torques

This section will explain the aerodynamic forces and torques applied to the models. Aerodynamic forces are created when an object moves at a relative speed to the wind. Therefore, the relative speed of the SPS compared to the wind must be known to determine the size and direction of the aerodynamic forces. The speed of the wind is defined as $\vec{v_w} = [v_{wx}, v_{wy}, v_{wz}]^T$, this can be subtracted from the velocity of the pendulum $\vec{R}$, which results in the relative wind speed $\vec{v_{rel}}$ in Equation (E.7).

$$\vec{v_{rel}} = \begin{bmatrix} \dot{x} - v_{wx} & \dot{y} - v_{wy} & \dot{z} - v_{wz} \end{bmatrix}^T \tag{E.7}$$

The angle between the relative wind speed $\vec{v_{rel}}$ and the YAW model's x-axis must be determined to estimate the drag and lift coefficients from smoothing functions. This angle has been called Γ and is calculated with Equation (E.8), which calculates the angle between two 2D vectors. Therefore, the assumption that the α angle is small is used, and drag and lift coefficients are solely based on the xy components.

$$\Gamma = cos^{-1}\left(\frac{\vec{v_{rel}} \cdot u_\Psi}{|\vec{v_{rel}}| \cdot |u_\Psi|}\right) \tag{E.8}$$

Equation (E.8) there calculates the angle between the xy components of the relative wind speed vector $\vec{v_{rel}}$ and a unit vector u_Ψ that points in the YAW model's x-axis. This vector can be described by equation (E.9).

$$u_\Psi = \begin{bmatrix} cos(\Psi) \\ sin(\Psi) \end{bmatrix} \tag{E.9}$$

Section 2.2 shows that the drag and lift coefficients can be estimated based on Rosendahl's or Pesavento's smoothing functions approximations. Rosendahl's smoothing function has been chosen for the model. Therefore, Equations (E.10) and (E.11) are used to determine the drag and lift coefficients for the pendulum box.

$$C_D(\Gamma) = C_{D.0} + (C_{D.90} - C_{D.0}) \cdot sin(|\Gamma|)^3 \tag{E.10}$$

$$C_L(\Gamma) = sin(\Gamma)^2 \cdot cos(\Gamma) \cdot C_D \tag{E.11}$$

The projected area of the SPS is used for calculating drag and lift, as described by equation (E.12), which is the same equation as from Equation (2.4). A sketch of the projected area can therefore be seen in Figure 2.5.

$$A_p(\Gamma) = (|L \cdot sin(\Gamma)| + |t \cdot cos(\Gamma)|) \cdot h \tag{E.12}$$

From this, the force from drag and lift can be calculated. The equations for this can be seen in equation (E.13) and (E.14).

$$\vec{F_D} = \frac{1}{2} \cdot C_D(\Gamma) \cdot \rho \cdot A_p(\Gamma) \cdot |\vec{v_{rel}}| \cdot \vec{v_{rel}} \tag{E.13}$$

$$\vec{F_L} = \frac{1}{2} \cdot C_L(\Gamma) \cdot \rho \cdot A_p(\Gamma) \cdot A_{90} \cdot (|\vec{v_{rel}}| \cdot \vec{v_{rel}}) \tag{E.14}$$

In Equation (E.14), a rotational matrix A_{z90} is required to rotate the lift force by 90 degrees such that lift and drag are perpendicular. The lift force will point towards the low-pressure zone, where the sign of $\vec{F_L}$ is accounted for when calculating $C_L(\Gamma)$. The rotational matrix is given by Equation (E.15)

$$A_{90} = \begin{bmatrix} cos(\frac{\pi}{2}) & -sin(\frac{\pi}{2}) \\ sin(\frac{\pi}{2}) & cos(\frac{\pi}{2}) \end{bmatrix} \tag{E.15}$$

The centre of pressure is located on the SPS nearest the direction of the wind or in the high-pressure zone. Equation (E.16) explains the logic to determine which side of the box is the high-pressure zone. This determines which side $\vec{v_{rel}}$ comes from to decide which side would build up a high-pressure. This can be seen sketched in Figure E.1. The sign is then multiplied by the u_Ψ to create a unit vector in the direction of COP.

$$\begin{aligned} if \quad \Gamma \leq \frac{\pi}{2} &\rightarrow sign_{COP} = u_\Psi \cdot (-1) \\ else \qquad\quad &\rightarrow sign_{COP} = u_\Psi \cdot 1 \end{aligned} \tag{E.16}$$

The COP changes with Γ where equation (E.17) is Rosendahl's correlation for the location of COP from COM.

$$x_p = 0.25 \cdot (1 - sin(\Gamma)) \cdot L \tag{E.17}$$

Multiplying Equation (E.16) and (E.17) results in a vector from COM to COP (E.18). This vector can be seen as the orange line in Figure E.1.

$$\vec{COP} = x_p \cdot sign_{COP} \tag{E.18}$$

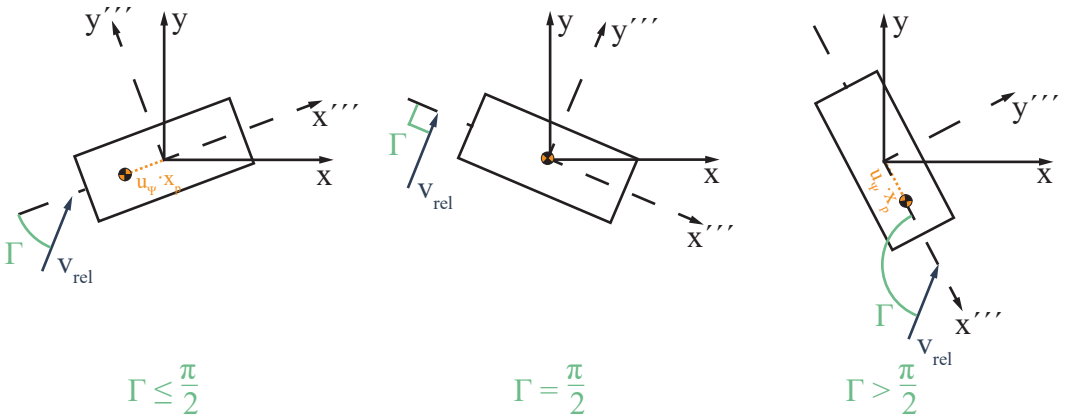


Figure E.1. Location of COP

E.3 Mechanical model behavior verfication

To ensure that the mechanical model works and behaves as intended, some tests have been carried out on it to ensure that it functions as expected. Some tests have, therefore, been run to see if the pendulum behaves as it should, if forces are applied correctly, and if this translates to approved responses.

E.3.1 Period verfication

The model can be tested to determine if the pendulum reacts as a pendulum. This is done by defining $M_{\alpha=0\ Nm}$ and $M_{\gamma=0\ Nm}$ in Equations (E.5) and (E.6). A wire length of $L = 10\ m$ and an initial condition of $\alpha_0 = 65°$ have been specified. This should result in a system with no losses, and all potential energy should be converted back and forth between potential and kinetic energy. This should result in a sinusoidal wave with a period that matches the period for a simple pendulum given by $T \approx 2\pi\sqrt{\frac{L}{g}}$. This test has been carried out for the angle of γ_0 from 0 to 180° so that it can be ensured that the pendulum has the same reaction to all angles of γ. The simulation results can be seen in Figure E.2.

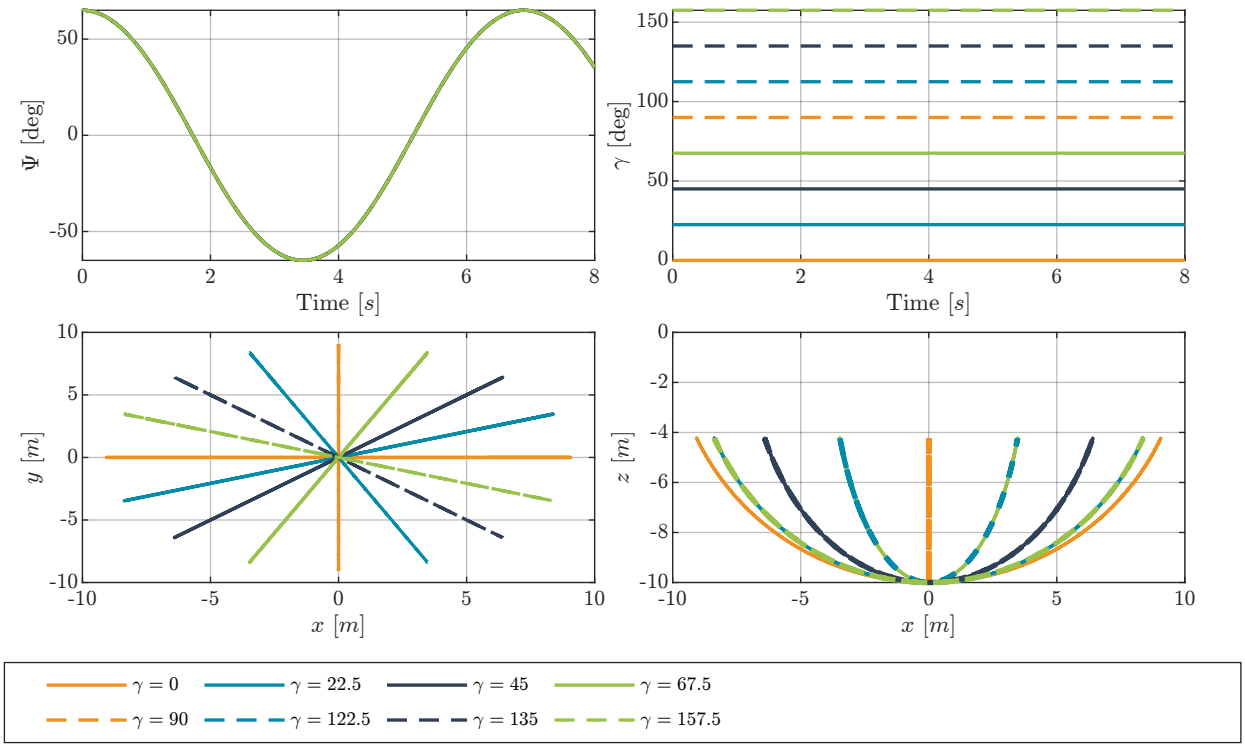


Figure E.2. Simulation results where M_α and M_γ are set to 0 Nm

Here it can be seen that the period of α is given by 6.89 s, which somewhat matches the 6.34 s from the equation for a simple pendulum. Here it can be noted that this is an approximation and that the true period of a simple pendulum is given by equation (E.19).

$$T = 2\pi\sqrt{\frac{L}{g}}\left[\sum_{n=0}^{\infty}\left(\frac{(2n)!}{2^{2n}(n!)^2}\right)^2 sin^{2n}(\frac{\alpha_0}{2})\right] \tag{E.19}$$

At $n = 2$, the period is calculated to 6.89 s. The simulation matches the calculated theoretical period at two decimal places, which is sufficient for using the model in this report.

E.3.2 Breaking verfication

The pendulum will be slowed down by applying forces to the SPS. Only drag forces are considered for this test, where Equation E.2 is applied to Equation (E.5) and (E.6). An initial angle of $\alpha_0 = 90°$ has been tested for γ_0 from 0 to 360°. This is done to see if the drag forces affect the pendulum similarly in all directions, slowing it down equally. The angle of the YAW model is defined by $\Psi = \gamma - 90°$ for all the simulations, as this results in roll swings and the maximum drag force. The estimates from Section 2.2 specify the drag coefficients used in the simulation. The simulation results can be seen in Figure E.3.

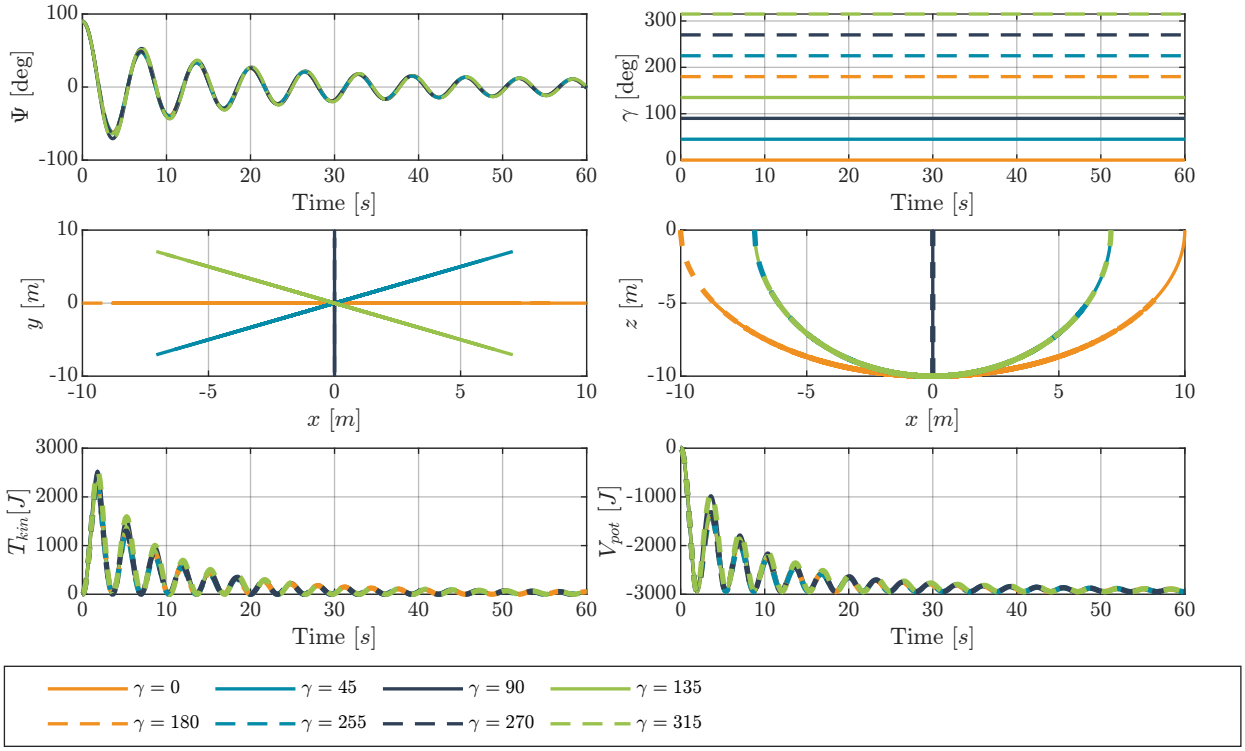


Figure E.3. Simulation results where $\alpha_0 = 90°$ and the windspeed is set to $\approx 0\ \frac{m}{s}$

The α angle is affected similarly in all directions of γ, where a damped sinusoidal movement with a decreasing amplitude over time is obtained. The periods are $\approx 6.49\ s$, which aligns with the pendulum period theory. The kinetic energy decreases during the simulation and correlates with the decline in amplitude of the pendulum. The potential energy becomes a steady value as the oscillations decrease and less energy is converted into kinetic energy. The final potential energy correlates with the expected potential energy for a pendulum hanging 10 m below the measurement point.

E.3.3 Windforce equlibrium verfication

To see if the wind forces are implemented correctly, it would be interesting to see if the simulation recreates the equilibrium from Section 2.2. The Rosendahl function is implemented in the

simulation, and it is desired to achieve an angle of 15.8°, a windspeed of 12 m/s is applied to the system. An initial angle of $\alpha_0 = 0°$ is specified for γ_0 from 0 to 360°. For all the tests $\Psi = \gamma - 90°$, to test roll swigs. The results of the simulations are shown in Figure E.4.

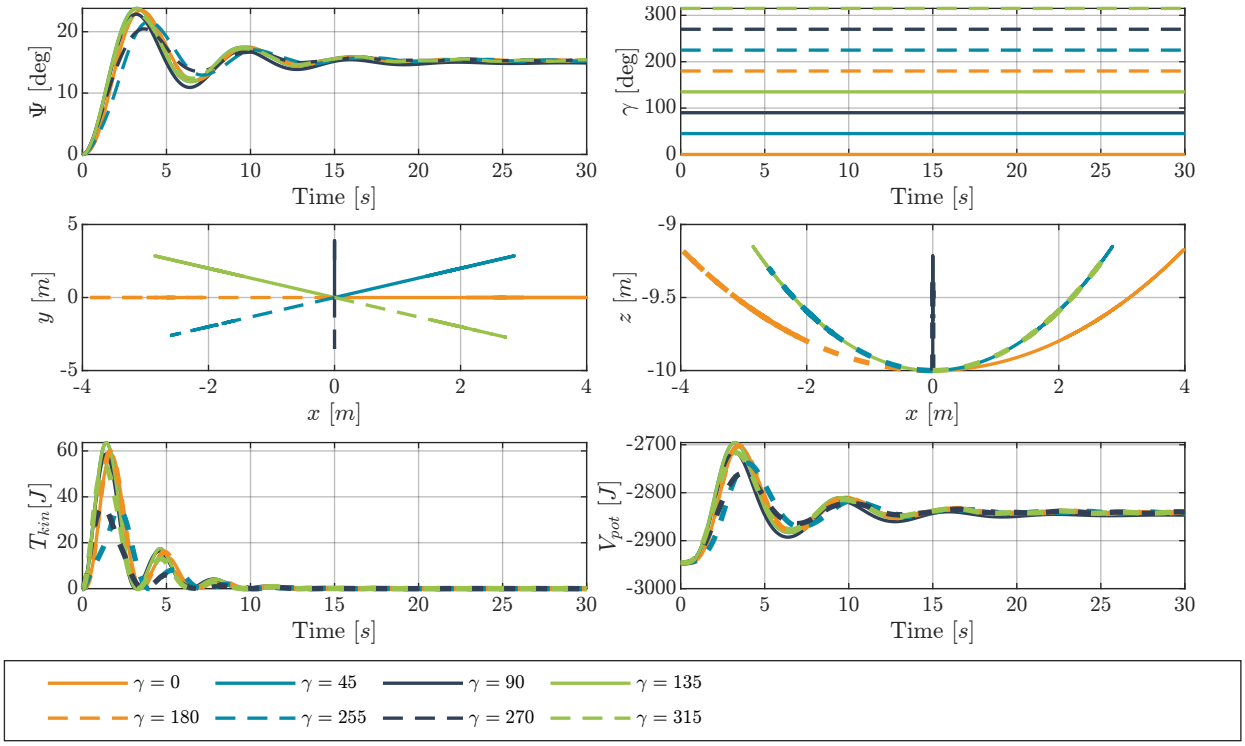


Figure E.4. Simulation results where $\alpha_0 = 0°$ and the windspeed is set to $\approx 12\ \frac{m}{s}$

It can be seen that the system reaches and settles at an equilibrium of $\alpha = 15.3°$, which happens at an applied force of $\approx 81\ N$ from the wind speed of 12 $\frac{m}{s}$. When using the approach in Section 2.2, an equilibrium of $\alpha = 15.3°$ is obtained when the same wind speed is applied.

Parameter estimations F

In this appendix, the different experiments conducted to find parameters for the models used in the report can be found and described in further detail with the results.

F.1 Determination of BLDC confirguration

This test aims to determine whether the BLDC motor is in a star or delta configuration, as this will influence the following electrical motor parameters and model the structure of the BLDC model.

F.1.1 Hypothesis

Figure F.1 shows a star and a delta configuration. If a voltage is applied on the U line and the W line is grounded, the current will run in phase a and c if it is a star configuration. If it is a delta configuration, the current will run in all three phases since they are connected to each other as represented in Figure F.1.

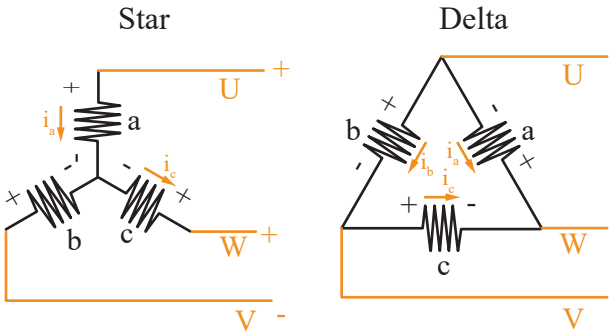


Figure F.1. How currents run through the phases of a star and delta configuration when voltage is applied to the U line and W is grounded.

For the star configuration, the same amount of current will run through the phases a and c. Phase b will not have any current running through it and therefore will not heat up. For the delta configuration, the voltage drop over phase a is the same as that over phases b and c combined. The current in each phase can be calculated by Ohm's law $U = R \cdot I$, resulting in twice the current in phase a compared to phase b or c as they have half the voltage drop.

Applying a DC voltage to heat the windings makes it possible to determine the winding configuration using a thermal camera. If $\frac{2}{3}$ of the windings heat, the winding configuration must be a star. If all windings heat up, but $\frac{1}{3}$ of the windings are heating to twice the temperature as the rest, it can be concluded that the winding configuration must be delta.

F.1.2 Material list

The following materials have been used for the experiment.

Item	Manufacture & model	Number of items
• BLDC motor	Tmotor F40proIV	1
• Thermal camera	FLIR-E64501	1
• Power supply	Gw Instek GPS-4303	1

F.1.3 Exsperimental procedure

- The rotor of the BLDC motor is removed from the stator for visual access to the windings.
- One line of the BLDC motor is connected to the positive channel on the power supply, and a different line is connected to the ground channel on the power supply.
- The power supply is current limited to a safe value within the limits of the BLDC motor so as not to damage the windings during the heating. The current limit is set low as the thermal mass of the rotor has been removed, and no cooling of the rotor will cool the windings down. The Tmotor F40proIV BLDC motor can sustain a peak current of $42.9A$ for 60 seconds (TMOTOR 2024), the power supply is therefore limited to $6A$.
- The thermal camera is aimed at the BLDC motor coils. A measurement of the temperature of the coils is made once the phases have been heated sufficiently. If $\frac{2}{3}$ of the coils are heated, it is a star configuration. If all the phases heat up but $\frac{1}{3}$ of the coils are heated more, it is a delta configuration.

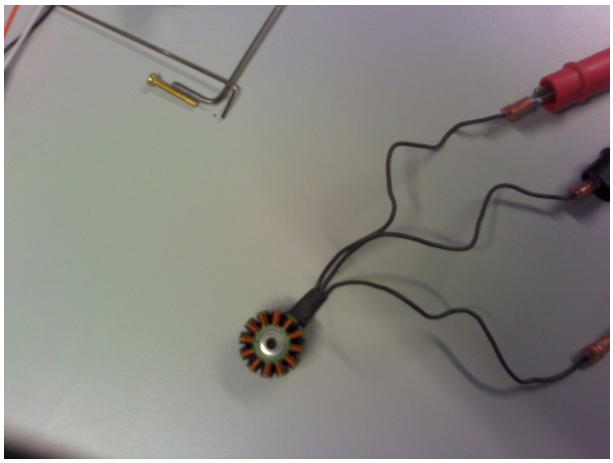


Figure F.2. Experimental setup used for determining if the motor is star or delta configured.

F.1.4 Results

In Figure F.3, the thermal image of the BLDC motor can be seen with a $6A$ DC current applied.

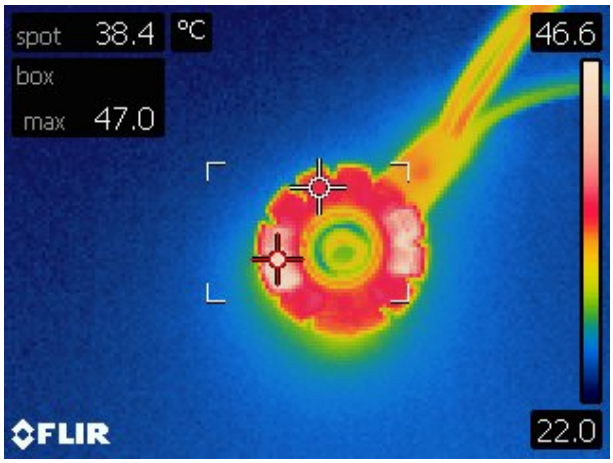


Figure F.3. Thermal image of the BLDC motor with a $6A$ DC current applied

The Tmotor F40proIV has 12 coils (TMOTOR 2024), and in Figure F.3 it can be seen that 4 coils are heated more than the rest. $\frac{4}{12} = \frac{1}{3}$ of the coils are hotter than the others, indicating the BLDC motor is in a delta configuration.

The room temperature is approximately $22°C$, whereas the hottest coils are $47°C$. The remaining coils are measured to be $38.4°C$, resulting in the temperature change shown in Table F.1.

Coils	ΔT
Hottest	$25°C$
Remaining	$16.4°C$

Table F.1. Table of delta temperature to room temperature at $22°C$

Table F.1 further provides evidence for the motor being configured in a delta configuration, as the hottest coils are approximately twice as hot as the rest.

F.1.5 Discussion

The results show that four coils are heated more than the remaining coils, indicating a single phase receives a higher current than the remaining when a DC voltage is applied between two lines, indicating the BLDC is wound in a delta configuration. Table F.1 shows the ΔT of the coils, further proving that the winding configuration is delta, as the hottest coils have heated up twice as much as the remaining coils. Although the relation between the hottest and remaining coils was expected to be closer to 2, this is most likely due to the coils not being thermally isolated and the hottest coils heating the rest of the coils. This is further indicated by the green hue around the motor in FigureF.3

F.1.6 Conclusion

This experiment successfully determined that the BLDC motor is configured in a delta configuration. The conclusion is made via both visual confirmation from Figure F.3 and temperature measurements in Table F.1, both indicating that the winding configuration is delta.

F.2 Measuring resistance of BLDC

This experiment aims to accurately measure the internal resistance of the BLDC motor (line-to-line resistance). From where the phase resistance can be calculated for use in simulating the BLCD motor. In Appendix Section F.1, it was determined that the configuration of the BLDC was a delta configuration. This fact will be used to determine the resistance of the BLDC.

F.2.1 Hypothesis

Tmotor provides a resistance of 45 $m\Omega$ (TMOTOR 2024) but does not explain whether it is a phase or line resistance.

The voltage equation for one phase of the BLDC motor is given by equation (F.1). A BLDC motor needs motor control to rotate, which is unwanted when measuring the internal resistance. Applying a constant voltage between two lines, the coils will magnetise and the rotor will orient itself towards the nearest opposite pole. A constant current will run through the wires, eliminating the second term in Equation (F.1) since $\frac{dI}{dt} = 0$, and the last term is also eliminated because there is no induced back-EMF as the rotor is stationary and $E = 0$. The remaining term is thereby Ohm's law.

$$U = R_{eq} \cdot I + L_{eq} \cdot \frac{dI}{dt} + E \tag{F.1}$$

By measuring the voltage drop between the two lines and the current in the wire, it is possible to calculate the resistance with Ohm's law by isolating for R_{eq} resulting in Equation (F.2), where R_{eq} is the equivalent resistance of the windings when measuring between two lines.

$$R_{eq} = \frac{U}{I} \tag{F.2}$$

BLDC motors are designed to have the same resistance in each phase $R_a = R_b = R_c$, and for a delta wound motor, the equivalent resistance is a parallel-series resistance as seen in Figure F.4.

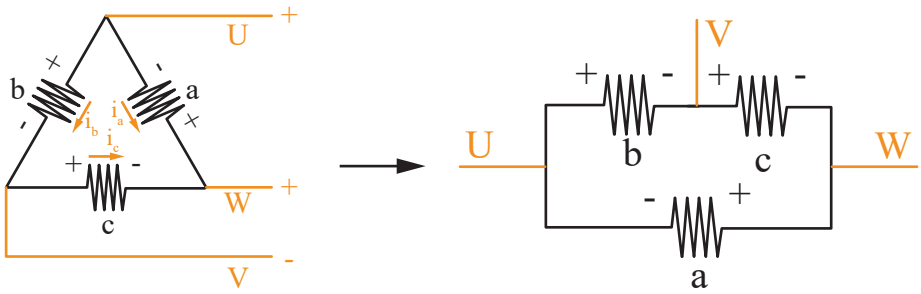


Figure F.4. Rearranging the phase resistors to calculate the equivalent resistance between line U and W

The equivalent resistance can therefore be calculated as Equation (F.3).

$$R_{eq} = \left(\frac{1}{R_a} + \frac{1}{R_b + R_c} \right)^{-1} \tag{F.3}$$

Because all resistances are equal, Equation (F.3) can be simplified to Equation (F.4), where R_s is the phase resistance $R_s = R_a = R_b = R_c$.

$$R_{eq} = \frac{2}{3} \cdot R_s \tag{F.4}$$

The phase resistance can be calculated by isolating Equation (F.4) for R_s, which yields the phase resistance, which can be calculated as Equation (F.5).

$$R_s = \frac{3}{2} \cdot R_{eq} \tag{F.5}$$

F.2.2 Material list

The following materials have been used for the experiment.

