Cape Peninsula
University of Technology

DEVELOPMENT AND STABILIZATION OF AN UNMANNED VERTICAL TAKEOFF AND LANDING TECHNOLOGY DEMONSTRATOR PLATFORM

by

CYPRIAN OGONNA ONOCHIE

Thesis submitted in fulfilment of the requirements for the degree Master of Engineering: Mechanical Engineering

in the Faculty of Engineering

at the Cape Peninsula University of Technology

Supervisor: Prof Oscar Philander
Co-supervisor: Mr Mornay Riddles

Bellville
February 2017

DECLARATION

I, **Cyprian Ogonna Onochie**, declare that the contents of this thesis represent my own unaided work, and that the thesis has not previously been submitted for academic examination towards any qualification. Furthermore, it represents my own opinions and not necessarily those of the Cape Peninsula University of Technology.

Cyprian Onochie	26/02/2017
Signed	**Date**

ABSTRACT

Small and micro unmanned aerial vehicles (UAV) are rapidly becoming viable platforms for surveillance, aerial photography, firefighting and even package delivery. While these UAVs that are of the rotorcraft type require little to no extra infrastructure for their deployment, they are typically saddled with short ranges and endurance, thus placing a restriction on their usage. On the other hand, UAVs that are of fixed wing type generally have longer range and endurance but often require a runway for take-off and landing which places a restriction on their usage.

This project focuses on the development of a vertical take-off and landing (VTOL) UAV demonstrator suitable for integration on a small or mini flying wing UAV (a fixed wing UAV) to counteract the take-off and landing limitations of fixed wing type UAVs.

This thesis first presents a propulsion characterisation experiment designed to determine the thrust and moment properties of a select set of propulsion system components. The results of the characterisation experiment identified that the propulsion set of a Turnigy C6374 – 200 brushless out runner electric motor driving a 22 x 10 inch three bladed propeller will provide approximately 79N (8kg) of thrust at 80% throttle (4250rpm). Therefore, two of these propulsion set would satisfy the platform requirement of 12kg maximum take-off mass (MTOM).

The result of the abovementioned experiment, together with the VTOL platform requirements were then used as considerations for the selection of the suitable VTOL method and consequently the design of the propulsion configuration. Following a comparison of VTOL methods, the tilt-rotor is identified as the most suitable VTOL method and a variable speed twin prop concept as the optimal propulsion configuration.

As system stabilisation is an integral part of a VTOL UAVs control strategies, this work goes on to propose a stability augmented control algorithm for the developed propulsion configuration. This control algorithm provides for the control command allocation, attitude correction, and system propulsion dynamics; and for simulation purposes, the rigid body dynamics of the vehicle.

Part of this work presents a prototype control hardware and software for the VTOL platform. The hardware uses a dedicated microcontroller for decoding command input therefore freeing up computational resources on the main processor. The control hardware also features an inertia measurement unit of 3-axis accelerometers, 3-axis angular rate sensors, 3-axis magnetometer and a pressure sensor. These are used for attitude estimation. The main controller reads the sensor data and estimates attitude, adds the response of the estimated attitude to the control command and outputs the control signals to the propulsion hardware.

Finally, the control algorithm is validated by a nonlinear MATHWorks Simulink simulation using models for the control command allocation, attitude correction, system propulsion dynamics and the rigid body dynamics of the developed VTOL platform. This validation was executed using five nonlinear simulations that observe the altitude, altitude rate, roll, pitch and yaw-rate responses to destabilizations conducted for 5 seconds each. The result of the simulation reveals the controller’s ability to stabilize the platform.

ACKNOWLEDGEMENTS

I wish to thank God for providing me with His grace throughout the duration of this study:

In no particular order, I wish to thank the following:

- My supervisor, Prof Dr O. Philander for providing me with the opportunity to execute this study under the auspices of the Adaptronics AMTL research centre at the Faculty of Engineering, Department of Mechanical Engineering at the Cape Peninsula University of Technology.
- Mr E. Erfort and Mr M. Riddles for the mentorship they provided and continue to provide.
- Paul and Diana Onochie for their support and encouragement.
- Ngozi Khanya Mafa for providing the fire that brought this to an end.

TABLE OF CONTENTS

GLOSSARY

Terms and Abbreviations	Definition or Explanation
AMTL	Advanced Manufacturing Technology Laboratory
AUAV	Autonomous Unmanned Aerial Vehicles
BCS	Body Coordinate System
C	System state measurement function
CG	Centre of Gravity
CPUT	Cape Peninsula University of Technology
DOF	Degrees of Freedom
e	Error signal
E-NED	Earth North-East-Down
e_z	Prop tilt eccentricity
F_i	Force acting on CG along the i_{th} axis with respect to the BCS
FPGA	Field-programmable gate array
I_{ii}	Moment of inertias
IMU	Inertial Measurement Unit
K_c	Controller gain
k_{col}	Collective command factor
k_{lat}	Lateral command factor
k_{long}	Longitudinal command factor
K_m	Moment coefficient
k_p	Proportional gain
K_{ped}	Pedal command factor
K_t	Thrust coefficient
m	Mass
MARG	Magnetic, angular rate and gravity
MCU	Microcontroller Unit
M_i	Moment on CG about the i_{th} axis with respect to the BCS
MPU	Microprocessor Unit
MTOM	Maximum take-off mass
N	Rotational speed in rpm
PID	Proportional Integral Derivative
PPM	Pulse Position Modulation
PWM	Pulse Width Modulation
QTW	Quad Tilt Wing
RC	Radio Control
SACS	Stability augmented control systems
t	Time
T_D	Rate time
T_i	Integral time
UAV	Unmanned Aerial Vehicles
US DoD	United States Department of Defence
U_t	Input or motion command
VC-NED	Vehicle-carried North-East-Down

Terms and Abbreviations	**Definition or Explanation**
v_i	Translational velocity of the UAV about the i_{th} axis
VTOL	Vertical Take-off and Landing
X_{est}	State estimate
X_{t-1}	Prior state
Z_t	Actual measurement
α and β	Tilt angles
δ_{col}	Collective control command
δ_{lat}	Lateral control command
δ_{long}	Longitudinal control command
δ_{ped}	Pedal control command
ε_x	State error or noise estimation
θ	Pitch angle
ϕ	Roll angle
ψ	Yaw angle
ω_i	Rotational velocity of the UAV about the i_{th} axis

LIST OF FIGURES

LIST OF TABLES

.

CHAPTER ONE: INTRODUCTION

1.1 Introduction

CPUT’s Advanced Manufacturing Technology Laboratory (AMTL) has the development of Unmanned Aerial Platforms (the Guardian UAV platform) as one of its research interests. The development of Unmanned Aerial Vehicle’s (UAVs) was initiated to provide a platform to test and improve its (AMTL’s) aero morphing technology. These vehicles, which are small or mini fixed wing type UAVs have after their inception, evolved primarily into unmanned platforms for aerial surveillance.

The need for a long runway for the landing of a small UAV designed to glide, impairs its implementation and application as a runway may not be readily available in all instances. This impairment creates the need for the development of Vertical Take-off and Landing (VTOL) propulsion system UAVs. A VTOL (or a STOL) platform when incorporated to a fixed wing UAV will provide the aircraft with the ability to be launched or deployed and retrieved from any location.

The development of UAVs by CPUT’s AMTL was initiated to provide a platform to test and improve its (AMTL’s) aero morphing technology. A significant criterion for this development of a VTOL demonstrator is it’s suitability for integration on a flying wing aircraft. The motivation for this criterion stems from the discussion that the novelty of aero morphing technology is reduced when validated on a fixed wing aircraft configuration. This is because it can be argued that hinging the wing or part of it will achieve a slightly dissimilar effect to aero morphing (Riddles, 2011). Hence, the blended wing of a flying wing aircraft is a perfect test bench for aero morphing technology.

Given the above reason, the Guardian UAV platform is projected to evolve into a flying wing UAV. This project therefore focuses on the development of a VTOL UAV technology demonstrator to be integrated to current and future Guardian UAVs.

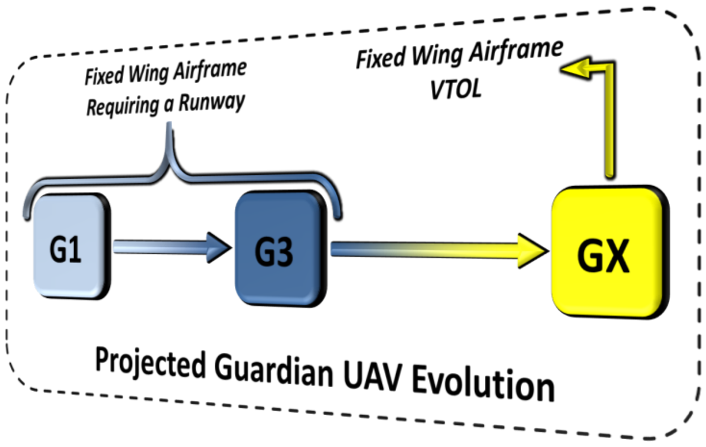


Figure 1.1: Projected Guardian UAV evolution

1.1.1 The Guardian UAV Platform

The Guardian in its current iteration (G2), is a small fixed wing UAV with a wingspan of 3.6m, a length of 2.1m and a maximum take-off mass of 12kg. It is an aerial surveillance system deployable in applications ranging from civil to military operations and can be remotely piloted or autonomously operated.

