REPLICATION OF AN UNDERACTUATED PROPELLER FOR ATTITUDE CONTROL USING STANDARD COMPONENTS

By

JACKSON THOMPSON

SUPERVISOR:

DR DAVID PARLEVLIET

A thesis submitted to Murdoch University School of Engineering and Information Technology to fulfil the requirements for the Bachelor of Engineering Honors Degree in the disciplines of Industrial Computer Systems Engineering and Instrumentation and Control Engineering
Authors Declaration

I declare that this thesis is my own account of my research and contains as its main content work which has not previously been submitted for a degree at any tertiary education institution.

X

Jackson Thompson
Abstract

Micro Air Vehicles (MAV) are unmanned air vehicles based on full sized aircraft designs and used for various purposes in many industries. 3D printing and low-cost hobby MAV components have expedited design innovations but rotary wing MAV research has continued to focus on quad rotor designs. Alternative MAV designs seek to control key flight components including: forces, moments, lift and the centre of pressure.

This thesis expands on the work of Paulos and Yim’s 2013 paper on the use of an underactuated propeller for attitude control which reduces cost compared to the original design. The concept for this thesis was to use the modulation of torque applied to a passively hinged, underactuated rotor assembly to create controllable lateral forces. A replicated rotor hub assembly was used and duplicated using 3D design software, a 3D printer and hobby drone rotor blades.

The controllable torques are generated using a direct current brushless motor, an electronic speed controller and a Raspberry Pi single board computer. This control required the replication of the DShot protocol through a Serial Peripheral Interface. Code to superimpose the sine wave to the desired motor speed was written on the Raspberry Pi.

A testing platform, consisting of a 3D printed assembly, load cells for measuring force and a laser beam for measuring the rotor speed was designed and built. This system used LabVIEW and a National Instruments Digital Acquisition Card for the measurement and logging of the electrical signals produced by the platform. Conversions were calculated for these signals to produce force measurements and the system was calibrated. Testing was performed on the constructed rotor assembly to analyse the generation and control of lateral forces. Control was established for the magnitude of the lateral forces produced, but not for their direction.

This work generated options for future development, including improvements direction control, the testing platform and areas for further testing which will all assist with the ultimate goal of independent flight using the underactuated model.
[This page was intentionally left blank]
Acknowledgements

This thesis is dedicated to the memory of Mr Iafeta “Jeff” Laava who sadly passed away at the end of 2018. Jeff was always generous with his time, friendship and biscuits and will be greatly missed.

I would like to thank my supervisor and lecturer, Dr David Parlevliet for his support and guidance throughout the undertaking of this thesis and my past studies. My thanks as well to the rest of the Murdoch University staff who taught me over the years. Specific mention to Professor Parisa Arabzadeh Bahri, Associate Professor Graeme Cole and Dr Linh Vu for all their time and patience with me.

To the technical staff, Mr Graham Malzer, Mr Mark Burt, Mr Will Stirling and Mr David Morrison, thank you for your friendship and letting me use your equipment as if it was my own.

To my family and friends, thank you for all your support as I traded my accounting career for the study books again. Special thanks to Joseph Allsopp for helping me realise what I could achieve and always being there to have a laugh.

To my partner, Laura Tarrier, thank you for your constant support and love. I couldn’t have done this without you and I hope I can support you with all your studies as well as you have supported me.
Contents

AUTHORS DECLARATION ........................................................................................................ II

ABSTRACT ................................................................................................................................. IV

ACKNOWLEDGEMENTS ........................................................................................................ VI

CONTENTS .............................................................................................................................. VII

TABLE OF FIGURES ................................................................................................................ X

TABLE OF EQUATIONS .......................................................................................................... XII

TABLE OF TABLES ................................................................................................................ XII

LIST OF ABBREVIATIONS .................................................................................................... XIII

1 INTRODUCTION .................................................................................................................... 1

2 BACKGROUND TO FLIGHT ............................................................................................... 3

2.1 FORCES ............................................................................................................................. 3

2.2 MOMENTS ......................................................................................................................... 4

2.3 LIFT .................................................................................................................................. 5

2.4 THE CENTRE OF PRESSURE ......................................................................................... 6

2.5 CYCLIC CONTROL .......................................................................................................... 6

3 EXAMINED MAV DESIGNS .............................................................................................. 8

3.1 THE MUFLY DESIGN ...................................................................................................... 8

3.1.1 CHANGING THE AIR FLOW ORIENTATION ............................................................. 9

3.1.2 CHANGING THE CENTRE OF GRAVITY ................................................................. 10

3.1.3 CHANGING ROTOR ORIENTATION ........................................................................... 11

3.2 THE CO-AXIAL Y6-ROTOR MAV DESIGN ..................................................................... 13

3.3 UNDERACTUATED PROPELLER DESIGN .................................................................... 14

3.3.1 DEFINITION .................................................................................................................. 14

3.3.2 CONCEPT ...................................................................................................................... 14

3.3.3 TESTING ....................................................................................................................... 15

3.3.4 CONTROL ...................................................................................................................... 16

3.3.5 RESULTS COMPARISON ......................................................................................... 16
# Contents

4 EVALUATION AND CREATION OF UNDERACTUATED PROPELLER PROTOTYPE .......................... 19

4.1 DESIGN JUSTIFICATION ........................................................................................................ 19
4.2 ROTOR HUB DESIGN ......................................................................................................... 20
4.3 CONCEPT EXPLANATION .................................................................................................. 24
   4.3.1 THRUST CREATION ..................................................................................................... 25

5 DESIGN AND DEVELOPMENT ............................................................................................ 29

5.1 DIRECT CURRENT BRUSHLESS MOTORS ....................................................................... 29
5.2 ELECTRONIC SPEED CONTROLLERS ............................................................................... 30
   5.2.1 DSHOT PROTOCOL .................................................................................................. 31
   5.2.2 PACKET STRUCTURE ............................................................................................... 33
5.3 COMMUNICATION DEVICES .............................................................................................. 34
   5.3.1 ARDUINO AMEBA RTL8195 .................................................................................. 34
   5.3.2 RASPBERRY PI MODEL B+ ..................................................................................... 35
   5.3.3 SERIAL PERIPHERAL INTERFACE .......................................................................... 36

6 METHOD AND IMPLEMENTATION ...................................................................................... 38

6.1 PROTOCOL CALCULATION AND VERIFICATION USING MICROSOFT EXCEL ................. 38
   6.1.1 BIT REPLICATION ..................................................................................................... 38
6.2 ARDUINO IMPLEMENTATION .......................................................................................... 41
   6.2.1 AMEBA ISSUES ....................................................................................................... 42
6.3 RASPBERRY PI IMPLEMENTATION .................................................................................. 44
   6.3.1 SPI SETUP ................................................................................................................ 45
   6.3.2 PROTOCOL CALCULATION .................................................................................... 46
   6.3.3 SPEED TESTING ...................................................................................................... 46
   6.3.4 SINE WAVE CREATION ............................................................................................ 47
   6.3.5 WAVEFORM IMPLEMENTATION ............................................................................. 48

7 TESTING & MEASUREMENT ............................................................................................... 50

7.1 TESTING PLATFORM .......................................................................................................... 50
   7.1.1 LOAD CELLS ............................................................................................................. 51
   7.1.2 MOUNTING .............................................................................................................. 52
   7.1.3 RPM MEASUREMENT .............................................................................................. 53
   7.1.4 LABVIEW DIGITAL ACQUISITION ......................................................................... 55
   7.1.5 CALIBRATION ......................................................................................................... 59

8 RESULTS ............................................................................................................................... 62

8.1 LATERAL FORCE MAGNITUDE CONTROL ...................................................................... 63
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.2</td>
<td>LATERAL FORCE DIRECTION CONTROL</td>
<td>64</td>
</tr>
<tr>
<td>8.3</td>
<td>THESIS COSTINGS</td>
<td>64</td>
</tr>
<tr>
<td>9</td>
<td>FUTURE WORK</td>
<td>66</td>
</tr>
<tr>
<td>9.1</td>
<td>MEASUREMENT</td>
<td>66</td>
</tr>
<tr>
<td>9.2</td>
<td>DIRECTION CONTROL</td>
<td>67</td>
</tr>
<tr>
<td>9.3</td>
<td>PHOTOGRAPHIC OBSERVATION</td>
<td>67</td>
</tr>
<tr>
<td>9.4</td>
<td>THRUST COMPARISON</td>
<td>68</td>
</tr>
<tr>
<td>9.5</td>
<td>ROTOR ASSEMBLY MODIFICATIONS</td>
<td>68</td>
</tr>
<tr>
<td>9.6</td>
<td>CONTROLLED FLIGHT</td>
<td>69</td>
</tr>
<tr>
<td>10</td>
<td>CONCLUSION</td>
<td>70</td>
</tr>
<tr>
<td>11</td>
<td>REFERENCES</td>
<td>72</td>
</tr>
</tbody>
</table>

APPENDIX A.IEEE COPYRIGHT NOTICE | 78   |
APPENDIX B.AMPLITUDE TESTING CODE – C | 79   |
APPENDIX C.PHASE TESTING CODE – C | 84   |
APPENDIX D.LABVIEW CODE FOR TESTING PLATFORM | 89   |
Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Reaction to thrust of rotor on a helicopter. Adapted from [8]</td>
</tr>
<tr>
<td>2</td>
<td>Angled moment calculation. Adapted from [9]</td>
</tr>
<tr>
<td>3</td>
<td>Use of a tail rotor to counteract rotor torque [12]</td>
</tr>
<tr>
<td>4</td>
<td>Lift is generated from the reaction to the acceleration of air. Adapted from [8]</td>
</tr>
<tr>
<td>5</td>
<td>Feathering mechanism [8]</td>
</tr>
<tr>
<td>6</td>
<td>Placement effects of the centre of pressure [8]</td>
</tr>
<tr>
<td>7</td>
<td>Cyclic control of blade pitch and tilting effect [8]</td>
</tr>
<tr>
<td>8</td>
<td>Google scholar publications referring to UAVs [14]</td>
</tr>
<tr>
<td>9</td>
<td>Cut through of an adaptive ducted fan design [17] © 2013 IEEE</td>
</tr>
<tr>
<td>10</td>
<td>BHNC-1 'Ducted Fan' MAV [22]</td>
</tr>
<tr>
<td>11</td>
<td>Centre of Gravity steering concept and CoaX 2 MAV Design [17] [23] © 2013 IEEE</td>
</tr>
<tr>
<td>12</td>
<td>SMART Active Rotor and trailing edge flap design [27] [28]</td>
</tr>
<tr>
<td>13</td>
<td>Side view of pulse torque rotor design [29]</td>
</tr>
<tr>
<td>17</td>
<td>Coaxial MAV controlled by an underactuated propeller [5] and most recent model [31] © 2013 IEEE</td>
</tr>
<tr>
<td>18</td>
<td>Comparison of 227g swashplateless design against CX2 coaxial helicopter and standard quadcopter [32] © 2013 IEEE</td>
</tr>
<tr>
<td>19</td>
<td>Quad rotor to Single Rotor Lifting Area Comparison. Adapted from: [35]</td>
</tr>
<tr>
<td>20</td>
<td>Computer Aided Designs produced in Fusion360</td>
</tr>
<tr>
<td>21</td>
<td>Constituent Bodies to Complete Component</td>
</tr>
<tr>
<td>22</td>
<td>Murdoch University MakerBot Replicator 2</td>
</tr>
<tr>
<td>23</td>
<td>Failed 3D Prints</td>
</tr>
<tr>
<td>24</td>
<td>DJI Mavic 2 Propeller (Complete) and Disassembled</td>
</tr>
<tr>
<td>25</td>
<td>Rotor blade mounting to Hub assembly using steel wire</td>
</tr>
<tr>
<td>26</td>
<td>Blade Response to changes in Torque</td>
</tr>
<tr>
<td>27</td>
<td>Impulsive and Retrograde Torque Generation using Sin Wave Superimposition</td>
</tr>
<tr>
<td>28</td>
<td>Side View of Unbalanced Thrust creation through Torque Application</td>
</tr>
<tr>
<td>29</td>
<td>Overhead View of Unbalanced Thrust creation through Torque Application</td>
</tr>
<tr>
<td>30</td>
<td>Motor Control Components. Adapted from: [41]</td>
</tr>
<tr>
<td>31</td>
<td>Brushed and Brushless Motor Comparison. Adapted from: [41]</td>
</tr>
<tr>
<td>32</td>
<td>Turnigy D2836/8 1100KV Brushless Motor</td>
</tr>
<tr>
<td>33</td>
<td>Wraith32 32bit 35A ESC V2</td>
</tr>
<tr>
<td>34</td>
<td>ESC Protocol Speed Comparison. Adapted from [43]</td>
</tr>
<tr>
<td>35</td>
<td>DShot1200 Packet Structure</td>
</tr>
<tr>
<td>36</td>
<td>Standard vs DShot Bit Structures</td>
</tr>
</tbody>
</table>
Table of Equations

Equation 1: Momentum Equation .................................................................19
Equation 2: Kinetic Energy Equation .............................................................19
Equation 3: Momentum Theory, Power for Thrust Equation [38] ....................20
Equation 4: Sine Wave equation amended for Code ......................................47
Equation 5: Rotor Period relationship to Code Variables ................................49