Item	Manufacture & model	Number of items
• Digital Oscilloscope	Tektronix TBS 2000B SERIES	1
• Voltage probe	Tektronix TPP0100 Voltage probe	1
• Current probe	Tektronix TCP0030A	1
• BLDC motor	Tmotor F40proIV	1
• Power supply	Gw Instek GPS-4303	1

F.2.3 Exsperimental procedure

- Turn on the oscilloscope and connect the voltage and current probes. The voltage probe is calibrated to the correct attenuation by connecting the probe to the probe comp and ground. The calibration is done when the 5 V square wave matches the 5 V line.
- The voltage probe is connected to the lines, and the resistance is measured. The current probe is placed on the + wire with the arrow pointing towards the motor for positive measurements. Once the current probe is connected, it is degaussed by pressing the button on the device and is scaled to 5 A.
- The voltage is applied, and the voltage and current are measured as an average over 100 ms.
- The power supply is current-limited as it yields the most consistent results. The current is increased from 1 A to 5.5 A in increments of 0.5 A, resulting in 10 measurements for each line.
- Steps 2 to 4 are repeated for lines UV, VW, and WU.
- Linear regression is performed on the measurements where the slope equals the resistance.

F.2.4 Results

The measurements for line UV can be seen in Table F.2, for line VW in Table F.3, and for line WU in Table F.4.

	1	2	3	4	5	6	7	8	9	10
Voltage (mV)	43	81	103	129	164	197	208	244	278	310
Current (A)	1.011	1.551	2.017	2.495	3.089	3.562	4.040	4.536	5.012	5.548

Table F.2. Voltage and current measurements for line UV

	1	2	3	4	5	6	7	8	9	10
Voltage (mV)	44	81	100	126	163	184	203	244	262	296
Current (A)	1.014	1.553	2.047	2.532	3.028	3.573	4.042	4.528	5.039	5.548

Table F.3. Voltage and current measurements for line VW

	1	2	3	4	5	6	7	8	9	10
Voltage (mV)	43	75	86	126	164	190	205	243	276	304
Current (A)	1.057	1.553	1.992	2.526	3.088	3.570	4.044	4.529	5.088	5.565

Table F.4. Voltage and current measurements for line WU

Applying linear regression to the data will result in the resistance being the slope of the linear function in the form $y = R_{line-2-line} \cdot x + b$. For the regression, the slope must pass $(0, 0)$ to fulfil Ohm's law. The b term in the linear function becomes $b = 0$. Figure F.5 shows the data points and the regression results.

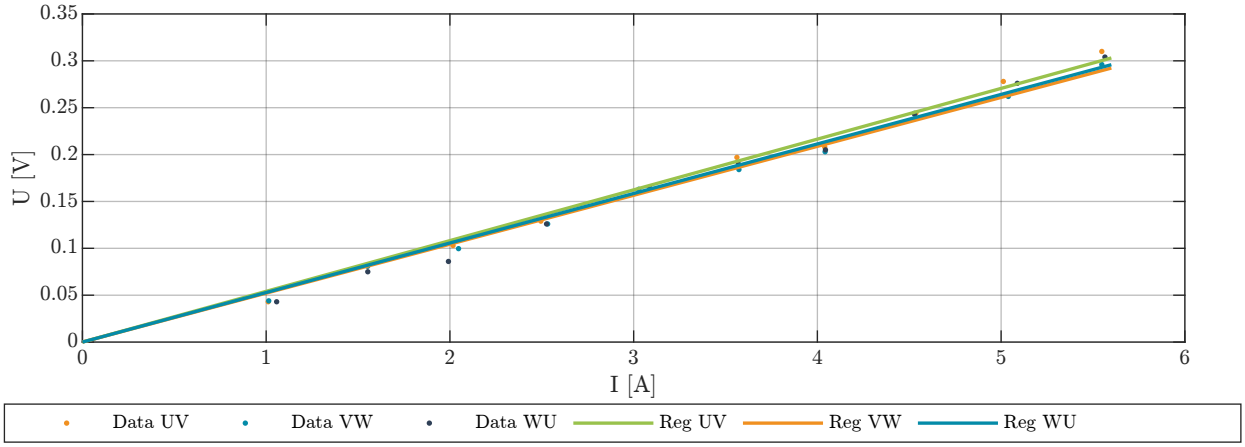


Figure F.5. Regression results and data for calculating line resistances

The calculated resistances are shown in Table F.5.

R_{UV}	0.0571 Ω
R_{VW}	0.0522 Ω
R_{WU}	0.0528 Ω

Table F.5. line resistances measurements

Due to measurement noise and inconsistencies, the line resistances differ, and the average value can better represent the line resistance. The line resistance is given by Equation (F.6).

$$R_{line} = \frac{R_{UV} + R_{VW} + R_{WU}}{3} = 0.053\ \Omega \tag{F.6}$$

The phase resistance is desired for simulation, which is calculated by Equation (F.5). The phase resistance is calculated in Equation (F.7).

$$R_s = \frac{3}{2} \cdot R_{line} = 0.0796\ \Omega \tag{F.7}$$

F.2.5 Discussion

The measured line resistances are slightly higher than those specified by the manufacturer. This could be due to the resistance specified by the manufacturer being the result of simulations or the specified resistance being measured closer to the windings. For the measurements made in this experiment, the wire clamp connectors were connected to the wires of the BLDC motor. Additionally, wires from the power supply to the motor were used. All of these factors slightly increase the resistance, and when the line resistance only differs by 0.008 Ω from what is specified by the manufacturer, the results are deemed sufficiently accurate.

Data were collected on 100 ms as a mean value of 20.000 data points, to eliminate the influence of noise as much as possible. When conducting the experiment, it was observed that the measurements still fluctuated. The oscilloscope was therefore triggered so that the voltage and current measurements matched.

It was observed that the linear regression of the data points did not pass through $(0, 0)$ as expected from Ohm's law. The regression was therefore forced to pass through the point $(0, 0)$ to fulfil Ohm's law, which decreased the fit of the regression. It was observed that forcing the regression through $(0, 0)$ slightly lowered the resistance and thus brought it closer to the expected resistance of the manufacturer. The resistance calculated from the regression, if it were not forced through $(0, 0)$, would be 0.0566 Ω.

F.2.6 Conclusion

The motor's line resistances have been measured in a way that would eliminate most measurement uncertainties, as an average made over a range of current inputs. The results are slightly higher than the manufacturer's specifications but within a reasonable error. Increasing the data gathering time could potentially improve the results slightly, but because so many measurements have been made, this is deemed an insignificant improvement.

F.3 Measuring back-EMF of BLDC

This experiment aims to measure the back-EMF constant of the BLDC motor. For a delta-wound BLDC motor, the back-EMF is expected to be trapezoidal with a 60° flat top.

F.3.1 Hypothesis

The back electromotive force, or back-EMF, is the voltage induced in the stator during the operation of the motor. In a BLDC motor, the permanent magnets induce the back-EMF and are ideally trapezoidal. For a delta-wound BLDC, the trapezoid has a 60° flat top. Due to the BLDC not being ideal and delta-wound configurations having circulating currents due to triple harmonics, it is not expected that the back-EMF waveform will be pure trapezoidal (Emetor 2024) (ASHISH and VINCENT 2014).

The back-EMF constant is a linear relation between the peak voltage induced by the permanent magnets and the rotational speed of the rotor. The equation for calculating the back-EMF constant K_m is given by equation (F.8).

$$K_m = \frac{V_{pk}}{\omega_m} \tag{F.8}$$

V_{pk} is the peak voltage induced by the back-EMF and ω_m is the rotational speed of the rotor.

By driving the rotor at a constant speed, measuring the back-EMF waveform on an oscilloscope is possible. By connecting differential probes to the lines as shown in Figure F.6, it is possible to measure the phase voltage. This is because the line voltage equals the phase voltage on a delta-wound BLDC motor.

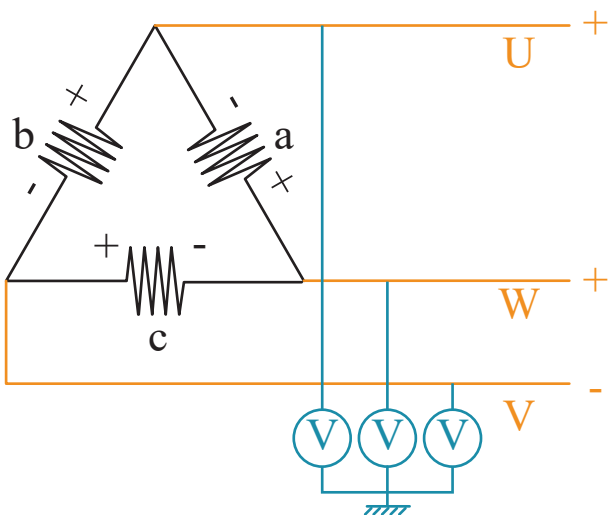


Figure F.6. Circuit for differential voltage probes to create an artificial neutral point for measuring the back-EMF constant

Equation (F.8) is only dependent on the peak voltage and the rotational speed of the rotor, which for an accurate calculation needs to be measured accurately. Equation (F.11) can be rewritten to Equation (F.9), which calculates the electrical back-EMF constant of the BLDC. The relation

between the back-EMF constant K_m and the electrical back-EMF constant K_e calculated in Equation (F.9) is the number of pole pairs $\frac{P}{2}$, where P is the number of poles which is the calculation from mechanical speed ω_m to electrical speed ω_e.

$$K_e = \frac{V_{pk}}{\omega_e} \rightarrow \frac{V_{pk} \cdot T_e}{2\pi} \rightarrow \frac{V_{pk-pk} \cdot T_e}{4\pi} \tag{F.9}$$

K_e is the electrical back-EMF constant, T_e is the period of the back-EMF wave, V_{pk-pk} is the peak-to-peak voltage or $V_{pk-pk} = V_{pk} \cdot 2$ Bobek 2013.

The BLDC motor is a Tmotor F40PRO IV KV2400 TMOTOR 2024, where the K_v number is the inverse of the back-EMF constant, as it describes the amount of RPM per. voltage ($\frac{RPM}{V}$). Theback-EMFf constant should thereby be close to $K_v = 2400\ \frac{RPM}{V}$ when using equation (F.10).

$$K_{v.m} = K_e^{-1} \cdot \frac{60}{2 \cdot \pi} \cdot \frac{2}{P} \tag{F.10}$$

F.3.2 Material list

The following materials have been used for the experiment.

Item	Manufacture & model	Number of items
• Digital Oscilloscope	Tektronix TBS 2000B SERIES	1
• Differential voltage probe	Tektronix P5200 Differential Voltage Probe	3
• BLDC motor	Tmotor F40proIV KV2400	1
• Driving motor	Tmotor F40proIV KV2400	1
• Motor mount		2
• Motor coupling		1

F.3.3 Exsperimental procedure

- Turn on the oscilloscope and connect the differential voltage probes. The differential voltage probes are calibrated to the correct attenuation by connecting the probe to the probe comp and ground. The calibration is done when the 5 V square wave matches the 5 V line.
- The BLDC motors are connected to the motor coupling and mounts, then both motors are mounted securely on the workspace table.
- The differential probes are connected to the BLDC motor as shown in Figure F.6, the other motor is used to drive the BLDC motor. The faster the motor can be driven, the lower the voltage measurement error.
- The BLDC is driven as fast as possible at a constant speed, and the back-EMF waveform is measured on the oscilloscope.
- The peak-to-peak voltage and the period are calculated for each phase, and equation (F.9) is used to calculate the back-EMF constant.

F.3.4 Results

The measured back-EMF can be seen in Figure F.7.

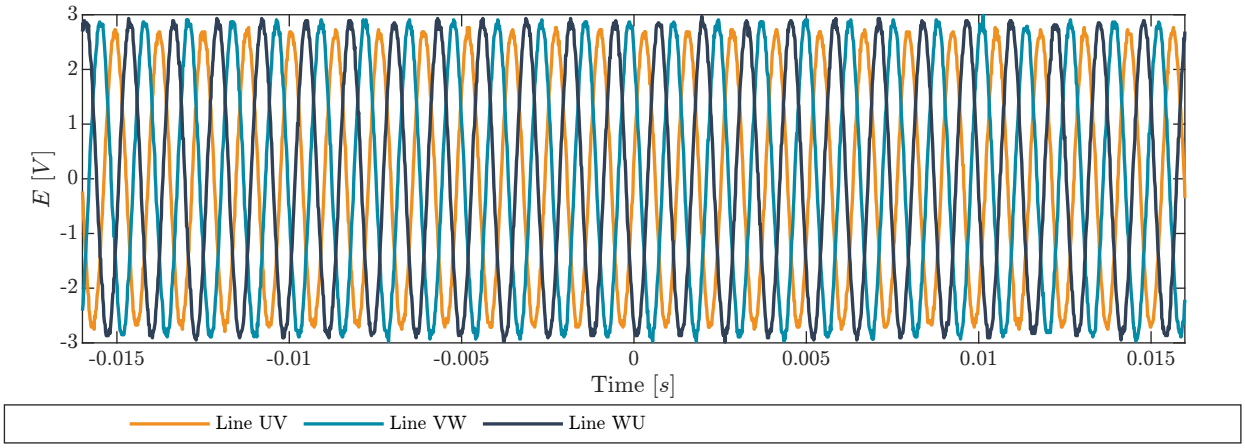


Figure F.7. Measured back-EMF for each line-2-line

A time span containing a full period for each line measurement is selected to measure the period time T_0 and calculate the peak-to-peak voltage. The results for each line can be seen in Table F.6.

Line	T_0	V_{pk-pk}
UV	0.0013 s	5.15 V
VW	0.0013 s	5.08 V
WU	0.0013 s	5.21 V

Table F.6. Measurement results for calculating back-EMF

Calculating the back-EMF for each line with Equation (F.9) results in the back-EMF constants shown in Table F.7.

Phase	K_e
UV	$5.3277e^{-4}$ $\frac{V \cdot s}{rad}$
VW	$5.2553e^{-4}$ $\frac{V \cdot s}{rad}$
WU	$5.3898e^{-4}$ $\frac{V \cdot s}{rad}$

Table F.7. Calculated back-EMF constant for each line-2-line.

The back-EMF constant K_e is calculated as the mean value for all three line-2-line calculations. The back-EMF constant is calculated using Equation (F.11).

$$K_e = \frac{K_{e.UV} + K_{e.VW} + K_{e.WU}}{3} = 5.32e^{-4} \ \frac{V \cdot s}{rad} \tag{F.11}$$

F.3.5 Discussion

Based on the expected harmonics in a delta-wound BLDC, the back-EMF was not expected to have a pure trapezoidal shape as the theory predicts. A single period of the line WU is

compared to a trapezoidal and sine wave waveform and can be seen in Figure F.8. From the figure, it can be seen that the measured back-EMF waveform resembles the trapezoidal waveform. However, it should be noted that this is not consistent over all periods, where some have a closer representation to the sine-wave waveform.

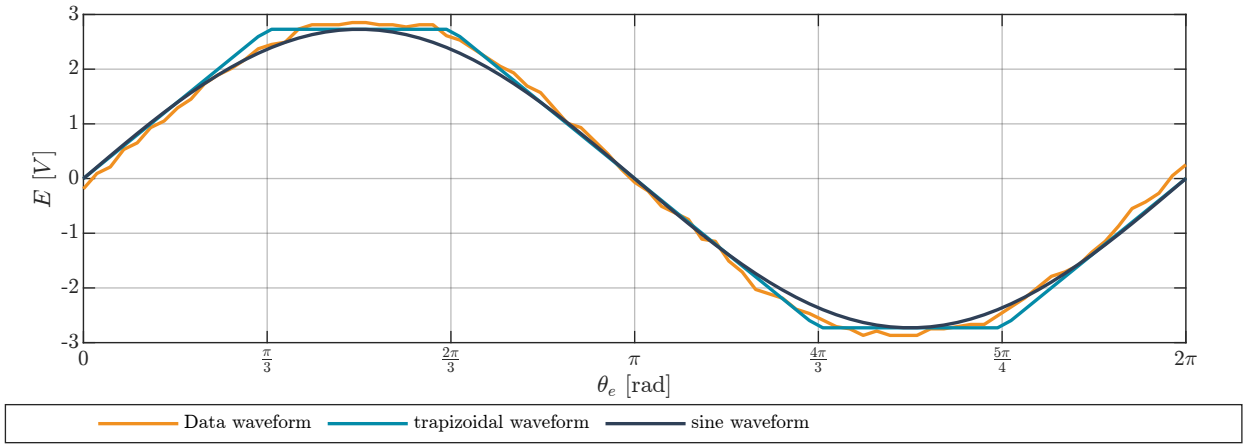


Figure F.8. Comparison of measured back-EMF waveform to trapezoidal and sine wave

For comparison between the K_v number given by Tmotors and the measured K_v number, equation (F.10) is used. $K_{v.m}$ is given by Equation (F.12). Tmotors give the number of poles on the BLDC as $P = 14$ (TMOTOR 2024), and is further confirmed by counting the poles on the motor.

$$K_{v.m} = \left(5.32e^{-4}\right)^{-1} \cdot \frac{60}{2 \cdot \pi} \cdot \frac{2}{14} = 2564 \ \frac{RPM}{V} \tag{F.12}$$

As calculated in equation (F.12) the measured K_v is very close to the expected K_v value given by Tmotors and differs only by a small amount, which could be due to measurement errors.

Referring to Figure F.8, the voltage at the given speed has been calculated that closely resembles the measured voltage, further proving that the calculated back-EMF constant is within a reasonable error of the BLDC motor. The maximum voltage measured and calculated from the back-EMF constant can be seen in Table F.8 at different speeds.

Speed	Measured	Calculated
292 $\frac{rad}{s}$	1.17 V	1.09 V
495 $\frac{rad}{s}$	1.97 V	1.84 V
690 $\frac{rad}{s}$	2.73 V	2.57 V

Table F.8. Measured and calculated back-EMF voltage at a constant speed

Line UV generated an average lower voltage than line VW and WU, as shown in Figure F.7. This is potentially due to an error in the winding or a measurement error.

It is also not possible to take the exact time value when the waveform has a zero crossing, which induces a small measurement error. Noise also influences the peak-to-peak voltage calculations. For both time and peak voltage data selection, selection points equally shifted from the zero crossing minimise this, and the voltage points selected have been approximately in the centre of the noise.

F.3.6 Conclusion

The back-EMF constant was calculated as an average of the measured back-EMF from lines UV, VW, and WU. The BLDC has been driven at three different speeds, and the peak voltages have been measured and compared to the voltage from the back-EMF constant K_e. Generally, the back-EMF constant predicts a slightly lower voltage, but this can be due to inaccuracies in speed measurements.

The $K_{v.m}$ number was calculated to be 2564 $\frac{RPM}{V}$ which is within error of the K_v number given by Tmotors of 2400 $\frac{RPM}{V}$ and the back-EMF constant calculated is therefore expected to be accurate since its only 6.8% off the value provide by the manufacture.

F.4 Measuring inductance of BLDC

The experiment aims to determine the inductance of the BLDC motor. This will be done by aligning to the d-q reference frame, which can then be converted to the ABC reference frame, which is used when modelling the BLDC motor.

F.4.1 Hypothesis

The BLDC has surface-mounted magnets on the rotor, which give inductance characteristics similar to those of a surface-mounted permanent magnet (PMSM) AC motor. For a PMSM, the synchronous inductance is almost equal in the d- and q-axis because the permanent magnets are surface mounted. This results in the relative permeability of the permanent magnets μ_{PM} being approximately equal to that of the relative permeability of the air μ_{air} in the airgap of the rotor (Bobek 2013).

$$\mu_{PM} \approx \mu_{air} \rightarrow L_d \approx L_q \tag{F.13}$$

The inductance L for the d—and q-axes needs to be determined, which requires aligning the rotor with them. The alignment is done according to the currents in the phases at the electrical angle. The d-axis alignment is done at 0° electrical, and the q-axis alignment is done at 90° electrical. A sketch of the currents in the phases can be seen in Figure F.9

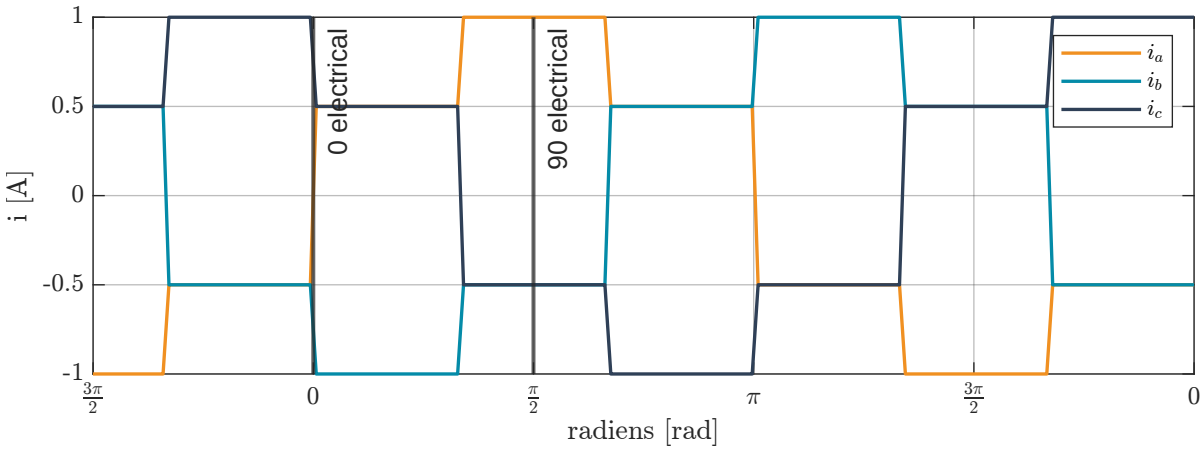


Figure F.9. Phase currents in a delta wound BLDC motor

To align with the d-axis phase c should have a positive current and phase b should have a negative current where phase a is 0. This is done by connecting lines U and W to the positive terminal, and line V is grounded. For q-axis alignment, phase a has a positive current, and phases b and c have a negative current. This is achieved by connecting line U to the positive terminal, line W is grounded, and line V is floating. This can be seen in Figure F.10.

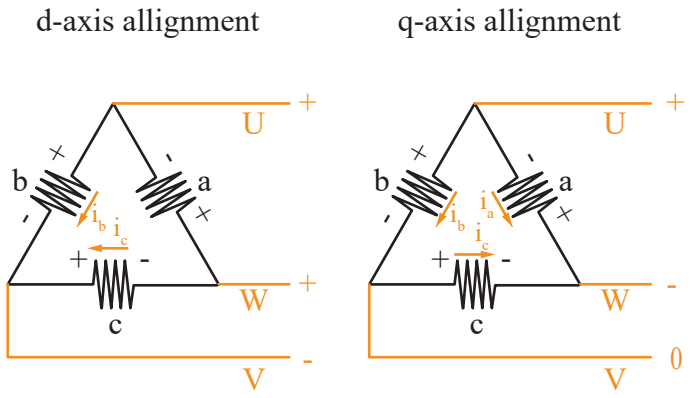


Figure F.10. d- and q-axis alignment of delta wound BLDC

Once the rotor is aligned with its respective axis, it is locked to remove the influence of the back-EMF. The current response of the system is then the first-order RL circuit in equation (F.14).

$$U = R \cdot i + L \cdot \frac{di}{dt} \tag{F.14}$$

The current response is measured through a line where a negative voltage step is given. When the d-axis is aligned, lines U and W are grounded, and line V is connected to the positive terminal. For the q-axis lines V and W are grounded and line U is connected to the positive terminal (Bobek 2013).

Equation (F.15) gives the current response of the circuit.

$$i = \frac{V}{R}(1 - e^{-\frac{t}{\tau}}) \tag{F.15}$$

t is the time, and τ is the circuit's time constant, which describes the system's rise time. τ is $\approx$ 63.2% of steady state. This can be calculated by letting $t = \tau$ which results in $(1 - e^{-1}) \approx 63.2\%$.

The definition of τ can be derived by converting the first-order RL system to the Laplace space and isolating for $\frac{I}{V}$ as seen in Equation (F.16), where in Laplace the time constant of a first-order system is multiplied by s.

$$V = R \cdot I + L \cdot \frac{di}{dt} \rightarrow V = R \cdot I + L \cdot I \cdot s \rightarrow \frac{I}{V} = \frac{1}{L \cdot s + R} \rightarrow \frac{I}{V} = \frac{1}{\frac{L}{R} \cdot s + 1} \tag{F.16}$$

The definition of τ is given by Equation (F.17).

$$\tau = \frac{L}{R} \tag{F.17}$$

By measuring the time constant τ and the resistance, Equation (F.17) can be isolated for L. The inductance measured is that of the equivalent circuit when making the voltage step. When making the voltage step for both the d- and q-axis, the two terminals have the same voltage,

and there is therefore no voltage drop across the phase in between. The equivalent inductance and resistance are therefore two phases in parallel, which can be calculated by Equation (F.18)

$$R_{eq} = \left(\frac{1}{R_s} + \frac{1}{R_s}\right)^{-1} = \frac{1}{2} \cdot R_s \tag{F.18}$$

The inductance is wanted for $L_{d/q}$ where the equivalent inductance is from the power supply seen as $\frac{1}{2} \cdot L_{d/q}$. Inserting this into Equation (F.17) and solving for $L_{d/q}$ results in Equation (F.19).

$$L_{d/q} = 2 \cdot \tau \cdot R_{eq} \tag{F.19}$$

An AA alkane battery has been used for the power supply, as it has minimal influence on the results by being a pure voltage source. A fast switch was used to apply the voltage step. When using a switch, there will be a voltage drop over it. The probes are placed according to Figure F.11 to minimise this influence on the measurements.

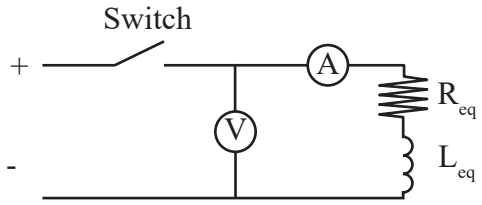


Figure F.11. circuit used for measuring current rise and voltage step.

The conversion from the induction in the dq-axis reference frame to the ABC frame used for modelling can be assumed to be $L_d \approx L_q \approx L_s$.

F.4.2 Material list

The following materials have been used for the experiment.

Item	Manufacture & model	Number of items
• Digital Oscilloscope	Tektronix TBS 2000B SERIES	1
• Voltage probe	Tektronix TPP0100 Voltage probe	1
• Current probe	Tektronix TCP0030A	1
• BLDC motor	Tmotor F40proIV	1
• AA battert	GP Super G-Tech	1
• Motor mount		1
• Rotor blocker		1

F.4.3 Exsperimental procedure

- Turn on the oscilloscope and connect the voltage and current probes. The voltage probe is calibrated to the correct attenuation by connecting the probe to the probe comp and ground. The calibration is done when the 5 V square wave matches up with the 5 V line.

- Depending on whether the d- or q-axis inductance is wanted to be measured, the rotor is aligned with that axis.
- The rotor is locked, and the current and voltage probes are connected. The current probe is connected to a single wire carrying current through the BLDC. The voltage probe is connected to the power supply.
- A negative step voltage is applied. The current runs in the opposite direction from when the rotor is aligned.
- The current step response is measured on the oscilloscope. From where the inductance can be calculated using Equation (F.19).

F.4.4 Results

The measured current responses and voltage steps for d- and q-axis alignments are seen in Figure F.12. The horizontal back line noted as τ_{line} is when the current reaches the time constant threshold of $\approx 63.2\%$.

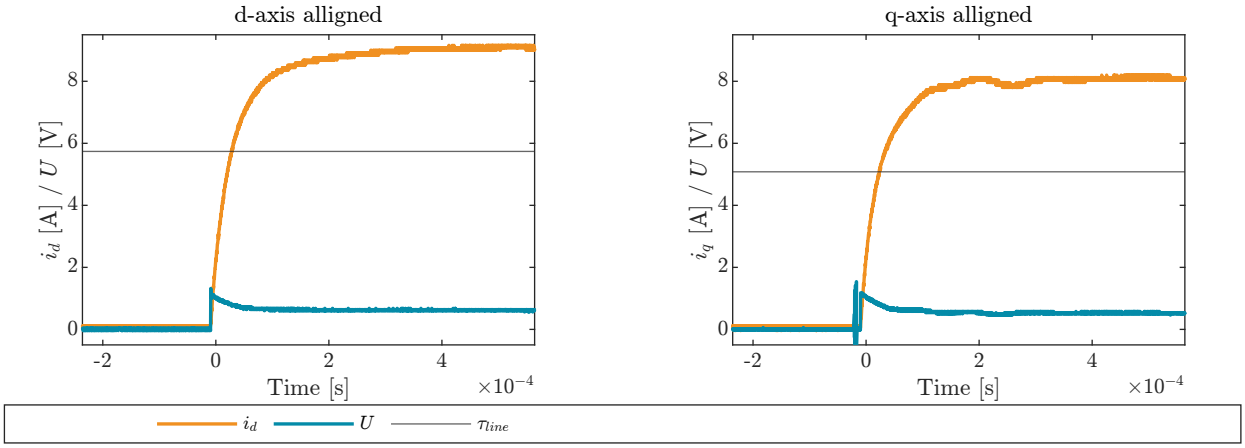


Figure F.12. Current response voltage step when the rotor is aligned to the d- and q-axis

Two time measurements are required to calculate the time constants for each alignment. The first is when the step starts, and the second is when the current crosses τ_{line}. Subtracting the two times from each other results in time constants τ_d and τ_q.