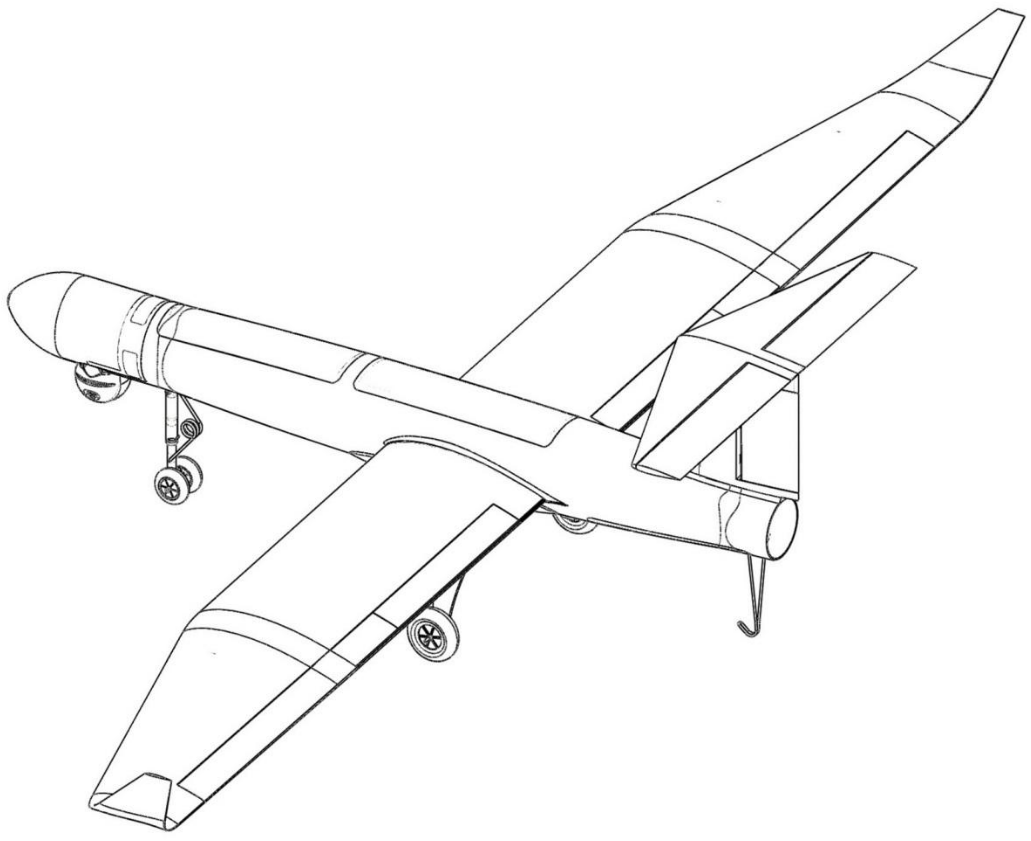


Figure 1.2: The Guardian II airframe

1.1.2 Problem Statement

The deployment of UAVs of the fixed-wing configuration type is greatly restricted by the necessity of a runway for take-off and landing (United States Department of Defence (US DoD), April 6, 2009). This restriction is intensified when the vehicle is designed to glide, as it then requires a longer runway for landing (Riddles, 2011).

This project focuses on the development of a VTOL UAV demonstrator suitable for integration on current (small fixed wing UAV) and future (flying wing) Guardian UAVs. This VTOL system will counteract the take-off and landing limitations of the UAVs developed by the AMTL of CPUT. This development is anticipated to be realised by taking into account the deliberations for achieving a smooth transition between vertical flight and horizontal flight stage of the future VTOL UAV's operation.

1.2 Objectives

The primary objective of this research is the development of a vertical take-off and landing platform to be integrated with current and future Guardian UAVs.

The secondary objectives essential to achieving the primary objective are:

1. To develope a suitable propulsion configuration and setup
2. To identify a prototype stabilization electronics system
3. The design of a mechanism to control and stabilize the VTOL system
4. The validation of the developed VTOL platform

1.3 Design Requirements

As the VTOL platform under development is intended for implementation on the guardian UAV, it is necessary that the specifications of this VTOL system meet that of the Guardian UAV. The applicable design requirements as developed from the Guardian UAV specification are listed below;

1. MTOM of 12kg
2. Ability to hover
3. Manoeuvrable in 4 degrees of freedom (DOF), i.e., X, Y, Z and yaw.
4. Stabilization in all 6 DOF
5. Low cost
6. Safe operation

The maximum take-off mass of 12kg is set in conformity to the specification of the Guardian UAV. However, specifications and requirements such as its endurance and autonomous control, together with operational characteristics such as its ascent and descent rate, maximum and loiter speeds, etc. are not applicable (to this project) as the research and development involved in meeting these requirements and specifications are beyond the scope of the objectives set for this project.

1.4 Methodology

1.4.1 Literature Review on VTOL UAV Systems and Control Technologies

The related study of the literature consists of the study of existing fixed wing VTOL UAV platforms and other VTOL system developments in recent years. It furthermore includes the research and study of existing rotorcrafts and VTOL platforms that are focused on the dynamics of the propulsion system implemented in these vehicles as well as the method utilized for transition between vertical and horizontal flight. This study also includes the control of autonomous systems applicable to small UAVs, attitude estimation or modelling techniques and sensor fusion algorithms.

1.4.2 Determine Design Requirements

The applicable platform design requirements are based on the specifications of the target (the Guardian UAV platform) and the objectives of the project.

1.4.3 Propulsion Configuration Development

Developing the propulsion configuration entails a design analysis of VTOL propulsion systems to select the best match to the platform requirements. It also includes selecting the system components for the identified propulsion configuration and specifying the placement of selected components.

1.4.4 Propulsion Configuration Characterization

This phase of the project takes account of the design of an experiment for propulsion system characterization; this experiment provides the thrust and torque generated by the selected propulsion system. In addition, the energy consumption as well as other operating conditions was noted for further analysis.

Further activities in this development phase include; the design, manufacture and procurement of the test equipment and the characterization experimentation.

1.4.5 Control and Stabilization System Development

The control and stabilization system development includes the design of the control algorithm and development of a prototype embedded system hardware architecture for the real-time control and stabilization of the system. The optimal embedded hardware architecture development and the system components were determined with a design decision analysis considering the following factors or characteristics:

1. Availability and cost of the system and system components
2. Suitability
3. Expandability
4. Ease of Manufacture

1.4.6 Validate VTOL Platform

Platform validation consists of the design of a simulation model in MATLAB Simulink using the mathematical relations that define the rigid body dynamics, control command allocation, the proposed attitude correction scheme and the propulsion system dynamics of the developed platform. This will be used to demonstrate the platform stability while in all modes of manoeuvrability.

1.5 Literature Review

The first powered unmanned aerial vehicle (The Ruston Proctor Aerial Target) was developed in 1916 as a means to provide a pilotless aircraft to be used against the Zeppelin (Taylor, et al., 1977 p. 28). All other subsequent UAV developments followed the direction of developing the UAVs as missile delivery drones or decoy drones until the early 1950s when the US Air Force Logistics Command's quick-reaction aircraft modification program began to modify aircrafts to airborne reconnaissance platforms (Ehrhard, 2010 p. 6). To date, UAVs have been used in numerous areas and for various situations.

All across the wide range of UAV deployment, the modern UAV is envisaged to possess air power characteristics such as; penetration, perspective, reach, technology, versatility and concurrent operations (Australian Defence Force, 2002), (Royal Air Force Centre for Air Power Studies, 2009).

Autonomous unmanned aerial vehicles (AUAV) are aircraft that are capable of carrying out flight missions without an on-board pilot. Such aircraft are equipped with on-board control systems consisting of a flight control computer, communication systems; navigation, inertial measurement and peripheral sensor sub systems.

UAVs are categorized by their size, cruising altitude and endurance. Small or mini UAVs, as defined by US Department of Defence (US DoD), are those with a total mass (payload and fuel included) of not more than 25kg (April 6, 2009). These are deployed for commercial aerial surveillance (Marcus UAV Corp, 2010), geomagnetic survey (Universal Wing, 2009), maritime surveillance (University of New South Wales, 2008), reconnaissance and search and rescue (airforce-technology.com, 2011).

The most common and practical types of mini UAVs are fixed wing and rotorcraft type. Fixed wing UAVs are used for their longer range and endurance while rotorcraft UAVs are deployed in situations requiring vertical take-off and hovering manoeuvres (United States Department of Defense (US DoD), April 6, 2009). The history of the development of Aircraft and UAVs has shown fixed wing aircraft configuration provides the best trade-off of simplicity of flight and air power. Equipping aircraft or UAVs of such configuration with a VTOL capability will, as has been shown, further enhance its air power (Bell Helicopter Textron Inc., 2011); (Hirschberg, 1999).

Launching systems for fixed wing UAVs as used for the catapult-launched British armed forces Phoenix UAV (Ministry of Defence, UK, 2008) and the hand launched Marcus UAV Corp Zephyr UAV (Marcus UAV Corp., 2010) obviously do not require a runway for take-off. Such systems, however, are not categorized as VTOL systems and as such will not be included in this literature survey.

1.5.1 Fixed Wing VTOL Methods

Although UAVs are by definition pilotless aircraft, they share the same characteristics (size, aerodynamic properties, operating conditions, etc.) as their manned counterparts. In the same manner, the technologies and principles implemented in manned aircraft are used for UAVs; and for mini UAVs, these technologies are scaled down with little adjustments (in parts like the propulsion and navigation system) to suit the size of the UAV.

1.5.1.1 The tilt-rotor

The most popular VTOL method in use is the tilt-rotor (Wilson, 2011). A tilt rotor aircraft utilizes two or more propellers with their axis of rotation on the vertical for vertical lift, after which the rotors/propellers are gradually tilted forward till the axis of rotation of the rotors becomes parallel to the craft’s longitudinal axis for a forward flight. The propellers, when rotating on the vertical axis provide vertical lift and the vehicle behaves as a rotorcraft. The propellers provide forward thrust when on the “horizontal” axis, while aerodynamic lift provided by the airflow over the craft’s wing provides vertical lift. This VTOL method is also referred to as a tilt-jet when a jet stream is used for propulsion.

Bell Helicopter demonstrated in 1998 the Eagle Eye tilt rotor UAV (Figure 1.3) powered by a turbo shaft engine driving two prop-rotors (three bladed) mounted on the gearbox spindles of rotor-gearbox located in the pylons at the tip of each wing. Bell Helicopter employs a clockwise rotating prop-rotor on the left wingtip and an anti-clockwise rotating prop-rotor on the right wingtip. The counter rotating prop-rotors cancel out the turning moment about the yaw axis produced by each propeller and manoeuvrability while hovering is attained by the utilization of variable pitch prop-rotors (Bell Helicopter Textron INC, 2005). The short wing of the Eagle Eye is designed to accommodate the wing-tip mounted prop-rotors; however, the reduction in wing area creates more power and speed demand in-flight than a longer wing of similar profile. Also, hazards occurring from the wing tip mounted prop-rotors will be greater than that of inboard mounted propellers.

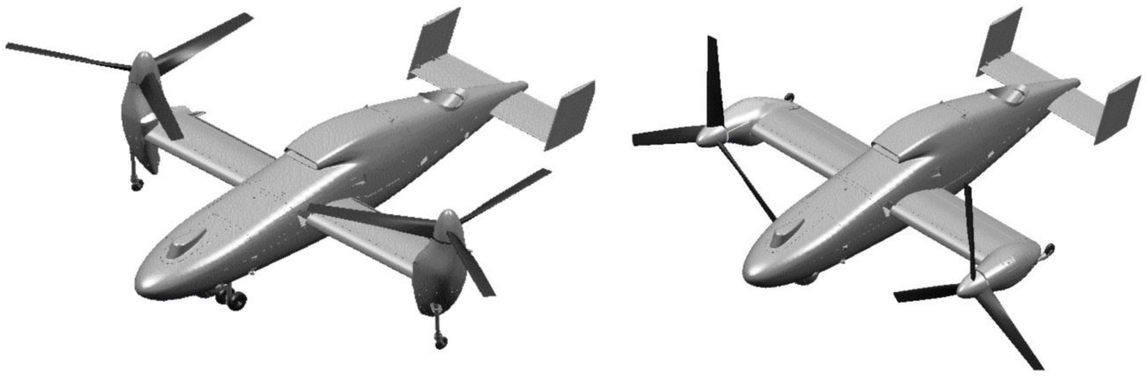


Figure 1.3: Bell Eagle Eye tilt-rotor UAV
In helicopter mode and airplane mode
(Bell Helicopter Textron INC, 2005)

The electrically driven Mini Panther tilt rotor UAV (Figure 1.4) of Israel Aerospace Industries Ltd (2002) uses three propellers each driven by an electric motor for propulsion. Two tiltable counter rotating propellers are mounted inboard, one at each wing; the third propeller is aft mounted with its axis fixed, parallel to the yaw axis of the UAV. Varying the speeds of all the electric motors as well as slight tilting of the two tiltable motors is used for hovering control and stabilisation (Israel Aerospace Industries Ltd, 2002). The aft mounted propeller is a dead weight in forward flight, as it does not provide propulsion. In addition, the bare rear propeller area increases the aerodynamic drag at the rear of the aircraft.

