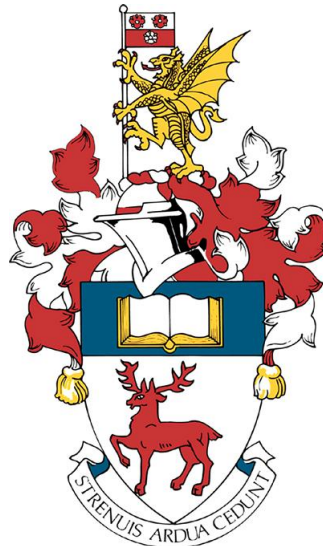


UNIVERSITY OF SOUTHAMPTON

UAV Smart Wing - active control of wing lift to suppress short-period oscillations

Author:
Bhagyesh GOVILKAR

Supervisor:
Dr. T. Glyn THOMAS



*This report is submitted in partial fulfillment of the requirements for the MEng
Aeronautics and Astronautics, Faculty of Engineering and the Environment, University
of Southampton*

April 14, 2018

Declaration

I, Bhagyesh Govilkar declare that this thesis and the work presented in it are my own and has been generated by me as the result of my own original research. I confirm that:

1. This work was done wholly or mainly while in candidature for a degree at this University;
2. Where any part of this thesis has previously been submitted for any other qualification at this University or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others, this is always clearly attributed;
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
5. I have acknowledged all main sources of help;
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
7. None of this work has been published before submission.

Acknowledgements

I would like to acknowledge my supervisor, Dr T. Glyn Thomas who provided me with guidance every step of the way and supported my ideas that enabled me to complete this project...

List of Abbreviations

LAH List Abbreviations Here
WSF What (it) Stands For

Physical Constants

Speed of Light $c_0 = 2.997\,924\,58 \times 10^8 \text{ m s}^{-1}$ (exact)

List of Symbols

a	distance	m
P	power	W (J s ⁻¹)
ω	angular frequency	rad

Contents

1	Introduction	1
1.1	Literature review	2
1.1.1	Phugoid suppression	2
1.1.2	Aeroelastic control in wind turbines	2
1.1.3	Aims and objectives	3
2	Theory	5
2.1	Flight dynamics	5
2.2	Mitigation of longitudinal dynamic modes	6
2.2.1	SPO regulations and qualification	6
2.3	Trailing edge flap	7
2.4	PID controller	8
3	Method	9
3.1	Pressure sensors	9
3.2	Actuator	10
3.3	Computer and programming language	11
3.4	Design of the wing and gust generator and CAD modelling	11
3.4.1	Main Wing	11
3.4.2	Gust Generator	12
3.4.3	Assembly	12
3.5	Electronic setup	13
3.6	Data filtering	13
3.7	Sensor calibration	13
3.8	Plant dynamics	13
3.8.1	Dynamics of the servo	13
3.8.2	Dynamics of the flap	15
3.8.3	MATLAB/Simulink first principles modelling	16
3.9	Lift Estimation	16
3.10	The code	16
4	Wind tunnel testing	17
4.1	Testing Methodology	17
4.2	Results	17
4.2.1	MATLAB data driven modelling	17
4.3	Discussion of results	17
5	Summary	19
6	Further recommendations	21

References	23
A High level	25
A.1 Low level	25

Chapter 1

Introduction

Unmanned Aerial Vehicles are becoming increasingly mainstream. With faster and more efficient ways of manufacturing such as 3D printing gaining momentum, this will only serve to bolster the emerging market.

There are several contemporary and potential applications of UAVs: they are widely used in agriculture to give farmers a detailed picture of the health of their farm, the amount of resources such as pesticides required and they can help cut down on labour costs; consumers and photographers are keen on buying products from UAV manufacturers such as DJI because these devices enable people to capture images/videos from extreme heights. The defence sector are using UAVs mostly to carry out surveillance and reconnaissance activities, a handful are even being used for offensive operations such as the MQ-9 Reaper. Commercial services such as Amazon Inc. have been known to conduct research in this technology. The company has an ambitious goal: using UAVs to transport goods to their customers.

With this increasing usage of UAVs, there is a strong need for a control system that can give the user a stable, reliable and robust aircraft. The aim is to reduce the frequency of UAVs crashing, increase the life-span and reduce the maintenance costs.

In this project, I will be looking specifically at a phenomenon known as a "short-period oscillation". This is one of the two longitudinal dynamic modes that occur on every aircraft, the other being a "phugoid oscillation". Short-period oscillations are characterised by the rapid decay and high frequency. A typical SPO can settle within a second and occurs at 1-2Hz. A pilot can induce a SPO by a sharp pitch input. They can also naturally occur due to a gust (impulsive increase/decrease in airspeed and/or angle of attack).

A UAV control system that does not have a SPO model is vulnerable to growing instability. It can act to reinforce the oscillation instead of cancelling them. Even if the system is proven to not introduce dynamic instabilities, the peak loading on the wing during a SPO event can test the structural limits of the aircraft making structural failures more likely. While this is not economically ideal, a more pressing concern is the health and safety risk created by such an event.

In this project, my goal will be to design a control system that minimises the effects of SPOs making UAVs safer to operate and also more economical.

1.1 Literature review

The knowledge we currently have and the applications of this knowledge need to be established first to identify the areas in which we lack an understanding and where improvements and optimisations can be made. In this chapter, I intend to do exactly that. By the end of this chapter, it will be clear that there has been little development in this area.

1.1.1 Phugoid suppression

It is worth looking at how we deal with phugoid oscillations first since suppressing phugoid oscillations has been well documented. This is achieved through the use of a Pitch Attitude Controller and has been described by Etkin [2]. More recently, a pitch-rate feedback control system was experimented with using an integral-separated PID controller [4]. In this controller, an error threshold is set below which the integral term in the PID calculation will be eliminated. This essentially makes it a PD controller until the pre-set threshold is crossed. The advantage of this is to limit the instability that the integral term brings but retain its tendency to remove steady-state errors. In both cases, the control system did well to suppress Phugoid oscillations and managed to weaken SPOs but relied on gyroscopes or other attitude sensors. This meant that the control system took action AFTER the flight dynamics caused a deviation in the pitch rate/angle from the nominal. This model is currently the state-of-the-art, but includes an intrinsic lag between the onset of the oscillations and the corrective action of the controller.

There has been little development in designing a control system that detects the disturbance at the source and takes action BEFORE the flight dynamics can cause an appreciable change in attitude. In this project, I will aim to achieve a control system that does not wait for a large change in attitude to occur before acting. This implies that the airflow around the wing will be controlled and the control of the flight trajectory is merely a consequence.

1.1.2 Aeroelastic control in wind turbines

Wind turbine blades are comparable to aircraft wings since both are lift generating devices and both run into similar issues. I looked at wind turbines and how active control systems are being used to improve performance, safety and robustness of the blades. There are multiple studies that look at aeroelastic control of wind turbine blades using trailing edge flaps and load sensors for input. There were two issues these studies dealt with: to suppress flutter[7], an aeroelastic phenomenon in which a structure in a fluid flow undergoes simple-harmonic motion; and to delay fatigue of the blades[8] by reducing the cyclic loading. The first issue is a safety concern while the second issue is about the economic sustainability of the wind turbines. Replacing large turbine blades can be difficult and expensive because of their size. These turbines can also be located in challenging environments like the North Sea to complicate the logistics even further. Therefore, a control system that can elongate the life of a single blade is highly sought-after.

These studies have shown great success. Experiments showed that under gusting environments, settling times can be reduced by 50%. In turbulence, the standard deviation of the load oscillation was reduced by 30%. In theory, this problem is quite similar to managing longitudinal dynamic modes in aircraft and the benefits are also similar. In both cases, simple harmonic motion is causing excessive loading on the wing or blade and reducing the life of the structure. Positive feedback can cause a disaster and is potentially a safety hazard. Performance of the system is hampered by these oscillations and this has an effect on the operational costs.