Table of Tables

Table 1: Mass and Velocity effect on Momentum and Energy ..........................19
Table 2: Worked Example to compare Single Rotor to Quad Rotor spacing inefficiencies ....20
Table 3: CRC Bit Calculation ........................................................................39
Table 4: Raspberry Pi - Standard SPI Speeds [51].........................................45
### List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D</td>
<td>Three Dimensional</td>
</tr>
<tr>
<td>Ameba</td>
<td>Arduino Ameba RTL8195</td>
</tr>
<tr>
<td>BLDC</td>
<td>Direct Current Brushless Motor</td>
</tr>
<tr>
<td>CoG</td>
<td>Centre of Gravity</td>
</tr>
<tr>
<td>CRC</td>
<td>Cyclic Redundancy Check</td>
</tr>
<tr>
<td>DAQ</td>
<td>Digital Acquisition</td>
</tr>
<tr>
<td>DShot</td>
<td>DShot Protocol</td>
</tr>
<tr>
<td>ESC</td>
<td>Electronic Speed Controller</td>
</tr>
<tr>
<td>FET</td>
<td>Field Effect Transistor</td>
</tr>
<tr>
<td>GPIO</td>
<td>General Purpose Input/Output</td>
</tr>
<tr>
<td>Hz</td>
<td>Hertz</td>
</tr>
<tr>
<td>IARC</td>
<td>International Aerial Robotics Competition</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>J</td>
<td>Joules</td>
</tr>
<tr>
<td>kg</td>
<td>Kilogram</td>
</tr>
<tr>
<td>kHz</td>
<td>Kilohertz</td>
</tr>
<tr>
<td>KV</td>
<td>Revolutions Per Volt</td>
</tr>
<tr>
<td>LabVIEW</td>
<td>Laboratory Virtual Instrument Engineering Workbench</td>
</tr>
<tr>
<td>LCD</td>
<td>Liquid Crystal Display</td>
</tr>
<tr>
<td>LDR</td>
<td>Light Dependent Resistor</td>
</tr>
<tr>
<td>m/s</td>
<td>Metres Per Second</td>
</tr>
<tr>
<td>m²</td>
<td>Metres Squared</td>
</tr>
<tr>
<td>m³</td>
<td>Metres Cubed</td>
</tr>
<tr>
<td>MAV</td>
<td>Micro Air Vehicle</td>
</tr>
<tr>
<td>Mhz</td>
<td>Megahertz</td>
</tr>
<tr>
<td>ms⁻²</td>
<td>Metres Per Second Per Second</td>
</tr>
<tr>
<td>N</td>
<td>Newton</td>
</tr>
<tr>
<td>NI</td>
<td>National Instruments</td>
</tr>
<tr>
<td>Nmm</td>
<td>Newton Millimetre</td>
</tr>
<tr>
<td>PWM</td>
<td>Pulse Width Modulation</td>
</tr>
<tr>
<td>RAF</td>
<td>Relative Air Flow</td>
</tr>
<tr>
<td>RPi</td>
<td>Raspberry Pi</td>
</tr>
<tr>
<td>RPM</td>
<td>Revolutions Per Minute</td>
</tr>
<tr>
<td>SPI</td>
<td>Serial Peripheral Interface</td>
</tr>
<tr>
<td>UAV</td>
<td>Unmanned Air Vehicle</td>
</tr>
<tr>
<td>V</td>
<td>Volts</td>
</tr>
<tr>
<td>W</td>
<td>Watts</td>
</tr>
</tbody>
</table>
1 Introduction

Large Unmanned Air Vehicles (UAVs) can trace their origins to 15 September 1924, when a Curtiss F-5L aircraft was flown remotely through all phases of flight [1]. This event occurred less than 20 years after the Wright brothers had made the first human-controlled, sustained flight of a powered, heavier-than-air aircraft [2]. Larger UAVs have since been used in both civilian and military operations.

Recent technological advancements, including; lightweight and energy dense batteries, low cost stabilisation systems, powerful microcontrollers and Field Effect Transistors (FET) [3] have led to the development of miniature UAVs and Micro Air Vehicles (MAV). Common design variants of rotary-wing models include quadrotor, coaxial helicopter and ornithopter [4]. These MAV variants closely follow the design principles of their larger UAV or aircraft counterparts.

Most of the work on multirotor UAVs has focused on quadrotor design. James Paulos [5] explored the potential of an underactuated rotor for attitude control as a simpler design that had fewer parts, was less complex and offered a cheaper, more reliable alternative. The goal of this project was to expand on Paulos’ work and determine whether his ideas could be replicated with low cost components. To achieve this a number of steps were required. These included:

1. An analysis of the key concepts to flight.
2. A review into what other designs where being researched.
3. A review into the research around the underactuated propeller design
4. An examination of the design and construction of the prototype.
5. An analysis of the components required to test the prototype.

To be successful the prototype had to achieve the forces required to generate flight using the underactuated propeller design. To meet this objective the following goals had to be accomplished:

1. Achieve control over the magnitude of any lateral forces generated.
2. Achieve control over the direction of any lateral forces generated.
3. Prove that the design can be cheaply replicated and tested.

These outlined steps and goals will be covered sequentially in this thesis. The conclusion will also consider the implications for future research.
2 Background to Flight

2.1 Forces

Vector quantities have a magnitude and a direction that identify them [6]. Scalar quantities have no direction and only consist of a magnitude. Velocity is therefore a vector and speed is a scalar. Acceleration is the rate of change of velocity, so inherits the vector quality of velocity [6].

Acceleration (and hence velocity) is linked to force by Newton’s Second Law of Motion, \( F = m \times a \) [7] which shows that the force required for a change in velocity is equal to the mass of the object multiplied by its acceleration. This defines the SI unit for force, the Newton (N) as the force that results in a one kilogram mass accelerating at one metre per second per second (ms\(^{-2}\)). [6] [8]

Helicopters (and MAVs) create force by accelerating air downwards to generate the opposite reaction upwards (as described by Newton’s Third Law of Motion). If this upwards force is equal to that of the downwards force of gravity, at roughly 9.81 ms\(^{-2}\) multiplied by the mass of the object, then it would be in equilibrium and would neither rise nor fall. Figure 1 demonstrates this concept. The helicopter will rise as the force from accelerating the air below the rotor exceeds that applied by gravity. [8]

![Figure 1: Reaction to thrust of rotor on a helicopter. Adapted from [8]](image-url)
2.2 Moments

Forces can also be considered when they do not act through a particular point, known as a moment. Moments are defined with respect to the point. A moment, with relation to a point is equal to the force multiplied by the distance perpendicular to the force, as shown at Figure 2. [6] [8]

![Moment Calculation Diagram]

\[
\text{Moment} = F \times L \cos a
\]

Figure 2: Angled moment calculation. Adapted from [9]

Couples (or torque) exist when two equal and opposite forces are applied at the same distance from a centre point. They produce angular acceleration at right angles to the plane of the couple [10]. Using the example of a helicopter, the engine causes the blades to spin using a couple produced at the rotor shaft. The reaction to this couple causes the helicopter to rotate in the opposite direction to rotor spin and must be countered if yaw control is desired. Traditional helicopter design uses a tail rotor, coaxial designs use two oppositely spinning rotors to counter-act each other. Figure 3 demonstrates how a tail rotor produces a moment that counters the couple produced by the main rotor, stopping undesirable rotation, or yaw. Consequently the helicopter drifts in the direction of the tail rotor thrust as the main rotor does not produce a lateral force in its couple which leaves the tail rotor thrust unbalanced. [11] [8]
2.3 Lift

Lift is the result of acceleration of air over a wing or rotor. For the purposes of this review and the literature discussed, the terms aerofoil and airfoil may be used interchangeably to refer to the physical components over which air passes to generate lift. [8]

Continuing with the helicopter example, the rotor blade’s airspeed is controlled primarily by their rotation. This is considered in terms of the vector, velocity, so is the net value of the all sources of airflow, including the wind velocity (with a direction). This net airflow is known as Relative Airflow (RAF), and is demonstrated in Figure 4. [8]
The importance of the angle of attack to the RAF is also demonstrated in Figure 4. The angle of attack is the angle relative to the RAF and differs from the term pitch, which is relative to the horizon. The greater the angle of attack the greater the resultant lift that is generated, noting the angle change of that lift as well. Traditional helicopters feather their blade pitch to control lift and can also change the RAF with the engine speed. Feathering is accomplished using bearings that connect the rotor to the shaft as shown at Figure 5. [8]

![Figure 5: Feathering mechanism](Figure: Feathering mechanism [8])

2.4 The Centre of Pressure

Lift is distributed over an aerofoil in a non-uniform manner. The centre of pressure is the point where the same lift effect would be generated if all the distributed force was condensed to that one point. In fixed wing aircraft this point is ahead of the centre of mass, providing them with a fixed angle of attack. This lift force generates a couple that tries to twist the blade upwards, increasing the angle of attack against the RAF which is then compounded as more lift is generated. Fixed wing aircraft overcome this torque in their wings using high strength materials that can withstand these stresses. [8] Similar materials cannot be used in helicopters as they would make the rotors too heavy so to overcome this problem the centre of pressure is moved slightly behind the centre of mass. This results in a couple that occurs in the opposite direction and reduces the angle of attack whenever it is increased by feathering. [8]
2.5 Cyclic Control

Lateral movement in a traditional helicopter is provided by the cyclic, a mechanism that imposes a sinusoidal component to the average blade pitch desired by the operator. The resultant rotor oscillation will be at the same frequency as the rotor, causing an increase on one side and a decrease on the other. This tilts the thrust vector producing a resultant sideways and slightly down force, as seen in Figure 7. The operator has to compensate for the loss in height but will achieve lateral movement. Full-size helicopters use a swashplate to achieve the direct cyclic control of the rotor blades and can also control the pitch of all the blades at the same time (collective pitch). [8]
3 Examined MAV Designs

The growth of small scale MAVs for hobby use has encouraged research into MAVs and removed many production challenges. This has inspired further MAV research into areas such as aerodynamics, flight control, electronics and materials design. It has also driven the formation of competitions and contests. The International Aerial Robotics Competition (IARC) [13] has been running for 27 years and continues to grow. Figure 8 demonstrates the increase in research publications regarding UAVs.

![Google Scholar Results for 'uav']

**Figure 8: Google scholar publications referring to UAVs [14]**

A 2014 study [15] found that 70% of research into MAVs used either a single rotor, multirotor, duct fan or coaxial rotor as their test platform over a fixed wing design, with the majority using multirotors. This review will particularly focus on research into newer and more original designs for MAV flight rather than those currently in common use.

3.1 The muFly Design

The muFly is a fully autonomous coaxial micro helicopter developed in 2005. Research into this design was instigated by the European Framework when it was identified as a specific targeted research project [16]. Much of the literature in this field was commissioned by similar projects, demonstrating the wider governmental and commercial interest in the subject. The muFly study had mass and size constraints set “comparable to a small bird” (target mass 50g, maximal dimension 10cm). Target capabilities included autonomous take off/hover/landing,
obstacle avoidance and mapping (all indoor) [17]. The research group chose to use a coaxial rotor configuration over the more common quadrotor or axial type.

The coaxial choice allowed for control of the yaw angle through the differential speed variation of each rotor. Altitude control is provided by the simultaneous rotor speed variation. To provide attitude control several methods were considered e.g., change the orientation of the down wash to create a moment on the fuselage, change the position of the helicopters Centre of Gravity (CoG), or change the orientation of one (or both) of the rotors, thereby changing the resultant thrust vector. The muFly study is unique in that it considers several novel designs to achieve attitude control and these are discussed below.

3.1.1 Changing the Air Flow Orientation

The first method, changing the down wash orientation, uses deflecting flaps controlled by servo motors to control steering moments. An example of this is shown at Figure 9. Commercial ducted fan designs include Cypher [18], iSTAR9 [19] and HoverEye [20] [21]. This methodology offers higher efficiency compared to normal rotor designs due to minimised tip losses (losses induced around the tip of rotor blades) and additional lift gained by the ducting of the setup. As noted by other designers [21], the ducted fan increases the flight safety by enclosing the rotor inside the duct. This reduces the chances of rotor failure due to hitting an object and reduces the danger of the rotors harming an individual or surface.

Figure 9: Cut through of an adaptive ducted fan design [17] © 2013 IEEE
Design disadvantages include component production and the relatively large size and mass required to achieve aerodynamic efficiency. It also restricts payload mounting. The ducted design is located at the lower part of the MAV, where payloads are normally attached. Options exist to mount payloads higher on the MAV (raising the CoG) or attaching additional structure to mount the payload, thereby increasing the total mass of the design.

Another design explored by Fu et al [22] is the use of a four-wing sitter vehicle. The BHNC, shown in Figure 10, offers controlled flight using a single rotor. A large range of complexities were found when modelling the aerodynamics of the rotor slipstream over the control vanes and their actuators. This was considered a weakness in the design by Fu et al as the complexity is noted as an impediment to the designing of the flight control system.

![Figure 10: BHNC-1 'Ducted Fan' MAV [22]](image)

### 3.1.2 Changing the Centre of Gravity

The second option explored by the muFly study was to change the MAV’s CoG [17], Figure 11. Moments are generated when the CoG is not located on the rotors’ axis of rotation [17]. This principle has been used to steer MAV designs and was successfully used for the CoaX 2 MAV [23]. The limitation to this methodology is the requirement for large changes in the CoG to generate the necessary moments that allow for steering. The response moment to moving this mass around is inherently slower than other designs which limits the control available.
3.1.3 Changing Rotor Orientation

The last option explored by the muFly team was to change the orientation of the rotor, which is traditionally achieved using a swash plate mechanism [8]. The literature identifies two key rotor blade properties that can be used to change the lift on a rotor blade; the angle or the blade profile geometry. To change the pitch angle V. Rostyne’s patent [24] has been considered. This patent involves the miniaturisation of the standard swashplate/cyclic design [8], replacing the normal armature with a geared system that adjusts the pitch of the blades.