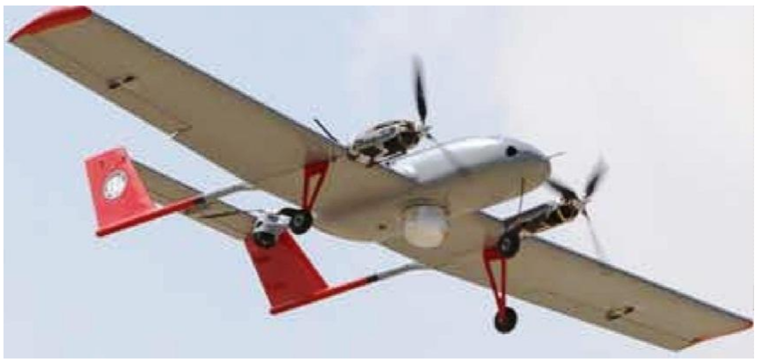


Figure 1.4: Mini Panther UAV
(Israel Aerospace Industries Ltd, 2002)

The tilt-rotor aircraft is a popular choice for achieving VTOL, based on the fact that it is relatively easy (compared to other methods) to develop, as the effect of tilting the rotors can be visualized and established while still in the design phase, consequently eliminating the extensive testing needed in the stabilization of other VTOL type aircrafts (Bell Helicopter Textron Inc., 2011).

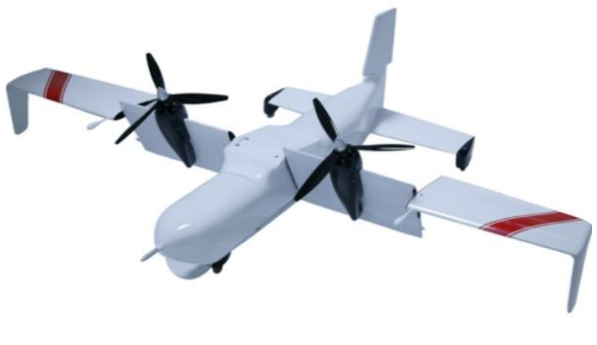
1.5.1.2 The tilt-wing

A tilt wing VTOL aircraft, as the name implies, has the wing or some part of the wing (inclusive of the propellers or rotors) set vertical for vertical take-off and landing. The wing and the propulsion system are turned to the longitudinal for forward flight. The tilting mechanism of this VTOL method will require more strength as compared to that of the tilt rotor; and this is because of the added weight of the wing that has to be tilted.

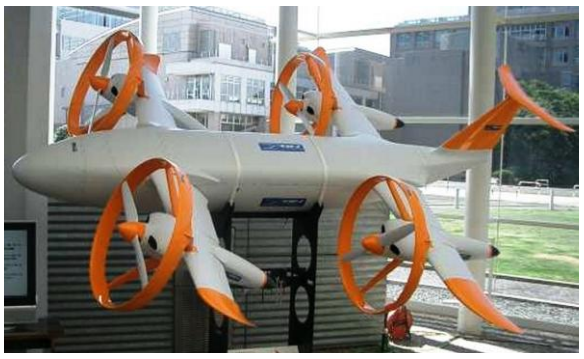
Known operational tilt-wing UAVs as of March 2011 are the Acuity Technologies AT-10 VTOL UAS (shown below in Figure 1.5 (a)) and the QTW (Quad Tilt Wing) UAV (Figure 1.5 (b)) developed by Chiba University and GH Craft Ltd (American Institute of Aeronautics and Astronautics, Inc., 2011).

The QTW UAV is powered by four electric motors driving two sets of counter-rotating propellers located in board on each wing of the tandem wing configuration (GH Craft, 2009). Hovering manoeuvrability is obtained by varying the speeds of the electric motors. The impact of the low energy density limitation of available electric batteries is observed in the relatively low flight endurance (as compared to other drive systems) of this UAV.

AT-10 UAS of Acuity Technologies Inc., (2011) has part of its wings tilt-able; two counter rotating propellers driven by an electric motor are mounted on the tilt-able part of the wings. Clark (2011) states that battery and fuel cell technologies are not currently adequate to support long endurance UAS missions, but he expects this to change in the next 3 to 5 years. For the present, Clark (2011) proposes the use of a heavy fuel engine as a power generation source for the electric propulsion system. The development of this UAS is still in progress.



(a)



(b)

Figure 1.5: Tilt-wing UAVs
(Acuity Technologies Inc., 2011) and (GH Craft, 2009)

1.5.1.3 The tail-sitter and coleopter

Tail-sitters employ a propeller or ducted fan mounted at the nose, wing or tail of the UAV, these (the propeller or ducted fan) provide vertical lift while the aircraft sits on its tail. The prop-wash over the wing and fin mounted control surfaces are used to achieve control during vertical flight. These same control surfaces can be used to initiate the tumble required to transition to forward flight.

The ring tailed SiCX 300V UAV (Figure 1.6) powered by a 2 cycle IC engine driving an aft mounted shrouded fan is an imaging and surveillance UAV under development by Guided System Technologies (2009). Its vertical flight control and stability is obtained with the aerofoils in the ring tail only, while the control surfaces on the wing and tail are used for forward flight control (Guided System Technologies Inc, 2009).

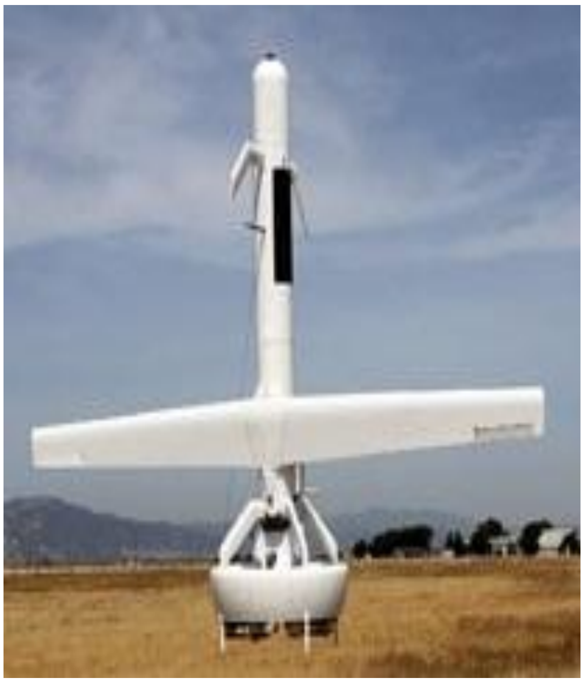


Figure 1.6: SiCX 300V Tail-siting UAVs
(Guided System Technologies Inc, 2009)

Coleopters employ a VTOL method which is a variation of the tail-sitter. A ducted fan in the main fuselage is used to provide vertical lift while the aircraft sits on its tail (Allen, 1965 p. 57), the ducts are then progressively reconfigured when the desired airspeed is attained to tilt the axis of the aircraft to the horizontal axis so the fan provides forward thrust while the aerodynamic lift provided by the airflow over its wing keeps it in the air.

1.5.1.4 **Lift fan method**

Lift fan VTOL aircraft use ducted fans for vertical lift, forward propulsion is provided by another propulsion source (propellers, jet or ducted fan) mostly aft mounted. The ducted fan is placed at the centre of gravity of the vehicle for a single lift fan system, while in a multi fan system; the fans are placed such that their collective thrust stabilizes the vehicle (Figure 1.7).

Lift fan VTOL UAVs so far do not employ wings for lift in forward flight mode; however, they feature control surfaces which are used for control and stabilization in vertical and forward flight modes (Aurora Flight Sciences Corporation, 2009), (Layton, 2007).

Layton (2007) proposed a blended wing lift fan UAV concept (shown in Figure 1.7) of three ducted lift fans; two are aft mounted while the third is fore mounted and an additional power plant for forward thrust. He proposes to maintain control of the vehicle in vertical flight mode by varying and vectoring the thrust produced by the fans while the aircraft will rely on its aerodynamic control surfaces for forward flight mode control and stabilization.

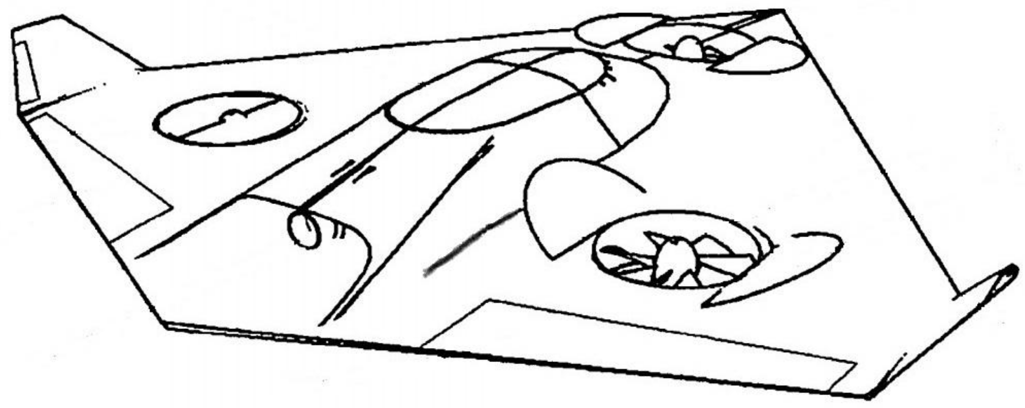


Figure 1.7: A lift fan VTOL UAV concept (Layton, 2007)

1.5.2 Control System

Accurate control of mini UAV systems enhances their air power characteristics (penetration, reach and versatility) and consequently plays an important role in their deployment. A precise estimation of the orientation and attitude of the vehicle needs to be provided to achieve this accurate control. The trend in autonomous UAV control is to utilize a stability augmented system for the control (Stone, et al., 1997), (Mellinger, et al., 2010), (Cordoba, 2007) and (Schafroth, et al., 2010) .