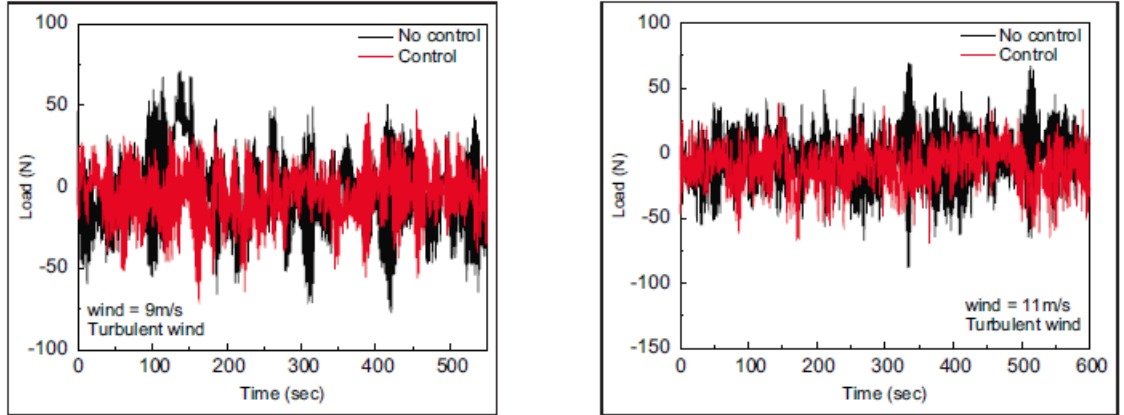


FIGURE 1.1: The load variation has shown decrease in both tests [8]

This review of literature will help me set realistic objectives for this project. I should be able to achieve similar results to the study discussed above.

1.1.3 Aims and objectives

I will aim to design a control system that manages to suppress the rapid change in lift production seen by the wing through a SPO.

The control system should also advance the settling time of the system compared to an existing control system that settles an SPO in 0.263 seconds. I will aim to settle the system within 50% of that time.

The peak load on the wing should be smaller in the gust than it would be without any controller.

Lastly, I will attempt to use first principles modelling to determine the optimal PID gains and design the control system. Then, based on wind tunnel data, I will use heuristic techniques driven by collected data to further improve the performance. Eventually, I will use trial and error to fine tune the controller gains. Comparing the gains determined by Simulink to the optimal gains determined by testing will verify whether the first principles model was valid. If not, I will be exploring the possible flaws in the model. This will be helpful for designing a control system for UAVs whose plants have not yet been constructed.

Chapter 2

Theory

In this section, relevant theory that has been well established will be outlined. Knowing this information is required to understand why SPOs occur. This information may also be necessary for parts of the project.

2.1 Flight dynamics

We need to understand flight dynamics to recognise the source of these oscillations.

I will use the longitudinal equations of motion as a starting point :

$$\begin{bmatrix} \Delta \dot{u} \\ \dot{w} \\ \dot{q} \\ \Delta \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{\dot{X}_u}{m} & \frac{\dot{X}_w}{m} & 0 & -g \cos \theta_0 \\ \frac{\dot{Z}_u}{m - \dot{Z}_{\dot{w}}} & \frac{\dot{Z}_w}{m - \dot{Z}_{\dot{w}}} & \frac{\dot{Z}_q + m U_\infty}{m - \dot{Z}_{\dot{w}}} & \frac{-m g \sin \theta_0}{m - \dot{Z}_{\dot{w}}} \\ \frac{1}{I_y} \left[\dot{M}_u + \frac{\dot{M}_{\dot{w}} \dot{Z}_u}{m - \dot{Z}_{\dot{w}}} \right] & \frac{1}{I_y} \left[\dot{M}_w + \frac{\dot{M}_{\dot{w}} \dot{Z}_w}{m - \dot{Z}_{\dot{w}}} \right] & \frac{1}{I_y} \left[\dot{M}_q + \frac{\dot{M}_{\dot{w}} (\dot{Z}_q + m U_\infty)}{m - \dot{Z}_{\dot{w}}} \right] & -\frac{M_{\dot{w}} m g \sin(\theta_0)}{I_y (m - \dot{Z}_{\dot{w}})} \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \Delta u \\ w \\ q \\ \Delta \theta \end{bmatrix} + \begin{bmatrix} \frac{\Delta \dot{X}_c}{m} \\ \frac{\Delta \dot{Z}_c}{m - \dot{Z}_{\dot{w}}} \\ \frac{\Delta \dot{M}_c}{I_y} + \frac{\dot{M}_{\dot{w}}}{I_y} \frac{\Delta \dot{Z}_c}{m - \dot{Z}_{\dot{w}}} \\ 0 \end{bmatrix} \quad (2.1)$$

The derivation of these equations are fairly straightforward but quite lengthy and can be found in virtually any flight mechanics/dynamics textbook [2]. They are based purely on classical/Newtonian mechanics.

These equations can be rewritten in the form:

$$\dot{\mathbf{x}} = \mathbf{Ax} + \mathbf{B} \quad (2.2)$$

Where x is the state vector, \mathbf{A} is the system matrix (constant) and \mathbf{B} is a vector of control forces and moments which can be considered 0.

$$\dot{\mathbf{x}} = \mathbf{Ax} \quad (2.3)$$

This is a first order ODE which means that the solutions are in the form of

$$\mathbf{x} = \mathbf{x}_0 e^{\lambda t} \quad (2.4)$$

$$\lambda \mathbf{x}_0 = \mathbf{Ax}_0 \quad (2.5)$$

$$(\mathbf{A} - \lambda \mathbf{I}) \mathbf{x}_0 = 0 \quad (2.6)$$

Non-trivial solutions occur when $\det(\mathbf{A} - \lambda\mathbf{I}) = 0$. Solving this equation for any given aircraft will give a quartic equation (4th order polynomial) which when solved will give two pairs of complex conjugates. One of the pairs corresponds to the phugoid mode while the other one corresponds to the SPO.

For example, we can consider the Navion aircraft [1]:

$$\mathbf{A} = \begin{bmatrix} -0.09148 & 0.04242 & 0 & -32.17 \\ 10.51 & -3.066 & 152 & 0 \\ 0.2054 & -0.05581 & -2.114 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad (2.7)$$

Computing the eigenvalue of this matrix: The matrix produces two pairs of complex

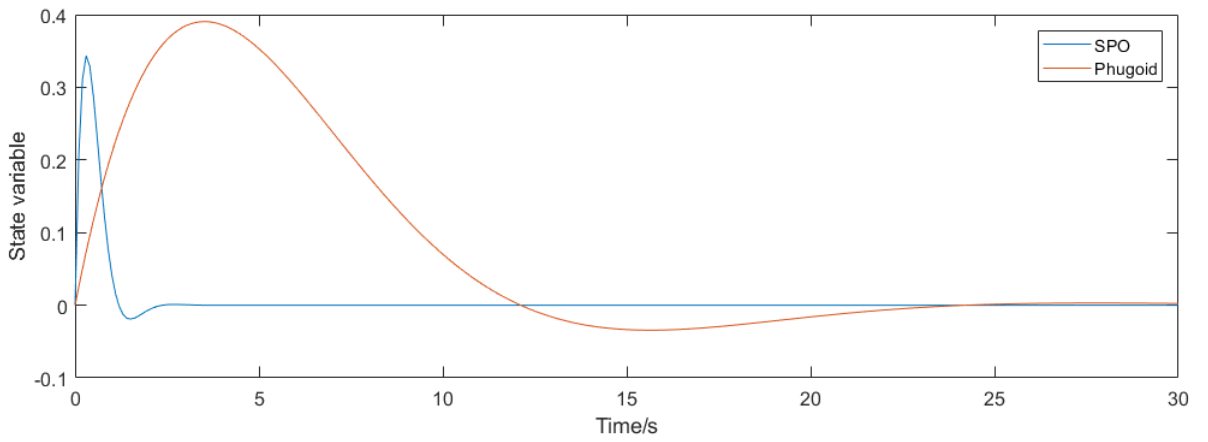


FIGURE 2.1: MATLAB output

conjugate eigenvalues. The first pair $\lambda_{1,2} = -2.4352 \pm 2.6461i$ is the SPO mode. The decay rate (the real part) is much greater than the other pair $\lambda_{3,4} = -0.2006 \pm 0.2593i$ and so is the frequency (coefficient of i).