To change the geometry of the blade profile two options were considered i.e. change the camber of the blade through the deformation of a piezoelectric rotor blade, or use active flaps on the rotor blades. Since publication, both options have been further pursued, although not for MAVs. Roh, Kim & Lee [25] have developed various models to explore the effectiveness of a rotor blade actuated by shape memory alloys to enhance aerodynamic performance. These models have shown greater aerodynamic performance than existing plain flap or slotted designs. Implementation with a MAV has yet to be completed with no recent literature demonstrating functional usage.

The use of active flaps in rotor blades is currently under development for full size helicopters in a joint research project between the US Military and the University of Maryland. F Straub et al have recently published wind tunnel testing data for their SMART Active Rotor [26], shown at Figure 12. The results from their studies have confirmed a material reduction of noise and vibration in the tested rotors. Although at full-size helicopter testing only, this confirmed it is possible to use active flaps to achieve attitude control. However, it is likely that
the miniaturisation of the actuators in the rotor blades will be a problem when scaling to MAV sizing.

One additional steering principle, the pulse torque rotor, was developed by Fujihira, Sasaki and Ando [29]. The design as shown in Figure 13 allows the blades to feather (14), this is controlled by the drag on the rod at (7). The drag on the rod is controlled by motor speed variations, allowing changes in the rotational speed to control the blade pitch. The muFly authors justification for not testing the design is the low control authority provided. However, this pulse torque concept has been further developed by Paulos and Yim [5] with much success and is discussed later in this review.
3.2 The Co-Axial Y6-Rotor MAV Design

The quadrotor design has been more popular than coaxial helicopter variants. The simplicity of a quadrotor’s multiple motors and fixed pitch propeller architecture is cheaper to produce than the advanced concepts considered in the design discussion for the muFly. Multirotor design has not stagnated. Czyba et al [30] have presented a coaxial tri-rotor design, (Figure 14), that has three equally spaced arms in a Y shape ending with coaxial rotor units. This design differs from others in that the controlling forces are on the vertices of the equilateral triangle produced, rather than on the vertices of a square as for normal quadcopters. The design eliminates the yaw effect of the rotors through the pair of rotors as coaxial.

Advantages to the design were an increase in thrust without increasing the size of the frame and the coaxial designs ability to counteract the couples generated by any one rotor. However, the increase in rotors did necessitate larger batteries for comparable flight time which counteracts the smaller frame size weight advantage.

As with most multirotor designs, pitch motion is provided by differing the thrust between front and rear motors. Without the ability to change blade pitch the frame of the craft follows this tilting motion, which other works [17] [5] have tried to reduce.

![Figure 14: Co-Axial Y6-Rotor MAV [30]](image-url)
3.3 Underactuated Propeller Design

3.3.1 Definition

James Paulos and Mark Yim have thoroughly researched the actuated rotor design. This design has been improved and documented in research over a period of five years. Due to the uniqueness of this design and the large body of work surrounding it, a comprehensive appraisal will be conducted. The first publication by Paulos and Yim details their use of an underactuated propeller for control of a MAV [5]. This concept references the work of Fujihira et al [29], discussed earlier, in achieving thrust and moment response from an indirect drive linkage to the rotor blades. Their use of the underactuated term is somewhat misleading. As [31] covers, almost all MAVs are underactuated by definition and very few designs use six or more actuators to control the six degrees of freedom of position and orientation. The research at [5] can still be considered underactuated however as it only uses two actuators compared to the four or more found on most quadcopter designs.

3.3.2 Concept

The work differs from traditional attitude control with the developed method of using the passive dynamic response of the rotor blades. This method uses “the mean applied torque to set the rotational speed and thrust and an applied oscillatory torque to induce the desired cyclic oscillation in blade pitch [5]”.

This physical componentry is limited to a unique hub and propeller design, as shown at Figure 15. When impulsive torque is applied to the centre hub the positive blade flexes backwards, increasing pitch. When retrograde torque is applied the positive blade flexes forwards, decreasing pitch. Due to the direction that the negative blade’s hinge lies, the opposite response is generated for that blade. The research found that a cyclic oscillation was induced in the blades by superimposing a sinusoidal torque at the rotor frequency on top of the steady torque desired. This cyclical oscillation was phase locked with the rotor position. In depth kinematic and aerodynamic modelling of the forces generated has been covered in [32] but are beyond the scope of this review.
The propeller hub was built using a Three Dimensional (3D) printer with steel wire hinges, shown in Figure 16. The rotor blades used were commercial blades sourced from E-flite, a producer of hobby MAV components.

3.3.3 Testing

Testing was conducted using an ATI Nano 17 [33] six-axis force and torque sensor and photographed using a strobe-based photography system. This involved long exposures of the system in a dark room and a xenon strobe flashed at the desired times. The force testing discovered that there were linear regions of torque amplitude that generated moments. There were also regions below a cut-in amplitude where no moment was generated, and one where the relationship plateaued and increasing the amplitude had no greater effect on the moment. The authors hypothesised that the cut-in threshold was due to the static friction in the hinge, however thought that this could be eliminated with further research [32]. The testing provided a maximum control moment of 30 Nmm (38.6cm rotor) which was
compared to cyclic toy helicopters with 45 Nmm (33cm rotor) and greater than coaxial versions with 1.7 Nmm (19cm rotor).

Scalability testing has been recently conducted in [34] with rotor sizes of ten centimetres and one metre. Testing was conducted at similar thrust ratings, which saw the larger rotor spin more slowly. At lower speeds the rotor was still controllable, however, the total bandwidth for control did drop due to the slower rotational speed.

3.3.4 Control

Using the original testing data in [5] the authors were able to implement a cascade control loop for attitude control. The inner loop was responsible for motor control and the outer loop was used for attitude control. The underactuated rotor was paired with a second fixed pitch rotor orientated below the first rotor to provide the yaw control common to coaxial systems. Overall operator control is provided by augmenting the pitch and roll moments provided by the control loop. The work in [31] continues this control concept, with the application of common quadcopter controllers used after inputs were mapped correctly. This was achieved in [31] by decoupling the attitude controller from either the propeller speed or motor drive update rate.

![Figure 17: Coaxial MAV controlled by an underactuated propeller [5] and most recent model [31] © 2013 IEEE](image)

3.3.5 Results Comparison

In [31] the authors completed load testing to compare the effectiveness of their design with others. Whilst the design did suffer from hover inefficiencies as first noted in [5], they were far lower than the increased power consumption that the mass of a traditional servo and linkage setup would incur. Asymmetric loading was also tested in [31]. This testing found the
need for constant attitude adjustments meant that the pulsing control was in constant use. This was verified by the increased power consumption of the design in this load case when compared to a traditional quadcopter.

With the refinement of design in [31] comparison against similar operability MAVs could be conducted. Figure 6 demonstrates how the removal of the swashplate components allows for a larger mass of batteries to be included, potentially increasing flight time depending on usage.

![Subsystem Mass Fraction](image)

*Figure 18: Comparison of 227g swashplateless design against CX2 coaxial helicopter and standard quadcopter [32] © 2013 IEEE*

In conducting these comparisons [31] also notes the reduction in noise levels from the new design when compared to a quadrotor. The median sound level in hover was 62 dBA for the pulsing coaxial helicopter and 70 dBA for a similar sized quadrotor. This is due to the slower operational speed of the underactuated single large rotor design, as opposed to the faster speeds required by the four small rotors of a quadcopter.

### 3.4 Future Research and Development

Recent MAV literature demonstrates that there are significant opportunities to propose and test new methods of controlling the fundamental principles of flight. The research conducted for this thesis highlighted the fact that much of the literature available relies on traditional quadcopter design, with little documented consideration of alternatives. This may be due to
the complexities of design work, and the temptation to adjust existing solutions to fix new problems. The research reviewed in the past chapter focuses on a smaller range of design alternatives that were evaluated with sufficient detail. These design alternatives are a dynamic area and a relevant case study to each has been included.

The work of Paulos and Yim is an excellent example of how large advances can be made into the field of MAV flight when novel ideas are explored. Although their research may be imitative of ideas in Fujihira’s work their final design outperformed the prior in both component simplicity and control bandwidth. Their design has provided the background framework for this thesis to determining replicability with standardised components within a limited budget.
4 Evaluation and Creation of Underactuated Propeller Prototype

4.1 Design Justification

The prevalence of cheap and easily pilotable multirotor drones may lead some to believe that the multirotor design is a more efficient or better design. Analysis of the mechanics of single and multirotor flight proves otherwise. Equation 1 and Equation 2 describe the energy exchange relationship between the rotors and the motor. This is further explored in Table 1 by using abstract values. The second and third rows of this table demonstrate the effect of either doubling the mass of air moved by the rotor or doubling the velocity of the air moved by the rotor. Both cases see the momentum double, however the energy required quadruples when velocity is changed, rather than only doubling when the mass of air is changed. This explains why a larger, slower-moving prop (found on single rotor designs) is more efficient than a smaller, faster moving prop (found in multirotor designs). [35]

\[ p = mv \]

*Equation 1: Momentum Equation*

\[ E_k = \frac{1}{2}mv^2 \]

*Equation 2: Kinetic Energy Equation*

<table>
<thead>
<tr>
<th>Mass (kg)</th>
<th>Velocity (m/s)</th>
<th>Momentum (kg.m/s)</th>
<th>Ek (J)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>10.0</td>
<td>10.0</td>
<td>50.0</td>
</tr>
<tr>
<td>2.0</td>
<td>10.0</td>
<td>20.0</td>
<td>100.0</td>
</tr>
<tr>
<td>1.0</td>
<td>20.0</td>
<td>20.0</td>
<td>200.0</td>
</tr>
</tbody>
</table>

Multirotors also suffer from interactions between each of the rotors. To eliminate these interactions, designers space the rotors further apart and this contributes to the overall multirotor design inefficiency. The interactions between rotors, caused by the blade vortices from each rotor, has led to a spacing guideline. This guideline states that each rotor should be spaced \( \sqrt{2} \times r \) apart, where \( r \) is the rotor radius [36], [37]. The effect of this guideline is displayed in Figure 19. Equation 3 is used to provide a quantitative evaluation at Table 2. These design issues illustrate how the quad rotor requires almost a third more lifting power due to limitations incurred by interactions between each rotor.
\[ P = \frac{T^\frac{3}{2}}{\sqrt{2 \rho A}} \]

Equation 3: Momentum Theory, Power for Thrust Equation [38]

### Table 2: Worked Example to compare Single Rotor to Quad Rotor spacing inefficiencies

<table>
<thead>
<tr>
<th>Single Rotor Area (m²)</th>
<th>Fuselage Area (m²)</th>
<th>Quad Rotor Area (m²)</th>
<th>Fuselage Area (m²)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.785</td>
<td>0.049</td>
<td>0.428</td>
<td>0.000</td>
</tr>
<tr>
<td>0.736</td>
<td></td>
<td>0.428</td>
<td></td>
</tr>
<tr>
<td><strong>P (W)</strong></td>
<td><strong>T (N)</strong></td>
<td><strong>p_{air} (kg/m³)</strong></td>
<td><strong>A (m²)</strong></td>
</tr>
<tr>
<td>Single Rotor</td>
<td>0.745</td>
<td>1</td>
<td>1.225</td>
</tr>
<tr>
<td>Quad Rotor</td>
<td>0.977</td>
<td>1</td>
<td>1.225</td>
</tr>
</tbody>
</table>

**Quad Rotor Power Increase:** 131.199%

### 4.2 Rotor Hub Design

As described in Section 2.4, thrust and attitude forces have been traditionally generated by single rotors using cyclic control of the rotor blade pitch. The design proposed by James Paulos, shown at Figure 16 in Section 3.3.2 required physical replication so that the overall concept could be tested and verified. James Paulos’ PhD is embargoed until 18 January 2021 [39] which limited the ability to accurately replicate the original design. Fortunately, 3D computer-aided design software, namely Fusion360, produced by Autodesk, Inc. was available. This software enabled the rapid iteration of designs that could be 3D printed at Murdoch University, assembled and then tested. The final design iteration is shown at Figure 20.
The construction of these designs involved breaking them down into constituent bodies such as the centre cylinder or the shapes that make up each hinged arm. An example of how the centre hub was constructed is shown at Figure 21. The advantage of this methodology was that individual bodies could be modified between 3D prints. This enabled changes to be made to particular aspects of the design, such as the freedom of movement for the hinged components or the angles at which they were hinged.
The 3D printer used was a MakerBot Replicator 2 Desktop 3D Printer, shown at Figure 22. The Replicator 2, and its associated software, MakerBot MakerWare was able to import the models produced by Fusion360 and replicate them with a 100 micron accuracy. The large 28.5cm x 15.3cm x 15.5m build volume provided by the printer allowed for multiple design iterations to be printed at one time. To achieve usable models, time was invested to perfect the Replicator 2 setup and printing process. Successful prints took 6 to 10 hours and were achievable approximately 75% of the time. A notable portion of project time was lost diagnosing the reason why prints failed, such as shown at Figure 23.