Stability augmented control systems (SAC System) consist of an inertial measurement unit (IMU) which consists of gyroscopes and accelerometers or a magnetic, angular rate and gravity (MARG) sensor which provides the attitude (orientation, angular and linear velocities and acceleration with respect to a reference coordinate system) and heading of the vehicle; a navigation system and a microprocessor to implement the sensor filtering and fusion scheme as well as perform the control calculations.

The wide range of commercially available control and sensor hardware (microprocessors, Micro-Electro-Mechanical System (MEMS) global positioning system (GPS) modules, gyroscopes, accelerometers, magnetometers, altimeter, etc.) provides a relatively small (as compared to available product range) amount of difference in the hardware performance. Therefore, any remarkable performance enhancement of the control system will have to be due to the optimizations on the software (sensor filtering, fusion and control algorithm) part of the development. The related literature presented below is focused on data filtering and sensor fusion technics.

1.5.2.1 Data filtering and sensor fusion

The Kalman Filter is the most widely used method of multi-sensor data fusion, integration and attitude estimation (Julier, et al., 1997). This filter is a recursive data processing algorithm that is optimal for linear control strategies. It estimates the mean and covariance for any given set of data through a direct application of Bayes' theorem (Kalman, 1960). The Kalman Filter is described by two linear equations, the state prediction and sensor prediction. The state prediction of a dynamic system at time *t* is given as:

$\boldsymbol{\bar{x}_t = A \cdot x_{t-1} + B \cdot u_t + \varepsilon_x}$ (Kalman, 1960 p. 40)

Equation 1.1

The state prediction $\bar{x}_t$ is the function of the prior state, x_{t-1} and the input or motion command, u_t. $\boldsymbol{A}$ and $\boldsymbol{B}$ are state transition and system state matrix, while ε_x, is the state error or noise estimation, which is Gaussian. The first assumption of the Kalman

Filter is that state prediction is a linear function of the prior state and the command is that of which the physics is (or will be) defined by the matrixes $\boldsymbol{A}$ and $\boldsymbol{B}$. Another assumption is that it has a Gaussian distribution in its noise.

The second linear equation, which is the sensor prediction at time, *t*, is given as:
$\bar{\boldsymbol{z}}_t = \boldsymbol{C} \cdot \bar{\boldsymbol{x}}_t + \boldsymbol{\varepsilon}_z$ (Kalman, 1960 p. 40)

Equation 1.2

The sensor prediction, $\bar{\boldsymbol{z}}_t$ contains the sensor measurement at time *t*, the measurement function, $\boldsymbol{C}$ relates the measurement to the system state and $\boldsymbol{\varepsilon}_z$ is the measurement noise, which is also Gaussian. The whole idea of the Kalman Filter can be summarised as;
$\boldsymbol{x}_{\mathrm{est}} = \bar{\boldsymbol{x}}_t + \boldsymbol{K}(\mathrm{z}_{\mathrm{t}} - \bar{\boldsymbol{z}}_t)$ (Kalman, 1960 p. 41)

Equation 1.3

This is that the state estimate, $\boldsymbol{x}_{\mathrm{est}}$ is a linear function of the predicted state, $\bar{\boldsymbol{x}}_t$ and the product of the difference between the actual measurement, z_{t} and the estimated measurement, $\bar{\boldsymbol{z}}_t$ and the Kalman gain *K*.

Fabiani, et al., (2007) provides an implementation of the Kalman filter in sensor fusion. They provided a state estimate using a non-stationary embedded Kalman filter, based on the relations between acceleration, speed and position using the estimates of attitude angles based on an IMU including accelerometers, gyroscopes and a magnetometer. The state estimation is completed by the determination of the inertial speed and position using the data given by a GPS and an altimeter. The system state and transition (with resulting rigid body dynamics) feedback are parameterized with Euler angles in this implementation.

Another approach in parameterizing the system state and transition is by using unit quaternions (Guerrero-Castellanos, et al., 2011) and consequently eliminating from the implementation the singularities that occur (gimbal lock) as pitch or roll angle approaches $\pm 90^{o}$ in the Euler angles parameterization. Guerrero-Castellanos, et al., concludes that this results in a controller suitable for an embedded application where low computational resources are available.

Kalman-centred sensor fusion algorithm requires large state vectors to linearize the dynamics and kinematics of the system state in three dimensions. This impedes its suitability for implementation on a system with limited computational resources.

Sung (2003) presents a fuzzy, logic-based algorithm and Mahony, et al., (2008) presents nonlinear complimentary filters for sensor fusion. Both algorithms are reasoned to address the high computational requirements of the Kalman Filter implementation.

Another sensor data fusion and attitude estimation technique presented by Madgwick, et al., (2011) uses a quaternion representation, thus allowing accelerometer and magnetometer data to be used in an analytically derived and optimised gradient-descent algorithm to compute the direction of the gyroscope measurement error as a quaternion derivative. They demonstrated the algorithm to be computationally inexpensive, effective at low sampling rates and contain a maximum of two adjustable parameters defined by observable system characteristics. Madgwick, et al., benchmarked their algorithm against propriety Kalman-based algorithm for orientation sensing, obtaining results that indicate the filter achieves levels of accuracy exceeding that of the Kalman-based algorithm. They conclude that the implications of the low computational load and ability to operate at low sampling rates open new opportunities for the use of IMU and MARG sensor arrays in real-time applications of limited power or processing resources.

Stability augmented control implementation is not limited to the use of a processor, microprocessor or a microcontroller in performing sensor fusion, attitude estimation or control calculations. Cordoba (2007) presents a SACS of three MEMS rate-gyros and three magneto-resistive transducers that send its outputs to a field-programmable gate array (FPGA) digital neural network in which a strategy was developed for attitude estimation to replace the Kalman Filter in the integration of MEMS IMU inertial sensors and Magneto resistive sensors signals. Cordoba concludes that the FPGA digital neural network strategy overcomes the adaptability limitations of the Kalman Filter implementation in non-linear systems.

1.5.2.2 **System Controllers**

The various types of controllers in use in the industry can generally be divided into two major groups: conventional (PID, Otto-Smith, etc.) and unconventional (fuzzy and neuro-fuzzy) controllers. A major characteristic of conventional controllers is that the mathematical model of the process must be known in order to implement the controller, whereas unconventional controllers employ methods to the controller strategy in which the mathematical model of a process are not known. VTOL aircraft stabilization processes are nonlinear and so is confounding to completely define mathematically. However, it is known that these (VTOL aircraft stabilization) processes can be controlled using PID controllers providing that controller parameters are properly tuned (Bang, et al., 2003), (Yoo, et al., 2010).

PID controllers consist of three basic modes: the Proportional mode (*P*), the Integral mode (*I*) and the Derivative mode (*D*). This controller is generally implemented as *P*, *I*, *PD*, *PI* and *PID* algorithm, the choice of the modes to implement hinge on the process characteristics (Westphal, 2001 p. 97).

The mathematical representation of a time domain proportional integral derivative algorithm is,

$$\boldsymbol{mv(t) = k_c\left[e(t) + \frac{1}{T_i}\int e(t)dt + T_D\frac{de(t)}{dt}\right]} \text{ (Westphal, 2001 p. 97)}$$

Equation 1.4

This representation is the standard PID realisation. The tuneable controller parameters are; the controller gain ($\boldsymbol{k_c}$) which is applied on all three terms of the algorithm, the integral time ($\boldsymbol{T_i}$) and the rate time ($\boldsymbol{T_D}$).

The proportional term of the algorithm ($\boldsymbol{k_c \cdot e(t)}$) adjusts the output signal in direct proportion to the controller input which is the error signal (*e*). The error signal is the difference between a reference signal (the set-point) and the system output (process variable).

A proportional controller reduces error but will not return the process variable to the set-point as it responds only to a change in error. The integral term ($\boldsymbol{k_c \cdot \frac{1}{T_i}\int e(t)dt}$) corrects the offset that may exist between the set-point and the process variable, while the derivative term ($\boldsymbol{k_c \cdot T_D\frac{de(t)}{dt}}$) increases the dynamic response of the process to the error signal.

The major demerit of this realisation is that a sudden change in set-point, which will lead to a change in the error, will cause the derivative term to become very large and thus provide a "derivative kick" which is undesirable for the output signal. An alternative realisation to mitigate this demerit is the parallel form of the algorithm, which is represented mathematically as:

$$\boldsymbol{mv(t) = k_p e(t) + k_i \int e(t)dt + k_D \frac{de(t)}{dt}}$$ (Ghosh, 2013 p. 302)

Equation 1.5

In this realisation, the proportional gain ($\boldsymbol{k_p}$) acts on the error, while the integral and derivative gain acts on the integral and derivative terms respectively.

The proportional and derivative terms can also be based on the process variable rather than on the error. This is particularly suitable for systems with a constant (fixed) set-point as the changes in the process variable will be the same as the changes in error.

1.5.3 Rigid Body Modelling and Dynamics

1.5.3.1 Coordinate Systems and Transformation

The equations of motion governing the rigid body dynamics presented here defines the dynamics of the body coordinate system (BCS) or body frame with respect to a vehicle-carried North-East-Down coordinate system (VC-NED). The BCS is vehicle-carried and fixed on the aircraft body. Its origin and axes as shown in Figure 1.8 is defined as (Stevens, et al., 1992 pp. 61-63);

1. The origin $\mathbf{O_B}$ is located on the centre of gravity (CG) of the UAV.
2. The X-axis $\mathbf{X_B}$ (roll axis) points forward lying in the symmetric plane of the UAV.
3. The Y-axis $\mathbf{Y_B}$ (pitch axis) is points towards the right side of the vehicle.
4. The Z-axis $\mathbf{Z_B}$ (yaw axis) points downwards to comply with the right hand rule

The vehicle carried NED has its origin $\mathbf{O_N}$ also on the CG of the UAV, and as the name implies its X-axis $\mathbf{X_N}$ is pointing north, the Y-axis $\mathbf{Y_N}$ pointing east and the Z-axis $\mathbf{Z_N}$ normal to the northeast plane pointing downward.

Euler angles yaw, pitch and roll are used to define the orientation of the rigid body. The transformation between the vehicle carried NED coordinate system and the BCS is achieved by the 3–2–1 Euler angles rotation sequence. These Euler angles also shown in Figure 1.8 are defined as (Stevens, et al., 1992 p. 64);

1. Pitch angle $\boldsymbol{\theta}$ is the angle from the BCS Z-axis to the vehicle-carried NED Z-axis projected on the X-Z plane of the vehicle-carried NED frame of the UAV.
2. Roll angle $\boldsymbol{\phi}$ is the angle from the BCS Y-axis to the vehicle-carried NED Y-axis projected on the Y-Z plane of the vehicle-carried NED frame of the UAV.
3. Yaw angle $\boldsymbol{\psi}$ is the angle from the BCS X-axis to the vehicle-carried NED X-axis projected on the X-Y plane of the vehicle-carried NED frame of the UAV.