2.2 Mitigation of longitudinal dynamic modes

2.2.1 SPO regulations and qualification

The United States of America's Federal Aviation Authority regulates SPOs in FAR Part 23.181 [5] as follows: Any short period oscillation not including combined lateral-directional oscillations occurring between the stalling speed and the maximum allowable speed appropriate to the configuration of the airplane must be heavily damped with the primary controls -

- (1) Free; and
- (2) In a fixed position.

United Kingdom's Civil Aviation Authority regulates SPOs in a very similar manner and can be found in CAP482 S181.

The quality of handling is largely determined by pilot opinion. This is visualised by pilot opinion charts such as the one shown below.

From SPO approximation [6] we can determine the natural frequency and damping ratio:

$$\omega_n = \sqrt{\dot{Z}_w \dot{M}_q + \dot{M}_w \dot{Z}_q} \quad (2.8)$$

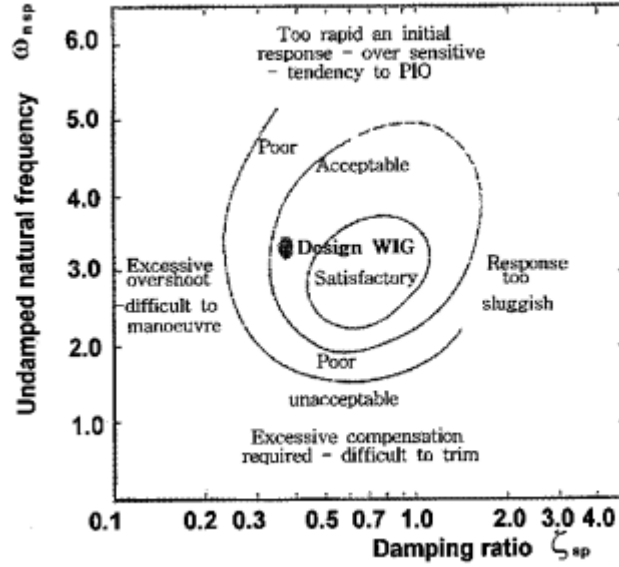


FIGURE 2.2: Short-period pilot opinion contours [3]

$$\zeta = \frac{\dot{M}_q + \dot{Z}_w}{\omega_n} \quad (2.9)$$

Therefore, simply by adjusting the aerodynamic derivatives, it is possible to conform to the regulations. The regulations do not make the existence of a control system with a SPO dedicated model mandatory. This may explain the lack of development in this field. Thus, to fill this gap I will be developing a system that will focus on minimising the impact of SPOs.

2.3 Trailing edge flap

Flaps are typically used as high-lift devices deployed in conditions where the airspeed is low. Increased lift is desirable to minimise the distance to take off or to achieve slower and steeper approaches during landing because extending flaps will lower the stall speed of the aircraft. The principle of operation is quite straightforward. Deflecting the flap downwards will increase the C_L and deflecting it upwards will decrease it. This in turn will change the lift produced:

$$L = \frac{1}{2} \rho V^2 S C_L \quad (2.10)$$

For a plain flap, the increase in lift can also be understood by realising that the airspeed under the wing will be slowed down when the flap is deflected downwards. Therefore, some of the dynamic pressure will be converted to static pressure increasing the pressure under the wing and consequently the lift. Deflecting the flap upwards will have the opposite effect: the speed at the top of the of the airfoil will be slowed down which will increase the static pressure here. This increased pressure acts in the direction that will reduce the lift.

A plain flap can both increase and decrease a lift which will be useful to regulate lift through a SPO.

2.4 PID controller

While PID controller (or its variations) is not the only controller in existence, it is by far the most commonly used controller in industrial applications. The reason for its popularity is most likely because it is easy to implement and gives the user the ability to fine tune the performance of the control system. For example, if the user desires a fast but less robust (allows overshoot) system, he/she can increase the integral gain.

By adjusting the gains, you can change the rise time, peak time, settling time, percentage overshoot and the steady state error (if applicable).

Chapter 3

Method

In this chapter, major design decisions and developing the tools required for this project will be discussed. There are for major decisions that need to be made: Pressure sensors, actuators, micro-controller, and the programming language.

3.1 Pressure sensors

This is one of the most important design decision for this project because they determine the ability to detect a change in airflow which is essential for a closed-loop feedback system.

There are a variety of sensors to choose from. There are absolute pressure sensors that simply measure the gauge or absolute pressure of air at a single point. Then there are pressure sensors that measure differential pressure. These have two ports and the sensor will measure the difference in pressure between them.

To calculate the lift on the wing, we will need to know the pressure distribution along the top and the bottom of the wing. A single differential pressure sensor can tell the difference in pressure at a certain chord position on the wing. For the same information, two absolute pressure sensors will be required. Furthermore, the airspeed in the wind tunnel will need to be measured. Using a differential sensor, one port can be subjected to the total/stagnation pressure from a pitot probe, and the second port will be subjected to the static pressure. The sensor will instantly give the difference between the two pressures which is the dynamic pressure. The airspeed can then be easily calculated using Bernoulli's principle:

$$V = \sqrt{\frac{2(p_0 - p)}{\rho}} \quad (3.1)$$

Thus, for convenience a differential type sensor will be used. TE connectivity makes a sensor called MS4524DO in both differential and absolute types. There is also the option to choose between the I2C and SPI data transmission buses.

There are merits to using both buses. The I2C bus is very common and many devices use it to communicate with a 'master'. The bus typically runs at 100kbps but can reach 400kbps on fast mode. It uses a data line (SDA) and a clock line (SCL) and is therefore sometimes called 'two wire interface' (TWI).

SPI on the other hand runs much faster at around 10Mbps. The interface requires a clock line (SCLK), a MISO line (Master in Slave out), sometimes a MOSI line (Master out Slave in) and a slave select pin.

The slave select pin allows communication with a specific device on the line. In this case, we will have four pressure sensors but we will need to talk to one sensor before moving on to the next.

I2C relies on the device address. Each I2C device is given a specific address which needs to be provided before data transmission can happen. All available I2C MS4525DO sensors have the address 0x28 which creates a conflict. The I2C bus will not know which sensor to communicate with because they're all at the same address. A 'switching' mechanism will be needed to make this option viable.

The most suitable bus for this application is obviously SPI. However, there is a severe lack of availability of these sensors at a reasonable price. A compromise in speed had to be made by choosing the I2C sensor which are widely available. However, for my purpose the speed exceeds Nyquist criterion (test described in chapter) so the lower bus speed will not be an issue. As mentioned above, a mechanism to resolve the address conflict will be needed.

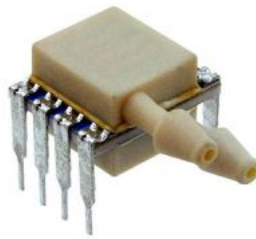


FIGURE 3.1: Differential type MS4525DO sensor

3.2 Actuator

The SPO is a short lived and fast changing phenomenon. It is important to have a mechanism that does not create too much lag between the generation of the control signal and the actuation completing. A significant lag can even reinforce the oscillation and create dynamic instability. In this case the system response would have to be slowed down to stabilise the system. This will increase the settling time and would be a great compromise.

I will use a BMS-631 servo coupled to the trailing edge flap of the wing. The manufacturer states that it can turn 60° in 0.10 secs at 5V input voltage and 0.08 secs at 6V. This is extremely fast compared to other servos on the market and is the most suitable device for this project. The manufacturer does not provide information on exactly how the angular displacement changes with time. The speed mentioned above is not sufficient information to give an accurate representation of the servo dynamics which many sources claim is a first order function. I will need to measure the servo's performance so that I have a close approximation of the plant before testing commences.

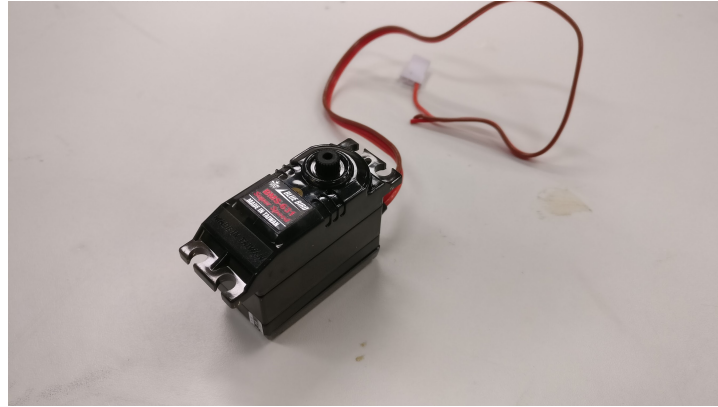


FIGURE 3.2: BMS-631

3.3 Computer and programming language

At the moment, there are two options for an embedded system: Arduino and Raspberry Pi.