Figure 22: Murdoch University MakerBot Replicator 2
Noting the problems encountered by Aubrey Cason with 3D printing rotor blades, [40] the decision was made to use cheap, commercially available blades which could attach to the designed rotor hub. DJI Mavic 2 low-noise propellers were chosen due to their robust construction and simplistic mounting. Their adaption to the designed rotor hub is shown at Figure 24 and Figure 25. This design choice allowed for the same set of blades to be tested on each of the design iterations of the rotor hub. Stainless steel wire of 1.57mm diameter was used for all hinges and connections.
4.3 Concept Explanation

Once a physical model of the system had been created it became easier to understand the control concept put forward by Paulos [5] and discussed in Section 3.3.2. Figure 26 explains Paulos’ concept using the model developed for this thesis. The positive (green) and negative (red) blades and hinges respond in opposite manners to impulsive and retrograde torques. This response changes the pitch of the blade on that side, thereby changing the amount of thrust generated. Impulsive and retrograde torques are generated by sinusoidal changes in speed to a steady rotor frequency. These changes are achieved via control of the motor. As shown in Figure 27, definite regions of impulsive and retrograde torque are creatable with the average value for the cycle being the desired rotation speed. Figure 27 also demonstrates how changing the phase of the sine wave can change the position of a torque type relative to the start of that waveform.

![Figure 25: Rotor blade mounting to Hub assembly using steel wire](image)

![Figure 26: Blade Response to changes in Torque](image)
4.3.1 Thrust Creation

The effect of superimposing a sine wave to the desired speed is the generation of an unbalanced thrust force centred over 50% of the revolution of the rotor. Figure 28 and Figure 29 demonstrate the three key responses to this concept; the equilibrium state when no sine wave is superimposed and then the positive and negative blade responses to impulsive and retrograde torque. The first component of these two figures demonstrates that when a constant torque is applied the blade pitches are equal and the thrust generated over each is in equilibrium. A perfectly balanced rotor (noting that manufacturing irregularities introduce small aerodynamic imbalances) would thereby have no attitudinal forces generated.
Figure 28: Side View of Unbalanced Thrust creation through Torque Application

0° Rotation, Constant Torque
Blade pitches balance producing equal thrust

0°-180° Rotation, Impulsive Torque applied
Increase pitch on ‘positive’ blade
Decrease pitch on ‘negative’ blade

180°-360° Rotation, Retrograde Torque applied
Decrease pitch on ‘positive’ blade
Increase pitch on ‘negative’ blade
Figure 29: Overhead View of Unbalanced Thrust creation through Torque Application

0°–360° Rotation, Constant Torque
Same pitch on ‘positive’ and ‘negative’ blades
Balanced thrust generation

0°–180° Rotation, Impulsive Torque
Increases pitch on ‘positive’ blade,
Increases thrust in that sector of rotation
Decrease pitch on ‘negative’ blade

180°–360° Rotation, Retrograde Torque
Decreases pitch on ‘positive’ blade,
Increases pitch on ‘negative’ blade,
Increases thrust in that sector of rotation
The second and third components of each figure demonstrate how unbalanced thrust forces are generated over an area of revolution. These unbalanced forces lead to the generation of the desirable attitudinal forces on the rotor assembly in a similar manner to a swashplate. The final designs maximum hinge angle is ±43° which allowed for a considerable range of pitch variation. These extremes were not encountered as their positioning would require significant amounts of impulsive or retrograde torque which would only occur during the first few revolutions at motor start up. Control of the angle of the hinges, and thereby the pitch of the blade, is achievable by controlling the amplitude of the sine wave. This increases the magnitude of the impulsive or retrograde torque force which is opposed by the aerodynamic forces and rotational motion acting to balance the blade pitch. This concept demonstrates that control of the magnitude of the attitudinal forces may be gained by controlling the amplitude of the superimposed sine wave.

To achieve directional control of this attitudinal force, a phase shift to the sine wave could be applied. The result of this phase shift would be to change the location of the sector of greater thrust proportional to the phase shift. As long as the frequency of the sine wave is matched to the frequency of the rotor then the relationship between phase shift and attitudinal force direction should be the same. Figure 27 shows the effect of a 180° phase shift to a sine wave. When this is applied to the example in Figure 29 the effect would be to flip the segment with greater thrust 180° to the opposite side.
5 Design and Development

To recreate the impulsive and retrograde torques described, a high level of motor control had to be established. This required multiple components ranging from a motor that was powerful enough to drive the rotor assembly, an electronic speed controller that could power and control said motor, to a single board computer which would set the desired speed and facilitate the sine wave generation. Advances in hobby drones and their racing has led to the development of cheaply available, high quality versions of all of these components. After a range of research and experimentation, suitable modules were chosen and tested in order to provide the desired forces that would facilitate attitudinal control.

![Diagram of Motor Control Components](image)

*Figure 30: Motor Control Components. Adapted from: [41]*

5.1 Direct Current Brushless Motors

Many modern hobby level drones use direct current brushless motors (BLDC) over brushed motors due to their higher power to weight ratio, low cost and longer lifespan. Their design is based on the same electromagnetic principles of brushed motors, with the key design change being the location of the windings. As seen in Figure 31 brushed motors have windings on the rotor whilst brushless motors have them on the stator. For all their advantages the removal of the brushes has introduced the requirement of components to control the power to the motor. This Electronic Speed Controller (ESC) replaces the brushes original functionality
of rotor position detection through the use of additional sensors or back electromagnetic force. Accurate detection is fundamental for their high efficiency. [41] [42]

The motor used for this thesis was a Turnigy D2836/8 1100KV Brushless Motor. This is a cheaply available, hobby brushless motor suitable for drones. It was chosen as a low cost option that could be independently and reliably sourced for future replication if required. The 1100KV rating indicates the speed the motor should rotate for every volt applied. The electrical speed controller chosen used a 14.8V supply. The resultant maximum unloaded motor Revolutions Per Minute (RPM) of 16,280RPM (14.8V x 1100KV) was deemed sufficient for the operating range that would be required.

5.2 Electronic Speed Controllers

As discussed in the prior section the inclusion of a BLDC motor necessitated the inclusion of an ESC. Most hobby level ESCs take input motor commands, normally sent using Pulse Width
Modulation (PWM) and convert this to the power switching actions that are required to drive the BLDC motor at a particular speed. The Wraith32 ESC was chosen for this thesis due to its implementation of the Digital Shot (DShot) protocol and its 32 bit Cortex-M0 Micro Controller Unit operating at 48 MHz. This ESC changed from the V1 to V2 model during testing. The differences between the two were mainly with the telemetry response from the ESC to communication device. This telemetry response was not used so had little impact.

5.2.1 DSHOT Protocol

The DShot protocol used by the ESC provided an accurate way to transmit speed values to the ESC and has a number of advantages over the existing PWM signal method. Firstly, the digital protocol and its transmission method eliminates the jitter that high frequency PWM signals sustain. Secondly, as a digital protocol the information transmitted can include a Cyclic Redundancy Check (CRC) that can assist with the detection and rejection of corrupted data. Lastly, as seen in Figure 34, the protocol is currently the fastest hobby level protocol available at nearly double the speed of its nearest competitor, Multishot [43]. This high speed communication is critical for the replication of the sine wave superimposition, allowing for the changing of rotor speeds multiple times during one cycle of revolution.

Figure 33: Wraith32 32bit 35A ESC V2
The DShot protocol was originally developed in an open source environment, centred on the RCGroups.com forum [44]. The author’s original goals included eliminating signal jitter, providing a higher resolution of speed control, eliminating oscillator drift (and the need to calibrate the ESC), providing more resistance to signal noise and implementing a CRC. The protocol was developed in association with Flyduino and the Betaflight Group and led to it moving to a closed source environment. Fortunately, original iterations of the code are still available on the internet and much of the communication methodology can be worked back from other open source examples, namely Cleanflight [45]. Cleanflight is open source 32 bit flight controller software made available for community development. From analysis of the Cleanflight controller software and the posts from its original development a large amount of the technical details regarding the protocol were able to be sourced.
5.2.2 Packet Structure

The DShot protocol consists of a relatively simple packet structure and a unique bit structure. 16 bit packets are sent from controller to the ESC. The 16 bits are made up of 11 representing the ESC command, the 12th as a telemetry response request indicator and the last four area a CRC calculated from the preceding 12 bits. This structure can be seen in Figure 35.

<table>
<thead>
<tr>
<th>BYTE ONE</th>
<th>BYTE TWO</th>
</tr>
</thead>
<tbody>
<tr>
<td>11 12 13 14 15 16 17 18</td>
<td>2.1 2.2 2.3 2.4 2.5 2.6 2.7 2.8</td>
</tr>
<tr>
<td>11 bit Command</td>
<td>1 bit Telo</td>
</tr>
<tr>
<td>$2^{11} = 2048$ commands; $0-47$ flight controller parameters; $48-2047$ throttle options (2030)</td>
<td>XOR bits 1,5,9 XOR bits 2,6,10 XOR bits 3,7,11 XOR bits 4,8,12</td>
</tr>
</tbody>
</table>

Figure 35: DShot1200 Packet Structure

One of DShot’s greatest strengths as a protocol comes from the unique bit structure implemented by its authors. The problems with PWM, namely jitter, are timing related. The DShot bit structure decouples the need to have a consistent time base between the sending and receiving devices. This is achieved by changing the requirement to use an accurate timing function when sending binary low or high voltages. Instead, the bit structure is defined by the combination of irregular timed pulses of high voltage and low voltage. The bit value is thereby determined by whichever pulse was longer. A graphical comparison of a standard bit structure against the DShot structure is shown at Figure 36, noting the red bars indicate the distinction between individual bits. The advantage of this bit structure is that considerable tolerance is introduced in sending and receiving bits. This is important with cheaper hobby level equipment when low voltage situations, such as when battery packs are near depletion, may induce changes in the clock speeds of the controlling devices. DShot enabled devices are thereby able to remove the need for separate clock lines or advanced encoding such as Manchester encoding. The drawback to this design is that a single DShot bit takes three times longer to send than a standard bit.
5.3 Communication Devices

Two communication devices were used during this thesis to control the ESC. Each of the devices had their own set of advantages and disadvantages and are discussed independently. Many hobby drones use similar devices as the main control unit for their design.

5.3.1 Arduino Ameba RTL8195

The Arduino Ameba RTL8195 (Ameba) is a cheap Internet of Things-enabled microcontroller board that allows users to create programs in C and carry out dedicated tasks. The Ameba is a recent advancement in the Arduino family and for its $25.00 cost features a powerful 32-bit Arm Cortex M3 chip that runs at 166MHz. As a design option it offered a Serial Peripheral Interface (SPI) which could be modified to use the DShot protocol, had inbuilt WiFi and a large software resource base. This was deemed important as initial research had indicated that the protocol may have already been replicated and could be used for this thesis.

The Ameba also offered a number of digital and analog inputs and outputs that could be used to enhance the usability of the device. Coding for the Ameba is conducted through the Arduino Integrated Development Environment (IDE), which is a software application run on a local computer that supports the development of code. Code is written in this IDE, compiled locally into a format that the Ameba accepts and then transferred to the Ameba. When the
Ameba is powered on it executes whatever code was written to it for as long as the code loops and it is powered.

5.3.2 Raspberry Pi Model B+

The Raspberry Pi (RPI) represents the next step in power and complexity for hobbyist computing. It is marketed as a $55.00 credit card size educational computer and features a powerful 1.4GHz 64-bit quad core processor. This additional computing power means that the RPI is able to run an operating system (Debian) which supports direct input via keyboard and mouse and support visual displays via HDMI. The quad core processor enables multiple
tasks to be run concurrently, compared to the Ameba’s single task ability. The RPi fills its educational role, and differs itself from other microcomputers by allowing users to write code on the RPi that will directly interact with its various inputs and outputs. Similar to the Ameba these include SPI, PWM, I2C, Serial and General Purpose Input/Output (GPIO) pins. Multiple languages are supported by the RPi, including C and Python with the choice depending on the user’s requirements.

Figure 38: Raspberry Pi Model B+

5.3.3 Serial Peripheral Interface

The SPI is a serial communication interface specification developed by Motorola in the 1980s. It is still used in Secure Digital cards (SD cards) and Liquid Crystal Displays (LCDs). The main premise is the use of given size shift registers, normally in multiples of eight bits, that move data one bit at a time dependent on a clock cycle. Multiple shift registers, on different devices, can be connected such that as a bit moves out from the first device into the second device the same happens from the second devices output to the first’s input.

Using the one byte example shown in Figure 39, it can be seen that after eight clock cycles the byte from the master would be in the slave’s shift register with the slave’s byte now
located in the master’s shift register. The definition of master comes from the device controlling the common clock signal which synchronises the rate at which devices send and receive their bits. When multiple devices are connected as slaves, additional slave select lines must be added. These lines communicate the requirement to each slave device to connect or disconnect itself from the serial connection as its slave select line goes high or low. [46]

*Figure 39: SPI Shift Register Example [47]*
6 Method and Implementation

6.1 Protocol Calculation and Verification using Microsoft Excel

Once the communication devices had been chosen, further research was conducted to establish if there was an existing communication library for DShot that could be used or adapted for the purposes of this thesis. Unfortunately, the libraries available were locked to specific drone control packages which would not support the sine wave superimposition required. Following the analysis of the bit structure and noting the similarity to multiples of three standard bits it was theorised that the bit structure could be replicated using a normal bit pattern. This would remove the requirement to learn and code new low level DShot bit structure communications and allow the usage of the more common SPI specification.