Table F.9 contains the measured times for calculating the time constant and the calculated time constant for each alignment.

	T_{step}	$T_{\tau_{line}}$	τ
d-axis	$-8.82e^{-6}$ s	$2.80e^{-5}$ s	$3.68e^{-5}$ s
q-axis	$-9.55e^{-6}$ s	$2.37e^{-5}$ s	$3.33e^{-5}$ s

Table F.9. Time steps used for calculating the time constant and time constant

The equivalent resistance is calculated with Equation (F.18) and the phase resistance measured in Appendix section F.2, and Equation (F.19) is used to calculate the inductance in the d- and

q-axis. The d- and q-axis inductances are calculated in Equation (F.20).

$$\begin{aligned} L_d &= 2 \cdot 3.68e^{-5}\ s \cdot 0.0398\ \Omega = 2.93e^{-6}\ H \\ L_q &= 2 \cdot 3.33e^{-5}\ s \cdot 0.0398\ \Omega = 2.64e^{-6}\ H \end{aligned} \tag{F.20}$$

For calculating the phase inductance, the average of L_d and L_q is used and is calculated in Equation (F.21).

$$L_s = \frac{L_d + L_q}{2} = 2.79e^{-6}\ H \tag{F.21}$$

F.4.5 Discussion

Referring to Figure F.12, the q-axis current and voltage have some fluctuations. These inconsistencies could influence the calculations of the inductance. It is deemed that this will have minimal influence as the current has a smooth curve from the step up over τ_{line}, and the steady state value used for calculating the location of τ_{line} is also smooth. The fluctuations should therefore not influence the results.

In Figure F.12, it is also observed that the voltage on the d and q axes decreases after the initial step. The peak of the step voltage is ≈ 1.18V, which is lower than the 1.5V the battery is rated to deliver.

To validate that the calculated inductances are correct, Equation (F.15) can be applied to validate that the calculated inductances are correct. The equivalent resistance, the steady-state voltage, and the time constant are inserted. The step response for the currents on the d and q axes is divided by $\sqrt{3}$ to convert the delta wound line currents to phase currents $i_{line} = i_{phase} \cdot \sqrt{3}$. The step response for the aligned currents on the d and q axes can be seen in Figure F.13.

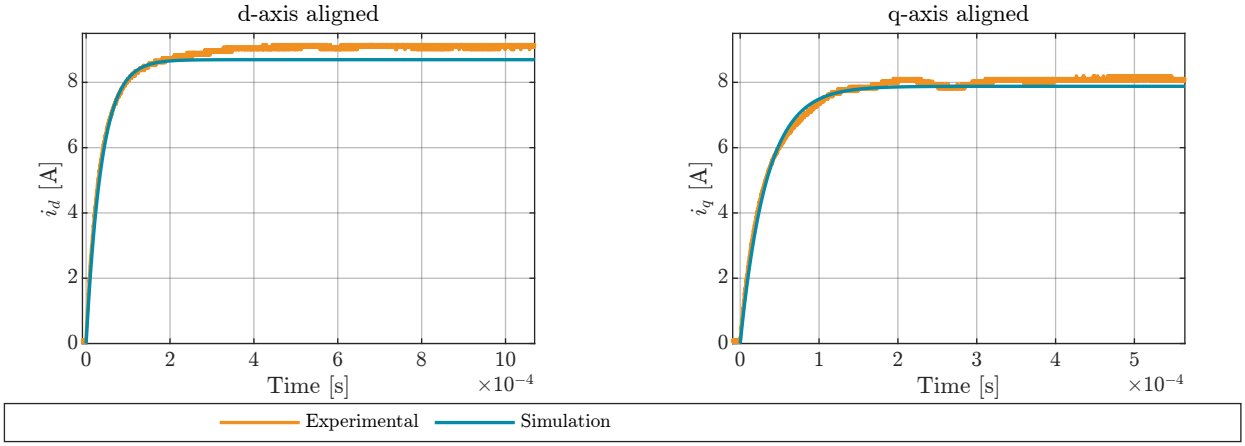


Figure F.13. Current step response for d- and q-axis aligned currents.

The curvatures for the currents on the d- and q-axes match the measured currents. The simulated d-axis currents do not reach the same steady-state value as the experimental data. This is believed to be caused by the resolution of the oscilloscope, which has been identified to be 0.04 V, and because the resistance is so low, a small change of 0.04V will result in ≈ 0.94 A. The calculated current is then divided by $\sqrt{3}$, so the change would be ≈ 0.54 A.

Furthermore, the experimental d-axis current rises after the simulated result finds a steady state. This potentially increases the time constant as the τ_{line} is at a higher current. This is not deemed the case because the experimental and simulated data match very well during the rise.

For the q-axis aligned currents, the results match well. The experimental and simulated results diverge slightly between $0.5e^{-4} - 1e^{-4}$ s, most likely due to the same fluctuations observed on the experimental data at steady state.

F.4.6 Conclusion

In Equation (F.20), the inductances of the d and q axes have been calculated, whereas the single-phase inductance L_s has been calculated in Equation (F.21) to be given as $2.79e^{-6}$ H. The results have been validated in Figure F.13, where it is deemed that the calculated time constants match the experimental data and thus that the calculated inductances are accurate to the BLDC.

Through the validation of the time constants, some measurement errors were observed, likely due to the experimental procedure used in this experiment. The influence on the results is considered minimal from the validation of the time constants in Figure F.13.

F.5 Estimating mechanical parameters of BLDC

This series of experiments aims to determine the mass moment of inertia, viscous, and Coulomb frictions. Additionally, the propeller's resistive torque is speed-dependent. The mass moment of inertia, friction, and torques can be determined by performing a series of calculations and experiments.

F.5.1 Hypothesis

The mechanical model for the BLDC motor is given by equation (F.22).

$$J_{tot} \cdot \dot{\omega}_m = \tau_e - \tau_w - C_v \cdot \omega_m - C_c \cdot sign(\omega_m) \tag{F.22}$$

J_{tot} is the total inertia of the BLDC motor's rotor and the propeller, τ_e is the torque produced by the motor, τ_w is the work performed by the motor in the application. When there is no load $\tau_w = 0$, but when a propeller is connected, τ_w can be found by Equation (F.23).

$$\tau_w = C_p \cdot \omega^2 \tag{F.23}$$

C_p is the resistive torque coefficient for the propeller, C_v is the viscous friction coefficient, and C_c is the Coulomb friction coefficient. ω_m is the rotor velocity in $\frac{rad}{s}$ and $\dot{\omega}_m$ is rotor acceleration in $\frac{rad}{s^2}$

Firstly, the moment of inertia needs to be estimated. This can be done for the rotor J_r and the propeller J_p, see Equation (F.26). The rotor moment of inertia can be estimated by Equation (F.24), where all the mass is assumed to be distributed as an outer ring. The propeller inertia is calculated by Equation (F.25), where the inertia of each blade is calculated as a bar with uniform mass density. The inertia is then multiplied by the number of blades.

$$J_r = \frac{1}{2} \cdot m_r \cdot (r_{r.i}^2 + r_{r.o}^2) \tag{F.24}$$

$$J_p = N_b \cdot (\frac{1}{3} \cdot \frac{m_p}{N_b} \cdot r_b^2) \tag{F.25}$$

$$J_{tot} = J_r + J_p \tag{F.26}$$

m_r is the rotor mass, m_p is the total mass of the propellor, N_b is the number of blades in the propellor, $r_{r.i}$ is the inner radius of the outer ring, $r_{r.o}$ is the outer radius of the outer ring and r_b is the length of a blade on the propellor from the axis of rotation. A sketch of the measurements used in the calculation of the mass moment of inertia can be seen in Figure F.14

Figure F.14. The dimensions used for calculating rotor- and propeller inertia

The friction will stay constant no matter the load. Therefore, it is possible to identify viscous and Coulomb frictions by running the motor at no load, where all the torque produced is used to counteract the friction. A speed curve can be measured by measuring the speed of the motor from a constant speed and then cutting power to let the motor freewheel until it stops. The only torque slowing down the motor is the friction, and by using optimisation tools, it is possible to fit C_v and C_c in a mechanical model so that the dynamics seen in the experiment can be recreated. This test is done at no load and therefore without the propeller, and Equation (F.27) is used to fit the data.

$$J_r \cdot \dot{\omega} = -C_v \cdot \omega - C_c \cdot sign(\omega) \tag{F.27}$$

When C_v and C_c have been identified, a similar experiment can be conducted to identify C_p, where the motor is run at a constant speed and then let freewheel until it stops. An optimisation tool is used to fit C_p to the measured curve. The equation used to fit C_p is Equation (F.22) where $\tau_e = 0$. The optimisation tool used in this test is the "Parameter estimator" in Simulink.

F.5.2 Material list

The following materials have been used for the experiment.

Item	Manufacture & model	Number of items
• BLDC motor	Tmotor F40proIV KV2400	1
• ESC	TMOTOR F60A MINI 4IN1	1
• Battery	Turnigy Graphene 2200mAh 4S 45C Lipo Pack	1
• Hall sensor		1
• Propellor	T5146	1
• Motor mount		1

F.5.3 Exsperimental procedure

- The mass of the propeller and rotor is measured and the moment of inertia estimated with Equation (F.24), (F.25), and (F.26).

- The motor is sped up by applying specified throttle values with and without a propeller. When the motor has reached a steady state, the power to the motor can be cut. In the case of this report, the power supply is a battery, so the power supply is cut at the anti-spark plug. The slowdown is measured through the hall sensor until the rotor stops.
- The previous item is repeated for several different throttle values to verify the friction fit at all speeds.
- C_v and C_c are first fitted. In this report, Matlab's curve-fitting tool is applied.
- C_p is fitted with Matlab's curve-fitter tool.

(Specified Throttle values: $[10, 20, 30, 40, 50, 60, 70, 80, 90, 95]$)

F.5.4 Results

The rotor is measured to weigh 11.36 g and has an outer/inner diameter of 27.9 mm/23 mm. The mass moment of inertia can be estimated as $1.85e^{-6}$ $\frac{kg}{m^2}$. The propeller has been measured to weigh 5 g and a blade length of 65 mm, resulting in a mass moment of inertia of $7.0417e^{-6}$ $\frac{kg}{m^2}$. To get a more precise rotor mass moment of inertia, the rotor was modelled in SolidWorks, see Figure F.15. Here the mass was estimated to be 12.92 g when the hub is made from steel, the shaft from titanium and the magnets of neudunium this results in a mass moment of inertia of $1.6e^{-6}$ $\frac{kg}{m^2}$ it can be seen that since more mass is placed closer to the centre the moment of inertia is smaller, therefore, the Solidwork estimate is used.

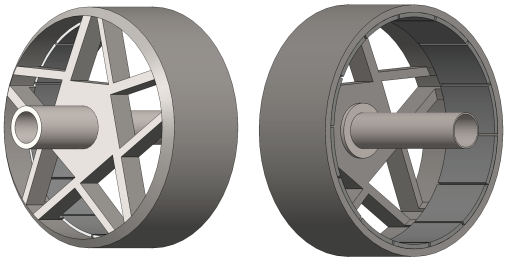


Figure F.15. CAD model of the motor's rotor

The test where no load was placed on the motor resulted in the dynamics seen in Figure F.16. This data was then run through the parameter estimator on the 50% throttle test. This resulted in simulation results in Figure F.16. It was estimated that column friction is given by $C_c = 5.955e-04\ Nm$. The viscous friction is provided by $C_v = 3.374e-06\ Nm \cdot s$ it can be seen that the simulation fits well at 50% throttle and in general gives an okay approximation from 30% throttle to 80% throttle It can be seen that in general that the lower throttle values below 50% throttle seam to stop too fast and the Throttle values above 50% seams to stop too late.

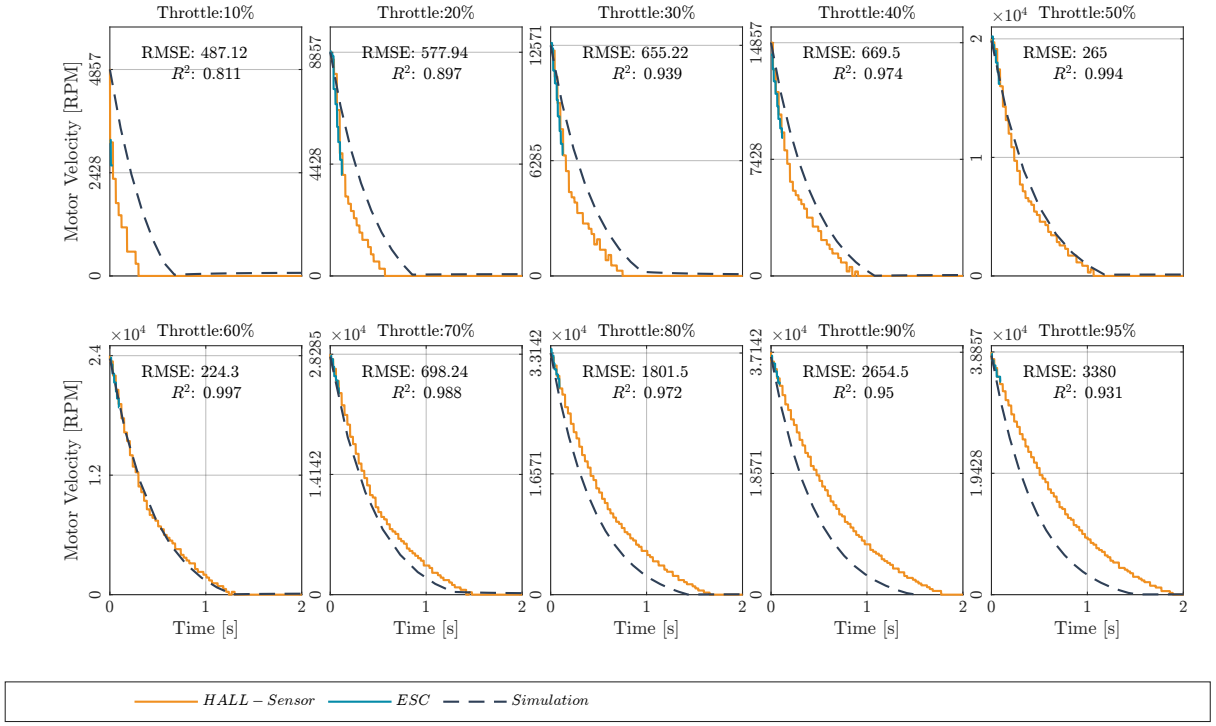


Figure F.16. Results of fit of C_v and C_c at different throttle percentages

The propeller was mounted on the motor, and the test was run at only throttle values: $[40, 50, 60, 70]$. The results can be seen in Figure F.17. This data was also run through the parameter estimator on the 50% throttle test. From this, it was determined that $C_p = 2.5442e-8\ Nm \cdot s^2$. This resulted in the simulations that can be seen in Figure F.17. Here, it can be seen that all four of the test seams seem to fit rather well.

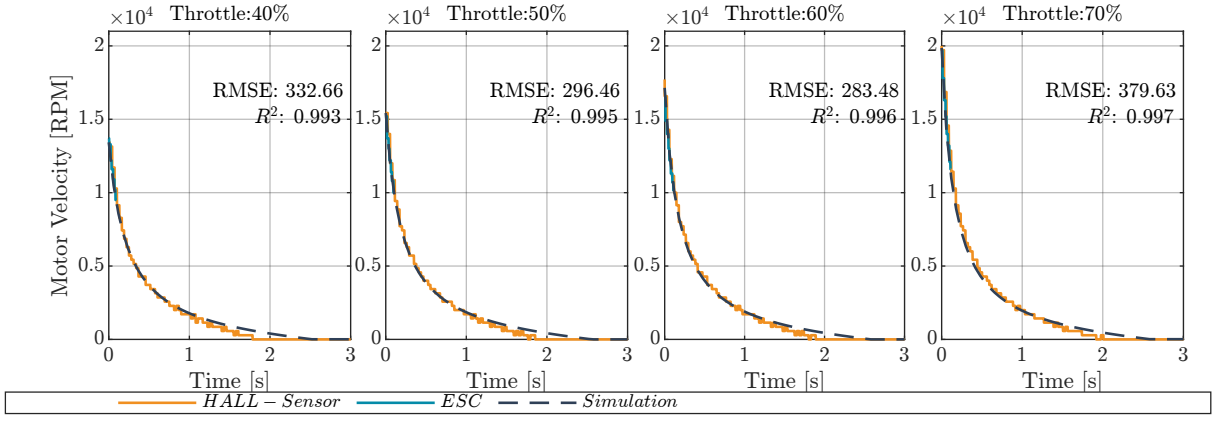


Figure F.17. Result of fit of C_p at different throttle percentages.

F.5.5 Discussion

As seen in Figure F.16, the friction seems too low at lower speeds and too high at higher speeds. This indicates that the friction affecting the motor is not linear, since this would not be fixed by changing the slope of the friction. However, the simulations' results seem close to the experimental results at 40−70% throttle. Therefore, the Coulomb friction of $C_c = 5.955e{-}04\ Nm$ and the viscous friction of $C_v = 3.374e-06\ Nm \cdot s$ give a fair approximation of the system's

dynamics. It can also be said that the coefficient for the load of $C_p = 2.5442e-8\ Nm \cdot s^2$ also provides a simulation with results reasonably close to the experimental results.

F.5.6 Conclusion

It can be concluded that the motors' dynamics when no load can be estimated by equation (F.27) with a Coulomb friction of $C_c = 5.955e-04\ Nm$ and the viscous friction of $C_v = 3.374e-06\ Nm \cdot s$. The motors with load can be estimated by equation (F.22), where the coefficient for the load of $C_p = 2.5442e-8\ Nm \cdot s^2$.

F.6 Thrust test

This test aims to determine what thrust is generated by the motor at different motor velocities/throttle percentages. The motor has therefore been tested with the propeller mounted at different throttle percentages to see the corresponding motor velocity and thrust.

F.6.1 Hypothesis

Section 2.3 shows that the more power added to the propeller, the more thrust is generated. This should mean that since J_{tot} is constant, and from Appendix Section F.5 it is known that the moment applied to the propeller from friction and load is increased. It's known that power in a mechanical system can be found by $P = \tau_e \cdot \omega_m$, so when both the torque and the motor velocity increase, the thrust will increase.

A load cell can measure thrust as a weight in grams. This is done by taring the load cell with the weight of the motor and the propeller on it. When the propeller spins and generates thrust, it can be measured on the load cell as a load in grams. The kilogram measurement can then be multiplied by g to find the thrust in N.

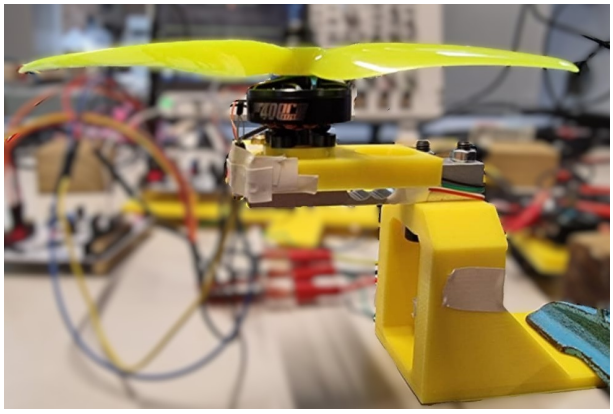


Figure F.18. The test setup with the motor mounted on the load cell, and the propeller mounted to the motor. A Hall sensor can be seen mounted with tape

F.6.2 Material list

The following materials have been used for the experiment. Figure F.18 shows a picture of the experimental setup.

Item	Manufacture & model	Number of items
• BLDC motor	Tmotor F40proIV KV2400	1
• ESC	TMOTOR F60A MINI 4IN1	1
• Battery	Turnigy Graphene 2200mAh 4S 45C Lipo Pack	1
• Hall sensor		1
• Propellor	T5146	1
• Motor mount		1
• Load cell	TAL220	1
• Load cell amplifier	HX711	1
• Load cell mount		1

F.6.3 Exsperimental procedure

- The load cell is calibrated with a known weight.
- The load cell is tared so that the load cell gives a reading of 0 g before the test starts.
- The motor is then run at specified throttle percentages, and the velocities and measured thrust are noted.

(Specified throttle values: $[-95, -90, -80, -70, -60, -50, -40, -30, -20, -17.5, -15, -12.5, -10, -7.5, -5, -2.5, 2.5, 5, 7.5, 10, 12.5, 15, 17.5, 20, 30, 40, 50, 60, 70, 80, 90, 95]$)

F.6.4 Results

The following graphs can be made, where Figure F.19 shows the motor velocity achieved at the different throttle percentages. The results are compared with the values given on T-motor's website for the motor with this specific propeller (TMOTOR 2024).

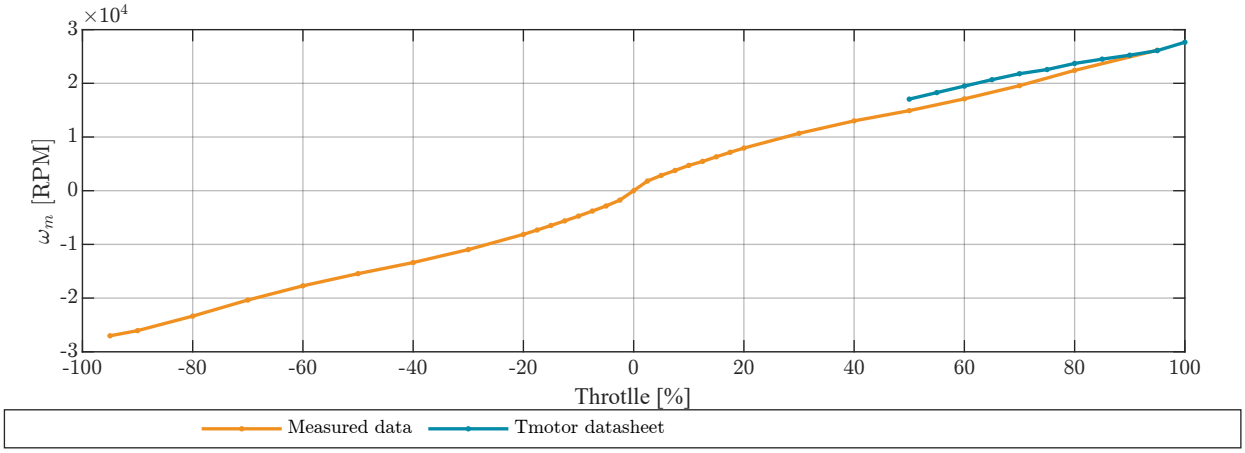


Figure F.19. Throttle percentage and corresponding motor velocity in RPM

The graph in Figure F.20 shows the thrust achieved at the different throttle percentages.

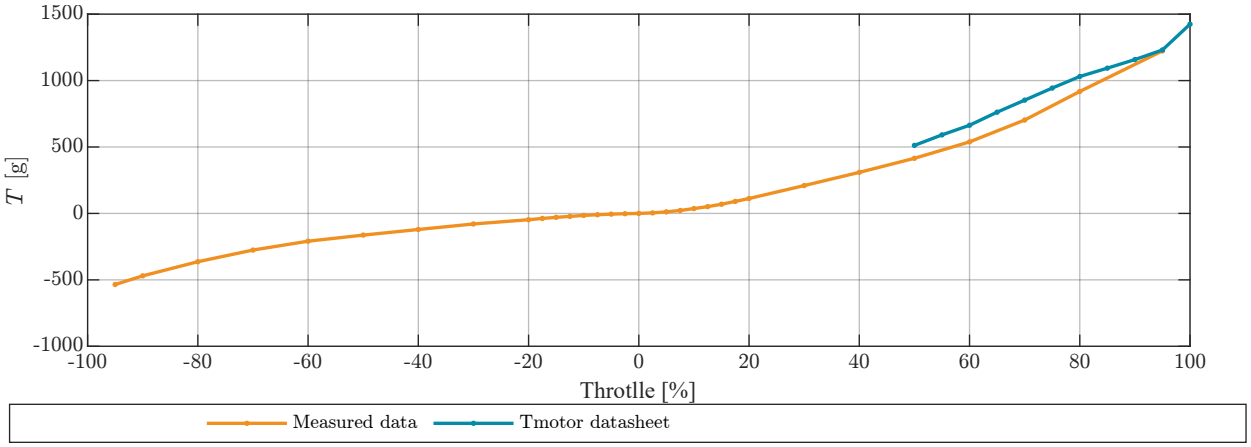


Figure F.20. Throttle percentage and corresponding thrust in grams

The last graph in Figure F.21 shows the thrust achieved at different motor velocities ω_m.

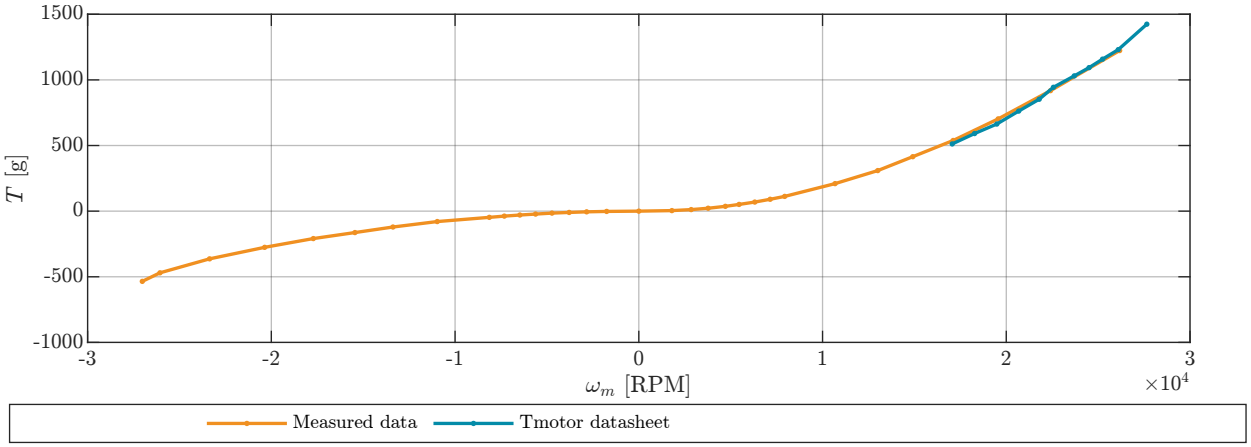


Figure F.21. Motor velocity in RPM and the corresponding thrust in grams

The maximum thrust is measured in the positive direction as 13.98 N, and the maximum thrust in the negative direction as 5.25 N.

F.6.5 Discussion

It can be seen in Figure F.21 that the thrust generated at a velocity matches the data given by the manufacturer at the higher motor velocities. This indicates that the measured thrusts correspond to the manufacturer, even though the archived velocities at specific throttle percentages in Figure F.21 differ slightly. This is mainly based on battery charge level. Since the battery voltage lowers, the motor velocity should decrease. In this test, a single fully charged battery was used, and it might have discharged a bit during the test, impacting the results. However, it can still be seen that the results match the manufacturer's.

F.6.6 Conclusion

It can be concluded that the relation between motor velocity and thrust has been found. The motors do not produce equal thrust in the positive and negative directions, the maximum positive thrust is 13.98 N, and the maximum negative thrust is 5.25 N.

Controller Simulation G

The BLDC motor controller tuned in Section 6.1 was tested in simulation to see how well the controller behaves when given a continuous input. This appendix uses the same frequencies as those in the controller tests. Therefore, a sinusoidal wave of amplitude of 500 rad ($4.77e^3$ RPM) with an offset of 1250 rad ($1.19e^4$ RPM) is given at frequencies of 0.4 Hz, 1.2 Hz, 2.4 Hz. This can be seen in Figure G.1.

Here it can be seen that at 0.4 Hz, the simulations are out of phase just like the test on the real system seen in Subsection 6.1.2. Using the "asin" function, it can be calculated that the simulated controller is 63 ms out of phase, whereas the implemented system is 67 ms out of phase. The simulation stays out of phase at faster frequencies with 63 ms, whereas the implemented system increases to $\approx$ 80 ms.