The Arduino uses an Atmel ATmega 2560 processor with a clock speed of 16MHz. The program for the Arduino is written in a subset of the C/C++ programming language on a desktop computer and copied to the onboard flash memory over USB typically. The Arduino IDE can be used to compile and write the executable to the flash memory. It also allows for serial communication with the device which is useful for debugging.

The Raspberry Pi has a 4-core 1.2GHz Broadcom BCM2837 64bit CPU running at 1.2 GHz which is much faster than the Arduino Mega. The extra processing power allows the user to use the Raspberry Pi as a full fledged computer. The programs can even be written on the device itself. However, it will be easier to program on a desktop and push the code to the Raspberry Pi over WiFi using the Secure Shell (SSH) protocol.

The Raspberry Pi also gives the user a choice of programming languages to use unlike the Arduino. Python is most commonly used to code for the Raspberry Pi. However, it would be more suitable to use a compiled language like C/C++ rather than an interpreted language like Python. While C can be difficult and more time consuming than Python to write and debug, it does run faster. The table below [9] shows the results from a Mandelbrot Set test (a recursive algorithm) for Python 3 and C++ with the g++ compiler.

<i>source</i>	<i>secs</i>	<i>mem</i>	<i>gz</i>	<i>cpu</i>	<i>cpuload</i>
<i>Python3</i>	225.24	15736	688	899.25	100%100%100%100%
<i>C++g++</i>	1.64	27636	1002	6.51	100%99%100%99%

Speed is important to control the short-lived SPO and thus, C++ executed on the Raspberry Pi 3 is the most suitable option.

3.4 Design of the wing and gust generator

This section will focus on the physical design of the wing that will be placed in the wind tunnel.

3.4.1 Main Wing

To calculate the lift produced by the wing, we will need local static pressure readings from several points along the top and the bottom of the wing. The easiest way to do this is by creating pressure tappings at specified points along the wing surface.

Firstly, an aerofoil needs to be chosen. I chose the NACA2412 aerofoil. This decision was not arbitrary. There were two major reasons for choosing this aerofoil: the moderate thickness of 12% chord length and a camber of 2% at 40% chord length. This increases the L/D ratio and is good for low speed flight which is the regime the Bormatec Explorer and most UAVs fly in. Secondly, there is a study that outlines how the C_L of the aerofoil changes with flap deflection. This can be used to determine the plant transfer function for the MATLAB simulation based on first principles.

The wing has a chord of 0.25m and a span of 0.35m. The trailing edge flap begins at x/c of 0.75. The pressure tappings are at x/c of 0.2, 0.4 and 0.6. There are better locations for these pressure sensors where the lift estimates will be more accurate. However, finding the optimum locations is a large task and can be a separate project. For my project, the accuracy of the lift estimate is not highly important. The aim is to produce a controller that responds well to a change in airflow. Therefore, a lift estimator that can give values within 10% of the actual lift on average will be good enough to measure the response of the control system.

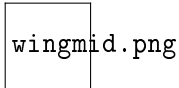


FIGURE 3.3: Midsection of the wing with the internal tubing highlighted

3.4.2 Gust Generator

To simulate an impulsive change in airspeed, a "gust generator" will be placed upstream from the wing. This consists of two NACA0015 airfoils with chord of 0.25 m and 0.5m span coupled to servos. The diagram below illustrates the configuration.

As the flaps move closer together, the airspeed of the flow between them will increase because of the decreasing cross-sectional area. This is simply due to conservation of mass.

$$\dot{m} = \rho UA \quad (3.2)$$

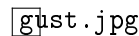


FIGURE 3.4: Gust generator in the wind tunnel

3.4.3 Assembly

The main wing was split into four parts: the two ends of the wings, the middle section and the flap. Different manufacturing processes had to be utilised to obtain these parts quickly and of sufficient quality.

The middle section is geometrically complex due to the routing of the pressure tappings. It is not very practical to hand craft the part and not feasible through 2D processes such

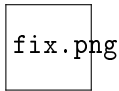


FIGURE 3.5: Fixture on the right hand side of the wind tunnel

as laser cutting or foam cutting due to the 3D geometric features. 3D printing is the only process that is suitable and available for this component. Unfortunately, the span of the wing is too great (30cm) for the 3D printer so the ends of the wing need to be extended. The 3D printed part was made 20cm wide.

To extend the wing by 5cm each side, the profile of the middle section was saved as a .dxf file. This profile was laser cut out of a 5mm thick plywood 20 times and glued together side-to-side.

The flap was made similarly. The profile was laser cut 60 times and glued together.

The main wing has been supported and mounted through two 6mm rods that go through the wing and attach to some fixtures on the walls of the tunnel. These fixtures were designed specifically for this project.

Looking upstream, the fixture disc (Figure 3.5) on the right hand side has holes for the carbon fibre rods, a cutout for the flap servo to be mounted in and several holes near the edges to allow changing the angle of attack of the wing to a desired setting. The angular separation between these holes is 5° .

The pressure tapings end at the right side of the middle section of the wing. Sections of a 4mm (outer diameter) copper pipe were placed in these holes and superglued in place. These created "ports" which allowed easy and secure connection of the pressure tapings to the pressure sensors. Six equal sections of 3mm a PVC tube(inner diameter) were cut to facilitate this connection. One end of each tube was secured to the pressure sensor ports, the other end had to be heated and expanded to fit onto the larger copper ports on the wing. This provided an excellent tight seal which is beneficial because less pressure will be lost.

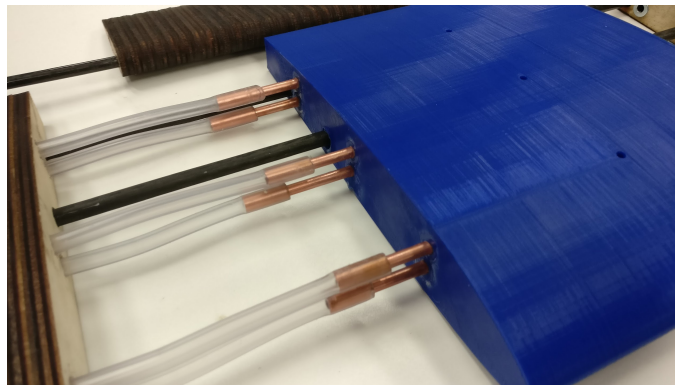


FIGURE 3.6: Picture of the PVC tubing and the midsection of the wing

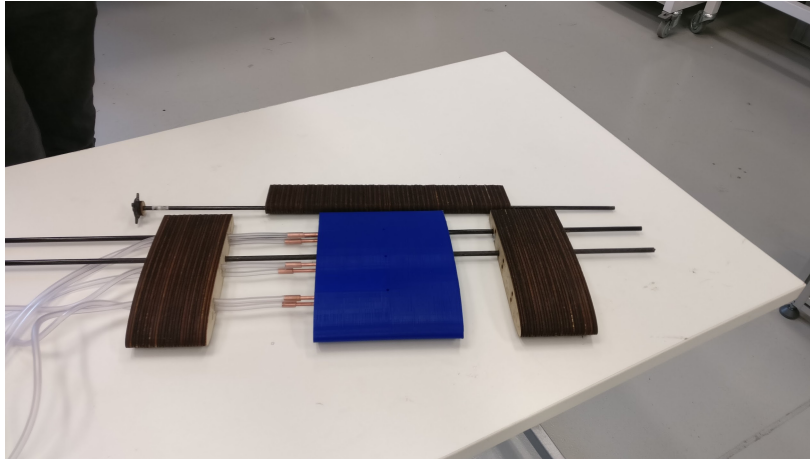


FIGURE 3.7: Picture showing how the different parts of the wing were integrated


wind.jpg

FIGURE 3.8: Picture showing the main wing assembly in the wind tunnel

3.5 Electronic setup

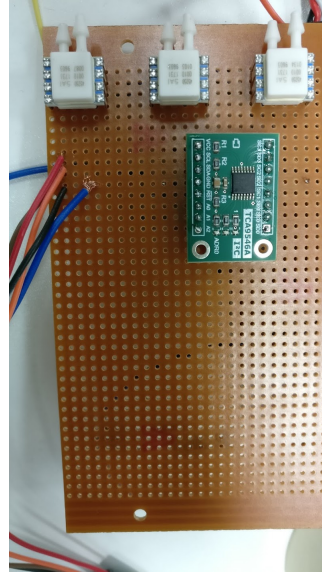


FIGURE 3.9: Photo of the breadboard with the multiplexer and the pressure sensors