6.1.1 Bit Replication

Using the example from Figure 36 it was theorised that each bit from the DShot bit structure could be replicated using three traditional bits. This would require a traditional bit communication medium, for example SPI, to communicate at a speed three times faster than the DShot protocol speed. A method of converting the packets made up of DShot bits to traditional bit would also have to be established. Figure 40 demonstrates this concept and the increased speeds that the traditional bit pattern device would have to achieve. Using this format a DShot 0 bit becomes ‘100’ and a DShot 1 bit becomes ‘110’.

![Figure 40: DShot bit conversion to Traditional bit pattern](image-url)
To experiment with this concept a Microsoft Excel (Excel) document was created that could track the changes for each bit and prove the concept. To achieve this and track the correct conversion of each bit a number of Excel’s built in commands were used. The first step was to calculate the binary representation of the 0 to 2047 numerical commands that made up the first 11 bits of each DShot packet. The Excel DEC2BIN function was unable to achieve this as its range is for numerical values between -512 and 511 (10 bit resolution). Fortunately, a Visual Basic for Applications function was available online for use. This allowed for the expansion of the range by using powers of two to convert the value [48]. Using the packet structure at Figure 35 a telemetry bit was then added to the first 11 bits, followed by the four bit CRC. This CRC was developed from the example found at Cleanflight [45] with its calculation found to be as per Table 3. This CRC is key to detecting accidental changes to the original information sent and ensures that the ESC does not erroneously change the speed. When the receiving device (the ESC) receives the packet in full it completes the same CRC calculation on the first twelve bits of the message and then compares its calculation to the CRC sent in the message to ensure accuracy.

**Table 3: CRC Bit Calculation**

<table>
<thead>
<tr>
<th>CRC Bit</th>
<th>Bit Calculation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>XOR(Bit1, Bit5, Bit9)</td>
</tr>
<tr>
<td>2</td>
<td>XOR(Bit2, Bit6, Bit10)</td>
</tr>
<tr>
<td>3</td>
<td>XOR(Bit3, Bit7, Bit11)</td>
</tr>
<tr>
<td>4</td>
<td>XOR(Bit4, Bit8, Bit12)</td>
</tr>
</tbody>
</table>

This Excel document provided a quick and understandable method to converting DShot packets into a format that could be sent using a standard communication method. This methodology tripled the two byte DShot bit packet to a six byte standard bit packet. These packets were precalculated and then coded to the Ameba for initial testing using the SPI medium.
<table>
<thead>
<tr>
<th>Std Bits</th>
<th>1 1 1 0 0 1 1 0 1</th>
<th>DShot Bit</th>
<th>1 2 3 4 5 6 7 8 9 10 11 1 1 2 3 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Std Bit Repn</td>
<td>110 110 110 100</td>
<td>Std Bit Concat</td>
<td>11011011 01001001 10110110 10011010 01001101 10100110</td>
</tr>
<tr>
<td>Std Bytes</td>
<td>219 73 182 154 77 166</td>
<td>Binary Representation (Text)</td>
<td>11100111010</td>
</tr>
<tr>
<td>Bit Length</td>
<td>6 48 6</td>
<td>Byte Length</td>
<td>48</td>
</tr>
<tr>
<td>DShot Bit Value</td>
<td>1 1 1 0 0 1 1 1 0 1 0 0 1 1 0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command Value</td>
<td>1850</td>
<td>Binary Representation (Text)</td>
<td>111001110110</td>
</tr>
<tr>
<td>Telemetry</td>
<td></td>
<td>Command Value</td>
<td>11100111010</td>
</tr>
<tr>
<td>Checksum</td>
<td></td>
<td>Binary Representation (Text)</td>
<td>11100111010</td>
</tr>
</tbody>
</table>
6.2 Arduino Implementation

With complete DShot packets generated the next stage was to test them with the ESC and motor. This setup was connected as shown in Figure 42 with power supplied by a variable voltage 2.5 amp DC supply, set at 14.8V. Analysis and modification of how the Ameba implemented SPI enabled initial testing to occur with relatively simple test code. The initial Ameba SPI connection to the ESC was successful with the generated packets causing the ESC to energise the motor rapidly in a way that made the motor emit a loud beep. This beeping was an indication that the ESC was ready to conduct its arming sequence. This arming sequence initialises the ESC, confirming the communication method to be used, either DSHOT or PWM and tests the ESCs connection to the motor. Fortunately, this arming sequence is still published, as shown at Figure 43, and was able to be conducted using the Ameba. The conduct of this arming sequence led to the discovery of a key issue with the Ameba and necessitated future control to use the RPi.

![Figure 42: Wiring guide for Ameba, ESC and Motor](image)

![Figure 43: Arming Sequence for ESC](image)
6.2.1 Ameba Issues

The initial working version of Ameba code is shown at Figure 44. This code was written to be as concise as possible to prove that the bit structure replication method could be used with SPI. The code that was implemented identified two key issues during its development. The first issue was that the SPI function written for the Ameba included an interbyte delay. This artificial delay caused the Ameba to pause for approximately 3.3μs between bytes. This pause introduced abnormal low voltage periods after every eight bits were sent, terminating the accurate replication.

Analysis of how the Ameba library implemented the SPI functions demonstrated the use of header files to declare the function and .cpp files to define the function. Evaluation of this code showed that the SPI transfer functions were also controlling the slave select pins. At the end of each byte the Ameba would lower the slave select pin then raise it again at the start.
of the next byte. This is a useful function for traditional SPI communication as it provides each device time to read and write new data to its shift register. This was not required by the Ameba as no data was being read by the master and both devices operated at speeds at least one order of magnitude higher than the communication rate. The solution to this problem was to comment out the portions of code inducing delays as shown at Figure 45.

```
byte SPIClass::transfer(byte _pin, uint8_t _data, SPITransferMode _mode)
{
    byte d;

    /* CODE WRITTEN OUT TO REMOVE INTERBYTE DELAY
    if (_pin != pinSS) {
        pinMode(_pin, OUTPUT);
        digitalWrite(_pin, 0);
    }
    */

    d = (byte) spi_master_write((spi_t *)pSpiMaster, _data);

    /* CODE WRITTEN OUT TO REMOVE INTERBYTE DELAY
    if (_pin != pinSS && _mode == SPI_LAST) {
        digitalWrite(_pin, 1);
    }
    */

    return d;
}
```

*Figure 45: Edited code snippet from Ameba, SPI.cpp*

The removal of this code enabled the Ameba to communicate successfully with the ESC. However, as the code was expanded with additional for and if loops it began to noticeably slow down. This introduced significant gaps between packets as the Ameba processed the conditional statements. This dramatically slowed the development of the code required to rapidly communicate the different motor speeds required to mimic a sine wave.

The issue was compounded by further testing to see whether the Ameba could also respond to button presses wired to other inputs. These buttons were to be added to allow external control of the motor speed and code without having to reset the Ameba. The addition of these button sense elements introduced delays to the communication whilst the button input was checked. These delays were long enough for the ESC to reset itself as if a communication loss had occurred. Significant time was invested in trying to resolve this problem, with attempts made to protothread the two functions. The Ameba, as a single-core chip with procedural code cannot be multithreaded as there is only one core on which tasks can be
executed. Protothreading is similar to hyperthreading, however rather than using a certain number of cycles as the decision to switch threads or tasks the system executes a main loop which conditionally run a large loop or a number of smaller loops if the large loop doesn’t require execution.

Analysis of the time spent coding in the Arduino IDE then waiting while it compiled and transferred to the Ameba suggested the need for another device. As a result, research was conducted as to whether other devices would be more conducive to the rapid iterations of the test code and could still support the communication method.

6.3 Raspberry Pi Implementation

The RPi was chosen as a suitable replacement for the Ameba due to its improved speed, keyboard and display support and easily accessible inputs and outputs. Connection the ESC and motor were similar as to the Ameba with the addition of a keyboard, display and resistor. The resistor addition occurred during testing when it was found that the SPI connection required the use of a pull down resistor to ensure the communication wire was pulled to a low logical level between signals. This pull down resistor combats the floating logic state that can occur due to the high impedance of the ESC. The addition of the resistor provides a way for positive charge built up at the ESC data input to drain back to ground when the RPi signal is dropped. The large resistance value ensures that only minimal voltage is lost through the resistor when the RPi switches the voltage to high.

![Wiring guide for Raspberry RPi, ESC and Motor](image)

*Figure 46: Wiring guide for Raspberry RPi, ESC and Motor*
6.3.1 SPI Setup

The RPi came preinstalled with a number of libraries, similar to the Ameba. Noting the difficulties with the Ameba, SPI function research was conducted to establish the best method of SPI utilisation. The RPi’s documentation noted that its SPI function was relatively basic and offered a limited number of driver speeds. These speeds are determined by clock divisors and are shown in Table 4. As the desired 3.6MHz speed did not correspond, the PIGPIO library was sourced [50]. The PIGPIO library is a library for the RPi that allows advanced control over the general purpose inputs and outputs. One key advantage is that the library has broken the SPI functions down into their constituent functions. This allowed the written code to transfer a desired number of bytes, at high speeds, back to back with no interbyte delay.

<table>
<thead>
<tr>
<th>Clock Divisor</th>
<th>Speed</th>
<th>Clock Divisor</th>
<th>Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>125.0 MHz</td>
<td>512</td>
<td>488 KHz</td>
</tr>
<tr>
<td>4</td>
<td>62.5 MHz</td>
<td>1024</td>
<td>244 KHz</td>
</tr>
<tr>
<td>8</td>
<td>31.3 MHz</td>
<td>2048</td>
<td>122 KHz</td>
</tr>
<tr>
<td>16</td>
<td>15.6 MHz</td>
<td>4096</td>
<td>61 KHz</td>
</tr>
<tr>
<td>32</td>
<td>7.8 MHz</td>
<td>8192</td>
<td>31 KHz</td>
</tr>
<tr>
<td>64</td>
<td>3.9 MHz</td>
<td>16384</td>
<td>15 KHz</td>
</tr>
<tr>
<td>128</td>
<td>1953 KHz</td>
<td>32768</td>
<td>7629 Hz</td>
</tr>
<tr>
<td>256</td>
<td>977 KHz</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The initial code written for the Ameba was translated to pure C with the PIGPIO functions to test communicability from the RPi to the ESC. Initial testing was not successful until it was noticed that the full packet was being transmitted at a greater speed than it should have been. This was discovered by using an oscilloscope to visualise the packets as seen in Figure 47 and measuring their length. The keyboard and mouse support from the RPi provided the ability to rapidly iterate through the coded SPI communication speeds and this issue was quickly resolved. The SPI configuration speed used in the code was found to be 2.15MHz for an actual speed of 3.6MHz. No documentation could be found for this error although it is theorised that it may be due to differences between the older RPi variant on which the library was originally coded and the model used for this thesis.
6.3.2 Protocol Calculation

The successful communication using the RPi led to the development of a function that would generate all 2000 DShot speed commands in one array, rather than using the previous Excel method. This would enable the RPi to pre-calculate all the commands that could be sent, save them to local memory and quickly pass them to the SPI shift register. This function, shown in Figure 48, was named packetcreation.

\[
\text{packetcreation}(\text{char res[]}[6], \text{int from}, \text{int to})
\]

Figure 48: Packet Creation Function

The packetcreation function, seen at the end of the code at Appendix B, takes a user specified matrix and fills it with the speed commands within the range of the integer ‘from’ value to the integer ‘to’ value. This also allows for the function to be easily used in other coding projects that involve the DShot protocol.

6.3.3 Speed Testing

Testing was then conducted using the system to establish a relationship between the ESC speed values (with a range of 0 to 2000) and the actual speed that the motor spun with the blade assembly. As seen in Figure 49 it was found that this resulted in a relatively linear
relationship through the first quartile of the ESCs speed values. Due to the high speeds achieved, the velocity to rotor area relationship discussed in Section 4.1, and the need for safety, it was decided to cap testing with a 2500 RPM maximum. This conversion was used as a reference during the thesis and not coded to the RPi as to keep the code motor independent.

![ ESC Speed Values to RPM](image)

**Figure 49: ESC Speed Value to Rotor RPM**

6.3.4 Sine Wave Creation

Sine waves are applied to the desired speed value by generating an array of values. These values are the product of the sine wave resolution, the amplitude desired, and the phase shift required. This array of values was calculated prior to each testing loop through the use of a function, wavecreation, shown in Figure 50.

```c
wavecreation(int elements, int res[elements], int amfact, int phasedeg)
```

**Figure 50: Wave Creation Function**

This function takes the resolution of the waveform, a user specified array to store the values in, an amplitude factor and a phase shift in degrees. The function uses the basic sine calculation as a function of the resolution of the waveform. This is then converted to the time domain when the main code sends a number of packets per revolution of the rotor.

$$y(\text{step #}) = \sin(2 \times \pi \times \frac{\text{step #}}{\text{resolution}} + \varphi) \times \left(\frac{\text{resolution}}{4}\right) \times \text{amplitude factor}$$

**Equation 4: Sine Wave equation amended for Code**
The reason for this amended equation, particularly the use of the amplitude factor as opposed to just an amplitude is the requirement to convert the floating point values from the sine equation to the integer steps in speed values that the ESC could achieve. This concept is shown in Figure 51 where step like responses are generated by low resolution amplitudes (dashed lines) as opposed to acceptable waveforms when higher resolution amplitudes are used (solid lines).