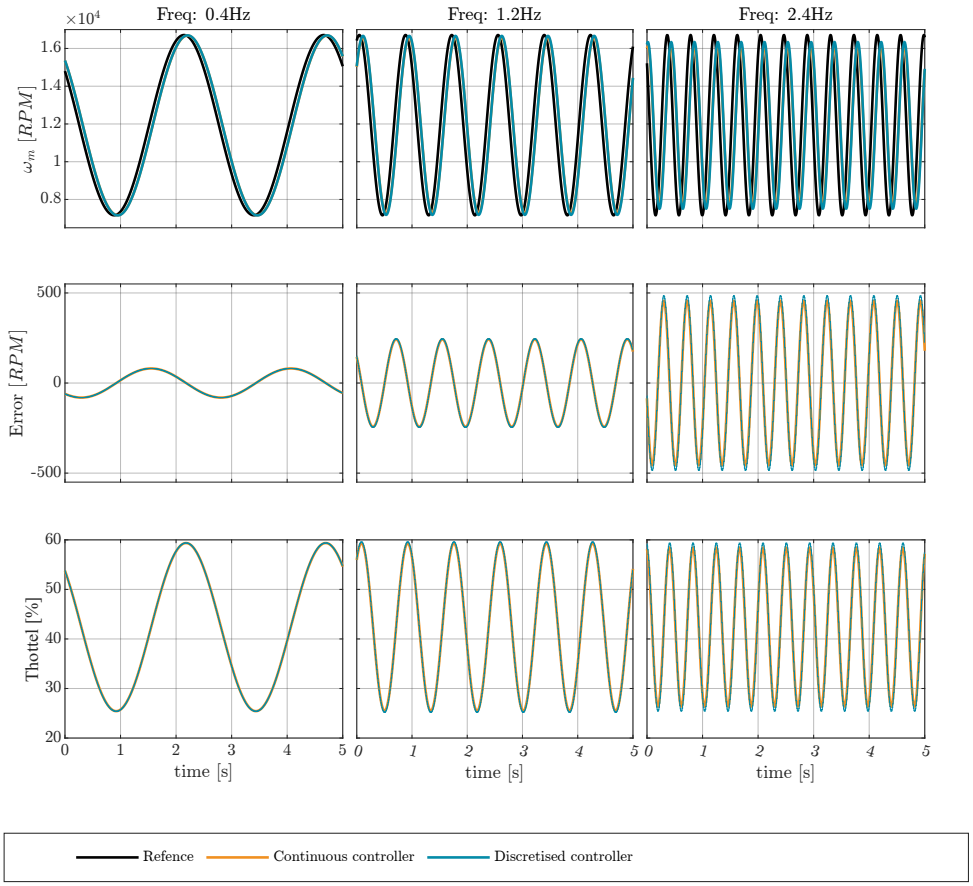


Figure G.1. Simulation results of BLDC controller at frequencys: $0.4Hz, 1.2Hz, 2.4Hz$

When given a reference with a maxium amplitude of $1.671e^4$ RPM at frequencies of 0.4 Hz, 1.2 Hz, 2.4 Hz the maximum measured amplitudes are $1.695e^4$ RPM,$1.668e^4$ RPM & $1.641e^4$ RPM. The errors are therefore within $\pm$500 RPM, which is a maximum error of $\approx$ 3%, This is similar to the implemented system. The largest difference between the simulation

and the implemented system is that the throttle is $\approx 6\%$ higher on the implemented system. The simulation, therefore, gives a higher velocity at a lower throttle, whereas the implemented system requires more throttle for the same results. This is most likely due to the battery voltage not being constant, therefore, requiring more throttle.

It can be seen that the controller tuned in Section 6.1 seems to be able to follow sinusoidal inputs up to $2.4Hz$, which should be enough to follow potential inputs from the pendulum controller.

To show that the discretised controller works, a C Function block was used in simulation, where the discretised controller was implemented with the following code:

```
float Rad_motor;
float w_ref;
float w_error_0_0,w_error_1_0;
float U_0_0,U_1_0;

w_error_1_0 = w_error_0_0;
w_error_0_0 = w_ref - Rad_motor;

U_1_0 = U_0_0;

U_0_0 = 0.00036125*w_error_0_0 + 0.00036125*w_error_1_0 + 1*U_1_0;
```

This Function block was set to run at a sample time of $85ms$ to match the sample time of the microcontroller. To get a closer look of the discretised controller a zomed in view of the simulations can be seen in Figure G.2.

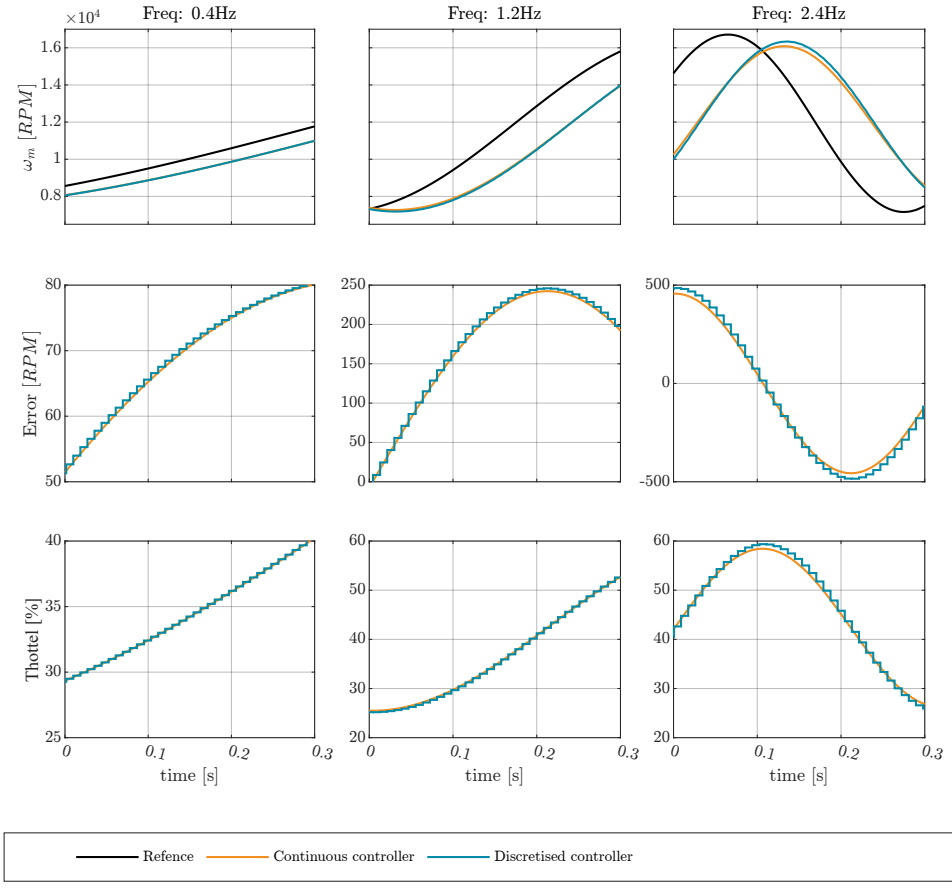


Figure G.2. Zomed in simulation results of BLDC controller at frequencies: $0.4Hz, 1.2Hz, 2.4Hz$

Although the reference and simulations differ, the continuous and discrete controllers match well. The discrete controller is deemed correctly implemented and can, therefore, be implemented on the MCU.

DSHOT H

In this Appendix some of different commands that the ESC can recieve can be seen listed in Tabel H.1. The Code for generating the DMA BUFFER is showed in a simplified state.

Nr.	Command	Remark
0	DSHOT_CMD_MOTOR_STOP	
1	DSHOT_CMD_BEEP1	Wait at least length of beep (260ms) before next command
2	DSHOT_CMD_BEEP2	Wait at least length of beep (260ms) before next command
3	DSHOT_CMD_BEEP3	Wait at least length of beep (260ms) before next command
4	DSHOT_CMD_BEEP4	Wait at least length of beep (260ms) before next command
5	DSHOT_CMD_BEEP5	Wait at least length of beep (260ms) before next command
6	DSHOT_CMD_ESC_INFO	Need 6x
7	DSHOT_CMD_SPIN_DIRECTION_1	Need 6x
8	DSHOT_CMD_SPIN_DIRECTION_2	Need 6x
9	DSHOT_CMD_3D_MODE_OFF	Need 6x
10	DSHOT_CMD_3D_MODE_ON	Need 6x
11	DSHOT_CMD_SETTINGS_REQUEST	
12	DSHOT_CMD_SAVE_SETTINGS	Need 6x wait (35ms) before next command
20	DSHOT_CMD_SPIN_DIRECTION_NORMAL	Need 6x
21	DSHOT_CMD_SPIN_DIRECTION_REVERSED	Need 6x

Table H.1. Tabel over DSHOT Commands (note Telemetry bit needs to be on)

The following code is for generating a singel DMA buffer. In the full code this is part of a function that can be run for each of the four motors. The CRC calculation is based on (Landa 2021).

```
uint16_t DMA_BUFF[18];
uint16_t value;
uint16_t packet;
bool Telemerty;
unsigned crc = 0;

//Add telemerty bit to packet;
packet = (value << 1)|(Telemerty ? 1 : 0);

// calculate checksum
crc = (packet ^(packet >> 4) ^ (packet >> 8)) & 0x0f;

// Attach checksum to packet
packet = (packet << 4) | CRC;

// convert bit to PWM length 14 for 1 and 7 for 0
for(int i = 0; i < FRAME_SIZE; i++){
DMA_BUFF[i] = (packet & 0x8000) ? 14:7;
packet <<=1;
}

// Two empty slots for in-between frames ()
DMA_BUFF[16] = 0;
DMA_BUFF[17] = 0;
```

Sensor implementation I

This appendix will describe how the control registers are configured, and how the values have been decided. A description of how the sensors have been calibrated will also be given.

I.0.1 Sensor inilisation

The gyroscope and accelerometer are combined in the sensor package "LSM6DS3" located on an Arduino Nano 33 IoT. The magnetometer is on a separate chip, "QMC5883L", on a separate board. Communication to the sensors is done via I^2C, enabling the single I^2C channel on the Arduino board to communicate with all sensors. Libraries have been applied and modified for the required use case for communication with the sensors. For communication with the "LSM6DS3" sensor, the "Arduino_LSM6DS3" library is used (Tillisch 2024), and for the "QMC5883L" sensor, the "QMC5883L Compass" Arduino Library is used (MPrograms 2023). Also, a linear algebra library has been used to simplify the multiplication of the matrices required for the EKF described in Section 7.4. The library is called "Basic Linear Algebra" (tomstewart89 2024).

Each sensor needs its control registers configured for operation. "Full scale" and "output rate selection" are two primary settings that must be chosen optimally for the specific use case. The data registers from which the measured data is read are each two bytes of unsigned integers. When selecting a higher scale, larger values can be measured, but this comes at the cost of resolution and an increase in noise. Likewise, increasing the output rate also increases noise (Tillisch 2024, p. 21). The output rate is selected to match the maximum frequency at which the Arduino Nano 33 IoT can run the code, which has been determined to be $100Hz$. The scale selection is based on the expected measurements that the system will be exposed to. The configuration of each sensor can be seen in Table 7.3. The units are later converted to SI units where the gravitational constant is converted to meters per second $g = 9.8\frac{m}{s^2}$, degrees per second are converted to radians per second $dps \approx \frac{0.0175rad}{s}$, and Gauss is converted to milli tesla $G = 100mT$.

I.0.2 Sensor calibration

The MEMS accelerometer measurements can be modelled by Equation (I.1) (Ghanipoor, Hashemi, and Salarieh 2020, p. 4133), and the same can be done for the MEMS gyroscope by Equation (I.2) (Ghanipoor, Hashemi, and Salarieh 2020, pp. 4132–4133), and MEMS magnetometer by Equation (I.3). (Ozyagcilar 2015, p. 3). a_g is the gravitational acceleration, a_l is linear acceleration including tangential accelerations, a_c is centripetal acceleration, which is present if the sensor is offset from the axis of rotation, b_{acc} is measurement bias, and η_{acc} is accelerometer measurement noise. The gyroscope measures the true rotational rates ω_{true}

with bias b_{gyro} and measurement noise η_{gyro}. B_p is the magnetic field vector measured by the magnetometer after rotations in yaw (Ψ), pitch (θ), and roll (ϕ), where B is the magnitude of the geomagnetic field vector and δ is the magnetic inclination. W and V respectively represent soft- and hard-iron effects and η_{mag} is measurement noise (Ozyagcilar 2015, pp. 5–9). The noise is assumed to be zero-mean Gaussian noise, which is additive (Titterton and Weston 2004, pp. 511–515)

$$\vec{a}_{acc} = a_g + a_l + a_c - b_{acc} + \eta_{acc} \tag{I.1}$$

$$\vec{\omega}_{gyro} = \omega_{true} - b_{gyro} + \eta_{gyro} \tag{I.2}$$

$$\vec{B}_p = W \cdot R_x(\phi) \cdot R_y(\theta) \cdot R_z(\Psi) \cdot B \cdot \begin{bmatrix} cos(\delta) \\ 0 \\ sin(\delta) \end{bmatrix} + V + \eta_{mag} \tag{I.3}$$

A low-pass filter (LP filter) can suppress most high-frequency noise. The low-pass filter is shown in Equation (I.4), where A is a value between $0 - 1$, in is the raw sensor input, and out is the filtered output. n indicates the iteration.

$$out_{filt.n} = A \cdot in_n + (1 - A) \cdot out_{filt.n-1} \tag{I.4}$$

The bias for the accelerometer and gyroscope can be measured by collecting m number of measurements and calculating the mean when the sensors are level and stationary. The expected measurements for the accelerometer and gyroscope are $\vec{a}_0 = [0, 0, -g]^T$ and $\vec{\omega}_0 = [0, 0, 0]^T$, and the biases are the offset from the values. The biases can be calculated by Equations (I.5).

$$\begin{aligned} b_{acc} &= \frac{\sum_{n=1}^{m} \vec{a}_{acc.n}}{n} - \vec{a}_0 \\ b_{gyro} &= \frac{\sum_{n=1}^{m} \vec{\omega}_{gyro.n}}{n} - \vec{\omega}_0 \end{aligned} \tag{I.5}$$

Calibrating the magnetometer is more difficult as it is affected by hard- and soft-iron effects. Measurements from a perfect magnetometer completely undisturbed by any hard- or soft-iron effects would result in a sphere centred at $(0, 0, 0)$. The sensor is located on a PCB which has permanently magnetised ferromagnetic components, which will offset the magnetometer reading. This effect is the hard-iron effect and can be compensated for by finding the offset and subtracting it from the measurements. This is only possible when the permanently magnetised ferromagnetic components rotate with the sensor, resulting in the same offset no matter the rotation. The soft-iron effects result from interfering magnetic fields induced by the geomagnetic field onto normally unmagnetised ferromagnetic components on the PCB (Ozyagcilar 2015, p. 2). The normally unmagnetised ferromagnetic components are still rotating with the sensor, but the geomagnetic field induces their magnetic field and thereby are dependent on the orientation of the PCB. The soft-iron effects transform the sphere into an ellipsoid.

The magnetometer calibration can be done with Equation (I.6), where W^{-1} is the inverse soft-iron matrix transforming the ellipsoid back into a sphere. V is the hard-iron effect offset of

the measurements and is subtracted from the raw magnetometer measurements B_p. B_c is the calibrated measurements (Ozyagcilar 2015, pp. 5–9).

$$B_c = W^{-1}(B_p - V) \tag{I.6}$$

Due to the complexity of the magnetometer calibration, it was done in Matlab by using the "magical" function, which returns the W^{-1} matrix and V vector. Raw magnetometer measurements were recorded on a PC while rotating the magnetometer sensor in all orientations. In Matlab, the "magical" function accepts the magnetometer measurements as a vector for each set of measurements. Figure I.1 shows the before B_p and after B_c of the magnetometer calibration.

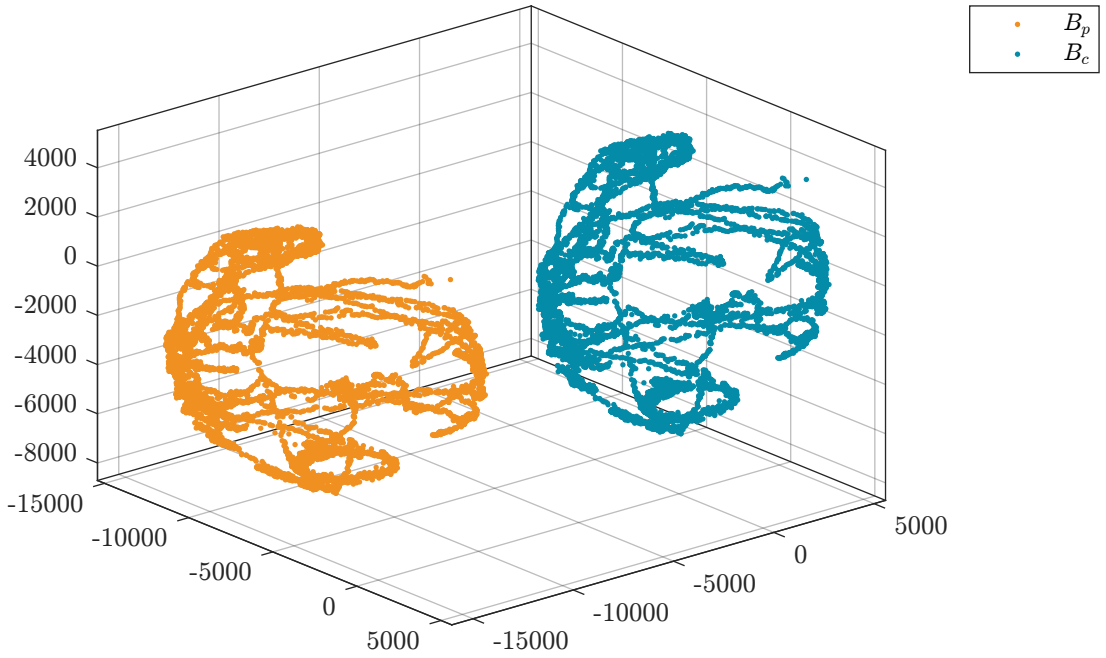


Figure I.1. Magnotometer data before (B_p) and after (B_c) calibration.

The values for V and W^{-1} can be seen in Equation (I.7).

$$V = \begin{bmatrix} -9746.457 \\ -9819.693 \\ -3622.959 \end{bmatrix}, \; W^{-1} = \begin{bmatrix} 0.9831 & -0.0247 & 0.0170 \\ -0.0247 & 0.9453 & 0.1057 \\ 0.0170 & 0.1057 & 1.0890 \end{bmatrix} \tag{I.7}$$

Test of System - addtional test J

In the following Appendix, the test results for the swing tests starting at a global swing plane angle of 270° are shown.

SPS - system start at 270°with wind & no load

In Figure J.1, the pitch swing results with the initial global swing plane angle of 270°. The tests reached stability in 8.42 s for the 5 $\frac{m}{s}$ test, 8.26 s for the 7.5 $\frac{m}{s}$ test, and the 10 $\frac{m}{s}$ test does not reach stability. A summary of the results and initial swing angle is shown in Table 8.2.

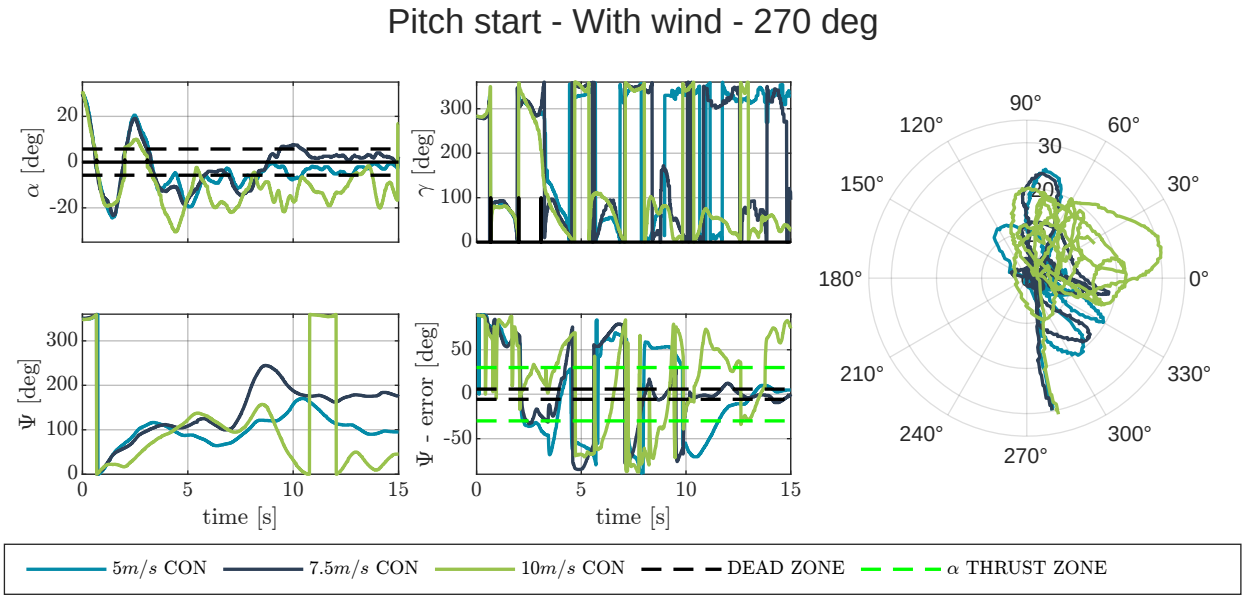


Figure J.1. Plot of SPS attitude measurements doing pitch disturbance test with wind starting at swing angle 270° (Video: `https://www.youtube.com/watch?v=f-xvM_0X6iE`)

In Figure J.2, the roll swing results with the initial global swing plane angle of 270°. The tests reached stability in 3.93 s for the 5 $\frac{m}{s}$ test, 3.01 s for the 7.5 $\frac{m}{s}$ test, and the 10 $\frac{m}{s}$ test does not reach stability. A summary of the results and initial swing angle is shown in Table 8.2.

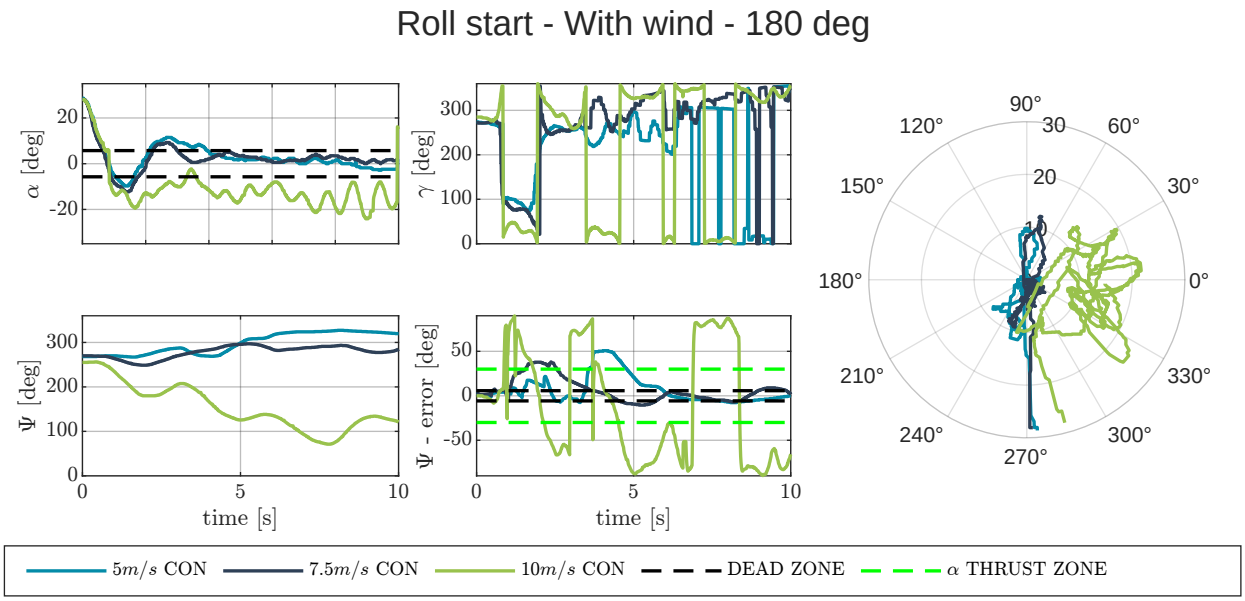


Figure J.2. Plot of SPS attitude measurements doing roll disturbance test with wind starting at swing angle 270° (Video: `https://www.youtube.com/watch?v=AxIT9htNtEE`)

SPS - system start at 270°with wind & load

In Figure J.2, the pitch swing with load results, with the initial global swing plane angle of 270°. The tests reached stability in 8.27 s for the 5 $\frac{m}{s}$ test, 15.5 s for the 7.5 $\frac{m}{s}$ test, and 14.01 s for the 10 $\frac{m}{s}$ test. A summary of the results and initial swing angle is shown in Table 8.4.

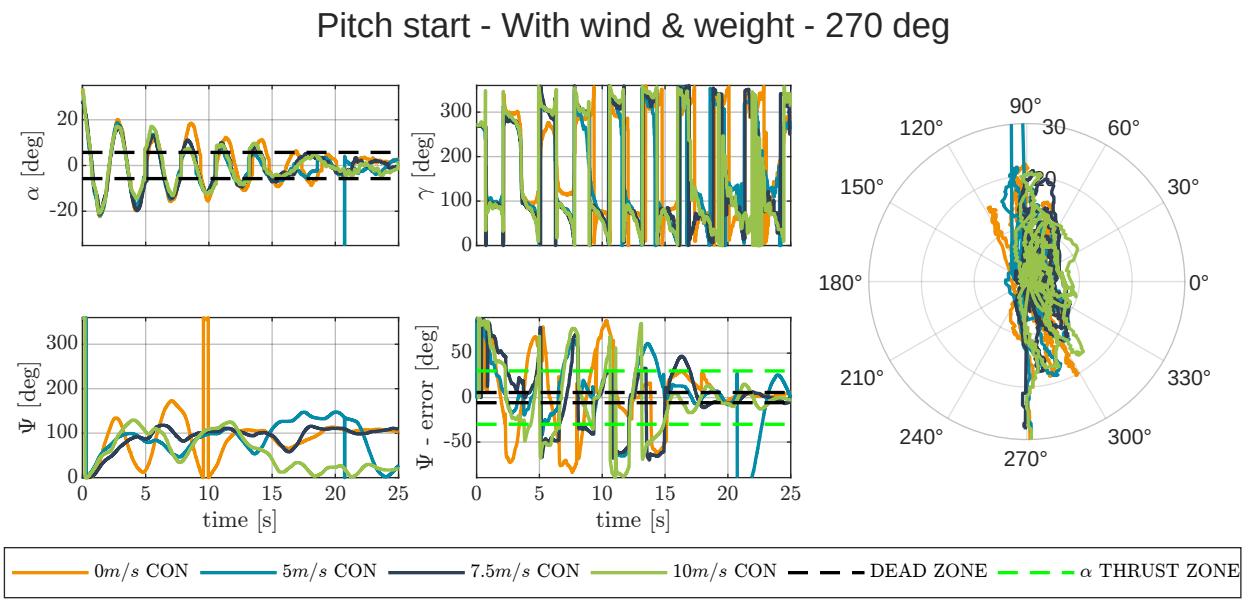


Figure J.3. Plot of SPS system with load attitude measurements doing pitch disturbance test with wind starting at swing angle 270° (Video: `https://www.youtube.com/watch?v=kcY45OP0uwM`)

In Figure J.2, the roll swing with load results, with the initial global swing plane angle of 270°. The tests reached stability in 12.78 s for the 5 $\frac{m}{s}$ test, 11.54 s for the 7.5 $\frac{m}{s}$ test, and 11.23 s for the 10 $\frac{m}{s}$ test. A summary of the results and initial swing angle is shown in Table 8.4.