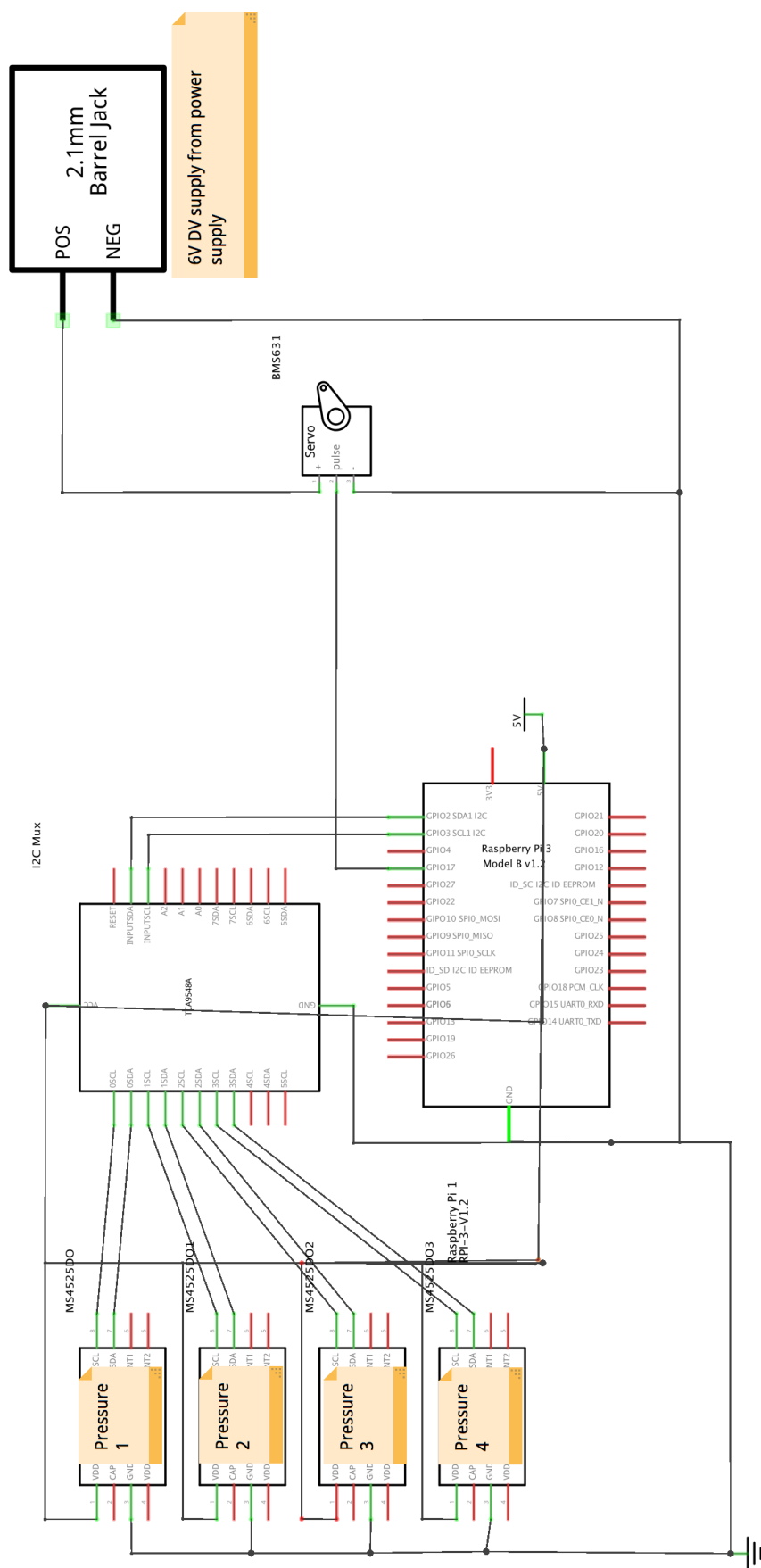
As mentioned in section 3.1, the I2C pressure sensors all have the same address so are bound to create a conflict unless a switching mechanism is constructed. I used an I2C multiplexer called TCA9546a which is an I2C device itself.

It has four channels thus can handle four I2C devices. To select a channel, a number is sent to the I2C address 0x77 (where the multiplexer is located).

Channel 0	0x01
Channel 1	0x02
Channel 2	0x04
Channel 3	0x08

Once the channel is selected, I2C communication between the Raspberry Pi and the selected pressure sensor can begin. Figure 3.3 below illustrates the connection between the multiplexer and the pressure sensors.

Note that the servo is powered externally via a 6V DC wall adapter, but the ground of the external power source and the Raspberry Pi power source are tied together because the servo and the Raspberry Pi need a common ground.



fritzing

FIGURE 3.10: Fritzing diagram illustrating the electronic set up

3.6 Sensor calibration

The pressure sensors mentioned in section 3.2 need to be calibrated against known values of pressures before they can be used. To achieve this, I used the principles of a manometer. The difference in pressure heads at the two ends of the tube is proportional to the difference in pressure.

$$\Delta P = \rho g \Delta h \quad (3.3)$$

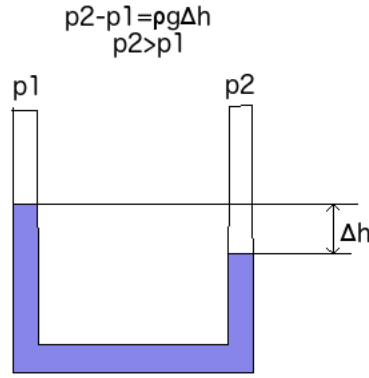


FIGURE 3.11: Fritzing diagram illustrating the electronic set up

Height differences of -10cm,-5cm,0cm,5cm and 10cm were created by raising the level of water in a PVC tube. Both ends were connected to the ports of the sensors to give pressure differentials of $0Pa$, $\pm 490.5Pa$ and $\pm 981Pa$. The pressure sensors output the number of decimal counts. Results were collected for each sensor and are plotted below. These allowed me to convert the sensor output to pressure in pascals. The test results are shown in the figure below.

psd.png

FIGURE 3.12: Counts vs Pressure

3.7 Data filtering

3.8 Plant dynamics

In this section, the dynamics of the plant will be estimated and analysed in MATLAB/Simulink to get initial estimates for the PID gains. There are two blocks in the plant for which we need a transfer function. We need to know how an input signal (PWM) translates to an angle in the servo. Then, we need to understand how the angle changes the C_L and consequently the lift production.

3.8.1 Dynamics of the servo

Rot.jpg

FIGURE 3.13: YUMO E6B2-CWZ3E

To measure the performance of the servo, I used a rotary encoder. This device converts the angular position into a digital signal. The typical quadrature rotary encoders work using two signals: A and B. These are 90 degrees out-of-phase and operate on two logic levels. For clockwise rotation the A and B transition occurs in 4 phases:

TABLE 3.1: Phases of quadrature rotary encoders

	A	B
Phase 1	0	0
Phase 2	0	1
Phase 3	1	1
Phase 4	1	0

Signals produced during anticlockwise rotation occur in exactly the opposite sequence: signal A leads signal B.

Each phase change increments a counter. Each counter increment corresponds to a small angular displacement. Thus, this device will be helpful to understand the dynamics of the servo.