![Sine Wave Creation - Double vs Integer Resolution](image)

*Figure 51: Sine Wave Creation - Double vs Integer Resolution*

6.3.5 Waveform Implementation

Waveform implementation was the final component to achieve the cyclical impulsive and retrograde torques that this thesis required. A series of nested for-loops were created that would iterate through each wave value, added to the desired speed value. This meant that communicated speed value sent to the ESC spent equal time below and above the desired speed value. The ‘steps’ variable controlled how many times the inner loop repeated. This inner loop continued to send the current speed value, as determined by the outer loop for the amount of iterations. The iteration of this inner loop was determined by the relationship...
shown at Equation 5, such that the product of the wave resolution, the ‘steps’ variable and the time to send one packet was equal to one revolution of the rotor. This ‘steps’ variable was manually pre-calculated using the relationship established in Figure 49.

\[ \text{Rotor Period} = \text{Wave Resolution} \times \text{‘steps’ variable} \times 23.9\mu s \]

*Equation 5: Rotor Period relationship to Code Variables*
7 Testing & Measurement

The work of Paulos [34] used a range of equipment for testing his design at the Modular Robotics Lab (ModLab) located at the University of Pennsylvania where he was completing his PhD. This lab supports advanced research into general robotics, automation, sensing and perception with a range of industry and government grants. One of the critical components for this testing was an ATI Industrial Automation Nano 17-E Transducer, being the smallest commercially available 6-axis transducers in the world [33]. This transducer, as seen at Figure 52, is used for measurement of very small forces, ranging down to 0.318 gram-force. They are commonly used in dental research and robotic surgery with a cost $11,058 if purchased by an academic organisation. This was outside of the budgetary confines of this thesis so an alternative method of measuring the lateral forces was created.

Figure 52: ATI Industrial Automation Nano17-E Transducer. Adapted from: [33]

7.1 Testing Platform

To measure the lateral forces generated by the rotor assembly a testing platform was created that met the low cost goal of this thesis. As the lateral forces were of key interest it was considered acceptable to focus on the flat X/Y plane without measuring the Z planes thrust axis. This was achievable through the design and implementation of a testing platform.

The concept for this testing platform used the trigonometric relationship between three vectors at known angles. If the forces generated by the rotor assembly could be measured, shown as x1,x2 and x3 in Figure 53, then the net X and Y axes forces could be calculated. To enable the construction of this testing platform a number of components were sourced or
created. These included; load cells for the measurement of the forces, a 3D designed and printed mount to house the motor and the load cells, and a Laboratory Virtual Instrument Engineering Workbench (LabVIEW) program to read the signals through Digital Acquisition (DAQ).

![Figure 53: Testing Platform Concept](image)

\[
\begin{align*}
X \text{ Axis} &= \cos(30^\circ) \times (x_2 - x_1) \\
Y \text{ Axis} &= \sin(30^\circ) \times (x_2 + x_1) - x_3
\end{align*}
\]

7.1.1 Load Cells

A load cell, according to [52], is “A device for measuring forces by means of strain gauges as opposed to the use of hydraulic pressure or mechanical means”. These are commonly available from hobby electronic suppliers for a small value. Load cells use strain gauges to convert the mechanical quantity of strain into an electrical signal. This is achieved through the associative change in resistance in the strain gauge as strain is applied. A Wheatstone bridge circuit, shown in Figure 54, uses the resistive values from four strain gauges. This arrangement allows for the measurement of the associated change in voltage and compensates for the effect of unwanted variables such as temperature change. [53] The load cells used were rated for a range of 0 to 100g.
7.1.2 Mounting

Building on the lessons learnt during the rotor hub design and construction a simple mounting system was designed using Fusion360. This system consisted of a base mount and a top mount, seen in Figure 55. The base mount had to include elements to allow for fixation to a benchtop and to fix the base of the load cells securely. The top mount had to be as evenly balanced as possible (to not induce unbalanced vibration forces during operation), to be able to fix to the top of the load cells, and to securely fix the motor and rotor assembly to itself.

Figure 55: Testing Platform Design
This design was 3D printed and assembled inside a laboratory where the environmental conditions could be kept within a consistent range. The completed physical platform, with rotor assembly is shown in Figure 56.

![Complete Testing Platform with Rotor Assembly](image)

**Figure 56: Complete Testing Platform with Rotor Assembly**

### 7.1.3 RPM Measurement

Measurement of the RPM was essential to tuning how long each sine wave value was held for by the controller. If this was not correct then the waveform would not match the one sine wave cycle per revolution of the rotor requirement. To conduct this measurement a laser RPM sensor was constructed using a red laser diode and a photosensitive Light Dependent Resistor (LDR) as shown in Figure 57. These parts were purchased from a local hobby electronics store for $10.90. A laboratory stand, a 3D printed mount and double sided tape were used to affix these two components so that the beam passed through the area that the rotors moved through with each rotation. As seen in Figure 58 this setup allowed for the measurement of the rotor RPM by counting the number of times that the LDR analog signal
significantly dropped its voltage per unit of time. This value was then divided by two to calculate the actual RPM as the rotor blades broke the laser beam twice per revolution.

Figure 57: Red Laser Diode and Photosensitive Light Dependent Resistor. Adapted from: [54] [55]

Figure 58: Rotor Blade Sensing Configuration
7.1.4 LabVIEW Digital Acquisition

LabVIEW is a development environment for the visual programming language developed by National Instruments (NI). This environment interfaces with a number of NI instruments, including DAQ cards that can be used to accurately measure and control both digital and analog values. With this software, and a PCI-6221 NI DAQ card, code was written to measure the analog load cell values and the responses from the LDR. This code is shown in Appendix D. The code was developed such that the sampling and logging of tests could be partly automated. This allowed for extensive tests that changed speeds, sine wave amplitudes and sine wave phase to be written on the RPi and then tested in one batch with minimal user involvement until the test was complete. The results of the test were then able to be output to an Excel document with the appropriate column headers so that the data could be analysed.

The LabVIEW code was written to provide the user immediate feedback as to what was being measured whilst tests were run. The format of the user display is shown at Figure 59. The top left chart consistently shows the last 100ms of information that the load cells output. The top right chart shows the last seconds worth of data that was sampled and is processed by a third order Butterworth filter to remove high frequency noise. The higher and lower limits for this filter are controlled by inputs at the bottom right of the display. The bottom chart displays the LDR signal being read. This is converted to the current RPM and a suitable ‘steps’ variable for the RPi code with both values being displayed in a box on the right hand side. The front page also includes control values for the number of samples to be taken every time the user selects ‘sample’, assisting with the automation of tests. A second tab, shown at Figure 60, is included which can be used to look at the data currently collected. All of this data is exported to Excel when the code is stopped. A wiring diagram for the testing platform is included at Figure 61.
Figure 59: LabVIEW DAQ Sampling Screen
Figure 60: LabVIEW Sampled Data Screen
7.1.5 Calibration

Calibration of the testing platform was required to convert the load cell voltages to a usable force measurement. This calibration was conducted using 10g weights, shown at Figure 62. This testing process consisted of suspending the weights using a pulley and aligning that with the X or Y plane of the testing platform. A number of measurements were conducted as the weight was stepped from 10g to 50g.
The weights were converted from grams to Nmm noting the small forces expected as per Paulos’ work [5]. Each 10g were equivalent to 98.07Nmm and a linear relationship was discovered through the calibration as seen in Figure 63 and Figure 64. The advantage to this linear relationship was that it could still be used even if the system zero was adjusted during manual handling. This calibration calculation was applied to the Excel spreadsheet data and was not included in the LabVIEW code.
Figure 63: X Axis Calibration

Figure 64: Y Axis Calibration
8 Results

The testing of the replicated system was successful in controlling the magnitude of the lateral forces, however was not successful in gaining control over their direction. Figure 65 demonstrates the cyclic response moments for the system with a superimposed sine wave at 1584RPM, 32% maximum amplitude and no phase offset. In this figure the black bars designate the period of revolution with the peaks of each axis representing the passing of a rotor blade over that axis. If the system were capable of untethered flight the average of these movements would result in pitch and roll responses relative to the X and Y axis of the frame. Each peak represents the peak moment generated by each rotor blade. The existence of major and minor peaks implies that each blade is not contributing equally to the system. This may be due to an imbalance of the printed rotor hub or may suggest that each blade hinge responds differently to the applied torques. Further evaluation of rotor hubs with different hinge angles may confirm this theory.

![Pitch and Roll Moments](image)

*Figure 65: Cyclic Torque and Response Moment*
8.1 Lateral Force Magnitude Control

Figure 66 demonstrates the success in the replication of control over the magnitude of the lateral forces generated by the system. This figure displays a direct correlation between lateral force generation and the amplitude of a superimposed sinewave to the motor speed. The forces generated are approximately quadruple those generated by the limited range of testing presented by Paulos [5], although there are substantial differences in the pitch and size of the blades used for his study. When the results of Paulos’ PhD are published further comparison may be made. A key point of differentiation between his study and this one was the inexistence of a plateau region that all of his tests discovered. This may be due to design differences in the hinge assembly allowing a greater level of freedom of movement around the hinge and the pitch generated. This theory could be tested by the same input tests conducted with hinge designs that limited the amount of movement available.

![Graph](attachment:graph.png)

*Figure 66: Moment Magnitude Mean Response*
8.2 Lateral Force Direction Control

Direction control was tested by applying phase shifts to the system at a constant RPM and amplitude. The results of these tests were plotted on the X and Y planes with measurements made of the angular direction change or each successive test point. Whilst the phase changes did result in directional change there was little correlation from the magnitude and direction of the phase shift to the directional change. This result may be due to the start of each sine wave cycle not being locked to the start of a rotation. This could be achieved by adding a sensor to the RPi and locking the sine wave’s start to a definite point in the rotors’ rotation every time.

<table>
<thead>
<tr>
<th>Phase Shift</th>
<th>Angular Direction</th>
<th>Direction Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>0°</td>
<td>-76°</td>
<td>-</td>
</tr>
<tr>
<td>45°</td>
<td>83°</td>
<td>-158°</td>
</tr>
<tr>
<td>90°</td>
<td>-59°</td>
<td>142°</td>
</tr>
<tr>
<td>135°</td>
<td>1°</td>
<td>-60°</td>
</tr>
<tr>
<td>180°</td>
<td>43°</td>
<td>-42°</td>
</tr>
<tr>
<td>225°</td>
<td>-11°</td>
<td>54°</td>
</tr>
<tr>
<td>270°</td>
<td>21°</td>
<td>-32°</td>
</tr>
<tr>
<td>315°</td>
<td>-21°</td>
<td>42°</td>
</tr>
<tr>
<td>360°</td>
<td>-60°</td>
<td>39°</td>
</tr>
</tbody>
</table>

*Figure 67: Directional Magnitude Response to Phase Shift*

8.3 Thesis Costings

One of the aims of this thesis was to replicate the studies of Paulos [5] at a substantially lower cost using hobby level components. Whilst this goal may have impacted the precision of the results gained it has enabled a unique flight control concept to be tested at a lower level. With regards to the testing of a single rotor design the costs are almost one quarter of what a standard quad rotor design would cost. This can be seen in Figure 68 with the rotor blades, motor and ESC being roughly half of the cost of the development. A quad design would have required three times as many of these components and brought the system cost to $250+ region.
The testing platform replicated was at a substantially lower cost to the Nano-17E used by Paulos. Whilst this platform in no way matches the abilities of the Nano-17E it was an acceptable system to establish the generation of the lateral forces. The costing for created palform was substantially cheaper than the $11,058 for the Nano-17E however consideration for that devices sensitivity should be given if more advanced testing was to be completed. Advanced testing may include high resolution mapping of the forces generated, as would be required for an accurate flight control system.

<table>
<thead>
<tr>
<th>Area</th>
<th>Components</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motor</td>
<td>Turnigy D2836/8</td>
<td>$19.90</td>
</tr>
<tr>
<td>ESC</td>
<td>Wraith32 ESC</td>
<td>$14.95</td>
</tr>
<tr>
<td>Flight Controller</td>
<td>Raspberry Pi</td>
<td>$55.00</td>
</tr>
<tr>
<td>Blades</td>
<td>DJI Quick Release Propellers</td>
<td>$17.00</td>
</tr>
<tr>
<td>Force Measurement</td>
<td>3x 100gm Load Cell</td>
<td>$36.00</td>
</tr>
<tr>
<td>Speed Measurement</td>
<td>Red Laser Diode Light Dependent Resistor</td>
<td>$4.95 $5.95</td>
</tr>
<tr>
<td>3D Printing Material</td>
<td>1kg PLA</td>
<td>$30.00</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td><strong>$183.75</strong></td>
</tr>
</tbody>
</table>

*Figure 68: Thesis Costing*

Figure 68 does not include a costing of the man hours spent for this thesis. It should be noted that Paulos’ work represented the studies of his PhD in a discipline more closely aligned to the control of flight. His work was assisted by another PhD student in advanced forms of motor control allowing them both to focus on their strengths. Paulos’ work was financially supported by a National Science Foundation grant and could leverage on the ‘ModLab’ with its staff and facilities. The evaluation of this uncosted element demonstrates the ability of smaller organisations to explore the concepts behind advanced areas of research as long as a level of realism is provided for the precision of the results.
9 Future Work

The overall aim of this thesis was to replicate lateral force control over a rotor blade assembly using sine wave superimposition to the driving motor. This aim was achieved in part with control over the magnitude of the forces generated but not the direction. The designed system was built from scratch and included advanced flight concepts which were learnt as the thesis developed. This has led to a number of areas of improvement being noted throughout the thesis and provides an avenue for future study. The future works identified include a number of existing features that should be consolidated and tested before new features are added.