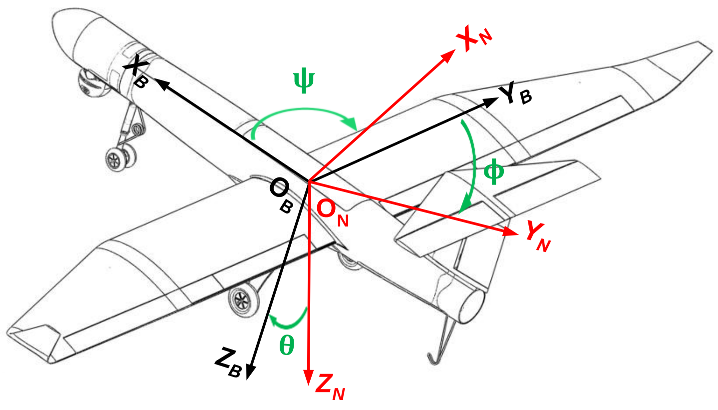


Figure 1.8: Coordinate systems

1.5.3.2 Rigid body dynamics

Small or mini UAVs are taken as rigid bodies, which are free to translate and rotate in three-dimensional space, and as such possess 6 degrees of freedom (6 DOF). The rigid body dynamics governing these 6 DOF derived from Newton's laws expressed as differential equations as defined by Stevens, et al (1992 p. 81) are:

<u>Force Equations</u>

$$\dot{v}_x = \omega_z \cdot v_y - \omega_y \cdot v_z - g \sin\theta + F_x/m$$
$$\dot{v}_y = -\omega_z \cdot v_x + \omega_x \cdot v_z + g \sin\phi \cos\theta + F_y/m$$
$$\dot{v}_z = \omega_y \cdot v_x - \omega_x \cdot v_y + g \cos\phi \cos\theta - F_z/m$$

Equation 1.6

<u>Moment Equations</u>

$$\dot{\omega}_x = \left[\left((I_{yy} - I_{zz})\omega_y \cdot \omega_z\right) + M_x\right]/I_{xx}$$
$$\dot{\omega}_y = \left[\left((I_{zz} - I_{xx})\omega_z \cdot \omega_x\right) + M_y\right]/I_{yy}$$
$$\dot{\omega}_z = \left[\left((I_{xx} - I_{yy})\omega_x \cdot \omega_y\right) + M_z\right]/I_{zz}$$

Equation 1.7

In these equations, (m) is the mass of the vehicle, F_x, F_y, F_z, M_x, M_y and M_z are the components of the accelerating forces and moments acting on the CG with respect to the BCS. v_x, v_y, v_z, ω_x, ω_y and ω_z are the translational and rotational velocities of the UAV. I_{xx}, I_{yy} and I_{zz} are the moment of inertias of the UAV.

To make this equation collection closed and resolvable, they are supplemented with the kinematic equations (Equation 1.8 and Equation 1.9) also expressed as differentials which define the transformation of the UAV's attitude θ, ϕ and ψ which are the pitch, roll, yaw angles and position x, y and z (Stevens, et al., 1992 p. 81).

Kinematics Equations of Rotation

$$\dot{\phi} = \omega_x + \omega_y \sin\phi \tan\theta + \omega_z \cos\phi \tan\theta$$
$$\dot{\theta} = \omega_y \cos\phi - \omega_z \sin\phi$$
$$\dot{\psi} = \omega_z \cos\phi \sec\theta + \omega_y \sin\phi \sec\theta$$

Equation 1.8

Kinematics Equations of Translation

$$\dot{x} = v_x + v_y \sin\phi \tan\theta + v_z \cos\phi \tan\theta$$
$$\dot{y} = v_y \cos\phi - v_z \sin\phi$$
$$\dot{z} = v_z \cos\phi \sec\theta + v_y \sin\phi \sec\theta$$

Equation 1.9

The kinematic equations of rotation define the rotational transformation of the UAV from the BCS to VC-NED, while the kinematic equation of translation defines the position of the vehicle with reference to an earth North-East-Down (E-NED) coordinate system. The E-NED has its origin arbitrarily placed on the earth surface, while its axis are oriented in the same manner as the axis of VC-NED defined in section 1.5.3.1 (Coordinate Systems and Transformation).

1.6 Relation of Work to Literature Survey

This project covers the development of a small UAV technology demonstrator VTOL. This development is to be integrated into AMTL's Guardian fixed-wing surveillance UAV as discussed in section 1.1.1. A major limitation to the deployment of this type (fixed-wing) of UAV as stated previously is the availability of a runway for take-off and landing. This limitation provides the motivation for this project. The first section of the related literature (section 1.5.1) explores the work done on similar platforms and presents existing options for VTOL implementation on UAVs and manned aircraft. Bell Helicopter Textron Inc. (2011) however asserts that the tilt-rotor is a preferred VTOL method due to its relative ease (compared to other methods) of development.

The stability of VTOL systems increases the UAV's controllability and consequently makes it highly deployable. Therefore the complete development of VTOL systems entails the development of a control subsystem to achieve this stability. Section 1.5.2 discussed the research conducted by Cordoba, (2007), Fabiani, et al., (2007), Stone, et al., (1997), Guerrero-Castellanos, et al., (2011), Madgwick, et al., (2011), Mellinger, et al., (2010), and Schafroth, et al., (2010). The focus of the works of the above authors is the development of the right suit of hardware; and the execution of the appropriate routine or algorithm to collect and fuse data from a suit of sensors in order to provide the required correction to the control signal to achieve stability in robotics systems. Expanding on the works of the authors above, this work will present a stability embedded control algorithm optimised for the developed VTOL UAV system.

The literature in section 1.5.3 defines the terms, system coordinates and mathematical relations of the dynamics, which are needed for the mathematical validation of the proposed VTOL method and control algorithm.

This work aims to initialise a successful VTOL platform that will serve as a framework for further development of small VTOL UAV systems. The developed platform will bring VTOL capabilities to AMTL's Guardian UAV and will greatly improve its ability to be deployed.

1.7 Scope of Thesis

The research and development conducted in the course of this project is presented in five chapters. The first chapter begins with the introduction and background as discussed above. The objectives and the methodology follows, then ending with a review of the related literature.

The second chapter presents the considerations and the selection of the VTOL method suitable for implementation on a fixed wing mini UAV. The chapter also presents the various propulsion configuration concepts (based on the selected VTOL method) with their control strategies and discussions of the iterative development that lead to the final propulsion configuration.

Chapter 3 covers the development of a stability augmented control system for a mini UAV. And the fourth chapter shows a low cost control electronics hardware prototype, highlighting the considerations and optimisations implemented in the development of the embedded attitude estimation and correction software.

The Fifth chapter presents the validation of the developments using a Simulink model of the motion dynamics of the VTOL platform. This is used to validate the control strategy of the developed platform and to demonstrate the stability of the selected VTOL configuration.

In the conclusions, a summary of the work presented in this thesis and recommendations for improvement are given. The dissertation concludes with a list of references and appendix.

CHAPTER TWO: PROPULSION CONFIGURATION

2.1 Introduction

The development of a propulsion configuration for a small UAV VTOL system requires the availability of operational characteristics for propulsion systems matching the design requirements. This set of data, however, is not readily available as it needs to be determined for every combination of power plant and propeller. Hence, the first section of this chapter contains the details of a propulsion characterisation experiment.

The following section presents considerations behind the selection process of the VTOL method with emphasis on the suitability (of the method) for implementation on current and future Guardian UAVs.

The last two sections present the various possible and suitable (implementable on current and future Guardian UAVs) propulsion configurations based on the selected VTOL method; the discussions and the iterative design development that led to the final propulsion configuration.

2.2 Propulsion Characterisation Experiment

This experiment logs the thrust and torque provided by any combination of power plant and propeller throughout their rotational speed range. To perform this experiment, a test-bench that allows for the measurement of thrust and torque with a force gauge and a static torque transducer was designed and manufactured.

In addition to the measurement of thrust and torque at each rotational speed point, the experiment measures the power consumed by the system and also executes a real-time monitor of the temperature conditions of the propulsion components.

A restriction on the characterisation of two propulsion system sets was placed on this activity due to the cost associated with procuring and executing this experiment on the wide combination of propulsion components possible.

Therefore, the characterisation was executed on two set of propulsion system based on a Turnigy C6374 – 200 brushless out runner electric motor driving a 22 x 10 inch three bladed propeller and later a 20 x 10 inch three bladed propeller.

2.2.1 Characterisation Equipment and Experiment Setup

Figure 2.1 shows a solid model render of the thrust torque test-bench. The test bench has the propulsion unit in a housing (**3**) fixed on a bed (**2**). Also fixed on the bed is a bracket (**5**) to which the static torque transducer (**4**) is mounted.

The four linear bearing unit (**7**) mounted on each corner of the bed allows it to slide on two shafts (**8**). These shafts are clamped to a bench (**1**) with four clamps (**9**) on each corner of the bench.

The force gauge (**6**) is mounted onto the bench and is connected to the sliding bed through the torque transducer bracket with a stud (**16**) with a left and right thread on both sides (see Figure 2.2).

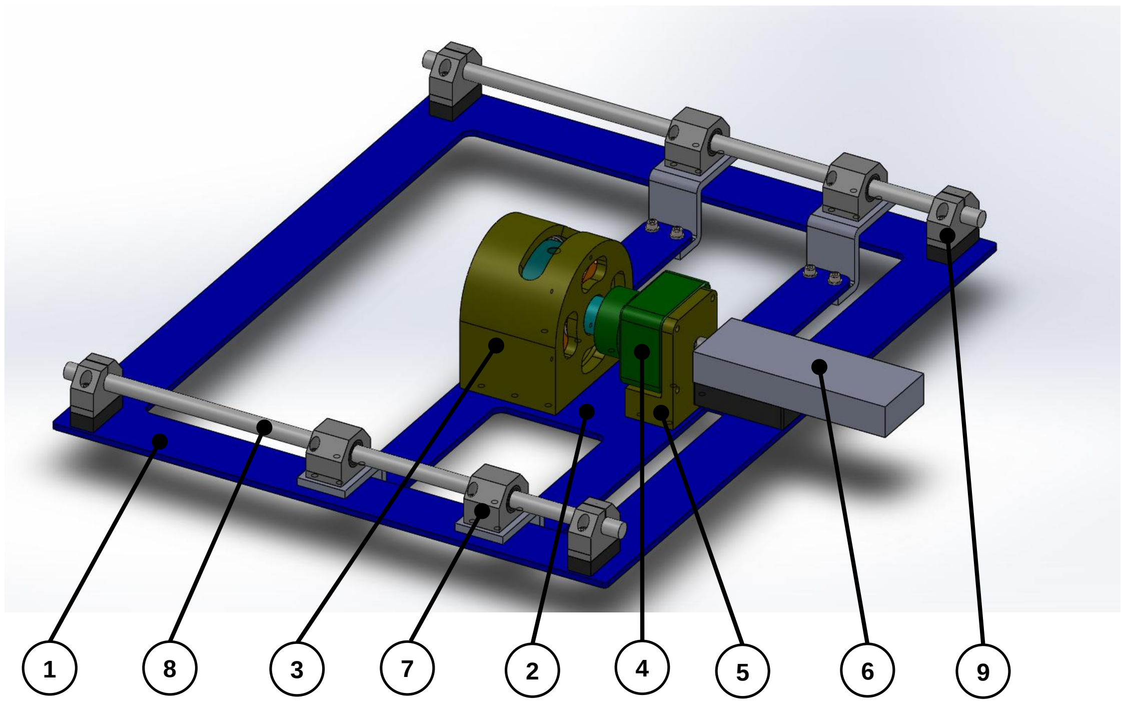


Figure 2.1: Thrust torque test-bench – Solid model render

The setup of the system is such that the pull/push generated by the propulsion system is transferred to a slide motion to the bed which will then pull/push on the force gauge through the stud.