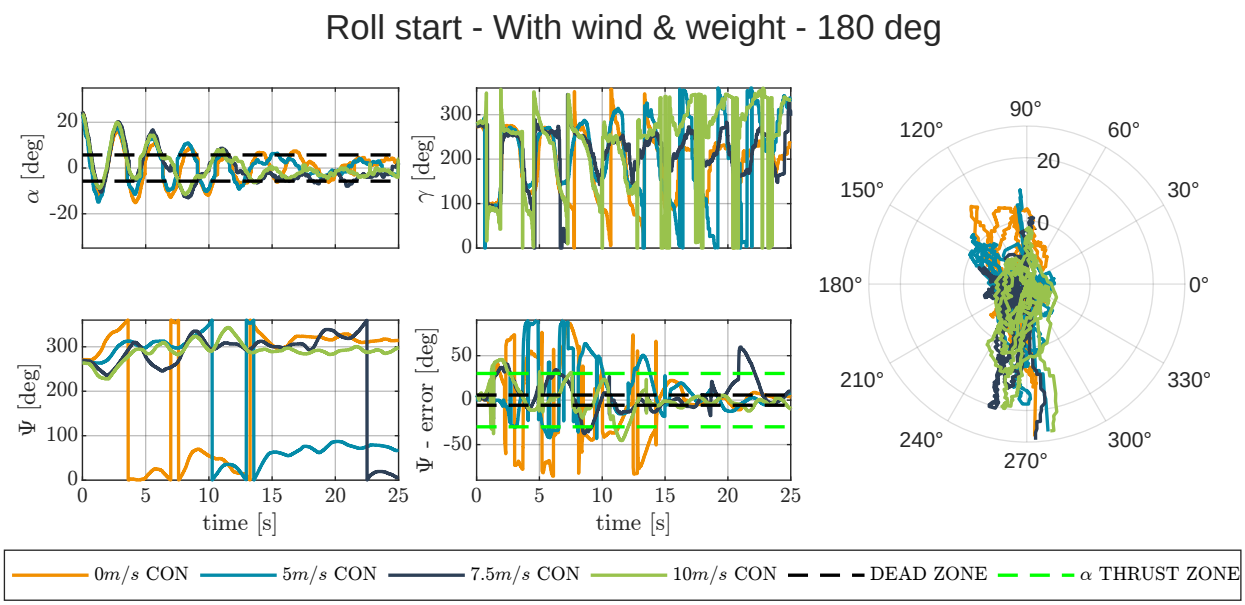


Figure J.4. Plot of SPS system with load attitude measurements doing roll disturbance test with wind starting at swing angle 270° (Video: `https://www.youtube.com/watch?v=tuDLWov3Sco`)

C-code K

In the following Appendix, the entire C code as implemented on the Nucelo bord can be seen further with the dashbord used to control the programme. In addition to this the C++ code implemented in the arduino can also be found.

K.1 STM32CubeMonitor dashbord

In Figure K.1, the dashboard used can be seen. The dashboard contains several plots for motor velocity and reference, as well as attitude and rotational rates. It is also possible to switch between the sys_states and start/stop logging data. Several LEDs are also displayed, indicating when different timers or dead zones are activated. The motor throttle and battery voltage are shown in gauges, illustrating the percentage.

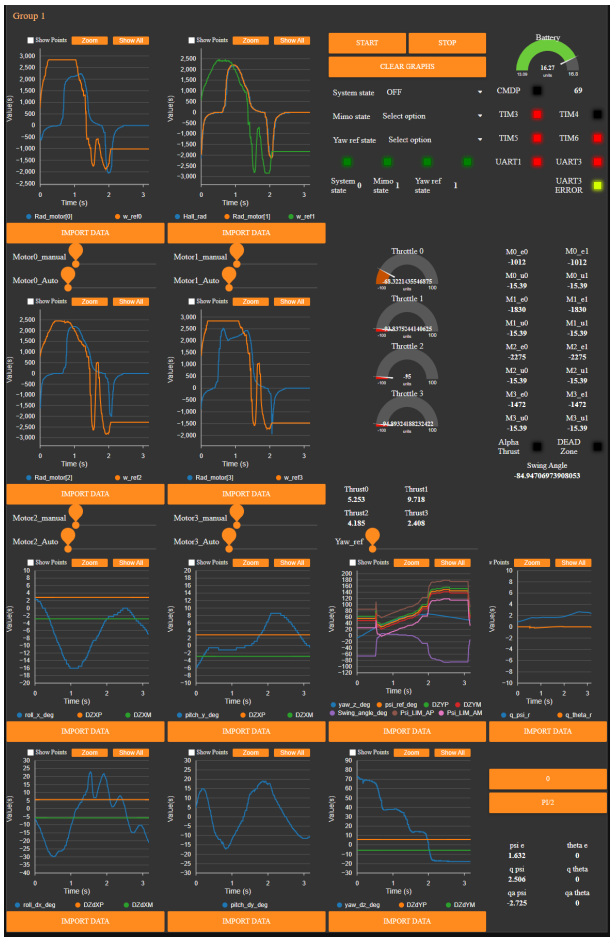


Figure K.1. The dashbord made in STM32CubeMonitor

K.2 main.c

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "dma.h"
#include "tim.h"
#include "usart.h"
#include "gpio.h"

/////////////////////////////////////////////////////////////////////////
// Include custom header with some functions
/////////////////////////////////////////////////////////////////////////
// Include EMSD_FUN.h header with Dshot functions
#include "EMSD_FUN.h"

/////////////////////////////////////////////////////////////////////////
// Maths
/////////////////////////////////////////////////////////////////////////
float    pi = 2 * acos(0.0);

/////////////////////////////////////////////////////////////////////////
// System variabels
/////////////////////////////////////////////////////////////////////////
int sys_state;
int MIMO_state = 1;
int Yaw_ref_state = 0;
int MIMO_state_restarter = 0;
int i;
float     Voltage_shotoff = 14.5;
float    Loop_time_3,Loop_time_4,Loop_time_6;

/////////////////////////////////////////////////////////////////////////
// Motor variabels
/////////////////////////////////////////////////////////////////////////

uint16_t motorvalues[4] = {0,0,0,0};
uint16_t motorvalues_test[4] = {0,0,0,0};
float     THROTT0, MAN_THROTT0, AUTO_THROTT0, w_ref0, man_w_ref0, w_error_0_0,
    w_error_1_0, w_error_2_0, U_0_0, U_1_0, U_2_0;
float     THROTT1, MAN_THROTT1, AUTO_THROTT1, w_ref1, man_w_ref1, w_error_0_1,
    w_error_1_1, w_error_2_1, U_0_1, U_1_1, U_2_1;
float     THROTT2, MAN_THROTT2, AUTO_THROTT2, w_ref2, man_w_ref2, w_error_0_2,
    w_error_1_2, w_error_2_2, U_0_2, U_1_2, U_2_2;
float     THROTT3, MAN_THROTT3, AUTO_THROTT3, w_ref3, man_w_ref3, w_error_0_3,
    w_error_1_3, w_error_2_3, U_0_3, U_1_3, U_2_3;
float     THROTT_LIM0 = 95;
float     THROTT_LIM1 = 95;
float     THROTT_LIM2 = 95;
float     THROTT_LIM3 = 95;
uint8_t   CMD_InPro    = 69;
uint16_t cmd_test,cmd_wait;
uint16_t cmd_nr = 6;
```

```
47 uint16_t CMD0_counter,CMD1_counter,CMD2_counter,CMD3_counter;
48 uint8_t  DIR0,DIR1,DIR2,DIR3;
49
50 ////////////////////////////////////////////////////////////////////
51 // Telemerty from ESC
52 ////////////////////////////////////////////////////////////////////
53 #define  Motor_NUM 4
54 #define  Tele_len  10
55 #define  UART3_LEN 37
56
57 uint16_t RxDataLen1;
58 bool     Tele_ON;
59 bool     Tele_force;
60 int      Tele_source = 0;
61 uint8_t  Telemerty_register[Motor_NUM] = {0,0,0,0};
62 uint8_t  UART_BUFF[Tele_len];
63 int      Tele_count;
64 uint8_t  Temperature_motor[Motor_NUM];
65 uint16_t Voltage_motor[Motor_NUM];
66 uint16_t eRpm_motor[Motor_NUM];
67 uint16_t Rpm_motor[Motor_NUM];
68 float    uRad_motor[Motor_NUM];
69 float    Rad_motor[Motor_NUM];
70
71 ////////////////////////////////////////////////////////////////////
72 // Telemerty from Adrino
73 ////////////////////////////////////////////////////////////////////
74
75 uint8_t  UART3_BUFF[UART3_LEN];
76 uint8_t  crc_check;
77 uint16_t RxDataLen3;
78 int      rx_x, rx_y,rx_z,rx_dx,rx_dy,rx_dz,rx_cmd,rx_sum,rx_sum_check;
79 float    roll_x,pitch_y,yaw_z,roll_dx,pitch_dy,yaw_dz,yaw_z_old,yaw_dz_old;
80
81 ////////////////////////////////////////////////////////////////////
82 // Swing detection
83 ////////////////////////////////////////////////////////////////////
84 float    Swing_angle;
85 float    in0,in1,out1;
86 float    alpha_angle,alpha_dangle;
87 float    yaw_plus;
88
89 ////////////////////////////////////////////////////////////////////
90 // MIMO Controller
91 ////////////////////////////////////////////////////////////////////
92
93 // MIMO control variables
94 float q_alpha,q_psi;
95 float alpha_ref,psi_ref;;
96 float alpha_error,psi_error;
97 float qa_alpha,qa_psi;
98 float alpha_arr[4];
```

```
99  float alpha;
100 float Thrust_ref0,Thrust_ref1,Thrust_ref2,Thrust_ref3;
101 float Thrust_ref0_bar,Thrust_ref1_bar,Thrust_ref2_bar,Thrust_ref3_bar;
102 float T_roll_bar,M_psi_bar;
103 float T_roll,M_psi;
104 float L1 = 0.2725;
105 float L2 = 0.4475;
106
107 // MIMO YAW control K^T
108 float   Kx1_yaw[4] = {1.953  ,1.953  ,-1.953  ,-1.953};
109 float   Kx2_yaw[4] = {1.789  ,1.789  ,-1.789  ,-1.789};
110 float   KI1_yaw[4] = {0.7409 ,0.7408 ,-0.7408 ,-0.7409};
111
112 // MIMO FULL control K^T
113 float   Kx1[4] = { 14.1181, 14.1181, 14.1181, 14.1181};
114 float   Kx2[4] = { 18.4659, 18.4659, 18.4659, 18.4659};
115 float   Kx3[4] = {  4.2447,  2.5848, -2.5848, -4.2447};
116 float   Kx4[4] = {  2.9256,  1.7815, -1.7815, -2.9256};
117 float   KI1[4] = { -6.3246, -6.3246, -6.3246, -6.3246};
118 float   KI2[4] = { -2.9587, -1.8017,  1.8017,  2.9587};
119
120 ////////////////////////////////////////////////////////////////////
121 // Lookup tabels
122 ////////////////////////////////////////////////////////////////////
123
124 // Lookup tabel for (Thurst -> rad) /////
125
126 // Look up table for positive propeller direction
127 #define  LUP_len 350
128 uint16_t Look_UP_PLUS[LUP_len] =
        {156,220,269,310,346,379,409,436,462,487,511,533,554,575,595,614,633,651,669,686,703,719,735
129 double   LUP_Step;
130
131 // Look up table for negative propeller direction
132 #define  LUM_len 350
133 uint16_t Look_UP_MINUS[LUM_len] =
        {158,222,271,313,349,382,412,440,467,491,515,538,559,580,600,620,638,657,674,692,708,725,741
134 double   LUM_Step;
135
136 ////////////////////////////////////////////////////////////////////
137 // Limits,saturations and dead zone
138 ////////////////////////////////////////////////////////////////////
139 // Saturate Thrust output
140 double LUP_max = 13.98;
141 double LUM_max = 5.252776920;
142 float  test_in,test;
143
144 // Telemerry update limit
145 double yaw_update_lim  = 0.1745;
146 double dyaw_update_lim = 0.4363;
```

```
147
148 // Deadzone
149 double DEADZ_psiERROR    = 0.1222;
150 double DEADZ_psispeed    = 0.1;
151 double DEADZ_alphaERROR = 0.05; //0.0262;
152 double DEADZ_alphaspeed = 0.1;
153
154 float psi_limit_for_alpha_thrust      = 0.2618*2;
155 float swing_angle_update_limit        = 0.1047;
156 float swing_angle_speed_update_limit = 0.2618;
157
158 ////////////////////////////////////////////////////////////////////
159 // Toggels
160 ////////////////////////////////////////////////////////////////////
161 uint8_t  Timer3_toggel;
162 uint8_t  Timer4_toggel;
163 uint8_t  Timer5_toggel;
164 uint8_t  Timer6_toggel;
165 uint8_t  UART1_toggel;
166 uint8_t  UART3_toggel;
167 uint8_t  DEADZ_toggel;
168 uint8_t  alpha_thrust_toggel;
169 uint8_t  UART3_ERROR_toggel;
170
171 ////////////////////////////////////////////////////////////////////
172 // Function prototypes
173 ////////////////////////////////////////////////////////////////////
174 uint8_t get_crc8(uint8_t *Buf, uint8_t BufLen);
175 uint8_t update_crc8(uint8_t crc, uint8_t crc_seed);
176 void Motor_bit_selector(uint16_t* motor_bits,uint8_t motor_nr,uint8_t* CMD_InP,
    float THROT,uint8_t* motor_dir, uint16_t* CMD_counter,float* RPM,float* u0,
    float* u1,float* u2,float* e0,float* e1,float* e2);
177 void Sinewave(double* Wave, int16_t Size);
178 double Force_to_Velocity(double Force,uint8_t motor_nr,uint16_t* LUP,uint16_t* LUM
    ,double LUP_Step, double LUM_Step);
179 float Saturate(float input,float min,float max);
180 void reset_variabels(float* var_THROTT0,float* var_THROTT1,float* var_THROTT2,
    float* var_THROTT3,float* var_AUTO_THROTT0, float* var_AUTO_THROTT1,float*
    var_AUTO_THROTT2, float* var_AUTO_THROTT3,float* var_w_ref0,float* var_w_ref1,
    float* var_w_ref2,float* var_w_ref3,float* var_w_error_0_0,float*
    var_w_error_1_0,float* var_U_0_0, float* var_U_1_0,float* var_w_error_0_1,
    float* var_w_error_1_1,float* var_U_0_1, float* var_U_1_1,float*
    var_w_error_0_2,float* var_w_error_1_2, float* var_U_0_2, float* var_U_1_2,
    float* var_w_error_0_3, float* var_w_error_1_3,float* var_U_0_3, float*
    var_U_1_3, float* var_psi_error, float* var_alpha_error,float* var_q_psi,
    float* var_qa_psi, float* var_q_alpha, float* var_qa_alpha, float*
    var_Thrust_ref0, float* var_Thrust_ref1,float* var_Thrust_ref2, float*
    var_Thrust_ref3,float* var_Thrust_ref0_bar,float* var_Thrust_ref1_bar, float*
    var_Thrust_ref2_bar, float* var_Thrust_ref3_bar);
181
182 int main(void)
183 {
```

```
184   /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
185 ESC  HAL_Init();
186
187   /* Configure the system clock */
188   SystemClock_Config();
189
190   /* Initialize all configured peripherals */
191   MX_GPIO_Init();
192   MX_DMA_Init();
193   MX_TIM8_Init();
194   MX_TIM1_Init();
195   MX_USART1_UART_Init();
196   MX_TIM2_Init();
197   MX_TIM3_Init();
198   MX_USART3_UART_Init();
199   MX_TIM4_Init();
200   MX_TIM5_Init();
201   MX_TIM6_Init();
202
203   ////////////////////////////////////////////////////////////////////
204   // Start Timers and peripherals
205   ////////////////////////////////////////////////////////////////////
206   // Configure Dshot timers 1 and 8
207   Timer_DSHOT_init(DSHOT300);
208
209   // Start timer 2 for counting Hall sensor events
210   HAL_TIM_Base_Start(&htim2);
211
212   // Start timer 3 with interrupt for numerical differentiation
213   HAL_TIM_Base_Start_IT(&htim3);
214
215   // Start timer 4 with interrupt for Motor controllers
216   HAL_TIM_Base_Start_IT(&htim4);
217
218   // Start timer 5 with interrupt for Updating Dshot input
219   HAL_TIM_Base_Start_IT(&htim5);
220
221   // Start timer 6 with interrupt for MIMO control system
222   HAL_TIM_Base_Start_IT(&htim6);
223
224   // Start UART1 for receiving motor telemerty
225   HAL_UARTEx_ReceiveToIdle_IT(&huart1,UART_BUFF,Tele_len);
226
227   // Start UART3 for receiving attitude data from adrino
228   HAL_UARTEx_ReceiveToIdle_IT(&huart3,UART3_BUFF,UART3_LEN);
229
230   ////////////////////////////////////////////////////////////////////
231   // Initial calculatiuon of some constants
232   ////////////////////////////////////////////////////////////////////
233   // Calculate timer 3 cycle time in [s]
234   Loop_time_3 = ((double)(htim3.Init.Period+1)*(htim3.Init.Prescaler+1))/
      APB1_TIM_clock;
```

```
235
236   // Calculate timer 4 cycle time in [s]
237   Loop_time_4 = ((double)(htim4.Init.Period+1)*(htim4.Init.Prescaler+1))/
        APB1_TIM_clock;
238
239   // Calculate timer 6 cycle time in [s]
240   Loop_time_6 = ((double)(htim6.Init.Period+1)*(htim6.Init.Prescaler+1))/
        APB1_TIM_clock;
241
242   // Calculate Lookup table x-axis step size for positive side
243   LUP_Step = ((double)13.987608/(double)LUP_len);
244
245   // Calculate Lookup table x-axis step size for negative side
246   LUM_Step = ((double)5.252776920/(double)LUM_len);
247
248   /* Infinite loop */
249   while (1)
250   {
251   }
252 }
253
254 void SystemClock_Config(void)
255 {
256   RCC_OscInitTypeDef RCC_OscInitStruct = {0};
257   RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
258
259   /** Configure the main internal regulator output voltage
260   */
261   __HAL_RCC_PWR_CLK_ENABLE();
262   __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
263
264   /** Initializes the RCC Oscillators according to the specified parameters
265   * in the RCC_OscInitTypeDef structure.
266   */
267   RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
268   RCC_OscInitStruct.HSIState = RCC_HSI_ON;
269   RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
270   RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
271   RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
272   RCC_OscInitStruct.PLL.PLLM = 8;
273   RCC_OscInitStruct.PLL.PLLN = 180;
274   RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
275   RCC_OscInitStruct.PLL.PLLQ = 2;
276   RCC_OscInitStruct.PLL.PLLR = 2;
277   if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
278   {
279     Error_Handler();
280   }
281
282   /** Activate the Over-Drive mode
283   */
284   if (HAL_PWREx_EnableOverDrive() != HAL_OK)
```

```
285   {
286     Error_Handler();
287   }
288 
289   /** Initializes the CPU, AHB and APB buses clocks
290   */
291   RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
292                                 |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
293   RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
294   RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
295   RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
296   RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
297 
298   if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
299   {
300     Error_Handler();
301   }
302 }
303 
304 /////////////////////////////////////////////////////////////////////
305 // Uart Callback events
306 /////////////////////////////////////////////////////////////////////
307 
308 // Uart callback event is called every time a interrupt is generated by a Uart
    peripheral
309 HAL_UARTEx_RxEventCallback(UART_HandleTypeDef *huart, uint16_t Size){
310   /////////////////////////////////////////////////////////////////////
311   // UART3
312   /////////////////////////////////////////////////////////////////////
313   // The following code is run when a UART3 interrupt is called
314   if(huart->Instance==USART3){
315     // Read length of Serial data
316     RxDataLen3 = Size;
317     // Activate UART3 Again
318     HAL_UARTEx_ReceiveToIdle_IT(&huart3,UART3_BUFF,UART3_LEN);
319 
320     /* Converted the UART3 Buffer to integeres based on the data from the adrino
    is send in the following format: "roll_x*100,pitch_y*100,yaw_z*100,roll_dx
    *100,pitch_dy*100,yaw_dz*100,cdm"*/
321     sscanf(UART3_BUFF,"%d,%d,%d,%d,%d,%d,%d,%d",&rx_x,&rx_y,&rx_z,&rx_dx,&rx_dy,&
    rx_dz,&rx_cmd,&rx_sum);
322 
323         // Clear Uart3 buffer
324     for (i = 0; i < 36; i++) {
325       UART3_BUFF[i] = 0;
326     }
327 
328       // Calculate sum_check
329     rx_sum_check = rx_x + rx_y + rx_z + rx_dx + rx_dy + rx_dz + rx_cmd;
330 
331       // Check sum_check
332     if(rx_sum == rx_sum_check){
```

```
333
334         // Convert the integer to floats
335         roll_x    = (float) rx_x/100;
336         pitch_y   = (float) rx_y/100;
337         yaw_z     = (float) rx_z/100;
338         roll_dx   = (float) rx_dx/100;
339         pitch_dy  = (float) rx_dy/100;
340         yaw_dz    = (float) rx_dz/100;
341
342       }else{
343         UART3_ERROR_toggel = UART3_ERROR_toggel ^ (uint)1;
344       }
345
346       UART3_toggel = UART3_toggel ^ (uint)1;
347       HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_10);
348     }
349
350     ////////////////////////////////////////////////////////////////////
351     // UART1
352     ////////////////////////////////////////////////////////////////////
353     // The following code is run when a UART1 interrupt is called
354     if(huart->Instance==USART1){
355       // Read length of Serial data
356       RxDataLen1 = Size;
357       // Activate UART1 Again
358       HAL_UARTEx_ReceiveToIdle_IT(&huart1,UART_BUFF,Tele_len);
359
360       // Source for ESC telemetry byte meaning
361       // http://www.rcgroups.com/forums/showatt.php?attachmentid=8524039&d
      =1450424877
362       // UART_BUFF[0] is 8bit Temperature
363       // UART_BUFF[1] is 8bit Voltage high byte
364       // UART_BUFF[2] is 8bit Voltage low byte
365       // UART_BUFF[3] is 8bit Current high byte       (NOT SUPPORTED BY ESC)
366       // UART_BUFF[4] is 8bit Current low byte        (NOT SUPPORTED BY ESC)
367       // UART_BUFF[5] is 8bit Consumption high byte   (NOT SUPPORTED BY ESC)
368       // UART_BUFF[6] is 8bit Consumption low byte    (NOT SUPPORTED BY ESC)
369       // UART_BUFF[7] is 8bit eRpm high byte
370       // UART_BUFF[8] is 8bit eRpm low byte
371       // UART_BUFF[9] is 8bit crc
372
373       // 1 Temperature_motor = 1deg celcius
374       // 1000 Voltage_motor  = 10V
375       // 1 eRpm_motor        = 100 eRpm
376       // 1 Rpm_motor         = 100 eRpm
377
378       // Calculate crc for the received data
379       crc_check = get_crc8(UART_BUFF, Tele_len);
380
381       // converting bytes to integers for current telemerty source
382       Temperature_motor[Tele_source]    =    UART_BUFF[0];
```

```
383     Voltage_motor[Tele_source]       = ((UART_BUFF[1] << 8) | UART_BUFF[2] & 0
        x00FF);
384     //Current_motor[Tele_source]     = ((UART_BUFF[3] << 8) | UART_BUFF[4] & 0
        x00FF)*100;
385     //Consumption_motor[Tele_source] = ((UART_BUFF[5] << 8) | UART_BUFF[6] & 0
        x00FF);
386     eRpm_motor[Tele_source]          = ((UART_BUFF[7] << 8) | UART_BUFF[8] & 0
        x00FF);
387
388     // Calculate unsigned motor velocity in [Rpm]
389     Rpm_motor[Tele_source]           = ((eRpm_motor[Tele_source]/7)*100);
390
391     // Calculate unsigned motor velocity in [RAD/s]
392     uRad_motor[Tele_source]          = (float)(Rpm_motor[Tele_source
        ]*0.1047197550);
393
394     // Change Telemerty Source to next motor
395     Tele_source = Tele_source + 1;
396
397     // Reset Telemerty counter
398     Tele_count  = 0;
399
400     // reset Telemerty Source back to source 0 if Telemerty has been colleted from
         all sources
401     if(Tele_source == 4){
402       Tele_source = 0;
403     }
404     UART1_toggel = UART1_toggel ^ (uint)1;
405   }
406 }
407
408 ////////////////////////////////////////////////////////////////////
409 // Functions in Main code
410 ////////////////////////////////////////////////////////////////////
411 // Function to select the bit sent via the Dshot protocol (Motor throttle and
        commands)
412 void Motor_bit_selector(uint16_t* motor_bits,uint8_t motor_nr,uint8_t* CMD_InP ,
        float THROT,uint8_t* motor_dir,uint16_t* CMD_counter,float* RPM,float* u0,
        float* u1,float* u2,float* e0,float* e1,float* e2){
413   float temp_Throt;
414
415   // Change the throttle value so that it is always positive
416   // This is done based on the direction setting of the motor
417   if(*motor_dir==1){
418     temp_Throt = THROT * (-1);
419   }else{
420     temp_Throt = THROT;
421   }
422
423   // The throttle value is converted to a bit value based on a macro mapping
        throttle to bits
424   // The macro is defined in EMSD_FUN.h
```

```
425   motor_bits[motor_nr] = ThrottlePRO(temp_Throt);
426
427   // code to send command to change the direction of the motor
428   // If the motor is set to run in the opposed direction that the throttle is
        indicating
429   // or if a direction change command has been started the following code is run
430   if(((*motor_dir == 0 && THROT<0)||(*motor_dir == 1 && THROT>0))||(*CMD_InP ==
        motor_nr)){
431
432     // The command to change the direction of a motor seams only to be reacheved
        if only one
433     // motor is given the command at a time. So a Command in progress parameter (
      CMD_InP) is used
434     // indicate witch motor is currently beeing commanded if no motor is beeing
      commanded it is set to 69
435     // The following if statement allows can start a new change dirrection command
        or let one in progress pass
436     if((*CMD_InP == 69)||*CMD_InP == motor_nr){
437
438       // If a change in motor direction is asked for and no command is running the
       CMD_InP is set to the
439       // motor currently beeing commanded
440       *CMD_InP = motor_nr;
441
442       // The motor only changes direction if the motor is stopped so if the motor
      velocity is larger than 0
443       // the motor bit is set to 0 to stop the motor
444       if(RPM[motor_nr] > 0){
445         motor_bits[motor_nr] = 0;
446       }
447
448       // To ensure that the motor is fully stoped the motor bit is set to 0 for a
      additional 15 cycles
449       if((*CMD_counter<15)&&(RPM[motor_nr]==0)){
450         motor_bits[motor_nr] = 0;
451         // Add to counter
452         *CMD_counter += 1;
453       }
454
455       // Send direction command  to motor 8 for reversed and 7 for normal this is
      done for 10 cycles
456       // even tough only 6 cycels is specified in the sorcue
457       if((*CMD_counter>=15)&&(RPM[motor_nr]==0)){
458         // If motor direction is normal send reverse signal
459         if(*motor_dir == 0){
460           motor_bits[motor_nr] = 8; // Reversed
461         }
462         // If motor direction is reversed send normal signal
463         if(*motor_dir == 1){
464           motor_bits[motor_nr] = 7; // Normal
465         }
466         // Add to counter
```

```
467         *CMD_counter += 1;
468       }
469
470       // When the 10 cycles of command signal has been send.
471       if((*CMD_counter==35)){
472         switch(*motor_dir){
473         // If motor was normal but has been commanded to reversed Set direction
    bit to reversed
474         // and set comand in progress to idel and comand counter to 0
475           case 0:
476             *motor_dir    = 1;
477             *CMD_InP      = 69;
478             *CMD_counter = 0;
479           break;
480         // If motor was reversed but has been commanded to normal Set direction
    bit to normal
481         // and set comand in progress to idel and comand counter to 0
482           default:
483             *motor_dir    = 0;
484             *CMD_InP      = 69;
485             *CMD_counter = 0;
486           break;
487         }
488         // Set all motor controller parameters to 0
489         *u0 = 0;
490         *u1 = 0;
491         *u2 = 0;
492         *e0 = 0;
493         *e1 = 0;
494         *e2 = 0;
495       }
496     }else{
497       // If the motor has been commanded to change direction but are in queue set
    motor output to 0
498       motor_bits[motor_nr] = 0;
499     }
500   }
501 }
502
503 // Function for lookup tabel to translate force to motor velocity
504 double Force_to_Velocity(double Force,uint8_t motor_nr,uint16_t* LUP,uint16_t* LUM
    ,double LUP_Step, double LUM_Step){
505   int VEL_INDEX;
506
507   // Since the motors are mounted in opposite dirrections the velocity needed to
     generate the force is diffent.
508   // Motor 1 and 2 are facing the same dirrection and Motor 0 and 3 are also a
     pair.
509
510   // For motor 1 and 2
511   if((motor_nr == 1)||(motor_nr == 2)){
512     // If the force is 0 then a output of 0 is given
```

```
513     if(Force==0){
514       return 0;
515     }
516 
517     // If the motor are asked to give a positive output
518     else if(Force>0){
519       VEL_INDEX = rint(Force/LUP_Step);
520       return LUP[VEL_INDEX-1];
521       // Equation for lookup tabel
522       //(2.3396e4 * pow(Temp_force,0.4935))*0.1047197550;
523     }
524     else{
525       VEL_INDEX = rint((Force*-1)/LUM_Step);
526       return -LUM[VEL_INDEX-1];
527       // Equation for lookup tabel
528       //-(3.7746e4 * pow(VEL_INDEX,0.4929))*0.1047197550;
529     }
530 
531   // For motor 0 and 3
532   }else if((motor_nr == 0)||(motor_nr == 3)){
533     if(Force==0){
534       return 0;
535     }
536     else if(Force>0){
537       VEL_INDEX = rint(Force/LUM_Step);
538       return LUM[VEL_INDEX-1];
539       // Equation for lookup tabel
540       //(2.3396e4 * pow(Temp_force,0.4935))*0.1047197550;
541     }
542     else{
543       VEL_INDEX = rint((Force*-1)/LUP_Step);
544       return -LUP[VEL_INDEX-1];
545       // Equation for lookup tabel
546       //-(3.7746e4 * pow(VEL_INDEX,0.4929))*0.1047197550;
547     }
548   }else{
549     return 0;
550   }
551 }
552 
553 // Saturate function
554 float Saturate(float input,float min,float max){
555   if(input>max){
556     return max;
557   }else if(input<min){
558     return min;
559   }else {
560     return input;
561   }
562 
563 }
564 
```