9.1 Measurement

To verify the results of the created rotor system, future work could be directed to improving the accuracy and usability of the testing system. With sufficient lead time and budgetary justification a higher quality transducer could be sourced to replace the current load cell arrangement. The advantage of this would be thrust measurement (along the Z plane) and a resolution improvement. Further research into the best options should be considered early in any future works given the lead time required for identifying and sourcing such a component.

If the current testing system is to be used again, research into the ground effect of the rotor should be considered. The ground effect with regards to aerodynamics is the increased lift that can be generated by rotors when they are close to ground. Due to the advanced flight concepts this was not explored in this thesis with all testing apparatus positioned 15cm above the table to try and counter it. A study into its effects would conclusively ascertain how they affect all testing with the current arrangement.

The LabVIEW code could be improved through additional error handling. Whilst this was not an issue for the testing conducted during this thesis a consistent method of error handling would allow for the code to be further added to by future students. This error handling would also assist with connecting the LabVIEW program to the RPi. This could be conducted through a SPI or WiFi. This would then allow the LabVIEW program to communicate with the RPi
testing system and remove the need for the user to make amendments to the RPi code between tests.

9.2 Direction Control
As discussed in Chapter 8 control over the direction of the lateral forces was not achieved. It is theorised that this is due to the lack of control over when a sine wave cycle starts relative to a particular position of the rotor. Control may be gained by locking the wave start to a sensed position on the rotor. This could be achieved in a number of ways. The easiest method would be to measure the laser sensor using the RPi as well as the LabVIEW program. This could be achieved by purchasing a RPi compatible analog-to-digital converter to measure the signal from the LDR. Every second iteration of the signal dipping would represent one of the blades (noting the signal dips twice per revolution). This could then be used to reset the wave signal and recalibrate its frequency based on the last RPM measured. This solution would only be for the short term however as a final goal would be to disconnect the flight system from the testing apparatus and achieve flight.

Instead of continuing with the laser sensor method, a rotary encoder could be implemented. This would involve the installation of the encoder to the existing system. It would be recommended that this is designed as a part of whatever flight system design is considered so that it would not have to be added later. This would be able to resolve the position of the rotor throughout the complete cycle of its revolution. This would not only assist with directional control but could be used to map the forces generated to the exact positions of the rotors revolution. This data may influence further design iterations of the rotor assembly, in particular the freedom of movement of the hinges.

9.3 Photographic Observation
Direct observation of the system would be advantageous to confirm the exact movements of the rotors throughout their revolution. Stroboscopy was experimented with during the thesis however the slow shutter speeds of available cameras captured significant motion blur so these images have not been included. The work of Paulos [5] implemented a xenon strobe with a duration of 1/13,000th of a second. Long exposures in darkened rooms were taken of
the system and then the bulb flashed to capture single images of the assembly. The signal to flash this bulb could be developed on the RPi if a rotary encoder is added. This would also allow for the exact positioning of the rotor blade for any photos taken.

9.4 Thrust Comparison

This thesis has not considered the effect that the changing torques has on thrust generated. If a thrust sensor or measurement of the Z plane was achieved then the hinged assembly could be evaluated against a fixed assembly using the same rotor blades. This would allow for quantitative comparison of the efficiencies of the underactuated system against other systems, be they fixed pitch blades or even a cyclic swashplate assembly to compare to other methods of blade actuation.

9.5 Rotor Assembly Modifications

The use of 3D design software and 3D printing substantially lowered the barriers to testing and modifying the rotor blade assembly. An avenue for further study could focus on changing the mechanical design of the system and then testing the effects of these changes. Key areas that could be investigated include; the materials used to make the assembly, the hinge angles relative to the rotor, the freedom of movement of the hinges and the effect of different blade sizes and pitches. A new design for the assembly was presented by Paulos [32] during the timescale of this thesis, as shown in Figure 69. The design uses the same concept as was replicated but adds another hinge assembly (highlighted in yellow) to each wing to improve blade tip dynamics. The original design could be modified to include this hinge allowing for comparison between the design used in this thesis and Paulos’ latest work.

Figure 69: New design for Rotor Assembly, New hinge highlighted in yellow. Adapted from: [32] © 2018 IEEE
9.6 Controlled Flight

The ultimate goal of all future works to this project would be to achieve independent flight using the concept replicated in this thesis. This would involve primary testing of the thrust generated by the rotor assembly. The thrust generated would in turn provide the limitations for the maximum weight of the entire design. This may involve multiple tests with different rotor sizes and speeds to provide a suitable range. The final design would have to include sufficient room for battery storage and a counter rotating motor and rotor blade to counteract the yaw moments generated by the primary rotor. It may be possible to use another cyclically controlled rotor assembly as the counter rotating rotor though due consideration should be given to the complexities of two of these systems in one flight vehicle. Flight control software will have to be designed or amended from existing software to provide control over the system as currently built. Figure 70 demonstrates this concept, involving an inner control loop consisting of the motor controller, the ESC, the motor and a model of the flight dynamics that the overall flight controller can provide thrust and lateral attitude commands to.

![Figure 70: Concept of overall Flight Control System](image)

With independent flight achieved, a regime of flight testing will have to be conducted. A similar sized quadcopter should be used to provide comparison against flight duration and system efficiency with testing conducted to compare flight abilities. If this goal is achieved then future goals may be able to incorporate further areas of research such as autonomy or multiple device swarm control.
10 Conclusion

The aim of this thesis was to recreate an underactuated propeller for attitude control using low cost components. The success of the results was measured in terms of the control achieved over the magnitude of the forces generated, the direction of the forces and the overall financial cost. This aim was not achieved in full. Directional changes force were generated, but not in a controllable manner.

The review of new MAV design literature conducted at the start demonstrated the need for an understanding of the principles of flight. It was interesting to note that whilst there is a substantial base of literature surround the principles of flight for full size aircraft there was surprisingly little for MAVs. It was also evident that a substantial amount of research relies on the traditional quadcopter design with limited consideration of alternatives. This may be due to the complexities of design work and the temptation to adjust existing solutions to fix new problems. The review of existing MAV designs demonstrates that the field is not stagnant. The novel approaches utilised by Paulos and further explored in this thesis attest to this.

The MAV design replicated in this thesis was broken down into its constituent parts. The rotor hub assembly was duplicated using 3D design software, a 3D printer and hobby drone rotor blades. This was used to confirm the theory behind the design and understand the impact of the torque modulation.

Controllable torques were generated using a direct current brushless motor, an electronic speed controller and a Raspberry Pi single board computer. This control required the replication of the DShot protocol. This was researched from the limited sources available, replicated in concept using Excel, tested on the Ameba and implement on the RPi through a Serial Peripheral Interface. Code was written on the RPi to superimpose the sine wave to the desired motor speed and facilitate the testing of the system.

A testing platform, consisting of a 3D printed assembly, load cells for measuring force and a laser beam for measuring the rotor speed was designed and built. This testing platform was a limitation to this thesis when compared to that used by Paulos, in terms of its reliability and
precision with regards to data collection. LabVIEW and a National Instruments Digital Acquisition Card were used by the platform for the measurement and logging of signals produced during testing. Calibration allowed the conversion of the signals into force measurements.

The testing platform was used with the constructed rotor assembly to test for the generation and control of lateral forces. Unfortunately, whilst control was gained over the magnitude of the lateral forces produced it was not gained over their direction. Analysis of the costs, against both multirotor designs and the original works, demonstrated it is possible to cheaply recreate advanced MAV designs. This thesis suggests that there are many avenues for future developments in this field. Projects could focus on perfecting the designs utilised in this thesis and advancing the ideas further with the goal of creating low cost, easily reproducible MAVs capable of independent flight.
11 References


Appendix A. IEEE Copyright Notice

As required by the IEEE the following copyright notice is shown for all material used with an IEEE copyright.

25/11/2018

Title: An underactuated propeller for attitude control in micro air vehicles
Author: James Paulos
Publisher: IEEE
Date: Nov. 2013
Copyright © 2013, IEEE

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author’s approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author’s university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity’s name goes here]’s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.
Appendix B.  
Amplitude Testing Code – C

1. /*AmplitudeTesting.c
2.  * Written by Jackson Thompson, 2018
3.  * Code is to be run on Raspberry Pi, 3B+
4.  * Pi requires PIGPIO library installed */
5.
6.
7. //Include predefined libraries for use in code
8.
9. #include <stdio.h>
10. #include <stdlib.h>
11. #include <stdbool.h>
12. #include <string.h>
13. #include <termio.h>
14. #include <unistd.h>
15. #include <pigpio.h>
16. #include <time.h>
17. #include <math.h>
18.
19. //Usage of definitions makes it easy to change key elements
20. #define SPEED 2150000 //SPI Sapeed, set at 2.15MHz, even though desired is 3.6MHz
21. #define BYTES 6 //Bytes to send, remember DShot bits x 3 std bts
22. #define AMPLITUDE 0 //Create initial value for amplitude of sine wave
23. #define RPS 250 //Default speed for motor
24. #define STEPS 32 //Amount of times a speed packet is sent for each step of sinewave
25. #define PHASE 0 //Initial phase offset for sine wave
26. #define AMPLOW 1 //Amplitude testing start point, remember is multiplied by 1/4 of resolution
27. #define AMPHIGH 15 //Amplitude testing start point, remember is multiplied by 1/4 of resolution
28. #define WAVERES 32 //Resolution of sine wave (how many points in sine wave)
29.
30.
31. /* Function Declarations */
32. bool kbhit(void); //Wait for key press function
33. void packetcreation(char res[])[6],int from, int to); // Packet creator function
34. void wavecreation(int elements, int res[elements],int amfact, int phasedeg); //Sinewave creator function
35.
36. /* Main Code */
37. int main()
38. {
39.  /* Variable Declarations */
40.  int speed=SPEED;
41.  int bytes=BYTES;
42.  int amplitude=AMPLITUDE;
43.  int rps=RPS;
44.  int rpsa=RPS;
45.  int steps=STEPS;
46.  int phase=PHASE;
47.  int waveres=WAVES;
48.  int waveres=waveres/4
49.  int amplow=AMPLOW;
50.  int amphigh=AMPHIGH;
51.  char packets[2000][6]=0]; // Empty matrix for storing speed packets
52.  char zeros[10]=0]; //Empty array for sending zero values, simpler than using timing functions
53.  int wave[waveres]; //Array for holding waveform variations
54.  int h; //SPI channel reference
55.  int x; //Used in loops
56.  int y; //Used in loops
57.  int c = \0; //Used to reset keypress function
58.  int c = \0; //Used to reset keypress function
59.
/* Calling Packet Creation Function, create all possible speed packets */
packetcreation(packets,0,2000);
printf("Packets Created Successfully \n");

/* Calling Wave Form Creation Function, create achievable sine waves */
if (amplow < 1) { //Ensure amplitude lower value cannot be negative
    amplow = 1;
    printf("Warning: amplow was below zero, set to zero \n");
}
if (amphigh*4 > rps) { //Ensure amplitude higher value cannot force waveform below 0.speed
    amphigh = rps/4 - 1;
    printf("Warning: amphigh would force speed below zero, set to acceptable value \n");
}
wavecreation(waveres,wave,amplitude,phase);
printf("Waveforms Created Successfully \n");

/* Testing GPIO initialised correctly */
if (gpioInitialise() < 0) {
    printf("GPIO not initialised correctly, program exiting \n");
    return 1;
}
printf("GPIO initialised correctly \n\n");

/* Open SPI with type name ‘h’ */
h = spiOpen(0, speed, 3); /* channel, baud, flags */

/* Starting Motor - Arming Sequence 1 */
printf("When low beep is heard press any key \n\n");
while (!kbhit()) {
    spiWrite(h, packets[500], bytes); /* handle, buf, count */
}

/* Starting Motor - Arming Sequence 2 */
printf("When high beep is heard press any key \n\n");
while(kbhit()) = getchar(); /* Reset keyboard hit flag */
while (kbhit()) {
    spiWrite(h, packets[0], bytes); /* handle, buf, count */
}

/* Starting Motor - Waiting for user ready to start motor
* Motor should now be in armed state */
printf("When ready to start press any key \n");
printf("Testing at %d speed value \n \n",rps);
while(kbhit()) = getchar(); /* Reset keyboard hit flag */
while (kbhit()) {
    spiWrite(h, packets[0], bytes); /* handle, buf, count */
}

/* Amplitude Testing Area - Will run amplitude testing of sine wave from
* amplitude low to amplitude high value. Remember the amplitude is multiplied
* by 1/4 of the resolution as dealing with integers */
printf("When ready to change start amplitude testing press any key \n\n");
while (kbhit())
{
    spiWrite(h, packets[rps], bytes); /* handle, buf, count */
}

printf("When ready to increase amplitude press any key \n");

while (kbhit()) c=getchar(); /* Reset keyboard hit flag */

for (amplitude=1; amplitude<16; amplitude=amplitude+1) { //Outer loop for stepping between amplitudes
    wavecreation(waveres,wave,amplitude,phase); //Create sine wave for that amplitude test
    printf("Testing at %d speed value and %d amplitude \n \n",rps,amplitude);
    while (kbhit())
    {
        for (x=0; x<waveres;x=x+1) { //Outer loop, steps between waveform values
            rpsa=rps+wave[x];
            for (y=0; y<steps;y=y+1) { //Inner loop, holds speeds value until needs to be changed
                spiWrite(h, packets[rpsa], bytes); /*SPI Write function: handle, buf, count */
                spiWrite(h, zeros, 8); //The amount of zeros can be changed for shorter/longer delay between packets
            }
        }
    }
    printf("Test Finished %d \n",c);
    spiClose(h); //Closes the SPI connections
    gpioTerminate(); //Terminates PIGPIO control over IO module
    return 0;
}