To eliminate parasitic loads from the equipment, the test bench is designed such that the axis of the propulsion system is collinear with the axis of the force gauge and pull/push stud. These axes are also designed to be coplanar to the slide plane (plane formed by the axis of the two slide shafts). However, due to the slide resistance of the linear bearing system, a measurement calibration is needed.

The propulsion housing is designed such that the propulsion unit is mounted onto a thrust plate (**13**). This plate will transfer any push from the propulsion system to the housing through a thrust bearing (**14**) as shown in Figure 2.2. The plate is bolted onto a sleeve (**10**) hence any pull from the propulsion unit is transferred from the plate to the sleeve through a needle roller bearing (**11**) and an internal retaining ring (**12**) to the propulsion housing.

The combination of the relative motion due to the needle bearing between the housing and the sleeve and the thrust bearing between housing and the thrust plate allows the moment generated by the propulsion system to be transferred onto the static torque transducer by the socket (**15**) connected to the other end of the thrust plate.

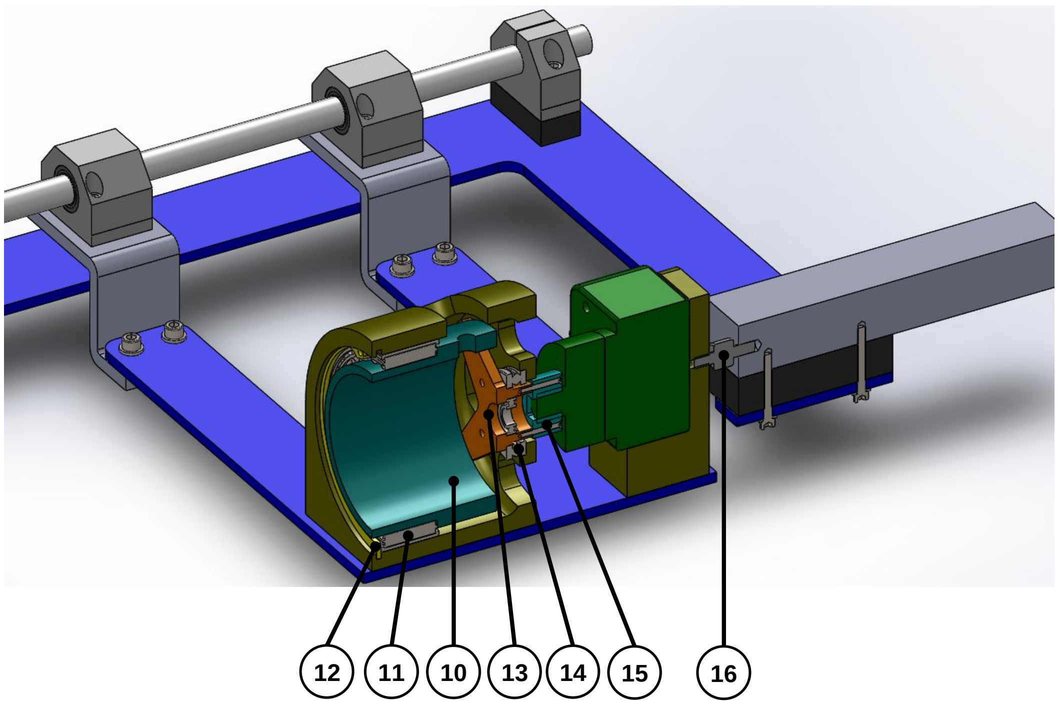


Figure 2.2: The propulsion housing unit

The remaining part of the experiment setup as shown in Figure 2.3 includes;

1. Two 5 cell 5000mAh lithium polymer batteries (**17**) connected in series to power the propulsion system.
2. A remote data unit (**18**) for measuring voltage, current, three external temperature sensor and rotational speed using a phase tachometer.
3. A control and logging unit (**20**) that logs the measurement onto a data storage device. This device also allows the scripting of the pulse width drive required for the speed control.
4. The electronic speed controller (**19**) which receives the throttle position as a pulse width drive from the control and logging unit.
5. The propulsion system mounted into the housing.
6. And other electrical connections that allows for the remote shutoff of the experiment.

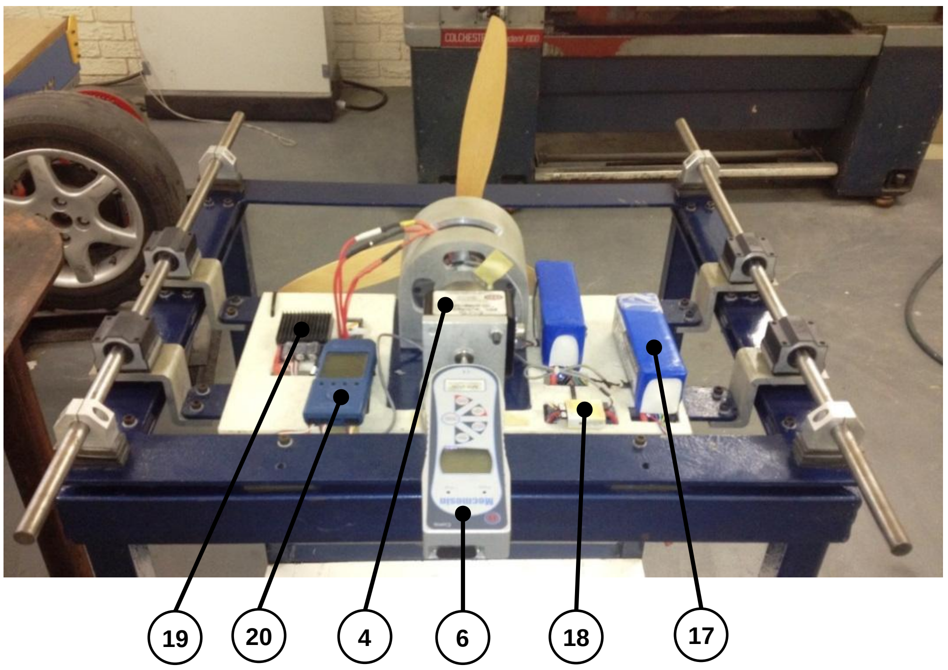


Figure 2.3: Thrust torque test-bench

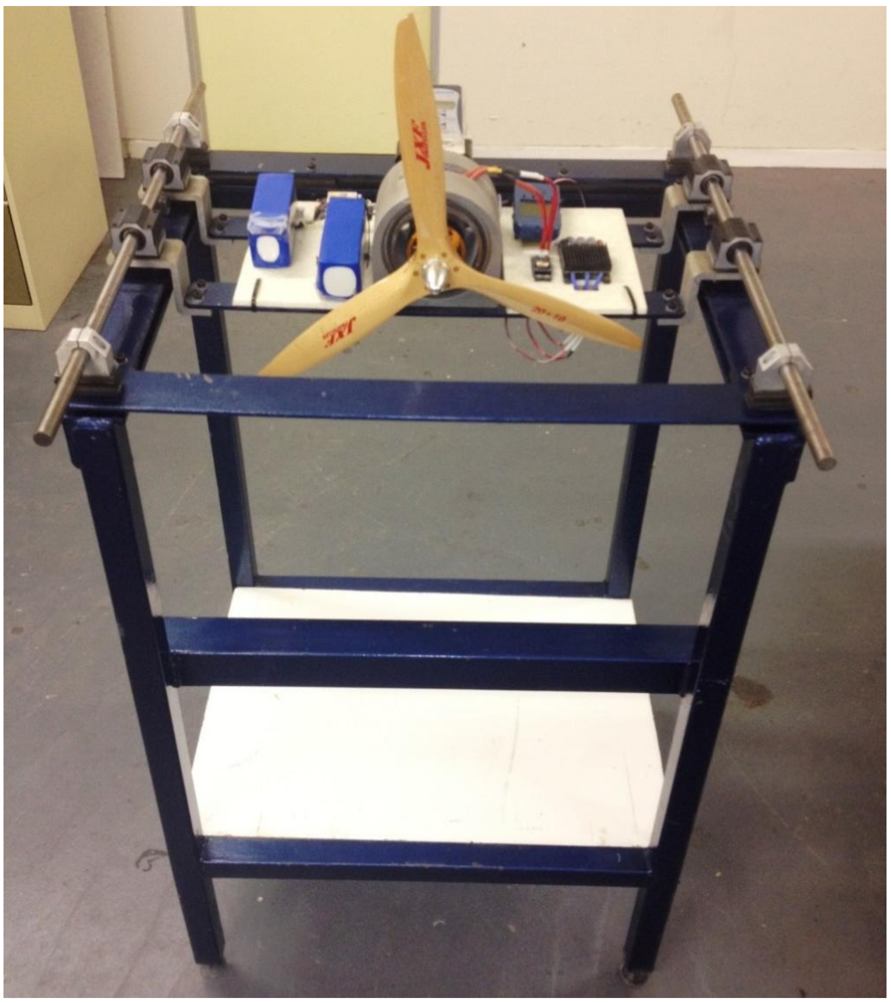


Figure 2.4: Characterisation experiment setup

The test equipment is designed to measure up to 200N of thrust and 150Nm of torque from the propulsion system. The propulsion system can be in the form of an internal combustion engine or an electric motor of up to a 7.5kW rated power and a propeller of up to 27 inches in diameter. As previously stated, the first procedure in the experiment is to calibrate the measurement setup of the system. This is achieved by applying a series of known loads onto the system in the manner of the propulsion unit and recording the corresponding gauge reading. This loading is applied over the range of expected experimental measurements.