The rotary encoder was coupled to the servo and the step response was measured. The highest point was assumed to be 120 degrees because the manufacturer claims that 120 degrees is the throw of the servo. The following graph shows how the angle changes with time. The initial part of the graph is linear which implies that the servo has clearly the maximum speed it can achieve because the angular velocity is fairly constant. Right before the servo angle converges to the final value, the first order dynamics of the servo are visible.

If we isolate that portion of the graph and analyse how it converges, we can find the time constant assuming that the servo follows first order dynamics (simple lag). The following expression is the transfer function for a first order system.

$$\frac{K}{Ts + 1}$$

When subjected to a step input $1/s$, we get the following transfer function.

$$\theta(s) = \frac{K}{s(Ts + 1)}$$

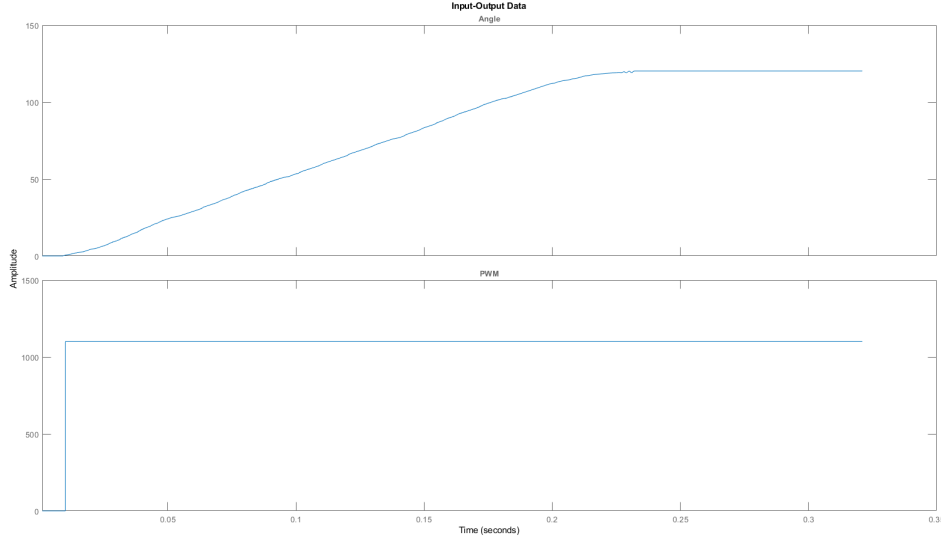


FIGURE 3.14: Input-Output relation

We can decompose this into partial fractions.

$$\theta(s) = \frac{K}{s} - \frac{KT}{Ts + 1}$$

Then we can apply an inverse Laplace transform to find the transfer function in the time domain.

$$\mathcal{L}^{-1} \left(\frac{K}{s(Ts + 1)} \right) = K(1 - e^{-\frac{t}{T}})$$

According to the data collected, the servo reaches to 94.799% within 0.195s. So the time constant can be found.

$$(1 - e^{-0.195/T}) = 0.94799$$

$$T = 0.06596s$$

The maximum speed can be found by taking the gradient at the straight line section of the graph.

$$\dot{\theta}_{MAX} = 0.599^0 ms^{-1}$$

This model of the servo (Figure 3.15) gives a very close approximation of the real servo. Figure 3.4 shows a screenshot of the Simulink model of the servo which incorporates the first order transfer function and a rate limiter set to 599. When a step input of 1100 is applied to the system, the output matches the graph of the measured data as shown in figure 3.3.

3.8.2 Dynamics of the flap

We now know how the angular displacement of the flap changes with time. We now need to know how that change in angle changes the coefficient of lift and consequently the lift.



FIGURE 3.15: Servo approximation

To do this, I used Xfoil to see how the C_L changes with deflection of the flap. It is then easy to calculate the lift:

$$L = \frac{1}{2} \rho V^2 S C_L$$

The graphing software shows that a cubic function is a good fit for the data. However, I chose not to use a cubic function to approximate the flap dynamics because at the maximum/minimum point, it is not clear which direction the flap must deflect to increase/decrease the lift coefficient.

A better approximation is a linear function with saturation on both ends.

$$C_L(\delta) = 0.04257\delta + 0.6665 \quad (3.4)$$

$$-1.35 \leq C_L \leq 2.08 \quad (3.5)$$

3.8.3 MATLAB/Simulink first principles modelling

The last two sections identified the dynamics of the plant. It is now possible to import the models developed into Simulink and use the PID tuner app in MATLAB to tune controller for output disturbance rejection (impulse in output). The image below is a screenshot of the Simulink model. The reference signal is a step signal that has a magnitude of 3.2 from 0s. The error signal is converted to a command signal or a 'control input' which ranges from -1 to 1. -1 corresponds to the flap being all the way up (-60 deg) which requires a PWM input of 0. On the other hand, for deflecting the flap all the way down (60 deg) a PWM value of 1100 is required.

Therefore, to convert the controller output to a PWM output, a gain of 550 is applied and a constant 550 is added.

The PWM signal is consequently fed into the plant.

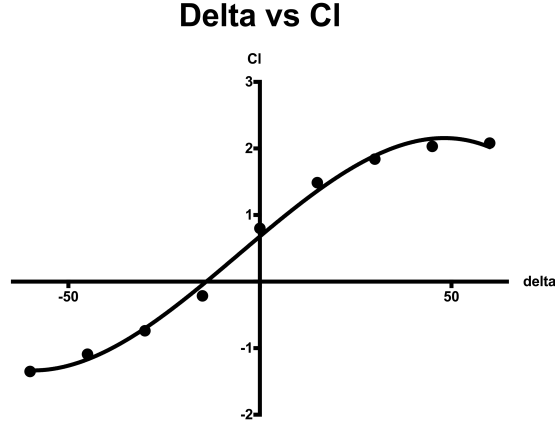


FIGURE 3.16: Flap deflection vs Coefficient of Lift

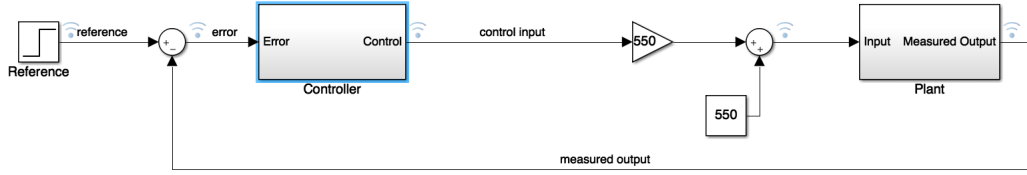


FIGURE 3.17: Simulink model

In figure 3.18, on the far left we have the controller generated PWM signal. This goes through a transfer function and a rate limiter as outlined in section 3.8.1. The output is the flap deflection angle. This signal then goes through a first order polynomial (refer to eq 3.1 and 3.2). The output is the coefficient of lift. This can be converted to lift by applying a gain of 6.615.

$$L = \frac{1}{2} \rho V^2 S C_L = \frac{1}{2} \cdot 1.225 \cdot 12^2 \cdot 0.30 \cdot 0.25 \cdot C_L = 6.615 C_L$$

This completes the model that I used for tuning my controller. To begin testing, I will need to find the optimal PID gains for my controller.

As mentioned earlier, I used the PID tuner app in MATLAB to determine these gains.