/*Check For Key Press Function
 * Note: Original Author unknown, refer to link for source

bool kbhit(void)
{
    struct termios original;
    tcgetattr(STDIN_FILENO, &original);
    struct termios term;
    memcpy(&term, &original, sizeof(term));
    term.c_iflag &= ~ICANON;
    tcsetattr(STDIN_FILENO, TCSANOW, &term);
    int characters_buffered = 0;
    ioctl(STDIN_FILENO, FIONREAD, &characters_buffered);
    tcsetattr(STDIN_FILENO, TCSANOW, &original);
    bool pressed = (characters_buffered != 0);
    return pressed;
}

/*Packet Creation Function*/
void packetcreation(char res[][],int from, int to)
int i; //Iteration variable
int j; //Iteration variable
unsigned short int speedoptions[2000] = {0}; //Array for holding DShot packets (Not for transfer)
char teleonoff = 0; //Telemetry on/off bit, set to 0
short int crc = 0; //Variable for CRC
short int crc_data = 0; //Variable for CRC Creation
unsigned short int fullpacketholder = 0;
char packetspeeds[2000][6] = {0};
for(i=from; i<to; i=i+1) {
   speedoptions[i]=i+48; //Creates speed values aligned with array ref (Control words 48 to 2047)
   if (i==1999) { //Error was occuring with 2000 speed value, this clear the error
      teleonoff=teleonoff;
   }
   speedoptions[i]=speedoptions[i] << 1 | teleonoff; //Adds telemetry request bit to end of message
}
/* CRC Creation Loop, adapted from Cleanflight Code */
for(i=from; i<to; i=i+1) {
   crc=0;
   crc_data=speedoptions[i];
   for (j=0; j<3; j=j+1) {
      crc ^= crc_data;
      crc_data >>= 4;
   }
   crc &= 0xf;
   speedoptions[i]=(speedoptions[i]<<4) | crc;
}
/* Conversion to normal bit structure from DShot structure
* Grabs bits one at a time, converts to 100 or 110 as required */
for(i=from; i<to; i=i+1) {
   speedholder=speedoptions[i];
   for (j=0; j<16; j=j+1) {
      if (32768 & speedholder) {
         fullpacketholder |= 6;
      } else {
         fullpacketholder |= 4;
      }
      if (j<15) {
         fullpacketholder <<= 3;
      }
      speedholder <<= 1;
   }
   //Separates entire message into byte sizes and allocates to byte matrix*/
   for (j=5; j>1; j=j-1) {
      packetspeeds[i][j]=fullpacketholder & 255;
      fullpacketholder >>= 8;
   }
   speedholder=0;
   fullpacketholder=0;
   /* Assigns each byte to matrix provided by user */
   for (j=0; j<6; j=j+1) {
      res[i][j]=packetspeeds[i][j];
   }
   // For printing out each packet to check
// printf("value: %d %d %d %d %d %d %d
",i,packetspeeds[i][0],packetspeeds[i][1],packetspeeds[i][2],packetspeeds[i][3],packetspeeds[i][4],packetspeeds[i][5]);

/*Sine Wave Creation Function
* Functions takes number of elements in wave, a pointer to user array, the wave amplitude and a phase shift (in degrees)*/

void wavecreation(int elements, int res[elements], int amfact, int phasedeg)
{
    /*Variables*/
    int x;
    double elementsdouble = elements;
    double xdouble;
    double sinval;
    double phase;
    /*Wave creation
    * Values are created as doubles, then converted to integers
    * so can relate to integer steps in speed */
    for(x=1;x<elements+1;x=x+1) {
        xdouble=x;
        xdouble=xdouble/elementsdouble; //Convert to ratio of number of steps
        phase=phasedeg*M_PI/180; //Convert phase to radians
        sinval=sin(xdouble*2*M_PI+phase)*(elements/4*amfact); //Multiply by sin value and amplitude
        sinval=round(sinval); //Round to achievable integer (noting speed values are all int)
        res[x-1]=sinval;
    }
}
 Appendix C.  Phase Testing Code – C

1. /*PhaseTesting.c
2. * Written by Jackson Thompson, 2018
3. * Code is to be run on Raspberry Pi, 3B+
4. * Pi requires PIGPIO library installed */
5. 
6. //Include predefined libraries for use in code
7. 
8. #include <stdio.h>
9. #include <stdlib.h>
10. #include <stdbool.h>
11. #include <string.h>
12. #include <termio.h>
13. #include <unistd.h>
14. #include <pigpio.h>
15. #include <time.h>
16. #include <math.h>
17. 
18. //Usage of definitions makes it easy to change key elements
19. #define SPEED 2150000  //SPI Speed, set at 2.15MHz, even though desired is 3.6MHz
20. #define BYTES 6  //Bytes to send, remember DShot bits x 3 std bts
21. #define AMPLITUDE 0  //Create initial value for amplitude of sine wave
22. #define RPS 250  //Default speed for motor
23. #define STEPS 32  //Amount of times a speed packet is sent for each step of sinewave
24. #define PHASE 0  //Initial phase offset for sine wave
25. #define AMPLOW 1  //Amplitude testing start point, remember is multiplied by 1/4 of resolution
26. #define AMPHIGH 15  //Amplitude testing start point
27. #define WAVERES 32  //Resolution of sine wave (how many points in sine wave)
28. 
29. /* Function Declarations */
30. 
31. bool kbhit(void);  //Wait for key press function
32. void packetcreation(char res[])[6], int from, int to);  //Packet creator function
33. void wavecreation(int elements, int res[elements], int amfact, int phasedeg);  //Sinewave creator function
34. 
35. /* Main Code */
36. int main()
37. {
38. /* Variable Declarations */
39. int speed=SPEED;
40. int bytes=BYTES;
41. int amplitude=AMPLITUDE;
42. int rps=RPS;
43. int rpsa=RPS;
44. int steps=STEPS;
45. int phase=PHASE;
46. int waveres=WAVERES;
47. int waveres=waveres/4
48. int amplow=AMPLOW;
49. int amplish=AMPHIGH;
50. char packets[2000][6]=0;  //Empty matrix for storing speed packets
51. char zeros[10]=0;  //Empty array for sending zero values, simpler than using timing functions
52. int wave[waveres];  //Array for holding waveform variations
53. int h;  //SPI channel reference
54. int x;  //Used in loops
55. int y;  //Used in loops
int c = '0'; // Used to reset keypress function

/* Calling Packet Creation Function, create all possible speed packets */
packetcreation(packets,0,2000);
printf("Packets Created Successfully \n");

/* Calling Wave Form Creation Function, create achievable sine waves */
packetcreation(packets,0,2000);

/* Testing GPIO initialised correctly */
if(gpioInitialise() < 0) {
    printf("GPIO not initialised correctly, program exiting \n");
    return 1;
}
printf("GPIO initialised correctly \n\n");

/* Open SPI with type name 'h' */
h = spiOpen(0, speed, 3); /* channel, baud, flags */

/* Starting Motor - Arming Sequence 1 */
while(!kbhit())
    spiwrit(h, packets[500], bytes); /* handle, buf, count */

/* Starting Motor - Arming Sequence 2 */
while(kbhit()) c=getchar(); /* Reset keyboard hit flag */
while(kbhit())
    spiwrit(h, packets[0], bytes); /* handle, buf, count */

/* Starting Motor - Waiting for user ready to start motor
   * Motor should now be in armed state */
while(kbhit())
    spiwrit(h, packets[0], bytes); /* handle, buf, count */

/* Amplitude Testing Area - Will run amplitude testing of sine wave from
   * amplitude low to amplitude high value. Remember the amplitude is multiplied

if (amplow < 1) { // Ensure amplitude lower value cannot be negative
    amplow = 1;
    printf("Warning: amplow was below zero, set to zero \n");
}
if (amphigh*4 > rps) { // Ensure amplitude higher value cannot force waveform below 0 speed
    amphigh = rps/4 - 1;
    printf("Warning: amphigh would force speed below zero, set to acceptable value \n");
}

wavecreation(waveres,wave,amplitude,phase);
printf("Waveforms Created Successfully \n");

while(kbhit())
    spiwrit(h, packets[0], bytes); /* handle, buf, count */

while(kbhit())
    spiwrit(h, packets[0], bytes); /* handle, buf, count */

while(kbhit())
    spiwrit(h, packets[0], bytes); /* handle, buf, count */

#define KEYBOARD_HIT 1
* by 1/4 of the resolution as dealing with integers */

print("When ready to start phase testing press any key \n\n");

while (kbhit())
{
    spiWrite(h, packets[rps], bytes); /* handle, buf, count */
}

printf("When ready to continue with each phase test press any key \n\n");

while (kbhit())
{
    spiWrite(h, zeros, 8);  //The amount of zeros can be changed for shorter/longer delay between packets
}

while (kbhit())
{
    if (kbhit())
    {
        //Outer loop, steps between waveform values
        rpsa=rps+wave[x];
        for (y=0; y<steps; y++)
        {
            spiWrite(h, packets[rpsa], bytes); /*SPI Write function: handle, buf, count */
        }
    }

    while (kbhit())
    {
        /* Test Finished %d \n",c);*/
    }
}

spiClose(h);  //Closes the SPI connections

gpioTerminate();  //Terminates PIGPIO control over IO module

return 0;

/*Check For Key Press Function
 * Note: Original Author unknown, refer to link for source

bool kbhit(void)
{
    struct termios original;
    tcgetattr(STDIN_FILENO, &original);

    struct termios term;
    memcpy(&term, &original, sizeof(term));

    term.c_lflag &= ~ICANON;
    tcsetattr(STDIN_FILENO, TCSANOW, &term);

    int characters_buffered = 0;
    ioctl(STDIN_FILENO, FIONREAD, &characters_buffered);

    tcsetattr(STDIN_FILENO, TCSANOW, &original);

    bool pressed = (characters_buffered != 0);

    return pressed;
}

/*Packet Creation Function*/
/* Conversion to normal bit structur
from DShot structure
e from DShot structure

void packetcreation(char res[], int from, int to)
{
  int i; //Iteration variable
  int j; //Iteration variable
  unsigned short int speedoptions[2000] = {0}; //Array for holding DShot packets (Not for transfer)
  char teleonoff = 0; //Telemetry on/off bit, set to 0
  short int crc = 0; //Variable for CRC
  short int crc_data = 0; //Variable for CRC Creation
  unsigned short int speedholder = 0;
  unsigned long long int fullpacketholder = 0;
  char packetspeeds[2000][6] = {0};

  for(i=from; i<to; i=i+1) {
    speedoptions[i] = i+48; //Creates speed values aligned with array ref (Control words 48 to 2047)
    if (i==1999) { //Error was occuring with 2000 speed value, this clear the error
      teleonoff = teleonoff;
    }
    speedoptions[i] = speedoptions[i] << 1 | teleonoff; //Adds telemetry request bit to end of message
  }
}

/* CRC Creation Loop, adapted from Cleanflight Code */
for(i=from; i<to; i=i+1) {
  crc=0;
  crc_data=speedoptions[i];
  for (j=0; j<3; j=j+1) {
    crc ^= crc_data;
    crc_data >>= 4;
  }
  crc &= 0xf;
  speedoptions[i] = (speedoptions[i] << 4) | crc;
}

/* Conversion to normal bit structure from DShot structure
1. * Grabs bits one at a time, converts to 100 or 110 as required */
for(i=from; i<to; i=i+1) {
  speedholder=speedoptions[i];
  for (j=0; j<16; j=j+1) {
    if (32768 & speedholder) {
      fullpacketholder |= 6;
    } else {
      fullpacketholder |= 4;
    }
    if (j<15) {
      fullpacketholder <<= 3;
    }
    speedholder <<= 1;
  }
  /*Separates entire message into byte sizes and allocates to byte matrix*/
  for (j=5; j>1; j=j-1) {
    packetspeeds[i][j]=fullpacketholder & 255;
    fullpacketholder >>= 8;
  }
  speedholder=0;
  fullpacketholder=0;

  /* Assigns each byte to matrix provided by user */
  for (i=0; i<6; i++) {
    res[i] = packetspeeds[i][j];
  }
  // For printing out each packet to check
// printf("value: %d %d %d %d %d %d %d
", i, packetspeeds[i][0], packetspeeds[i][1], packetspeeds[i][2], packetspeeds[i][3], packetspeeds[i][4], packetspeeds[i][5]);

/* Sine Wave Creation Function */

void wavecreation(int elements, int res[elements], int amfact, int phasedeg)
{

    /* Variables */
    int x;
    double elementsdouble = elements;
    double xdouble;
    double sinal;
    double phase;

    /* Wave creation */
    Values are created as doubles, then converted to integers
    so can relate to integer steps in speed */
    for(x=1; x<elements+1; x=x+1) {
        xdouble=x;
        xdouble=xdouble/elementsdouble; // Convert to ratio of number of steps
        phase=phasedeg*M_PI/180; // Convert phase to radians
        sinal=sin(xdouble*2*M_PI+phase)*elements/4*amfact; // Multiply by sin value and amplitude
        sinal=round(sinal); // Round to achievable integer (noting speed values are all int)
        res[x-1]=sinal;
    }
Appendix D. LabVIEW Code for Testing Platform