```
565 void reset_variabels(float* var_THROTT0,float* var_THROTT1,float* var_THROTT2,
        float* var_THROTT3,float* var_AUTO_THROTT0,float* var_AUTO_THROTT1,float*
        var_AUTO_THROTT2, float* var_AUTO_THROTT3,float* var_w_ref0,float* var_w_ref1,
        float* var_w_ref2,float* var_w_ref3,float* var_w_error_0_0,float*
        var_w_error_1_0,float* var_U_0_0, float* var_U_1_0,float* var_w_error_0_1,
        float* var_w_error_1_1,float* var_U_0_1, float* var_U_1_1,float*
        var_w_error_0_2,float* var_w_error_1_2, float* var_U_0_2, float* var_U_1_2,
        float* var_w_error_0_3, float* var_w_error_1_3,float* var_U_0_3, float*
        var_U_1_3, float* var_psi_error, float* var_alpha_error,float* var_q_psi,
        float* var_qa_psi, float* var_q_alpha, float* var_qa_alpha, float*
        var_Thrust_ref0, float* var_Thrust_ref1,float* var_Thrust_ref2, float*
        var_Thrust_ref3,float* var_Thrust_ref0_bar,float* var_Thrust_ref1_bar, float*
        var_Thrust_ref2_bar, float* var_Thrust_ref3_bar){
566
567   // Throttel control variables
568   *var_THROTT0 = 0;
569   *var_THROTT1 = 0;
570   *var_THROTT2 = 0;
571   *var_THROTT3 = 0;
572
573   *var_AUTO_THROTT0 = 0;
574   *var_AUTO_THROTT1 = 0;
575   *var_AUTO_THROTT2 = 0;
576     *var_AUTO_THROTT3 = 0;
577
578     // Motor controller variables
579     *var_w_ref0       = 0;
580     *var_w_ref1       = 0;
581     *var_w_ref2       = 0;
582     *var_w_ref3       = 0;
583
584     *var_w_error_0_0  = 0;
585   *var_w_error_1_0  = 0;
586   *var_U_0_0        = 0;
587   *var_U_1_0        = 0;
588
589   *var_w_error_0_1  = 0;
590   *var_w_error_1_1  = 0;
591   *var_U_0_1        = 0;
592   *var_U_1_1        = 0;
593
594   *var_w_error_0_2  = 0;
595   *var_w_error_1_2  = 0;
596   *var_U_0_2        = 0;
597   *var_U_1_2        = 0;
598
599   *var_w_error_0_3  = 0;
600   *var_w_error_1_3  = 0;
601   *var_U_0_3        = 0;
602   *var_U_1_3        = 0;
603
604   // MIMO Control variables
```

```
605   *var_psi_error     = 0;
606   *var_alpha_error   = 0;
607   *var_q_psi         = 0;
608   *var_qa_psi        = 0;
609   *var_q_alpha       = 0;
610   *var_qa_alpha      = 0;
611
612   *var_Thrust_ref0   = 0;
613   *var_Thrust_ref1   = 0;
614   *var_Thrust_ref2   = 0;
615   *var_Thrust_ref3   = 0;
616
617   *var_Thrust_ref0_bar = 0;
618   *var_Thrust_ref0_bar = 0;
619   *var_Thrust_ref0_bar = 0;
620   *var_Thrust_ref0_bar = 0;
621
622 }
623
624 ////////////////////////////////////////////////////////////////////
625 // Timer interupts
626 ////////////////////////////////////////////////////////////////////
627 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef* htim)
628 {
629   ////////////////////////////////////////////////////////////////////
630   // Motor controllers
631     ////////////////////////////////////////////////////////////////////
632   if(htim->Instance == TIM4){
633
634     // change the sign of motor velocity
635     if(DIR0 == 1){
636       Rad_motor[0] = uRad_motor[0]* (-1);
637     }else{
638       Rad_motor[0] = uRad_motor[0];
639     }
640     if(DIR1 == 1){
641       Rad_motor[1] = uRad_motor[1]* (-1);
642     }else{
643       Rad_motor[1] = uRad_motor[1];
644     }
645     if(DIR2 == 1){
646       Rad_motor[2] = uRad_motor[2]* (-1);
647     }else{
648       Rad_motor[2] = uRad_motor[2];
649     }
650     if(DIR3 == 1){
651       Rad_motor[3] = uRad_motor[3]* (-1);
652     }else{
653       Rad_motor[3] = uRad_motor[3];
654     }
655
656     //// Motor 0 controller ////////////////////////////////
```

```
657     // Motor 0 update error
658     w_error_1_0 = w_error_0_0;
659     w_error_0_0 = (float)(w_ref0 - Rad_motor[0]);
660 
661     // Motor 0 update controller output
662     U_1_0 = U_0_0;
663 
664     // discretized controller
665     U_0_0 = ((float)0.00036125*w_error_0_0)+((float)0.00036125*w_error_1_0) + ((
        float)1*U_1_0);
666 
667     // saturate motor 0 value
668     if(U_0_0>(THROTT_LIM0/100)* 16.2){
669       U_0_0 =  (THROTT_LIM0/100) * 16.2;
670     }
671     if((-THROTT_LIM0/100)* 16.2>U_0_0){
672       U_0_0 = (-THROTT_LIM0/100) * 16.2;
673     }
674 
675     AUTO_THROTT0 = (U_0_0/16.2)*100;
676 
677     ////////////////////////////////////////////////////////
678     //// Motor 1 controller ////////////////////////////////
679     // Motor 1 update error
680     w_error_1_1 = w_error_0_1;
681     w_error_0_1 = (float)(w_ref1 - Rad_motor[1]);
682 
683     // Motor 1 update controller output
684     U_1_1 = U_0_1;
685     // discretized controller
686     U_0_1 = ((float)0.00036125*w_error_0_1)+((float)0.00036125*w_error_1_1) + ((
        float)1*U_1_1);
687 
688     // saturate motor 1 value
689     if(U_0_1>(THROTT_LIM1/100)* 16.2){
690       U_0_1 =  (THROTT_LIM1/100) * 16.2;
691     }
692     if((-THROTT_LIM1/100)* 16.2>U_0_1){
693       U_0_1 = (-THROTT_LIM1/100) * 16.2;
694     }
695 
696     AUTO_THROTT1 = (U_0_1/16.2)*100;
697 
698     ////////////////////////////////////////////////////////
699     //// Motor 2 controller ////////////////////////////////
700     // Motor 2 update error
701     w_error_1_2 = w_error_0_2;
702     w_error_0_2 = (float)(w_ref2 - Rad_motor[2]);
703 
704     // Motor 2 update controller output
705     U_1_2 = U_0_2;
706     // discretized controller
```

```
707     U_0_2 = ((float)0.00036125*w_error_0_2)+((float)0.00036125*w_error_1_2) + ((
        float)1*U_1_2);
708     // saturate motor 2 value
709     if(U_0_2>(THROTT_LIM2/100)* 16.2){
710       U_0_2 =  (THROTT_LIM2/100) * 16.2;
711     }
712     if((-THROTT_LIM2/100)* 16.2>U_0_2){
713       U_0_2 = (-THROTT_LIM2/100) * 16.2;
714     }
715
716     AUTO_THROTT2 = (U_0_2/16.2)*100;
717
718     ////////////////////////////////////////////////////////
719     //// Motor 3 controller ////////////////////////////////
720     // Motor 3 update error
721     w_error_1_3 = w_error_0_3;
722     w_error_0_3 = (float)(w_ref3 - Rad_motor[3]);
723
724     // Motor 3 update controller output
725     U_1_3 = U_0_3;
726     // discretized at t = ?
727     U_0_3 = ((float)0.00036125*w_error_0_3)+((float)0.00036125*w_error_1_3) + ((
        float)1*U_1_3);
728
729     // saturate motor 3 value
730     if(U_0_3>(THROTT_LIM3/100)* 16.2){
731       U_0_3 =  (THROTT_LIM3/100) * 16.2;
732     }
733     if((-THROTT_LIM3/100)* 16.2>U_0_3){
734       U_0_3 = (-THROTT_LIM3/100) * 16.2;
735     }
736
737     AUTO_THROTT3 = (U_0_3/16.2)*100;
738
739     ////////////////////////////////////////////////////////
740   Timer4_toggel = Timer4_toggel ^ (uint)1;
741   }
742
743
744     ////////////////////////////////////////////////////////////////////
745     // Mode
746     ////////////////////////////////////////////////////////////////////
747   if(htim->Instance == TIM5){
748
749     switch(sys_state){
750         case 0: // off state (The motor is off)
751           motorvalues[0] = 0;
752           motorvalues[1] = 0;
753           motorvalues[2] = 0;
754           motorvalues[3] = 0;
755           Tele_ON = true;
756           break;
```

```
757         case 3: // Motors in manual Throttel control
758           // Assign manual values to the Throttel values
759           THROTT0 = MAN_THROTT0;
760           THROTT1 = MAN_THROTT1;
761           THROTT2 = MAN_THROTT2;
762           THROTT3 = MAN_THROTT3;
763           // Run Bit selector function that determines bit for throttel procentage
764           // and keeps track of the direction of the motor
765           Motor_bit_selector(motorvalues,0,&CMD_InPro,THROTT0,&DIR0,&CMD0_counter,
    uRad_motor,&U_0_0,&U_1_0,&U_2_0,&w_error_0_0,&w_error_1_0,&w_error_2_0);
766           Motor_bit_selector(motorvalues,1,&CMD_InPro,THROTT1,&DIR1,&CMD1_counter,
    uRad_motor,&U_0_1,&U_1_1,&U_2_1,&w_error_0_1,&w_error_1_1,&w_error_2_1);
767           Motor_bit_selector(motorvalues,2,&CMD_InPro,THROTT2,&DIR2,&CMD2_counter,
    uRad_motor,&U_0_2,&U_1_2,&U_2_2,&w_error_0_2,&w_error_1_2,&w_error_2_2);
768           Motor_bit_selector(motorvalues,3,&CMD_InPro,THROTT3,&DIR3,&CMD3_counter,
    uRad_motor,&U_0_3,&U_1_3,&U_2_3,&w_error_0_3,&w_error_1_3,&w_error_2_3);
769           // Turn Telemerty on
770           Tele_ON = true;
771           break;
772         case 4: // start Motor controller
773           // Set all controller values to 0 to reset the controllers
774           reset_variabels(&THROTT0,&THROTT1,&THROTT2,&THROTT3,&AUTO_THROTT0,&
    AUTO_THROTT1,&AUTO_THROTT2,&AUTO_THROTT3,&w_ref0,&w_ref1,&w_ref2,&w_ref3,&
    w_error_0_0,&w_error_1_0,&U_0_0,&U_1_0,&w_error_0_1,&w_error_1_1,&U_0_1,&U_1_1
    ,&w_error_0_2,&w_error_1_2,&U_0_2,&U_1_2,&w_error_0_3,&w_error_1_3,&U_0_3,&
    U_1_3,&psi_error,&alpha_error,&q_psi,&qa_psi,&q_alpha,&qa_alpha,&Thrust_ref0,&
    Thrust_ref1,&Thrust_ref2,&Thrust_ref3,&Thrust_ref0_bar,&Thrust_ref1_bar,&
    Thrust_ref2_bar,&Thrust_ref3_bar);
775
776           // go to SYS_state 5
777           sys_state   = 5;
778           break;
779         case 5: // Motors in Speed controller control
780           w_ref0 = man_w_ref0;
781           w_ref1 = man_w_ref1;
782           w_ref2 = man_w_ref2;
783           w_ref3 = man_w_ref3;
784
785           // Assign Automatical Throttel values to the Throttel values
786           THROTT0 = AUTO_THROTT0;
787           THROTT1 = AUTO_THROTT1;
788           THROTT2 = AUTO_THROTT2;
789           THROTT3 = AUTO_THROTT3;
790           // Run Bit selector function that determines bit for throttel procentage
791           // and keeps track of the direction of the motor
792           Motor_bit_selector(motorvalues,0,&CMD_InPro,THROTT0,&DIR0,&CMD0_counter,
    uRad_motor,&U_0_0,&U_1_0,&U_2_0,&w_error_0_0,&w_error_1_0,&w_error_2_0);
793           Motor_bit_selector(motorvalues,1,&CMD_InPro,THROTT1,&DIR1,&CMD1_counter,
    uRad_motor,&U_0_1,&U_1_1,&U_2_1,&w_error_0_1,&w_error_1_1,&w_error_2_1);
794           Motor_bit_selector(motorvalues,2,&CMD_InPro,THROTT2,&DIR2,&CMD2_counter,
    uRad_motor,&U_0_2,&U_1_2,&U_2_2,&w_error_0_2,&w_error_1_2,&w_error_2_2);
```

```
795           Motor_bit_selector(motorvalues,3,&CMD_InPro,THROTT3,&DIR3,&CMD3_counter,
    uRad_motor,&U_0_3,&U_1_3,&U_2_3,&w_error_0_3,&w_error_1_3,&w_error_2_3);
796           // Turn Telemerty on
797           Tele_ON = true;
798           break;
799         case 6:
800           reset_variabels(&THROTT0,&THROTT1,&THROTT2,&THROTT3,&AUTO_THROTT0,&
    AUTO_THROTT1,&AUTO_THROTT2,&AUTO_THROTT3,&w_ref0,&w_ref1,&w_ref2,&w_ref3,&
    w_error_0_0,&w_error_1_0,&U_0_0,&U_1_0,&w_error_0_1,&w_error_1_1,&U_0_1,&U_1_1
    ,&w_error_0_2,&w_error_1_2,&U_0_2,&U_1_2,&w_error_0_3,&w_error_1_3,&U_0_3,&
    U_1_3,&psi_error,&alpha_error,&q_psi,&qa_psi,&q_alpha,&qa_alpha,&Thrust_ref0,&
    Thrust_ref1,&Thrust_ref2,&Thrust_ref3,&Thrust_ref0_bar,&Thrust_ref1_bar,&
    Thrust_ref2_bar,&Thrust_ref3_bar);
801
802           sys_state     = 7;
803           break;
804         case 7:
805           w_ref0  = Force_to_Velocity(Thrust_ref0,0,Look_UP_PLUS,Look_UP_MINUS,
    LUP_Step,LUM_Step);
806           w_ref1  = Force_to_Velocity(Thrust_ref1,1,Look_UP_PLUS,Look_UP_MINUS,
    LUP_Step,LUM_Step);
807           w_ref2  = Force_to_Velocity(Thrust_ref2,2,Look_UP_PLUS,Look_UP_MINUS,
    LUP_Step,LUM_Step);
808           w_ref3  = Force_to_Velocity(Thrust_ref3,3,Look_UP_PLUS,Look_UP_MINUS,
    LUP_Step,LUM_Step);
809
810           THROTT0 = AUTO_THROTT0;
811           THROTT1 = AUTO_THROTT1;
812           THROTT2 = AUTO_THROTT2;
813           THROTT3 = AUTO_THROTT3;
814           // Run Bit selector function that determines bit for throttle percentage
815           // and keeps track of the direction of the motor
816           Motor_bit_selector(motorvalues,0,&CMD_InPro,THROTT0,&DIR0,&CMD0_counter,
    uRad_motor,&U_0_0,&U_1_0,&U_2_0,&w_error_0_0,&w_error_1_0,&w_error_2_0);
817           Motor_bit_selector(motorvalues,1,&CMD_InPro,THROTT1,&DIR1,&CMD1_counter,
    uRad_motor,&U_0_1,&U_1_1,&U_2_1,&w_error_0_1,&w_error_1_1,&w_error_2_1);
818           Motor_bit_selector(motorvalues,2,&CMD_InPro,THROTT2,&DIR2,&CMD2_counter,
    uRad_motor,&U_0_2,&U_1_2,&U_2_2,&w_error_0_2,&w_error_1_2,&w_error_2_2);
819           Motor_bit_selector(motorvalues,3,&CMD_InPro,THROTT3,&DIR3,&CMD3_counter,
    uRad_motor,&U_0_3,&U_1_3,&U_2_3,&w_error_0_3,&w_error_1_3,&w_error_2_3);
820           // Turn Telemerty on
821           Tele_ON = true;
822           break;
823
824         default: // Error unexpected sys_state turn motor off
825           motorvalues[0] = 0;
826           motorvalues[1] = 0;
827           motorvalues[2] = 0;
828           motorvalues[3] = 0;
829           Tele_ON = false;
830           break;
831         }
```

```
832
833         // End program if Throttle percentage exceed the allowed values
834       if ((THROTT0 > THROTT_LIM0) || (THROTT1 > THROTT_LIM1) || (THROTT2 >
    THROTT_LIM2) || (THROTT3 > THROTT_LIM3) || (THROTT0 < -THROTT_LIM0) || (
    THROTT1 < -THROTT_LIM1) || (THROTT2 < -THROTT_LIM2) || (THROTT3 < -THROTT_LIM3
    ) ){
835         sys_state = 0;
836       }
837
838       // Turn off system if battery is discharged.
839       if(Voltage_motor[0]<=Voltage_shotoff){
840         sys_state = 0;
841       }
842
843       // Activate UART so that it is on
844    HAL_UARTEx_ReceiveToIdle_IT(&huart1 ,UART_BUFF, Tele_len);
845    HAL_UARTEx_ReceiveToIdle_IT(&huart3 ,UART3_BUFF,UART3_LEN);
846
847       /* Change DSHOT channel that request trelemerty and turn on and off
    trelemerty */
848    Trelemertry_change(Telemerty_register,&Tele_source,&Tele_count,Tele_ON,
    Tele_force);
849
850       // Update DSHOT
851      Update_DSHOT(motorvalues, Telemerty_register);
852      Timer5_toggel = Timer5_toggel ^ (uint)1;
853  }
854
855    ////////////////////////////////////////////////////////////////////
856  // SPS controller
857    ////////////////////////////////////////////////////////////////////
858  if(htim->Instance == TIM6){
859
860
861    // While testing
862      Yaw_ref_state = 1;
863
864      switch(Yaw_ref_state){
865      case 1:
866        if(roll_x != 0){
867          if ( (((roll_x > swing_angle_update_limit)||(roll_x < -
    swing_angle_update_limit)) || ((pitch_y > swing_angle_update_limit)||(pitch_y
    < -swing_angle_update_limit)))&&((abs(pitch_dy)>swing_angle_speed_update_limit
    )||abs(roll_dx)>swing_angle_speed_update_limit)){
868
869            // save old values for low pass filter
870            out1 = Swing_angle;
871            in1 = in0;
872
873            // calculate nessary angel change for 0 pitch angle
874            in0 =  atan(pitch_y/roll_x);
875
```

```
876             //low pass filter for swing angel to not react to small angel changes
877             // 1/40
878             Swing_angle = 0.16667*in0 + 0.16667*in1 + 0.66667 * out1;
879 
880             // calculate psi refence
881             //psi_ref = fmod(yaw_z - Swing_angle,pi);
882 
883             if( ((yaw_z - Swing_angle)>pi) || ((yaw_z - Swing_angle)<0)){
884               psi_ref = yaw_z + Swing_angle;
885             }else{
886               psi_ref = yaw_z - Swing_angle;
887             }
888             }
889           }
890         break;
891       default:
892         // The psi_ref is not given by program but by user input
893         break;
894       }
895 
896 
897   // Finding the tracking error
898   // e = r - C * x
899   psi_error    = (psi_ref    - yaw_z );
900 
901   // Only passing the tracking error for alpha when psi error is within error
        limit
902   if(fabs(psi_error)>= psi_limit_for_alpha_thrust){
903     // Set tracking error to 0
904     alpha_error = 0;
905     // reset integrator state
906     q_alpha     = 0;
907 
908             // set messured roll and roll velocity to 0     roll_x  = 0;
909             roll_dx = 0;
910 
911     alpha_thrust_toggel = 0;
912   }else{
913     alpha_thrust_toggel = 1;
914     alpha_error = (alpha_ref - roll_x);
915   }
916 
917     // Dead zone
918     if( ((psi_error<DEADZ_psiERROR)&&(psi_error>-DEADZ_psiERROR))&&((yaw_dz<
        DEADZ_psispeed)&&(yaw_dz>-DEADZ_psispeed))&&((alpha_error<DEADZ_alphaERROR)&&(
        alpha_error>-DEADZ_alphaERROR))&&((roll_dx<DEADZ_alphaspeed)&&(roll_dx>-
        DEADZ_alphaspeed))       ){
919       DEADZ_toggel = 1;
920 
921       Thrust_ref0_bar = 0;
922       Thrust_ref1_bar = 0;
923       Thrust_ref2_bar = 0;
```

```
924       Thrust_ref3_bar = 0;
925
926       Thrust_ref0 = 0;
927       Thrust_ref1 = 0;
928       Thrust_ref2 = 0;
929       Thrust_ref3 = 0;
930
931       T_roll_bar = 0;
932       M_psi_bar  = 0;
933
934       T_roll = 0;
935       M_psi  = 0;
936
937     }else{
938       // set Dead zone toggle to off
939         DEADZ_toggel = 0;
940
941       // Adding anti windup
942         if( (((Thrust_ref0 - Thrust_ref0_bar)!=0)&&((Thrust_ref0 - Thrust_ref0_bar
    )>0))||(((Thrust_ref1 - Thrust_ref1_bar)!=0)&&((Thrust_ref1 - Thrust_ref1_bar)
    <0)) ){
943           qa_alpha = T_roll - T_roll_bar;
944         }else{
945           qa_alpha = 0;
946         }
947
948         //qa_alpha = 0;
949
950         qa_psi    =  M_psi - M_psi_bar;
951
952       // Integrating the error and anti windup
953       // q = q + qdot*time
954         q_alpha = q_alpha + (alpha_error + qa_alpha)*Loop_time_6;
955         q_psi   = q_psi   + (psi_error + qa_psi)*Loop_time_6;
956
957       //Calculating unsaturated controller outputs
958       // u = -KI*q - Kx*x
959         Thrust_ref0_bar = -(q_alpha * KI1[0] + q_psi *  KI2[0]) -  (Kx1[0]*roll_x
    + Kx2[0] *roll_dx + Kx3[0]*yaw_z + Kx4[0] *yaw_dz);
960         Thrust_ref1_bar = -(q_alpha * KI1[1] + q_psi *  KI2[1]) -  (Kx1[1]*roll_x
    + Kx2[1] *roll_dx + Kx3[1]*yaw_z + Kx4[1] *yaw_dz);
961         Thrust_ref2_bar = -(q_alpha * KI1[2] + q_psi *  KI2[2]) -  (Kx1[2]*roll_x
    + Kx2[2] *roll_dx + Kx3[2]*yaw_z + Kx4[2] *yaw_dz);
962         Thrust_ref3_bar = -(q_alpha * KI1[3] + q_psi *  KI2[3]) -  (Kx1[3]*roll_x
    + Kx2[3] *roll_dx + Kx3[3]*yaw_z + Kx4[3] *yaw_dz);
963
964       // Calculate saturation ratios
965         Thrust_ref0 = Saturate(Thrust_ref0_bar,-LUP_max,LUM_max);
966         Thrust_ref1 = Saturate(Thrust_ref1_bar,-LUM_max,LUP_max);
967         Thrust_ref2 = Saturate(Thrust_ref2_bar,-LUM_max,LUP_max);
968         Thrust_ref3 = Saturate(Thrust_ref3_bar,-LUP_max,LUM_max);
969
```

```
970         T_roll_bar  = Thrust_ref0_bar     + Thrust_ref1_bar     + Thrust_ref2_bar
        + Thrust_ref3_bar;
971         M_psi_bar   = Thrust_ref0_bar *L2 + Thrust_ref1_bar *L1 - Thrust_ref2_bar
    *L1 - Thrust_ref3_bar *L2;
972
973         T_roll      = Thrust_ref0     + Thrust_ref1     + Thrust_ref2     +
    Thrust_ref3;
974         M_psi       = Thrust_ref0 *L2 + Thrust_ref1 *L1 - Thrust_ref2 *L1 -
    Thrust_ref3 *L2;
975
976     }
977
978   // toggel timer 6 toggel
979   Timer6_toggel = Timer6_toggel ^ (uint)1;
980   }
981
982
983 }
984
985 void Error_Handler(void)
986 {
987 }
988
989 #ifdef  USE_FULL_ASSERT
990
991 void assert_failed(uint8_t *file, uint32_t line)
992 {
993 }
994 #endif
```

K.3 EMSD_FUN.h

```
1  #ifndef INC_EMSD_FUN_H_
2  #define INC_EMSD_FUN_H_
3
4  #include <stdbool.h>
5  #include <math.h>   // lrintf
6  #include <stdint.h>
7  #include <tim.h>
8  #include <time.h>
9
10 // MHZ to HZ calculator
11 #define MHZ_TO_HZ(x)  ((x)*1000000);
12
13 // Clock timing defination
14 #define SYS_clock   MHZ_TO_HZ(180);
15 #define APB1_clock      MHZ_TO_HZ(45);
16 #define APB2_clock      MHZ_TO_HZ(90);
17 #define APB1_TIM_clock  MHZ_TO_HZ(90);
18 #define APB2_TIM_clock  MHZ_TO_HZ(180);
19
```

```
20 // Multiply by two (Becouse then the times fit)
21 #define DSHOT150_HZ   MHZ_TO_HZ(1.5 *2);
22 #define DSHOT300_HZ   MHZ_TO_HZ(3   *2);
23 #define DSHOT600_HZ   MHZ_TO_HZ(6   *2);
24
25 // Name of Dshot types
26 typedef enum{
27   DSHOT150,
28   DSHOT300,
29   DSHOT600
30 }Dshot_type_enum;
31
32 // Definitions for motors
33 // Motor 1
34 #define MOTOR_1_TIMER (&htim1)
35 #define MOTOR_1_CHANNEL TIM_CHANNEL_1
36
37 // Motor 2
38 #define MOTOR_2_TIMER (&htim8)
39 #define MOTOR_2_CHANNEL TIM_CHANNEL_2
40
41 // Motor 3
42 #define MOTOR_3_TIMER (&htim8)
43 #define MOTOR_3_CHANNEL TIM_CHANNEL_4
44
45 // Motor 4
46 #define MOTOR_4_TIMER (&htim8)
47 #define MOTOR_4_CHANNEL TIM_CHANNEL_3
48
49 // Definitions of BITLengths
50 // (14/20) = 75% & (7/20) = 35%
51 #define BITLENGTH 20
52 #define BIT1     14
53 #define BIT0      7
54
55 // FRAME is 16 byte
56 #define FRAME_SIZE 16
57 // We use 18 byte to make pause between frames :)
58 #define BUFF_SIZE  17
59
60 // Throttle - range
61 #define Throttle0   48
62 #define Throttle100   2048
63 #define Throttle50 1048
64
65 #define Throttle0_3D   1000
66 #define Throttle100_3D   2000
67 #define Throttle50_3D 1500
68
69 // Throttle in percentage to bit value
70 #define ThrottlePRO(x) lrintf(x*(((float)Throttle100-Throttle0)/100) + Throttle0);
71 #define ThrottlePRO3D(x) lrintf(x*(((float)Throttle100_3D-Throttle0_3D)/200) +
```

```
    Throttle50_3D);
72
73
74 // Functions for use in main.c
75 void Timer_DSHOT_init(Dshot_type_enum dshot_type);
76 void Update_DSHOT(uint16_t* motorvalues,uint8_t* Telemerty_bool);
77 void Trelemertry_change(uint8_t* Telemerty_bools,int* Telemerty_Source,int*
    Telemerty_Counter,bool Telemerty_ON,bool Telemerty_force);
78
79 #endif /* INC_EMSD_FUN_H_ */
```

K.4 EMSD_FUN.c

```
1 #include "EMSD_FUN.h"
2
3 // Variables
    ------------------------------------------------------------------------
4 static int    TIM_INFO[3];
5
6 static uint16_t Motor_1_BUFF[BUFF_SIZE];
7 static uint16_t Motor_2_BUFF[BUFF_SIZE];
8 static uint16_t Motor_3_BUFF[BUFF_SIZE];
9 static uint16_t Motor_4_BUFF[BUFF_SIZE];
10
11
12 int datasentflag1=0;
13 int datasentflag2=0;
14 int datasentflag3=0;
15 int datasentflag4=0;
16
17 // Functions declaration
    ------------------------------------------------------------
18
19 // define functions for function Timer_Dshot_init
20   static void Timer_set(Dshot_type_enum dshot_type,int* TIM_info);
21   static uint32_t Choose_DSHOT(Dshot_type_enum dshot_type);
22   static void Start_PWM();
23   static void dshot_dma_tc_callback(DMA_HandleTypeDef *hdma);
24   static void dshot_put_tc_callback_function();
25
26   // define functions for function Update_DSHOT
27
28   static uint16_t Prepare_packet(uint16_t value,uint8_t telemerty);
29
30   static void Prepare_BUFF(uint16_t* DMA_BUFF,uint16_t value,uint8_t telemerty);
31
32   static void Prepare_ALL_BUFF(uint16_t values[4], uint8_t Telemerty_bools[4]);
33
34   static void Start_DMA();
35   static void Enable_DMA_REQUEST();
36
```