I started off by tracking the output disturbance rejection plot and then adjusting the response time until it was under 0.13s. The percentage overshoot was kept under 10%. The optimal gains found are shown in the table:

TABLE 3.2: Matlab predicted PID gains

K_p	0.14324
K_i	2.0572
K_d	-0.00047687

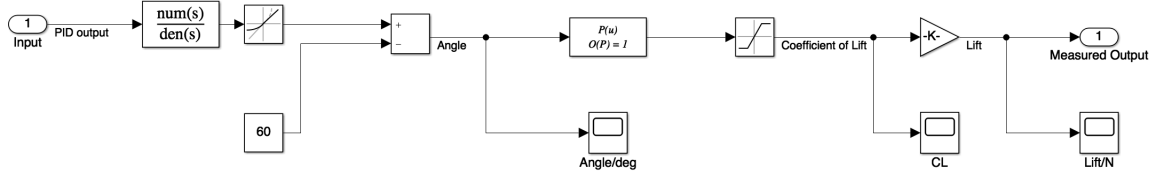


FIGURE 3.18: Simulink model of the plant

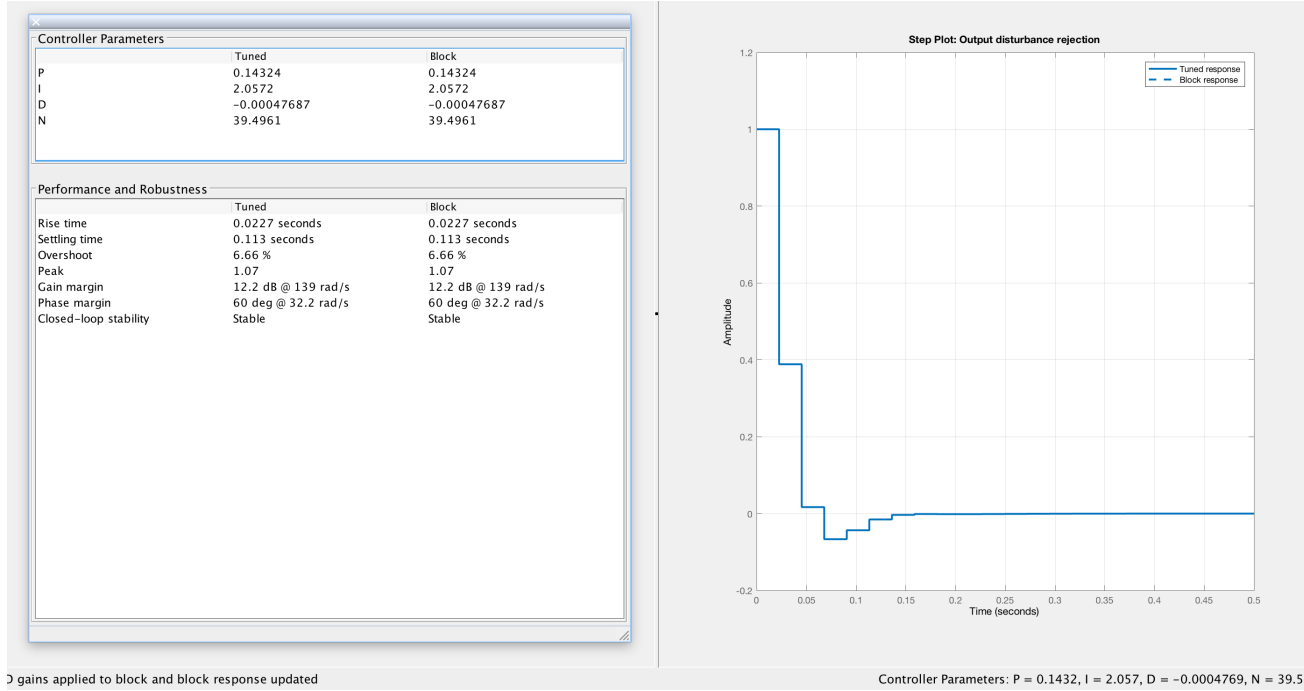


FIGURE 3.19: Tuner app with the performance parameters and the optimised gains in the bottom right

3.9 Lift Estimation

In section 3.6, I was able to accurately obtain pressure in pascals from the pressure sensors. This section will deal with using the pressure data from the sensors to calculate the lift produced by the wing. A good lift estimate is essential for feedback.

Lift and drag are both aerodynamic forces that are a result of an unequal pressure (thus force) distribution along the surface of the wing. If the force due to pressure normal to the direction of the free stream is resolved, the result would be the lift on the wing. Similarly, the component of pressure forces parallel to the free stream is resolved to find the drag.

$$\vec{F}_N = \oint p \hat{n} dS \quad (3.6)$$

$$L = \vec{F}_N \cdot \hat{n} \cos \theta \quad (3.7)$$

$$D = \vec{F}_N \cdot \hat{n} \sin \theta \quad (3.8)$$

These equations cannot be directly applied to measured data because we have pressure readings over finite discrete points along the wing surface. But if the wing has a large

number of pressure tappings, the lift can be estimated by just summing the product of the individual pressure readings and the distance between consecutive tappings.

$$L = \sum_{i=1}^n (p_{li} - p_{ui}) \delta A \quad (3.9)$$

Linear interpolation between consecutive points can further improve the estimation. The size of the wing in this project is unfortunately constrained due to the dimensions of the wind tunnel. Fitting a large number of pressure tappings within the wing was therefore not possible. A large number of pressure tappings also require a large number of pressure sensors which is not economically viable. So the lift estimation is expected to be a very rough approximation but the error should be under 10%. I will use an empirical approach to design the lift estimator.

We can assume that the lift is estimated by the following equation:

$$L = \sum_{i=1}^3 w_i (p_{li} - p_{ui}) \quad (3.10)$$

w_1, w_2 and w_3 are weights given to the differential pressure readings from each location on the wing. These weights will be determined using Xfoil and Excel. I will use Xfoil to sweep through α between -5 and 10 and δ between -60° and 60° . The coefficient of pressure at C_p at the top and bottom of the wing at x/c values of the pressure tappings will be recorded. These can be converted to pressure. The pressure values should roughly be what the pressure sensors will actually measure. The C_L is also recorded for each test. This can be used to find the actual lift on the wing.

The results were all entered into an excel spreadsheet. A column for the estimated lift took the xfoil predicted pressure at each tapping, multiplied it by the corresponding weight and added them. The square of the error was calculated by taking the difference between the actual lift and the estimated lift and squaring the resulting value.

Excel has a solver function which is often used for optimisation. The sum of the squared error values for all tests (across all angles of attacks and flap deflections) were minimised by allowing the solver to adjust the weights.

The following values were determined by excel solver. The average percentage error was about 7.3%.

$$\begin{aligned} w_1 & 0.032876442 \\ w_2 & -0.035263791 \\ w_3 & 0.03927127 \end{aligned}$$

Considering a wide range of flap deflection angles and angle-of-attack when estimating the weightings gave me a lift estimator that works fairly well for all operating conditions the system is likely to see during testing. The full Excel spreadsheet can be found in the appendix.

3.10 The code

Before testing can begin, the code has to be written that will collect the data from the sensors and use it to calculate the lift. It should then control the flap angle in accordance to the controller that I designed in the preceding sections.

It will need to follow a certain sequence. Figure 3.9 shows the sequence of tasks that need to be carried out when the code is compiled and executed.

The 'main' void in my code ran an iterative loop that called different voids, each with a dedicated purpose. The main void was also used to set up the I2C devices and the servo. Both the I2C devices and the servo were implemented using the wiringPi and wiringPiI2C libraries both of which are made publicly available by Gordon under GNU LGPLv3 license.

```

1  softServoSetup
   (0, 1, 2, 3, 4, 5, 6, 7) ;
2  int fd = wiringPiI2CSetup (0x28) ;
   // 0x28 I2C device address
3  int fd2 = wiringPiI2CSetup (0x77) ;
   // 0x77 I2C device address

```

The I2C devices at address 0x28 are the sensors and the device at 0x77 is the multiplexer (refer to section 3.5).