The result of this calibration can be seen in Appendix A1. From the calibration data, the relationship between and actual load and the gauge reading in Newton is:

$$\boldsymbol{Actual\ Load = (1.0834 \times gauge\ reading) + 2.1611}$$

Equation 2.1

The experimental data was imported to MathWorks MATLAB where the measurements were adjusted with the calibration results. Appendix A2 and A4 consist of the propulsion characterisation results while the details and outputs of the data analysis executed on the data set is recorded in Appendix A3 and A5. Table 2.1 is a summary of the outputs of the analysis on the characterisation results.

Table 2.1: Summary of the characterisation result

No	Description	Experiment 1 : Turnigy C6374-200 & 22 inch Prop			
		y_{Thrust}	R^2	y_{Moment}	R^2
1	Fit curve to plot of Load vs RPM	(4.5496e-06x^2) + (-0.001187x) + (1.9501)	0.99933	(1.7492e-07x^2) + (-0.00015829x) + (0.089176)	0.99922
2	Fit curve to plot of Load vs Square of RPM	(4.3595e-06x) + (0.48518)	0.99928	(1.4987-07x) + (-0.10855)	0.99846
3	Adjust fit to intercept at (0,0)	(4.3854e-06x)	0.99928	(1.4422e-07x)	0.99846
No	**Description**	**Experiment 2 : Turnigy C6374-200 & 20 inch Prop**			
		y_{Thrust}	R^2	y_{Moment}	R^2
1	Fit curve to plot of Load vs RPM	(3.8137e-06x^2) + (-0.0040205x) + (3.6957)	0.99938	(1.2769e-07x^2) + (-0.00012548x) + (0.072132)	0.99876
2	Fit curve to plot of Load vs Square of RPM	(3.218e-06x) + (-1.64)	0.99845	(1.0942e-07x) + (-0.098077)	0.99801
3	Adjust fit to intercept at (0,0)	(3.1433e-06x)	0.99845	(1.0508e-07x)	0.99801

For each experiment, the coefficient of the equation of the linear curve fitted to the plot of load vs the square of the rotational speed is the coefficient of thrust and moment for the thrust and moment curve respectively.

Hence for the Turnigy C6374-200 & 22 inch propeller propulsion system, given a rotational speed (**N** *[rpm]*), the thrust (**F** *[N]*) and moment (**M** *[Nm]*) developed by the system can be approximated with:

$$F = 4.3854^{-6} \times N^2 \qquad \& \qquad M = 1.4422^{-7} \times N^2$$

Equation 2.2

2.3 VTOL Method Comparison

The Guardian UAV platform is projected to evolve into a flying wing UAV; therefore the focus of this project is to develop a VTOL UAV technology demonstrator to be integrated onto current and future Guardian UAVs. The design analysis of the VTOL methods is thus based on each method's suitability for implementation on a fixed wing and a flying wing UAV configuration.

Each VTOL method is analysed as to its operation and merits. Possible limitations when used on the Guardian platform are explored, and a rating is assigned based on suitability. A rating of 10 means the method is highly suitable while 1 is not suitable.

This analysis and ratings applied due to it are not universal, but based on the judgment of the author, the requirements of the platform under development and the authors interpretation of the workings of each VTOL method as discussed in section 1.5.1. Table 2.2 shows the outcome of this analysis.

2.3.1 The Tilt–wing

The appeal of the tilt–wing VTOL method is based on the simplicity and practicality of having the propulsion on a fully or partially tilt-able wing. The tilt of the wings can be used to provide control in the direction of flight and the prop wash over the control surfaces on the wing ensures there is always some amount of controllability in both vertical and horizontal flight.

However, its suitability for use on a flying wing UAV is greatly reduced by the requirement to tilt all or a considerable part of the wing. Therefor a rating of 6 is assigned to this method.

2.3.2 The Tilt–rotor

The main deterrent against the implementation of the tilt-rotor VTOL method is the complexity of the mechanism involved. The issue of complexity, however, is greatly outweighed by the flexibility of the placement of the tilting components (the rotors) as they can be mounted at the wing tip, inboard, before the leading edge or after the trailing edge of the wing, can even be implemented on an aft mounted rotor. The complexity factor is further rendered mute by the relative ease (compared to other methods) of visualizing the effect of tilting the rotors which facilitates the platform stabilization. Given these considerations, a rating of 9 is given to this VTOL method as there are no likely specific limitations in its implementation on a flying wing UAV.

2.3.3 The Tail Sitter and Coleopter

Tail sitters have operational advantage over the tilt-rotor and the tilt-wing method as they require less mechanical complexity than both methods. It however requires that the propulsion and control components (the propellers and control surfaces) be placed in such way that the effect of the slipstream velocity over the control surfaces due to the prop wash, is enough to maintain control while hovering. The degree of control of this VTOL method while in transition from hover to forward flight (and vice versa), depends mostly on the slipstream velocity distribution over the control surfaces, the size of the control surfaces and position of the control surfaces relative to the vehicles Centre of Gravity (CG).

These control requirements are the main demerit of implementing this method (tail sitter) on a flying wing. The coleopter is also not implementable as a flying wing aircraft has no definite fuselage. Finally, the apparent difficulties involved with stabilizing the transition phase makes both methods impractical to investigate. Both VTOL methods are awarded a rating of 5.

2.3.4 Lift Fan/Jet Method

The main demerit of this method is the redundancy of the propulsion components (fans) not used for forward flight. The effects of using fans to provide lift in horizontal flight mode is not documented as this is a rarely utilised VTOL method. An effect of significant interest is the possible aerodynamic demerits of having some part of the wing open at high speeds. This VTOL method is awarded a rating of 4 due to the redundant transmission components required to implement it.

2.3.5 Method Selection

The discussions above show the tilt-rotor as the most suitable VTOL method for a small fixed wing and flying wing UAV.

Table 2.2: VTOL methods design evaluation result

VTOL Method	Suitability Rating
Tilt-wing	6
Tilt-rotor	9
Tail sitter and coleopter	5
Lift fan/jet	4

2.4 Propulsion Configuration Conceptualisation

The development of the propulsion configuration follows an evolving design approach. This approach starts with an initial concept providing the simplest structure of the predefined design feature (a tilt-rotor VTOL in this case), this concept is analysed and new design features are recognised. The emergent design features (new design features) are added to the design system to form a new design concept. The entire process is reiterated until concepts that provide the desirable set (for this system stability in all modes of manoeuvrability) of design features is actualised. These concepts are then analysed and the best option selected.

2.4.1 First Concept: The Fixed Prop Tri-copter

This concept (shown in Figure 2.5 and Figure 2.6) has two tilt-able propellers mounted fore of the wing; they can be located at the wing tip or mounted in board, a third larger propeller is located aft of the vehicle. All three propellers are powered independently. The front propellers rotate in the same direction while the rear rotates in the other direction. All propellers provide vertical lift when rotating with axis on or close to vertical, while to transition to forward flight, the propellers are gradually tilted in the direction shown in Figure 2.6. Forward thrust in forward flight can be provided by the combination of all propellers, the front propellers only or the rear propeller.

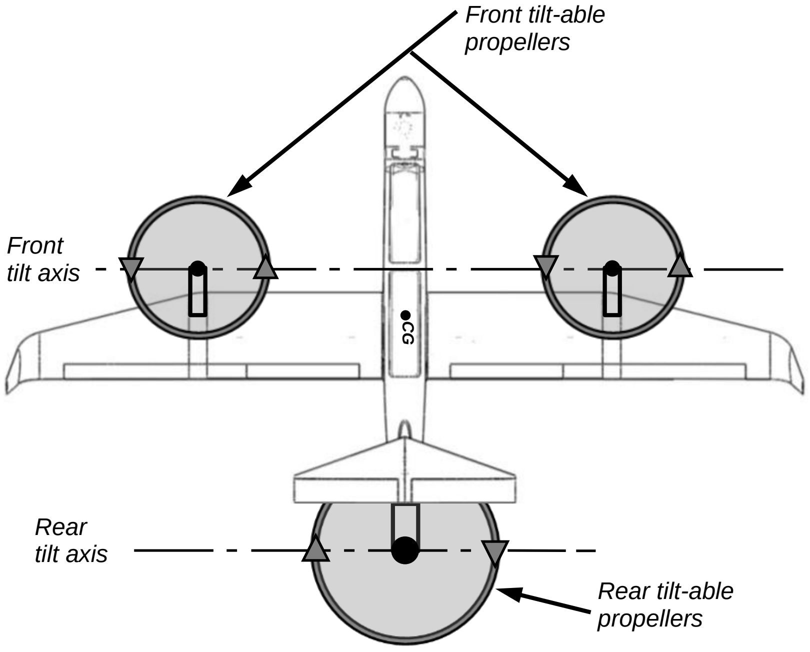


Figure 2.5: The fixed prop tri-copter concept (Superimposed on the guardian II airframe)