```
37
38
39  //void HAL_TIM_PWM_PulseFinishedCallback(TIM_HandleTypeDef *htim);
40
41 // Functions
    ------------------------------------------------------------------------
42
43  // init timers
44  void Timer_DSHOT_init(Dshot_type_enum dshot_type){
45    // Function to set prescaler and autoreload
46    Timer_set(dshot_type,TIM_INFO);
47    dshot_put_tc_callback_function();
48    Start_PWM();
49  }
50
51  // Update_Dshot
52  void Update_DSHOT(uint16_t* motorvalues,uint8_t* Telemerty_bools){
53    // Write Dshot Datapacket to buffer
54    Prepare_ALL_BUFF(motorvalues,Telemerty_bools);
55    // Sendt buffer to DMA and run PWM signal
56    Start_DMA();
57    Enable_DMA_REQUEST();
58  }
59
60  // Change Trelemertry source
61  void Trelemertry_change(uint8_t* Telemerty_bools,int* Telemerty_Source,int*
      Telemerty_Counter,bool Telemerty_ON,bool Telemerty_force){
62
63    if(Telemerty_force == true){
64        Telemerty_bools[0] = 1;
65        Telemerty_bools[1] = 1;
66        Telemerty_bools[2] = 1;
67        Telemerty_bools[3] = 1;
68    }else{
69
70    // Set Telemerty bit to 1 in one Dshot channel
71    if(*Telemerty_Counter > 0 && Telemerty_ON == true ){
72      Telemerty_bools[*Telemerty_Source] = 1;
73      *Telemerty_Counter = 0;
74    }else{
75      // Set all Telemerty bit to 0 when there are no Telsemerty requests
76      Telemerty_bools[0] = 0;
77      Telemerty_bools[1] = 0;
78      Telemerty_bools[2] = 0;
79      Telemerty_bools[3] = 0;
80        }
81    *Telemerty_Counter = *Telemerty_Counter +1;
82  }
83  }
84
85 // set prescaler and autoreload
86 static void Timer_set(Dshot_type_enum dshot_type,int* TIM_info){
```

```
87 uint16_t Prescaler_APB1;
88 uint16_t Prescaler_APB2;
89 uint32_t apb1_clock = APB1_clock;
90 uint32_t apb2_clock = APB2_clock;
91
92 // Calculate size of prescaler
93 // Prescaler found by clock_freq/desigered_freq - 1 (-1 since counting from 0)
94 Prescaler_APB1 = lrintf((float) apb1_clock/Choose_DSHOT(dshot_type) + 0.01f) - 1;
95 Prescaler_APB2 = lrintf((float) apb2_clock/Choose_DSHOT(dshot_type) + 0.01f) - 1;
96 TIM_info[0] = Prescaler_APB1;
97 TIM_info[1] = Prescaler_APB2;
98 TIM_info[2] = Choose_DSHOT(dshot_type);
99 // Set prescaler __HAL_TIM_SET_PRESCALER(&Adress,Prescaler)
100 __HAL_TIM_SET_PRESCALER(MOTOR_1_TIMER,Prescaler_APB1);
101 __HAL_TIM_SET_PRESCALER(MOTOR_2_TIMER,Prescaler_APB1);
102 __HAL_TIM_SET_PRESCALER(MOTOR_3_TIMER,Prescaler_APB1);
103 __HAL_TIM_SET_PRESCALER(MOTOR_4_TIMER,Prescaler_APB1);
104
105 // Set Autoreload
106 __HAL_TIM_SET_AUTORELOAD(MOTOR_1_TIMER,BITLENGTH);
107 __HAL_TIM_SET_AUTORELOAD(MOTOR_2_TIMER,BITLENGTH);
108 __HAL_TIM_SET_AUTORELOAD(MOTOR_3_TIMER,BITLENGTH);
109 __HAL_TIM_SET_AUTORELOAD(MOTOR_4_TIMER,BITLENGTH);
110 }
111
112 // Chose witch DSHOT Prodocol we use
113 static uint32_t Choose_DSHOT(Dshot_type_enum dshot_type){
114
115   switch(dshot_type){
116   case DSHOT600:
117     return DSHOT600_HZ;
118     break;
119   case DSHOT300:
120     return DSHOT300_HZ;
121     break;
122   default: //DSHOT150
123     return DSHOT150_HZ;
124     break;
125   }
126 }
127
128 static void Prepare_ALL_BUFF(uint16_t values[4], uint8_t Telemerty_bools[4]){
129   // Prepare BUFFER for all DSHOT Channels
130   Prepare_BUFF(Motor_1_BUFF,values[0],Telemerty_bools[0]);
131   Prepare_BUFF(Motor_2_BUFF,values[1],Telemerty_bools[1]);
132   Prepare_BUFF(Motor_3_BUFF,values[2],Telemerty_bools[2]);
133   Prepare_BUFF(Motor_4_BUFF,values[3],Telemerty_bools[3]);
134 }
135
136 static void Prepare_BUFF(uint16_t* DMA_BUFF,uint16_t value,uint8_t Telemerty_bool)
      {
137   // Create packet based on data and Telemerty
```

```
138   uint16_t packet;
139   packet = Prepare_packet(value,Telemerty_bool);
140 
141   // convert bit to PWM length 14 for 1 and 7 for 0
142   for(int i = 0; i < FRAME_SIZE; i++){
143     DMA_BUFF[i] = (packet & 0x8000) ? BIT1:BIT0;
144     packet <<=1;
145   }
146 
147   // Two emty slots for inbetween frames ()
148  DMA_BUFF[16] = 0;
149   //DMA_BUFF[17] = 0;
150 }
151 
152 static uint16_t Prepare_packet(uint16_t value,uint8_t Telemerty_bool){
153   uint16_t packet;
154   bool T_bool;
155 
156   if(Telemerty_bool == 1){
157     T_bool = true;
158   }else{
159     T_bool = false;
160   }
161 
162   //Add telemerty bit to packet;
163   packet = (value << 1)|(T_bool ? 1 : 0);
164 
165   // calculate checksum
166   unsigned crc = 0;
167   crc = (packet ^(packet >> 4) ^ (packet >> 8)) & 0x0f;
168 
169   // Attach checksum to packet
170   packet = (packet << 4) | crc;
171   return (uint16_t) packet;
172 }
173 
174 static void Start_DMA()
175 {
176 
177   HAL_DMA_Start_IT(MOTOR_1_TIMER->hdma[TIM_DMA_ID_CC1], (uint32_t)Motor_1_BUFF, (
      uint32_t)&MOTOR_1_TIMER->Instance->CCR1, BUFF_SIZE);
178   HAL_DMA_Start_IT(MOTOR_2_TIMER->hdma[TIM_DMA_ID_CC2], (uint32_t)Motor_2_BUFF, (
      uint32_t)&MOTOR_2_TIMER->Instance->CCR2, BUFF_SIZE);
179   HAL_DMA_Start_IT(MOTOR_3_TIMER->hdma[TIM_DMA_ID_CC4], (uint32_t)Motor_3_BUFF, (
      uint32_t)&MOTOR_3_TIMER->Instance->CCR4, BUFF_SIZE);
180   HAL_DMA_Start_IT(MOTOR_4_TIMER->hdma[TIM_DMA_ID_CC3], (uint32_t)Motor_4_BUFF, (
      uint32_t)&MOTOR_4_TIMER->Instance->CCR3, BUFF_SIZE);
181 
182 }
183 
184 static void Enable_DMA_REQUEST(){
185   __HAL_TIM_ENABLE_DMA(MOTOR_1_TIMER, TIM_DMA_CC1);
```

```
186   __HAL_TIM_ENABLE_DMA(MOTOR_2_TIMER, TIM_DMA_CC2);
187   __HAL_TIM_ENABLE_DMA(MOTOR_3_TIMER, TIM_DMA_CC4);
188   __HAL_TIM_ENABLE_DMA(MOTOR_4_TIMER, TIM_DMA_CC3);
189 }
190
191 static void Start_PWM()
192 {
193   // Start the timer channel now.
194     // Enabling/disabling DMA request can restart a new cycle without PWM start/
      stop.
195     HAL_TIM_PWM_Start(MOTOR_1_TIMER, MOTOR_1_CHANNEL);
196     HAL_TIM_PWM_Start(MOTOR_2_TIMER, MOTOR_2_CHANNEL);
197   HAL_TIM_PWM_Start(MOTOR_3_TIMER, MOTOR_4_CHANNEL);
198   HAL_TIM_PWM_Start(MOTOR_4_TIMER, MOTOR_3_CHANNEL);
199 }
200
201 static void dshot_put_tc_callback_function()
202 {
203   // TIM_DMA_ID_CCx depends on timer channel
204   MOTOR_1_TIMER->hdma[TIM_DMA_ID_CC1]->XferCpltCallback = dshot_dma_tc_callback;
205   MOTOR_2_TIMER->hdma[TIM_DMA_ID_CC2]->XferCpltCallback = dshot_dma_tc_callback;
206   MOTOR_3_TIMER->hdma[TIM_DMA_ID_CC4]->XferCpltCallback = dshot_dma_tc_callback;
207   MOTOR_4_TIMER->hdma[TIM_DMA_ID_CC3]->XferCpltCallback = dshot_dma_tc_callback;
208 }
209
210 static void dshot_dma_tc_callback(DMA_HandleTypeDef *hdma)
211 {
212   TIM_HandleTypeDef *htim = (TIM_HandleTypeDef *)((DMA_HandleTypeDef *)hdma)->
      Parent;
213
214   if (hdma == htim->hdma[TIM_DMA_ID_CC1])
215   {
216     __HAL_TIM_DISABLE_DMA(htim, TIM_DMA_CC1);
217   }
218   else if(hdma == htim->hdma[TIM_DMA_ID_CC2])
219   {
220     __HAL_TIM_DISABLE_DMA(htim, TIM_DMA_CC2);
221   }
222   else if(hdma == htim->hdma[TIM_DMA_ID_CC3])
223   {
224     __HAL_TIM_DISABLE_DMA(htim, TIM_DMA_CC3);
225   }
226   else if(hdma == htim->hdma[TIM_DMA_ID_CC4])
227   {
228     __HAL_TIM_DISABLE_DMA(htim, TIM_DMA_CC4);
229   }
230 }
```

K.5 Acc_Gyro_Main.ino

```
1 #include "Calculate_Gyro.h"
```

```
#include <stdlib.h>

float RPY[3], filt[3], G_rates[3], Ts = 0.01, Ax, Ay, Az;
int16_t data_str[8];
unsigned long Ts_prev = 100, Ts_current;
String str_Data = "";
byte cmd_byte = 0;

float Test[3];

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  Serial1.begin(115200); // init UART
  delay(2000);

  Gyro.InitGyroscope(Ts); // Init gyroscope + acc + magnotometer

  //Gyro.BIAS(); // Only used to calibration for bias

}

void loop() {
  // put your main code here, to run repeatedly:

  Ts_current = millis();
  if ((float)(Ts_current - Ts_prev) >= Ts*1000){ // Cycle time - 100
     Hz

    // Get attitude estimation from EKF
    Gyro.EKF(RPY[0], RPY[1], RPY[2], (float)(Ts_current - Ts_prev)
       /1000);

    Gyro.FromStorage(G_rates);


    // Update Ts for next sample time
    Ts_prev = Ts_current; P

    //Verify that no overflow in gyro rates occurs
    if (G_rates[0] == Gyro.MaxGyroRate()*PI/180){
      cmd_byte = cmd_byte | 0x04;
```

```
42      } else {
43        cmd_byte = cmd_byte & ~0x04;
44      }
45      if (G_rates[1] == Gyro.MaxGyroRate()*PI/180){
46        cmd_byte = cmd_byte | 0x02;
47
48      } else {
49        cmd_byte = cmd_byte & ~0x02;
50      }
51      if (G_rates[1] == Gyro.MaxGyroRate()*PI/180){
52        cmd_byte = cmd_byte | 0x01;
53      } else {
54        cmd_byte = cmd_byte & ~0x01;
55      }
56
57      // Convert values to int16_t and give 2 decimal place
58      data_str[0] = (int16_t)(RPY[0]*100); // Roll - pos
59      data_str[1] = (int16_t)(RPY[1]*100); // Pitch - pos
60      data_str[2] = (int16_t)(RPY[2]*100); // Yaw - pos
61      data_str[3] = (int16_t)(G_rates[0]*100); // Roll - rate
62      data_str[4] = (int16_t)(G_rates[1]*100); // Pitch - rate
63      data_str[5] = (int16_t)(G_rates[2]*100); // Yaw - rate
64      data_str[6] = cmd_byte; // command byte     (x, x, x, x, x,
           overflow roll, overflow pitch, overflow yaw)
65      data_str[7] = data_str[0] + data_str[1] + data_str[2] + data_str
           [3] + data_str[4] + data_str[5] + (int16_t)data_str[6]; // crc
66      // Create string to send
67      str_Data = String(data_str[0]) + ',' + String(data_str[1]) + ','
           + String(data_str[2]) + ',' + String(data_str[3]) + ',' +
           String(data_str[4]) + ',' + String(data_str[5]) + ',' + String
           (data_str[6]) + ',' + String(data_str[7]);
68      Serial1.println(str_Data); // Send string with UART
69
70      // Serial print
71      //Serial.println(str_Data);
72
73      /*
74      Serial.print(RPY[0],5);
75      Serial.print(",");
76      Serial.print(RPY[1],5);
77      Serial.print(",");*/
78      Serial.println(RPY[2],5);
```

```
79   }
80 }
```

K.6 Acc_Gyro.h

```
1
2 #include <Arduino.h>
3 #include <Wire.h>
4 #include <SPI.h>
5
6 #define LSM6DS3_ADDRESS          0x6A // Adresses of LSM6DS3 - omly
      used with wire
7 #define LSM6DS3_WHO_AM_I_REG     0X0F // Register fixed at 69h
     defining who LSM6DS3 is
8 #define LSM6DS3_CTRL1_XL         0X10 // Linear acceleration sensor
      control register
9 #define LSM6DS3_CTRL2_G          0X11 // Angular rate sensor
     control register
10 #define LSM6DS3_STATUS_REG       0X1E // Sensor status register (
    now data available or not)
11 #define LSM6DS3_CTRL6_C          0X15 // Angular rate sensor
     control register
12 #define LSM6DS3_CTRL7_G          0X16 // Angular rate sensor
     control register
13 #define LSM6DS3_CTRL8_XL         0X17 // Linear acceleration sensor
      control register
14 #define LSM6DS3_CTRL9_XL         0x18 // Linear acceleration sensor
      control register
15
16 #define LSM6DS3_OUT_TEMP_L       0X20 // Temperature data output
     resister low
17 #define LSM6DS3_OUT_TEMP_H       0X21 // Temperature data output
     resister high
18
19 #define LSM6DS3_OUTX_L_G         0X22 // Angular rate sensor pitch
     axis (X) low
20 #define LSM6DS3_OUTX_H_G         0X23 // Angular rate sensor pitch
     axis (X) high
21 #define LSM6DS3_OUTY_L_G         0X24 // Angular rate sensor roll
     axis (Y) low
22 #define LSM6DS3_OUTY_H_G         0X25 // Angular rate sensor roll
     axis (Y) high
```

```
23 #define LSM6DS3_OUTZ_L_G            0X26 // Angular rate sensor yaw
    axis (Z) low
24 #define LSM6DS3_OUTZ_H_G            0X27 // Angular rate sensor yaw
    axis (Z) low
25 #define LSM6DS3_OUTX_L_XL           0X28 // Linear acceleration sensor
    X-axis low
26 #define LSM6DS3_OUTX_H_XL           0X29 // Linear acceleration sensor
    X-axis high
27 #define LSM6DS3_OUTY_L_XL           0X2A // Linear acceleration sensor
    Y-axis low
28 #define LSM6DS3_OUTY_H_XL           0X2B // Linear acceleration sensor
    Y-axis high
29 #define LSM6DS3_OUTZ_L_XL           0X2C // Linear acceleration sensor
    Z-axis low
30 #define LSM6DS3_OUTZ_H_XL           0X2D // Linear acceleration sensor
    Z-axis high
31
32 class LSM6DS3 {                    // Some wierd C++ garbage of
    defining functions
33   public:    // Public functions
34     LSM6DS3(TwoWire& wire, uint8_t slaveAddress);
35     LSM6DS3(SPIClass& spi, int csPin, int irqPin);
36     virtual ~LSM6DS3();
37
38     int begin();
39     void end();
40
41     // Accelerometer
42     virtual int readAccelerationRaw(float& x, float& y, float& z); //
        Results are in g (earth gravity).
43     virtual int accelerationSampleRate(int& DataRate, int&
       ScaleSelection, int& AntiAliasing); // Sampling rate of the
       sensor.
44     virtual int accelerationAvailable(); // Check for available data
       from accelerometer
45
46     // Gyroscope
47     virtual int readGyroscopeRaw(float& x, float& y, float& z); //
       Results are in degrees/second.
48     virtual int gyroscopeSampleRate(int& DataRate, int&
       ScaleSelection, int& FullScaleAt); // Sampling rate of the
       sensor.
```

```
49      virtual int gyroscopeAvailable(); // Check for available data
            from gyroscope
50
51      // Temperature Sensor
52      virtual int readTemperature(float& t); // Results are in deg. C
53      virtual float temperatureSampleRate(); // Sampling rate of the
            sensor.
54      virtual int temperatureAvailable(); // Check for available data
            from temperature sensor
55
56    protected:  // Can only be used by functions in this header
57      int readRegister(uint8_t address); // read registre of address
58      int readRegisters(uint8_t address, uint8_t* data, size_t length);
            // Use SPI or wire to read registres from LSM6DS3
59      int writeRegister(uint8_t address, uint8_t value); //Use SPI or
            wire to write to register
60
61    private:  // Private variables can only be accessed in this class
62      TwoWire* _wire;
63      SPIClass* _spi;
64      uint8_t _slaveAddress;
65      int _csPin;
66      int _irqPin;
67
68      SPISettings _spiSettings;
69  };
70
71  extern LSM6DS3 Acc_Gyro; // Defining LSM6DS3 class to the name
        Acc_Gyro globally
72  #undef IMU // Undefine IMU such that IMU never existed
73  #define IMU Acc_Gyro // Define  IMU to Acc_Gyro class
```

K.7 Acc_Gyro.cpp

```
1  #include "Acc_Gyro.h"
2
3  LSM6DS3::LSM6DS3(TwoWire& wire, uint8_t slaveAddress): //Constructor
       declaration, is automatically called when an object of that class
       is created
4    _wire(&wire), // Initializes the _wire member variable to the
         addressof the wire object
5    _spi(NULL), //Initialised the _spi member to null, indicating that
```

```
    no SPI communication is configured
6   _slaveAddress(slaveAddress) // Address of LSM6DS3 device on the I2C
        bus
7 {
8 }
9
10 LSM6DS3::LSM6DS3(SPIClass& spi, int csPin, int irqPin) : //Contructor
    declaration but for SPI communation
11   _wire(NULL),
12   _spi(&spi),
13   _csPin(csPin),
14   _irqPin(irqPin),
15   _spiSettings(10E6, MSBFIRST, SPI_MODE0)
16 {
17 }
18
19 LSM6DS3::~LSM6DS3() // Is a destructor for the LSM6DS6Class, is
    automatically called when an object of the class is destroyed
20 {
21 }
22
23 int LSM6DS3::begin() // Initialise the sensor and check functionality
24 {
25   if (_spi != NULL) { // Check if _spi is not NULL, if not SPI
       Communication is used
26     pinMode(_csPin, OUTPUT); // SPI is used, configure the chip
         select pin and initialise the SPI interface
27     digitalWrite(_csPin, HIGH);
28     _spi->begin();
29   } else { // SPI is not used, initialise the I2C interface
30     _wire->begin();
31   }
32   if (!(readRegister(LSM6DS3_WHO_AM_I_REG) == 0x6C || readRegister(
       LSM6DS3_WHO_AM_I_REG) == 0x69)) {
33     end();
34     return 0; // Reads the WHO_AM_I_REG and checks the value
         depending of the sensor, return 0 to indicate failure
35   }
36
37   //set the gyroscope control register to work at 104 Hz, 1000 dps
38   writeRegister(LSM6DS3_CTRL2_G, 0x48);
39   // Set the Accelerometer control register to work at 104 Hz, 4 g
```

```
40   // low pass filter (check figure9 of LSM6DS3's datasheet)
41   writeRegister(LSM6DS3_CTRL1_XL, 0x48);
42   // set gyroscope power mode to high performance and bandwidth to 16
         MHz
43   writeRegister(LSM6DS3_CTRL7_G, 0x00);
44   // Set the ODR config register to ODR/4
45   writeRegister(LSM6DS3_CTRL8_XL, 0x00);
46   return 1;
47 }
48
49 void LSM6DS3::end() // Deinitialize the sensor and release any
       resources it may be using
50 {
51   if (_spi != NULL) { // Thecks if SPI communication is used
52     _spi->end(); // Deinitialize the SPI interface
53     digitalWrite(_csPin, LOW); // Set the chip select pin to low
54     pinMode(_csPin, INPUT); // and input mode
55   } else {
56     writeRegister(LSM6DS3_CTRL2_G, 0x00); // Power down gyroscope
57     writeRegister(LSM6DS3_CTRL1_XL, 0x00); // Powe down accelerometer
58     _wire->end(); // Denitialize the I2C interface
59   }
60 }
61
62 int LSM6DS3::readAccelerationRaw(float& x, float& y, float& z)
63 {
64   int16_t data[3];
65   if (!readRegisters(LSM6DS3_OUTX_L_XL, (uint8_t*)data, sizeof(data))
         ) {
66     x = NAN;
67     y = NAN;
68     z = NAN;
69     return 0;
70   }
71   x = data[0];
72   y = data[1];
73   z = data[2];
74   return 1;
75 }
76
77 int LSM6DS3::accelerationAvailable()
78 {
```

```
79   if (readRegister(LSM6DS3_STATUS_REG) & 0x01) {
80     return 1;
81   }
82   return 0;
83 }
84
85 int LSM6DS3::accelerationSampleRate(int& DataRate, int&
       ScaleSelection, int& AntiAliasing)
86 {
87   int8_t CTRLReg;
88   CTRLReg = readRegister(LSM6DS3_CTRL1_XL);
89
90   DataRate = (CTRLReg & 0xF0) >> 4; // Turn off first byte and
       rightshift - Acc data rate
91   ScaleSelection = (CTRLReg & 0x0C) >> 2; // Turn off all but bit 4:5
        and rightshift - Acc scale selection
92   AntiAliasing = (CTRLReg & 0x03); // Turn off all but bit 6:7 - Acc
       anti-aliasing
93
94   if (ScaleSelection == 0){
95     return 2; // Return 2g
96   } else if (ScaleSelection == 1){
97     return 16; // Return 16g
98   } else if (ScaleSelection == 2){
99     return 4; // Return 4g
100  } else if (ScaleSelection == 3){
101    return 8; // Return 8g
102  } else {
103  return 0; // Error
104  }
105
106 }
107
108 int LSM6DS3::readGyroscopeRaw(float& x, float& y, float& z)
109 {
110   int16_t data[3];
111   if (!readRegisters(LSM6DS3_OUTX_L_G, (uint8_t*)data, sizeof(data)))
        {
112     x = NAN;
113     y = NAN;
114     z = NAN;
115     return 0;
```

```
116   }
117
118   x = data[0];
119   y = data[1];
120   z = data[2];
121   return 1;
122 }
123
124 int LSM6DS3::gyroscopeAvailable()
125 {
126   if (readRegister(LSM6DS3_STATUS_REG) & 0x02) {
127     return 1;
128   }
129   return 0;
130 }
131
132 int LSM6DS3::gyroscopeSampleRate(int& DataRate, int& ScaleSelection,
      int& FullScaleAt)
133 {
134   int8_t CTRLReg;
135   CTRLReg = readRegister(LSM6DS3_CTRL2_G);
136
137   DataRate = (CTRLReg & 0xF0) >> 4; // Turn off first byte and
        rightshift - Gyro data rate
138   ScaleSelection = (CTRLReg & 0x0C) >> 2; // Turn off all but bit 4:5
        and rightshift - Gyro scale selection
139   FullScaleAt = (CTRLReg & 0x02) > 1; // Turn off all but bit 6 and
        right shift - Gyro some full scale
140
141   if (ScaleSelection == 0){
142     return 250; // Return 250dps
143   } else if (ScaleSelection == 1){
144     return 500; // Return 500dps
145   } else if (ScaleSelection == 2){
146     return 1000; // Return 1000dps
147   } else if (ScaleSelection == 3){
148     return 2000; // Return 2000dps
149   } else {
150     return 0; // Error
151   }
152
153 }
```

```
154
155 int LSM6DS3::readTemperature(float& t)
156 {
157   int16_t data[1];
158
159   if (!readRegisters(LSM6DS3_OUT_TEMP_L, (uint8_t*)data, sizeof(data)
        )) {
160     t = NAN;
161
162     return 0;
163   }
164
165   t = data[0] / 16.0 + 25;
166
167   return 1;
168 }
169
170 int LSM6DS3::temperatureAvailable()
171 {
172   if (readRegister(LSM6DS3_STATUS_REG) & 0x04) {
173     return 1;
174   }
175
176   return 0;
177 }
178
179 float LSM6DS3::temperatureSampleRate()
180 {
181   return 52.0F;
182 }
183
184 int LSM6DS3::readRegister(uint8_t address)
185 {
186   uint8_t value;
187
188   if (readRegisters(address, &value, sizeof(value)) != 1) {
189     return -1;
190   }
191
192   return value;
193 }
194
```

```
195 int LSM6DS3::readRegisters(uint8_t address, uint8_t* data, size_t
    length) // Reads multiple registers depending of the length
196 { // Reads multiple registers to store all 16 bit values for x, y,
    and z
197 // The address input is the first register for the gyro data. The
    sizeof(data) function used to define length returns 6 bits or 3
    sets of data, each containing 2 bytes
198
199   if (_spi != NULL) { // SPI is used
200     _spi->beginTransaction(_spiSettings); // initates an SPI
        transaction with the settings specified by _spiSetting
201     digitalWrite(_csPin, LOW); // Set the select pin low to select
        the sensor
202     _spi->transfer(0x80 | address); // Sens register address with the
         read bit set 0x80 to the sensor
203     _spi->transfer(data, length); // transfer length bytes of data
        from the sensor to the buffer data
204     digitalWrite(_csPin, HIGH); // Release the select pin by setting
        it high
205     _spi->endTransaction(); // End SPI transaction
206   } else { // I2C is used
207     _wire->beginTransmission(_slaveAddress); // Start I2C
        communication to the addres of the sensor
208     _wire->write(address); // Write register address t othe sensor
209     if (_wire->endTransmission(false) != 0) { // If non-zero value is
         returned it indicates an error
210       return -1;
211     }
212     if (_wire->requestFrom(_slaveAddress, length) != length) { // If
        non-zero value is returned it indicates an error
213       return 0;
214     }
215     for (size_t i = 0; i < length; i++) { // Requests length bytes
        from the sensor. Length is 6 bytes or 48bit
216       // This loop writes to the data variable, which is
          automatically allocated to each row in the array
217       *data++ = _wire->read(); // reads the requirested data from the
          I2C bus and store it in data buffer
218     } // bite starts from the least significant bit and iterates to
        the most significant bit
219     // *data++ increments the address of the pointer itself, making
        it point to the next memory location
```

```
220   }
221   return 1;
222 }
223
224 int LSM6DS3::writeRegister(uint8_t address, uint8_t value)
225 {
226   if (_spi != NULL) {
227     _spi->beginTransaction(_spiSettings);
228     digitalWrite(_csPin, LOW);
229     _spi->transfer(address); // Tell address
230     _spi->transfer(value); // write value to address
231     digitalWrite(_csPin, HIGH);
232     _spi->endTransaction();
233   } else {
234     _wire->beginTransmission(_slaveAddress);
235     _wire->write(address); // Tell address
236     _wire->write(value); // Write to address
237     if (_wire->endTransmission() != 0) {
238       return 0;
239     }
240   }
241   return 1;
242 }
243
244 #ifdef ARDUINO_AVR_UNO_WIFI_REV2 // Checks if the
    ARDUINO_AVR_UNO_WIFI_REV2 macro is defined
245   LSM6DS3 IMU(SPI, SPIIMU_SS, SPIIMU_INT); // If macro is defined,
       compile this function
246 #else
247   LSM6DS3 Acc_Gyro(Wire, LSM6DS3_ADDRESS); // If not defined compile
       this function
248 #endif
```

K.8 Calculate_Gyro.h