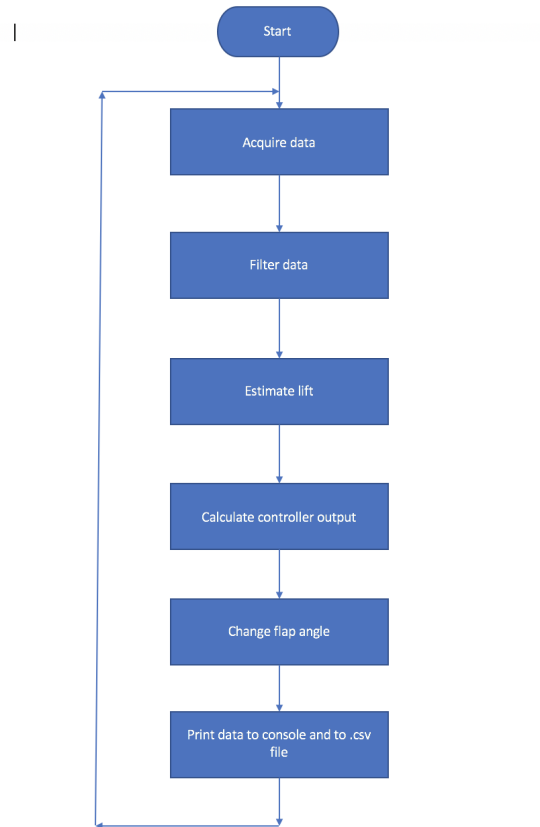


FIGURE 3.20: Flowchart showing the sequence of code blocks

The iterative loop mentioned above was done via an endless while loop as given below. The clock_gettime function was called at the beginning and the end of the loop to calculate the loop/sample time which is useful for the PID calculation and also to understand whether the program was running fast enough to keep up with the oscillation.

```

1  while(1){
2      clock_gettime(CLOCK_REALTIME, &gettime_now);
3      start_time = gettime_now.tv_nsec;
4      counter+=1;
5      daq(fd, fd2);
6      estimateLift();
7      control();
8      printdata();
9      clock_gettime(CLOCK_REALTIME, &gettime_now);
10     if(gettime_now.tv_nsec - start_time > 0){

```

```

11     dt = (gettime_now.tv_nsec - start_time)/1000000000;
12 }
13 }

```

The 'daq' void being called above was used to acquire data from each sensor and to filter the data using an average filter with a window of 20 samples.

```

1  while(i<20){
2      wiringPiI2CWrite (fd2, ch_2) ; // select channel
3      read(fd, &data2, 4); //request 4 byte data from device
4      pres2 = (((int)(data2[0] & 0x3f)) << 8) | data2[1]; // acquiring pressure
5      data
6      p_out2 = 1.098*pres2-8914 ; //conversion and calibration to give pressure in
7      Pa
8      ap1+=p_out2;
9      i+=1;
10 }
11 ap1=ap1/40;
12 i=0;

```

The code above is an example which shows how data was collected and averaged from sensor 2 after converting to pascals. The 'ch2' variable is equal to 0x02. When this value is sent to the multiplexer, the second channel is selected.

Once data from all sensors is acquired and filtered, the lift is estimated in the 'estimateLift' void.

```

1 void estimateLift(){ //to estimate lift using acquired pressure data
2     L=(w1*ap1+w2*ap2+w3*ap3);
3 }

```

This is fairly straightforward. The block of code above simply applies the equation 3.7. The weightings are global variables declared at the start of the file.

Once the lift is known, the error value and the controller output can be calculated.

```

1 void control(){ //run PID iteration and actuate
2     e=L-3.2;
3     P = Kp*e;
4     if(output<-1){
5     }
6     else if(output>1){
7     }
8     else{
9         int_e += e;
10    }
11    I = Ki*int_e*dt;
12    D = Kd*(e-e1)/dt;
13    output = (P+I+D);
14    if(output<-1){
15        output=-1;
16    }
17    else if(output>1){
18        output=1;
19    }
20    e1=e;
21    angle = -550*output+550;
22 }

```

The void above is called after the lift is estimated is estimateLift void. It implements a PID controller with anti-windup protection. This prevents the integral error from accumulating a large value during initialisation or scenarios where the error is temporarily so high

that the output would take a long time to return to nominal values even after the cause of the error is gone.

Lastly, the data needs to be timestamped and printed to the console for live monitoring and to a .csv file to analyse the data after testing. All of this is done in the 'printdata' void. The following code is responsible for displaying and storing the data.

```
1 clock_gettime(CLOCK_REALTIME, &gettime_now);  
2 printf("%f ; %f ; %d\n", gettime_now.tv_sec+((double)gettime_now.tv_nsec)  
   /1000000000 - init_time, L, angle);  
3 clock_gettime(CLOCK_REALTIME, &gettime_now);  
4 fprintf(fp, "%f ; %d ; %f ; %f ; %f ; %f\n", gettime_now.tv_sec+((double)  
   gettime_now.tv_nsec)/1000000000 - init_time, angle, ap1, ap2, ap3, L);
```

The wall time since program start, the lift production and the angle of flap are printed to the console. All of the aforementioned data as well as the filtered pressure values are written to the .csv file.

The code can be found in its entirety in the appendix.

Chapter 4

Wind tunnel testing

All the tools, equipment and resources required for the project have been gathered and assembled as discussed in chapter 3. This chapter will outline how I carried out the testing in chronological order. As with most things, the first iteration of any design is bound to be flawed. But this chapter will show how I was able to improve and optimise the system in each successive iteration.

4.1 Measuring wind tunnel performance

The wind tunnel is operated using a control panel which displayed a frequency. This frequency is proportional to the airspeed in the wind tunnel. However, the exact relation between this frequency and the airspeed was unknown. Thus, the performance of the wind tunnel had to be measured to make sure that I was keeping the test conditions the same throughout testing.

Using a pitot probe and one of the differential pressure sensors, I was able to establish a clear linear relation between the frequency and the airspeed. The following figure shows this relation.

To keep the test conditions constant, I will run all tests at a constant speed of 12ms^{-1} which corresponds to 14Hz. I will also keep the angle of attack at 5° which according to the lift estimator should give me a lift of 4.6 N. I will keep the setpoint at 3.2N to minimise the peak load on the wing.

4.2 Initial test

I began by testing the control system in steady state conditions. The wind tunnel was set to 14 Hz (12ms^{-1}). The controller should be able to hold the lift at roughly 3.2N.

Unfortunately, the controller was unable to stabilise the system and oscillated between the extreme flap angles. This was not entirely unexpected because the controller was tuned very aggressively. There is also a considerable amount of noise in the data to which the controller was hypersensitive. One solution to this issue would be to increase the averaging window size. However, this would mean that the sample rate decreases below the Nyquist frequency. It may also mask the gust so the system may not even detect the gust.

4.3 Cohen and Coon tuning

4.4 Discussion of results

Chapter 5

Summary

Chapter 6

Further recommendations

References

- [1] Catterall, R. (2003). State-Space Modeling of the Rigid-Body Dynamics of a Navion Airplane From Flight Data, Using Frequency-Domain Identification Techniques. [online] Trace.tennessee.edu. Available at: http://trace.tennessee.edu/cgi/viewcontent.cgi?article=3269&context=utk_gradthes [Accessed 2 Feb. 2018].
- [2] B. Etkin and L. Reid, Dynamics of flight. Chichester: Wiley, 1996, pp. 15, 174, 175.
- [3] H. Chun and C. Chang, "Longitudinal stability and dynamic motions of a small passenger WIG craft", Ocean Engineering, vol. 29, no. 10, p. 1161, 2002.
- [4] C. Huang, Q. Shao, P. Jin, Z. Zhu and B. Zhang, "Pitch Attitude Controller Design and Simulation for a Small Unmanned Aerial Vehicle," 2009 International Conference on Intelligent Human-Machine Systems and Cybernetics, Hangzhou, Zhejiang, 2009, URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5336043&isnumber=5335881>
- [5] "14 CFR 23.181 - Dynamic stability.", LII / Legal Information Institute, 2011. [Online]. Available: <https://www.law.cornell.edu/cfr/text/14/23.181>. [Accessed: 03- Feb- 2018].
- [6] B. Aliyu, C. Oshoku, P. Okeke, F. Opara and B. Okere, "Oscillation Analysis for Longitudinal Dynamics of a Fixed-Wing UAV Using PID Control Design", Advances in Research, vol. 5, no. 3, p. 6, 2015.
- [7] N. Li and M. Balas, "Aeroelastic Control of Wind Turbine Blade Using Trailing-Edge Flap", Wind Engineering, vol. 38, no. 5, p. 1, 2014.
- [8] J. Lee, J. Han, H. Shin and H. Bang, "Active load control of wind turbine blade section with trailing edge flap: Wind tunnel testing", Journal of Intelligent Material Systems and Structures, vol. 25, no. 18, pp. 2253-2255, 2014.
- [9] Python 3 vs C gcc (64-bit Ubuntu quad core) | Computer Language Benchmarks Game. [online] Available at: <https://benchmarksgame.alioth.debian.org/u64q/compare.php?lang=python3&lang2=gcc> [Accessed 13 Apr. 2018].

Appendix A

High level

A.1 Low level