```
1 #include "Acc_Gyro.h" // Gyro and accelerometer libary
2 #include "QMC5883LCompass.h" // Magnotometer libary
3 #include <math.h>
4 #include <BasicLinearAlgebra.h>
5
6 using namespace BLA;
7
```

```
8  class Calculate_GyroClass {
9    public:
10   Calculate_GyroClass() {
11         //Initialise buffer to 0
12         buffer[0] = 0.0f;
13         buffer[1] = 0.0f;
14         buffer[2] = 0.0f;
15     }
16
17   // functions
18   void InitGyroscope(float T);
19   virtual void AccRate(float& Ax, float& Ay, float& Az);
20   virtual void GyroRate(float& Gx, float& Gy, float& Gz);
21   virtual void AccAngle(float& AccPitch, float& AccRoll);
22   virtual void KFPos(float& Roll, float& Pitch, float& Yaw);
23   void EKF(float& Roll, float& Pitch, float& Yaw, float Ts);
24   void AccRaw(int& Ax, int& Ay, int& Az);
25   void GyroRaw(int& Gx, int& Gy, int& Gz);
26   void MagRaw(float& Mx, float& My, float& Mz);
27   void MagRate(float& Mx, float& My, float& Mz);
28   void MagFilt(float& Filt_Mx, float& Filt_My, float& Filt_Mz);
29   void AccFilt(float& Filt_Ax, float& Filt_Ay, float& Filt_Az);
30   void GyrFilt(float& Filt_Gx, float& Filt_Gy, float& Filt_Gz);
31   void BIAS();
32   float wrapToPi(float angle);
33   void ToStorage(float X, float Y, float Z);
34   float FromStorage(float* storage_buf);
35   float MaxGyroRate();
36
37   private: // Can only be accessed in this class
38   float AccScale, GyroScale;
39
40   int AccDataRate, AccScaleSelection, AccAntiAliasing, GyroDataRate,
        GyroScaleSelection, GyroFullScaleAt;
41
42   float bias_acc[3] = {0.01,0.03,0.02}, bias_gyr[3] =
        {0.09,0.17,0.18};
43
44   // Variables for Kalman filter
45   float x_predic[3], x_nl[3], P_predic[3], P_nl[3], y_hat[3], S[3],
        Kl[3], Ql[3] = {1, 3, 1}, Rl[3] = {0.2, 0.2, 0.2};
46   float Al=1, B, H=1;
```

```
47  unsigned long TStart, TEnd; // For integration
48
49  // Filters
50  float Temp_M[3], Temp_MO1[3], Temp_A[3], Temp_AO1[3], Temp_G[3],
       Temp_GO1[3];
51
52  // Storage
53  float buffer[3];
54
55  // EKF ____________________________________________
56
57  // Define Q and R as diagonal matrices
58  Matrix<3,3,float> Q;
59  Matrix<3,3,float> R_acc = {0.000023, 0, 0,
60                             0, 0.000024, 0,
61                             0, 0, 0.000022}; // Calculated as
                                variance of acc when stationary (unit
                                 is rad/s)
62  /*{0.000866, 0, 0,
63     0, 0.000866, 0,
64     0, 0, 0.000866}*/
65  Matrix<3,3,float> R_mag = {0.003253, 0, 0,
66                             0, 0.002501, 0,
67                             0, 0, 0.003991}; // Calculated as
                                variance of mag when stationary (unit
                                 is mT)
68
69  Matrix<3,3,float> I = {1, 0, 0,
70                         0, 1, 0,
71                         0, 0, 1}; // Identety matrice
72
73  Matrix<3,1,float> x_n;     // state vector
74
75  Matrix<3,3,float> P_n;  // Covariance matrix
76  Matrix<3,1,float> omega_m1; // Old body gyro rates
77  // Reference vector variables
78  float g_ = 9.8; // Reference vector only down in the ground -
       gravitational constant - vector is (0,0,-g)
79  float Vec_M[3];
80  float g_const = 9.8; // gravitational constant
81  float L = 1.9560;
82 };
```

```
83
84 extern Calculate_GyroClass Calculate_Gyro; // Defining LSM6DS3 class
       to the name Acc_Gyro globally
85 #undef Gyro // Undefine IMU, such that IMU never existed
86 #define Gyro Calculate_Gyro // Define  IMU to Acc_Gyro class
```

K.9 Calculate_Gyro.c

```
1  #include "Calculate_Gyro.h"
2
3  Calculate_GyroClass Calculate_Gyro;
4  QMC5883LCompass compass;
5
6  void Calculate_GyroClass::InitGyroscope(float T){
7    IMU.begin(); // Init IMU (gyro and acc)
8
9    if (!(IMU.accelerationSampleRate(AccDataRate,AccScaleSelection,
         AccAntiAliasing) == 0)){
10     AccScale = (float)IMU.accelerationSampleRate(AccDataRate,
           AccScaleSelection,AccAntiAliasing);
11   }
12   if (!(IMU.gyroscopeSampleRate(GyroDataRate,GyroScaleSelection,
         GyroFullScaleAt) == 0)){
13     GyroScale = (float)IMU.gyroscopeSampleRate(GyroDataRate,
           GyroScaleSelection,GyroFullScaleAt);
14   }
15
16   compass.init(); // init magnotometer
17
18   // Init constants
19   Q = {0.000000094*T, 0, 0, // define Q based on wanted Ts
20         0, 0.00000023*T, 0,
21         0, 0, 0.00000011*T};
22
23   P_n = {1 , 0, 0, 0, 1, 0, 0, 0, 1}; //Initial covariance matrix
24
25   // Initial values for filters
26   AccFilt(Temp_AO1[0], Temp_AO1[1], Temp_AO1[2]);
27   GyrFilt(Temp_GO1[0], Temp_GO1[1], Temp_GO1[2]);
28   MagFilt(Temp_MO1[0], Temp_MO1[1], Temp_MO1[2]);
29
30   // Mag reference vector
```

```
31
32   for (int i; i<100; i++){
33     MagFilt(Vec_M[0], Vec_M[1], Vec_M[2]);
34   }
35 }
36
37 void Calculate_GyroClass::AccRaw(int& Ax, int& Ay, int& Az){
38   float RawAccx, RawAccy, RawAccz;
39   IMU.readAccelerationRaw(RawAccx,RawAccy,RawAccz);
40
41   Ax = (int) RawAccx;
42   Ay = (int) RawAccy;
43   Az = (int) RawAccz;
44 }
45
46 void Calculate_GyroClass::AccRate(float& Ax, float& Ay, float& Az){
47   float RawAccx, RawAccy, RawAccz;
48   IMU.readAccelerationRaw(RawAccx,RawAccy,RawAccz);
49
50   Ax = (RawAccy*AccScale / 32767.0 - bias_acc[0]);
51   Ay = (RawAccx*AccScale / 32767.0 - bias_acc[1]);
52   Az = (RawAccz*AccScale / 32767.0 - bias_acc[2]) *-1;
53 }
54
55 void Calculate_GyroClass::AccFilt(float& Filt_Ax, float& Filt_Ay,
     float& Filt_Az){
56   float alpha_A = 0.5;
57   AccRate(Temp_A[0],Temp_A[1],Temp_A[2]);
58
59   Filt_Ax = (alpha_A*Temp_A[0] + (1-alpha_A)*Temp_AO1[0]);
60   Temp_AO1[0] = Filt_Ax;
61
62   Filt_Ay = (alpha_A*Temp_A[1] + (1-alpha_A)*Temp_AO1[1]);
63   Temp_AO1[1] = Filt_Ay;
64
65   Filt_Az = (alpha_A*Temp_A[2] + (1-alpha_A)*Temp_AO1[2]);
66   Temp_AO1[2] = Filt_Az;
67 }
68
69 void Calculate_GyroClass::GyroRaw(int& Gx, int& Gy, int& Gz){
70   float RawGyrox, RawGyroy, RawGyroz;
71   IMU.readGyroscopeRaw(RawGyrox,RawGyroy,RawGyroz);
```

```
72
73    Gx = (int) RawGyrox;
74    Gy = (int) RawGyroy;
75    Gz = (int) RawGyroz;
76  }
77
78  void Calculate_GyroClass::GyroRate(float& Gx, float& Gy, float& Gz){
79    float RawGyrox, RawGyroy, RawGyroz;
80    IMU.readGyroscopeRaw(RawGyrox,RawGyroy,RawGyroz);
81
82    Gx = ((RawGyroy * GyroScale / 32767.0) - bias_gyr[0]) *-1; // -1 is
        for matching with angle direction and += is for calibration
83    Gy = ((RawGyrox * GyroScale / 32767.0) - bias_gyr[1]) *-1; // For
       some reason is x and y switched according to the datasheet
84    Gz = ((RawGyroz * GyroScale / 32767.0) - bias_gyr[2]);
85  }
86
87  void Calculate_GyroClass::GyrFilt(float& Filt_Gx, float& Filt_Gy,
      float& Filt_Gz){
88    float alpha_G = 0.5;
89    GyroRate(Temp_G[0],Temp_G[1],Temp_G[2]);
90
91    Filt_Gx = alpha_G*Temp_G[0] + (1-alpha_G)*Temp_GO1[0];
92    Temp_GO1[0] = Filt_Gx;
93
94    Filt_Gy = alpha_G*Temp_G[1] + (1-alpha_G)*Temp_GO1[1];
95    Temp_GO1[1] = Filt_Gy;
96
97    Filt_Gz = alpha_G*Temp_G[2] + (1-alpha_G)*Temp_GO1[2];
98    Temp_GO1[2] = Filt_Gz;
99  }
100
101 void Calculate_GyroClass::MagRaw(float& Mx, float& My, float& Mz){
102   compass.read();
103
104   Mx = compass.getX();
105   My = compass.getY();
106   Mz = compass.getZ();
107 }
108
109 void Calculate_GyroClass::MagRate(float& Mx, float& My, float& Mz){
110   compass.read();
```

```
111
112   Mx = (compass.getX() * 2)/32767.0*100; // Convert 2 gauss to mT
113   My = (compass.getY() * 2)/32767.0*100*-1;
114   Mz = (compass.getZ() * 2)/32767.0*100;
115 }
116
117 void Calculate_GyroClass::MagFilt(float& Filt_Mx, float& Filt_My,
      float& Filt_Mz){
118   float alpha_M = 0.5;
119   MagRate(Temp_M[0],Temp_M[1],Temp_M[2]);
120
121   Filt_Mx = alpha_M*Temp_M[0] + (1-alpha_M)*Temp_MO1[0];
122   Temp_MO1[0] = Filt_Mx;
123
124   Filt_My = alpha_M*Temp_M[1] + (1-alpha_M)*Temp_MO1[1];
125   Temp_MO1[1] = Filt_My;
126
127   Filt_Mz = alpha_M*Temp_M[2] + (1-alpha_M)*Temp_MO1[2];
128   Temp_MO1[2] = Filt_Mz;
129 }
130
131 void Calculate_GyroClass::EKF(float& Roll, float& Pitch, float& Yaw,
      float Ts){
132   float Ax, Ay, Az, Gx, Gy, Gz, Mx, My, Mz;
133   // Measurements
134   AccFilt(Ax,Ay,Az);
135   GyrFilt(Gx,Gy,Gz);
136   MagFilt(Mx,My,Mz);
137
138   // Convert to ENU by flipping y - is done in functions
139   float p = Gx * PI/180; // Convert to rad/s
140   float q = Gy * PI/180;
141   float r = Gz * PI/180;
142
143   // Scale to g = 9.8
144   Ax *= g_const;
145   Ay *= g_const;
146   Az *= g_const;
147
148   // Prediction step
        ------------------------------------------------------------
149
```

```
150  // Common trig terms
151  float sx = sin(x_n(0)); float cx = cos(x_n(0));
152  float sy = sin(x_n(1)); float cy = cos(x_n(1));
153  float sz = sin(x_n(2)); float cz = cos(x_n(2));
154  float ty = tan(x_n(1));
155  // f(x,u) non-lin input function
156  /*Matrix<3,1,float> f = {p + r*cx*ty + q*ty*sx,
157                         q*cx - r*sx,
158                         (r*cx/cy) + (q*sx/cy)};*/
159  Matrix<3,1,float> f = {p - r*cx*ty + q*ty*sx,
160                         q*cx + r*sx,
161                         (r*cx/cy) - (q*sx/cy)};
162  ToStorage(f(0), f(1), f(2));
163
164  // Predict states x_n+1 = x_n-1 + T * f(x,u)
165  x_n = x_n + Ts * f;
166
167  // Trig terms are recomputed with estimated states
168  sx = sin(x_n(0)); cx = cos(x_n(0));
169  sy = sin(x_n(1)); cy = cos(x_n(1));
170  sz = sin(x_n(2)); cz = cos(x_n(2));
171  ty = tan(x_n(1));
172  // Jacobian of f(x,u) - linearize at current states
173  /*Matrix<3,3,float> F_ = {(Ts*sy*(q*cx - r*sx))/cy, (Ts*(r*cx + q*
     sx))/(cy*cy), 0,
174                          -Ts*(r*cx + q*sx), 0, 0,
175                          (Ts*(q*cx - r*sx))/cy, (Ts*sy*(r*cx + q*sx)
                               )/(cy*cy), 0};*/
176  Matrix<3,3,float> F_ = {Ts*ty*(q*cx + r*sx), -(Ts*(r*cx - q*sx))/(
     cy*cy), 0,
177    Ts*(r*cx - q*sx), 0, 0,
178    -(Ts*(q*cx + r*sx))/cy, (Ts*ty*(r*cx - q*sx))/cy, 0};
179  // Transpose of F
180  Matrix<3,3,float> FT = {F_(0,0), F_(1,0), F_(2,0),
181                          F_(0,1), F_(1,1), F_(2,1),
182                          F_(0,2), F_(1,2), F_(2,2)};
183
184  //Update covariance P_n+1 = F*P_n-1*FT + Q;
185  P_n =  F_*P_n*FT + Q;
186
187  // Update step for accelerometer
```

```
188
189 // Output funcion h(x,u) for accelerometer
190 Matrix<3,1,float> h_acc = {-g_*sy, cy*g_*sx, -cx*cy*g_};
191
192 // Jacobian of h(x,u) - linearize at estimated states
193 Matrix<3,3,float> H_acc = {0, -cy*g_, 0,
194                            cx*cy*g_, -g_*sx*sy, 0,
195                            cy*g_*sx, cx*g_*sy, 0};
196
197 Matrix<3,3> H_accT = {H_acc(0,0), H_acc(1,0), H_acc(2,0),
198                       H_acc(0,1), H_acc(1,1), H_acc(2,1),
199                       H_acc(0,2), H_acc(1,2), H_acc(2,2)};
200
201 // Kalman gain K = P_n * H^T / (H*P_n*H^T + R)
202 Matrix<3,3,float> K_acc = P_n * H_accT * Inverse(H_acc*P_n*H_accT +
        R_acc);
203
204 // Updata covariance matrix P_n+1 = (I - K*H)*P_n
205 P_n = (I - K_acc*H_acc) * P_n;
206
207 // Updata estimates x_n+1 = x_n + K*(y-h)
208 Matrix<3,1,float> y_acc = {Ax, Ay, Az}; // Acc measurements
209 x_n = x_n + K_acc*(y_acc-h_acc);
210
211 // Update step for magnetometer
       ------------------------------------------------
212
213 // Compute additional common trig terms
214 sx = sin(x_n(0)); cx = cos(x_n(0));
215 sy = sin(x_n(1)); cy = cos(x_n(1));
216 sz = sin(x_n(2)); cz = cos(x_n(2));
217
218 // Output funcion h(x,u) for magnotometer
219 Matrix<3,1,float> h_mag = {Vec_M[2]*sy + Vec_M[0]*cy*cz - Vec_M[1]*
        cy*sz,
220   Vec_M[0]*(cx*sz + cz*sx*sy) + Vec_M[1]*(cx*cz - sx*sy*sz) - Vec_M
        [2]*cy*sx,
221   Vec_M[0]*(sx*sz - cx*cz*sy) + Vec_M[1]*(cz*sx + cx*sy*sz) + Vec_M
        [2]*cx*cy};
222
223 // Jacobian of h(x,u) - linearize at estimated states
224 Matrix<3,3,float> H_mag = {0, Vec_M[2]*cy - Vec_M[0]*cz*sy + Vec_M
```

```
      [1]*sy*sz, -Vec_M[1]*cy*cz - Vec_M[0]*cy*sz,
225   -Vec_M[0]*(sx*sz - cx*cz*sy) - Vec_M[1]*(cz*sx + cx*sy*sz) -
        Vec_M[2]*cx*cy, Vec_M[2]*sx*sy - Vec_M[1]*cy*sx*sz + Vec_M[0]*
        cy*cz*sx, Vec_M[0]*(cx*cz - sx*sy*sz) - Vec_M[1]*(cx*sz + cz*
        sx*sy),
226   Vec_M[0]*(cx*sz + cz*sx*sy) + Vec_M[1]*(cx*cz - sx*sy*sz) - Vec_M
        [2]*cy*sx, Vec_M[1]*cx*cy*sz - Vec_M[0]*cx*cy*cz - Vec_M[2]*cx
        *sy, Vec_M[0]*(cz*sx + cx*sy*sz) - Vec_M[1]*(sx*sz - cx*cz*sy)
        };
227
228   Matrix<3,3,float> H_magT = {H_mag(0,0), H_mag(1,0), H_mag(2,0),
229                               H_mag(0,1), H_mag(1,1), H_mag(2,1),
230                               H_mag(0,2), H_mag(1,2), H_mag(2,2)};
231
232   // Kalman gain K = P_n * H^T / (H*P_n*H^T + R)
233   Matrix<3,3,float> K_mag = P_n * H_magT * Inverse(H_mag*P_n*H_magT +
        R_mag);
234
235   // Updata covariance matrix P_n+1 = (I - K*H)*P_n
236   P_n = (I - K_mag*H_mag) * P_n;
237
238   // Updata estimates x_n+1 = x_n + K*(y-h)
239   Matrix<3,1,float> y_mag = {Mx, My, Mz};
240   x_n = x_n + K_mag*(y_mag-h_mag);
241   /*Serial.print(y_mag);
242   Serial.print(",");
243   Serial.println(h_mag);*/
244   // Output states
245   Roll = x_n(0);
246   Pitch = x_n(1);
247   Yaw = x_n(2);
248 }
249
250 float Calculate_GyroClass::wrapToPi(float angle){
251     while (angle > PI){
252       angle -= 2 * PI;
253       }
254     while (angle < -PI){
255       angle += 2 * PI;
256     }
257     return angle;
258 }
```

```
259
260 void Calculate_GyroClass::ToStorage(float X, float Y, float Z){
261   buffer[0] = X;
262   buffer[1] = Y;
263   buffer[2] = Z;
264
265 }
266
267 float Calculate_GyroClass::FromStorage(float* storage_buf){
268   storage_buf[0] = buffer[0];
269   storage_buf[1] = buffer[1];
270   storage_buf[2] = buffer[2];
271 }
272
273 float Calculate_GyroClass::MaxGyroRate(){
274   return GyroScale;
275 }
276
277 void Calculate_GyroClass::BIAS(){
278   float buf_acc[3] = {0,0,0}, cal_acc[3] = {0,0,0}, buf_gyr[3] =
        {0,0,0}, cal_gyr[3] = {0,0,0};
279   Serial.println("Begin bias measurements");
280   // Take measurements
281   for (int k=0; k<1000; k++){
282   Gyro.GyroRate(buf_gyr[0],buf_gyr[1],buf_gyr[2]);
283   Gyro.AccRate(buf_acc[0],buf_acc[1],buf_acc[2]);
284
285   cal_gyr[0] += buf_gyr[0];
286   cal_gyr[1] += buf_gyr[1];
287   cal_gyr[2] += buf_gyr[2];
288
289   cal_acc[0] += buf_acc[0];
290   cal_acc[1] += buf_acc[1];
291   cal_acc[2] += buf_acc[2];
292
293   }
294
295   // Average measurements
296   cal_gyr[0] /= 1000;
297   cal_gyr[1] /= 1000;
298   cal_gyr[2] /= 1000;
299
```

```
300   cal_acc[0] /= 1000;
301   cal_acc[1] /= 1000;
302   cal_acc[2] /= 1000;
303
304   // Measurements for the gyroscope should be around 0 when
        stationary
305   Serial.print("Bias for gyroscope: X: ");
306   Serial.print(cal_gyr[0]);
307   Serial.print(" Y: ");
308   Serial.print(cal_gyr[1]);
309   Serial.print(" Z: ");
310   Serial.println(cal_gyr[2]);
311
312   //Measurements for accelerometer should be 0 for X and Y, and -1
        for Z
313   Serial.print("Bias for accelerometer: X: ");
314   Serial.print(cal_acc[0]);
315   Serial.print(" Y: ");
316   Serial.print(cal_acc[1]);
317   Serial.print(" Z: ");
318   Serial.println(cal_acc[2] + 1);
319
320   while(1){}
321 }
```

K.10 QMC5883LCompass.h

```
1 #ifndef QMC5883L_Compass
2 #define QMC5883L_Compass
3
4 #include "Arduino.h"
5 #include "Wire.h"
6
7 class QMC5883LCompass{
8   public:
9   QMC5883LCompass();
10          void init();
11   void setADDR(byte b);
12   void setMode(byte mode, byte odr, byte rng, byte osr);
13          void setReset();
14   void read();
15          float getX();
```

```
16          float getY();
17          float getZ();
18
19    private:
20    void _writeReg(byte reg,byte val);
21          void _applyCalibration();
22    void _Normalize();
23          float _get(int index);
24    byte _ADDR = 0x0D;
25          int _vRaw[3] = {0,0,0};
26    float W_inv[3][3] = {{0.9831,    -0.0247,    0.0170},
27                         {-0.0247,    0.9453,    0.1057},
28                         {0.0170,     0.1057,    1.0890}};
29    float V[3] = {-9746.457, -9819.693, -3622.959};
30          float _vCalibrated[3];
31    float _vNormalized[3], _Norm;
32  };
33  #endif
```

K.11 QMC5883LCompass.c

```
1  /*
2  ==========================================================================
3  QMC5883LCompass.h
4  Library for using QMC5583L series chip boards as a compass.
5  Learn more at [https://github.com/mprograms/QMC5883LCompass]
6
7  Supports:
8
9  - Getting values of XYZ axis.
10 - Calculating Azimuth.
11 - Getting 16 point Azimuth bearing direction (0 - 15).
12 - Getting 16 point Azimuth bearing Names (N, NNE, NE, ENE, E, ESE, SE
       , SSE, S, SSW, SW, WSW, W, WNW, NW, NNW)
13 - Smoothing of XYZ readings via rolling averaging and min / max
       removal.
14 - Optional chipset modes (see below)
15
16 ==========================================================================
17
18 v1.0 - June 13, 2019
19 Written by MPrograms
```

```
20 Github: [https://github.com/mprograms/]
21
22 Release under the GNU General Public License v3
23 [https://www.gnu.org/licenses/gpl-3.0.en.html]
24
25 ================================================================
26 FROM QST QMC5883L Datasheet [https://nettigo.pl/attachments/440]
27 ----------------------------------------------------
28  MODE CONTROL (MODE)
29          Standby                 0x00
30          Continuous              0x01
31
32 OUTPUT DATA RATE (ODR)
33          10Hz            0x00
34          50Hz            0x04
35          100Hz           0x08
36          200Hz           0x0C
37
38 FULL SCALE (RNG)
39          2G              0x00
40          8G              0x10
41
42 OVER SAMPLE RATIO (OSR)
43          512             0x00
44          256             0x40
45          128             0x80
46          64              0xC0
47
48 */
49 #include "Arduino.h"
50 #include "QMC5883LCompass.h"
51 #include <Wire.h>
52
53 QMC5883LCompass::QMC5883LCompass() {
54 }
55 /**
56          INIT
57          Initialize Chip - This needs to be called in the sketch setup
               () function.
58          @since v0.1;
59 **/
60 void QMC5883LCompass::init(){
```

```
61          Wire.begin();
62          _writeReg(0x0B,0x01);
63          setMode(0x01,0x0C,0x00,0x80);
64  }
65  /**
66          SET ADDRESS
67          Set the I2C Address of the chip. This needs to be called in
               the sketch setup() function.
68          @since v0.1;
69  **/
70  // Set I2C Address if different then default.
71  void QMC5883LCompass::setADDR(byte b){
72          _ADDR = b;
73  }
74  /**
75          REGISTER
76          Write the register to the chip.
77          @since v0.1;
78  **/
79  // Write register values to chip
80  void QMC5883LCompass::_writeReg(byte r, byte v){
81          Wire.beginTransmission(_ADDR);
82          Wire.write(r);
83          Wire.write(v);
84          Wire.endTransmission();
85  }
86  /**
87          CHIP MODE
88          Set the chip mode.
89          @since v0.1;
90  **/
91  // Set chip mode
92  void QMC5883LCompass::setMode(byte mode, byte odr, byte rng, byte osr
       ){
93          _writeReg(0x09,mode|odr|rng|osr);
94  }
95  /**
96          RESET
97          Reset the chip.
98          @since v0.1;
99  **/
100 // Reset the chip
```

```
101 void QMC5883LCompass::setReset(){
102         _writeReg(0x0A,0x80);
103 }
104 /**
105         READ
106         Read the XYZ axis and save the values in an array.
107         @since v0.1;
108 **/
109 void QMC5883LCompass::read(){
110         Wire.beginTransmission(_ADDR);
111         Wire.write(0x00);
112         int err = Wire.endTransmission();
113         if (!err) {
114                 Wire.requestFrom(_ADDR, (byte)6);
115                 _vRaw[0] = (int)(int16_t)(Wire.read() | Wire.read()
                        << 8);
116                 _vRaw[1] = (int)(int16_t)(Wire.read() | Wire.read()
                        << 8);
117                 _vRaw[2] = (int)(int16_t)(Wire.read() | Wire.read()
                        << 8);
118                 _applyCalibration();
119         }
120 }
121 /**
122     APPLY CALIBRATION
123         This function uses the calibration data provided via @see
                setCalibration() to calculate more
124         accurate readings
125         @author Claus Naveke - TheNitek [https://github.com/TheNitek]

126         Based on this awesome article:
127         https://appelsiini.net/2018/calibrate-magnetometer/
128         @since v1.1.0
129
130 **/
131 void QMC5883LCompass::_applyCalibration(){
132   _vCalibrated[0] = W_inv[0][0]*(_vRaw[0]-V[0]) + W_inv[0][1]*(_vRaw
        [1]-V[1]) + W_inv[0][2]*(_vRaw[2]-V[2]);
133   _vCalibrated[1] = W_inv[1][0]*(_vRaw[0]-V[0]) + W_inv[1][1]*(_vRaw
        [1]-V[1]) + W_inv[1][2]*(_vRaw[2]-V[2]);
134   _vCalibrated[2] = W_inv[2][0]*(_vRaw[0]-V[0]) + W_inv[2][1]*(_vRaw
        [1]-V[1]) + W_inv[2][2]*(_vRaw[2]-V[2]);
```

```
135
136   // Bc = W^-1*(Bp - V)
137   /*
138   _vCalibrated[0] = _vRaw[0];
139   _vCalibrated[1] = _vRaw[1];
140   _vCalibrated[2] = _vRaw[2];
141   */
142 }
143 /**
144         GET X AXIS
145         Read the X axis
146                 @since v0.1;
147         @return int x axis
148 **/
149 float QMC5883LCompass::getX(){
150         return _get(0);
151 }
152 /**
153         GET Y AXIS
154         Read the Y axis
155         @since v0.1;
156         @return int y axis
157 **/
158 float QMC5883LCompass::getY(){
159         return _get(1);
160 }
161 /**
162         GET Z AXIS
163         Read the Z axis
164         @since v0.1;
165         @return int z axis
166 **/
167 float QMC5883LCompass::getZ(){
168         return _get(2);
169 }
170
171 /**
172         GET SENSOR AXIS READING
173         Get the smoothed, calibration, or raw data from a given
               sensor axis
174         @since v1.1.0
175         @return int sensor axis value
```

```
176 **/
177 float QMC5883LCompass::_get(int i){
178   //_Normalize();
179   //return _vNormalized[i];
180         return _vCalibrated[i];
181 }
182
183 void QMC5883LCompass::_Normalize(){
184   _Norm = sqrt(_vCalibrated[0]*_vCalibrated[0] + _vCalibrated[1]*
        _vCalibrated[1] + _vCalibrated[2]*_vCalibrated[2]);
185   _vNormalized[0] = _vCalibrated[0]/_Norm;
186   _vNormalized[1] = _vCalibrated[1]/_Norm;
187   _vNormalized[2] = _vCalibrated[2]/_Norm;
188 }
```