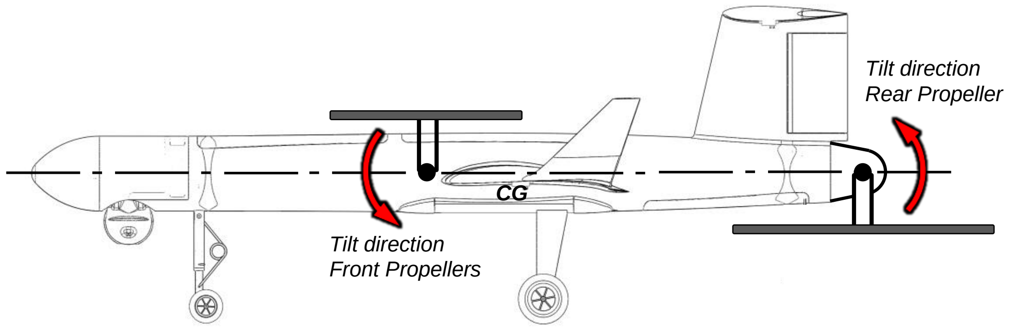
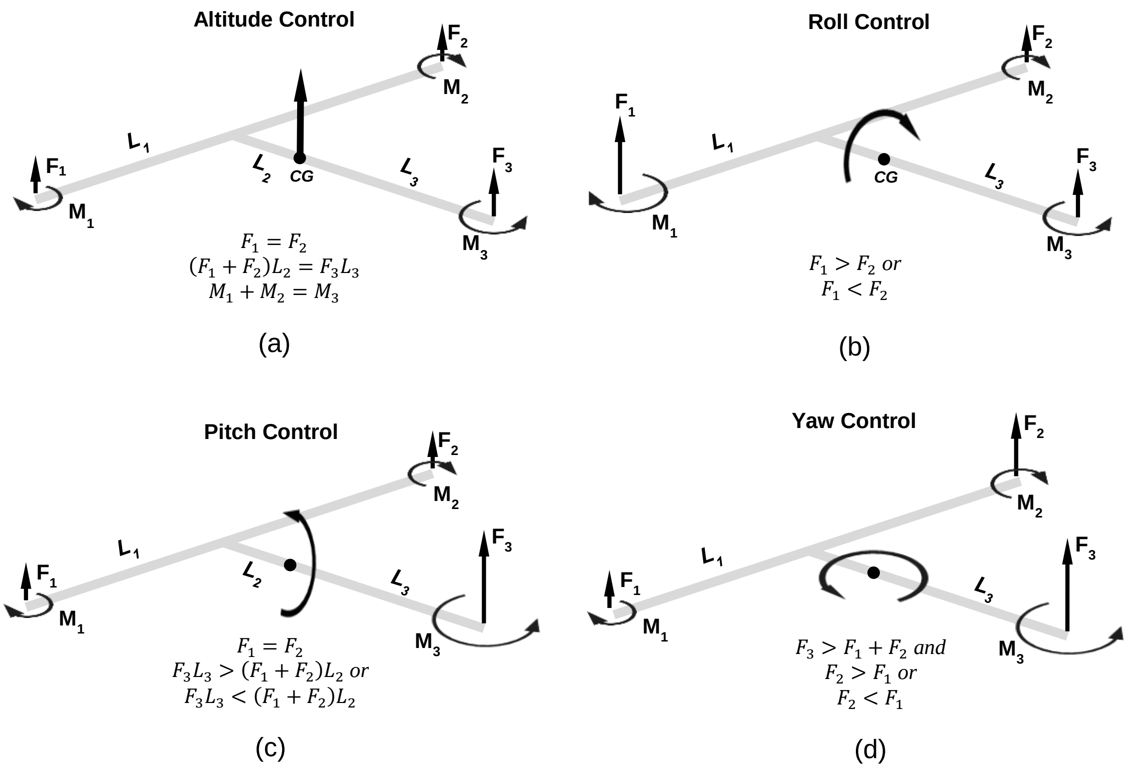


Figure 2.6: Propeller tilt directions

2.4.1.1 Control strategy

As translational motion on the X_B and Y_B axis of the platform is dependent on the pitch and roll angle respectively, the motion control of this concept is split into altitude, pitch, roll, and yaw control. Increasing the speed of all three propellers leads to an increase in altitude. A stable altitude control (Figure 2.7(a)) is attained by increasing or decreasing the speeds such that; the thrust developed by both front propellers are equal, there is no moment about the pitch axis and the collective torque of the front propellers is equal to that of the rear propeller. Roll control is attained by varying the speeds of the two front propellers. Figure 2.7(c) shows the pitch control generated by varying the speed of the rear propeller while keeping the front propeller speeds equal. Yaw control is achieved by the coupling of roll and pitch control.



$F_1 = F_2$
$(F_1 + F_2)L_2 = F_3L_3$
$M_1 + M_2 = M_3$

$F_1 > F_2$ or
$F_1 < F_2$

$F_1 = F_2$
$F_3L_3 > (F_1 + F_2)L_2$ or
$F_3L_3 < (F_1 + F_2)L_2$

$F_3 > F_1 + F_2$ and
$F_2 > F_1$ or
$F_2 < F_1$

Figure 2.7: Proposed attitude control for the first VTOL concept

Although all proposed control strategies are based on varying the thrust developed by the propellers, the author is aware that the propellers alongside developing thrust also develops a turning moment (yawing moment when the propeller axis is parallel to the yaw axis) about their axis of rotation (shown as M_i in Figure 2.7). These moments, however, were shown by initial propeller characterisation investigation results (APPENDIX A) to be insufficient in maintaining yaw control hence the use of coupled pitch and roll for attaining yaw control.

The main drawback of this concept is the absence of pure yaw control and consequently, possible instability about the yaw axis during hover and other manoeuvres. Another demerit of this concept is the twofold effect of the change in the rotational speed of a propeller. Although a change in rotational speed is necessary to achieve the desired variation of thrust needed to provide roll or pitch control, it will (while providing this control) introduce instability in the yaw axis. Given this consideration, it can be argued that pure and independent altitude, pitch, roll and yaw control is not attainable from this concept.

2.4.2 Second Concept: The Tilting Prop Tri-copter

The concept has also two tilt-able propellers mounted fore of the wing and a third propeller located aft of the vehicle. The front propellers are counter rotating cancelling out the moment developed by each other. All propellers like in the first concept provide vertical lift when rotating with axis on or close to vertical, and in order to obtain forward flight, the propellers are gradually tilted.

2.4.2.1 Control strategy

The motion control of this concept is also split into altitude control, pitch, roll and yaw. Pitch, roll and altitude control is attained in the same manner as the initial concept. Yaw control is attained by tilting the front propellers in opposite directions by the propeller tilt angle, α (see Figure 2.8). This is also used to maintain yaw stability while hovering as some amount of tilting from both front propellers is required to counter the yawing moment produced by the rear propeller. The tilting angle, α is of immense importance in the stability and control of this VTOL concept as it plays a part in all modes of control.

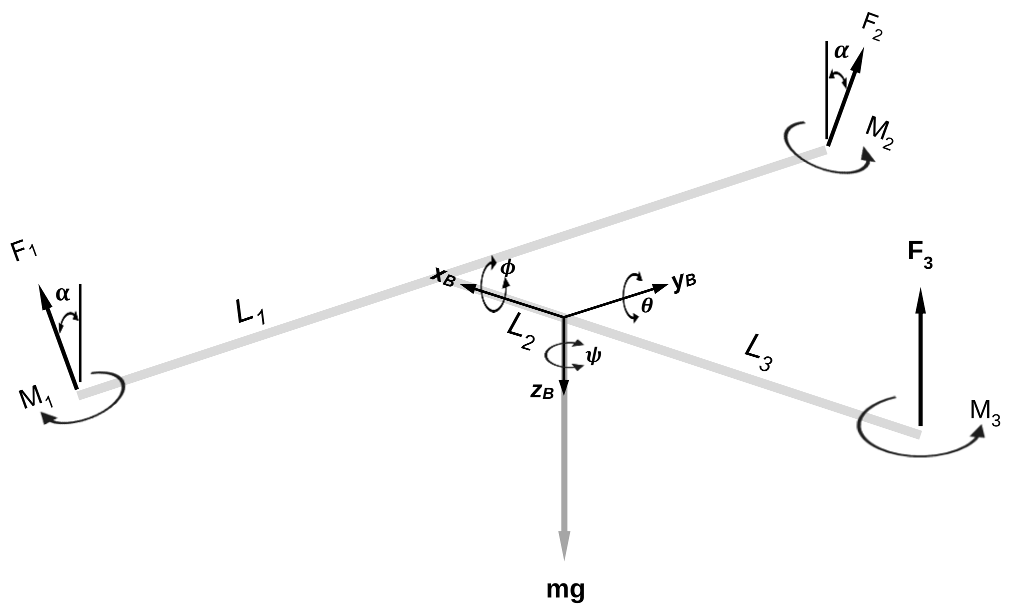


Figure 2.8: Free body diagram of the second VTOL concept

In the same manner as in the first concept, the propellers are independently powered. This brings about a high degree of simplicity to the propulsion system as the propeller can be fixed directly to the shaft's power plant. However, the extra weight added reduces the payload of the platform which is an important air power characteristic. Also, given that the vehicle is a surveillance UAV which by the nature of the on-board surveillance sensor payload is vibration intolerant, the choice of power plant for this small UAV (especially for the wing mounted propellers) is constrained to electric motors.

This concept, apart from its need for multiple power plants, has been shown to suffice in providing a VTOL platform with complete and independent manoeuvrability (Israel Aerospace Industries Ltd, 2002), (Yoo, et al., 2010).

2.4.3 Third Concept: The Twin Prop Tilt-rotor

In this propulsion concept, power from a single plant is provided to two variable pitch propellers placed at the tip of the wing. The propellers are counter rotating therefore cancelling out the turning moment about the yaw axis produced by each prop (Figure 2.9). Control is maintained by varying the pitch and tilt of the propellers.

A variation of this concept employs two non-variable pitch propellers located in the same manner as described above. The propellers, however, are driven separately. In this case, control of the UAV is maintained by varying the rotational speed of the propellers and tilting the propellers.

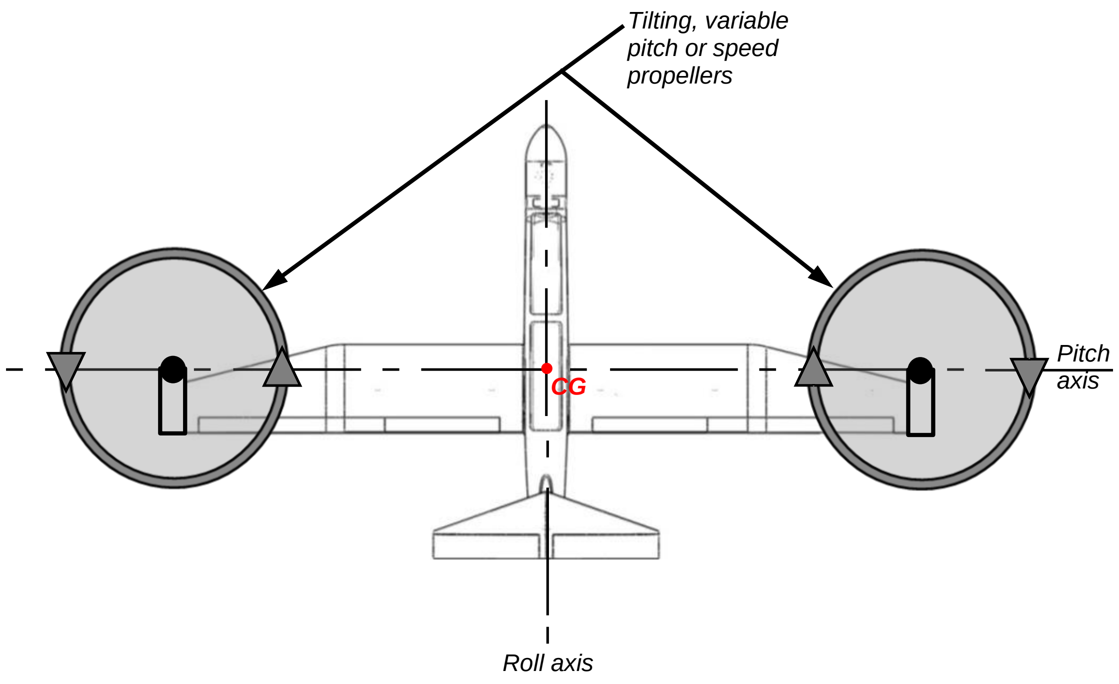


Figure 2.9: The twin prop tilt-rotor (superimposed on the guardian II airframe)

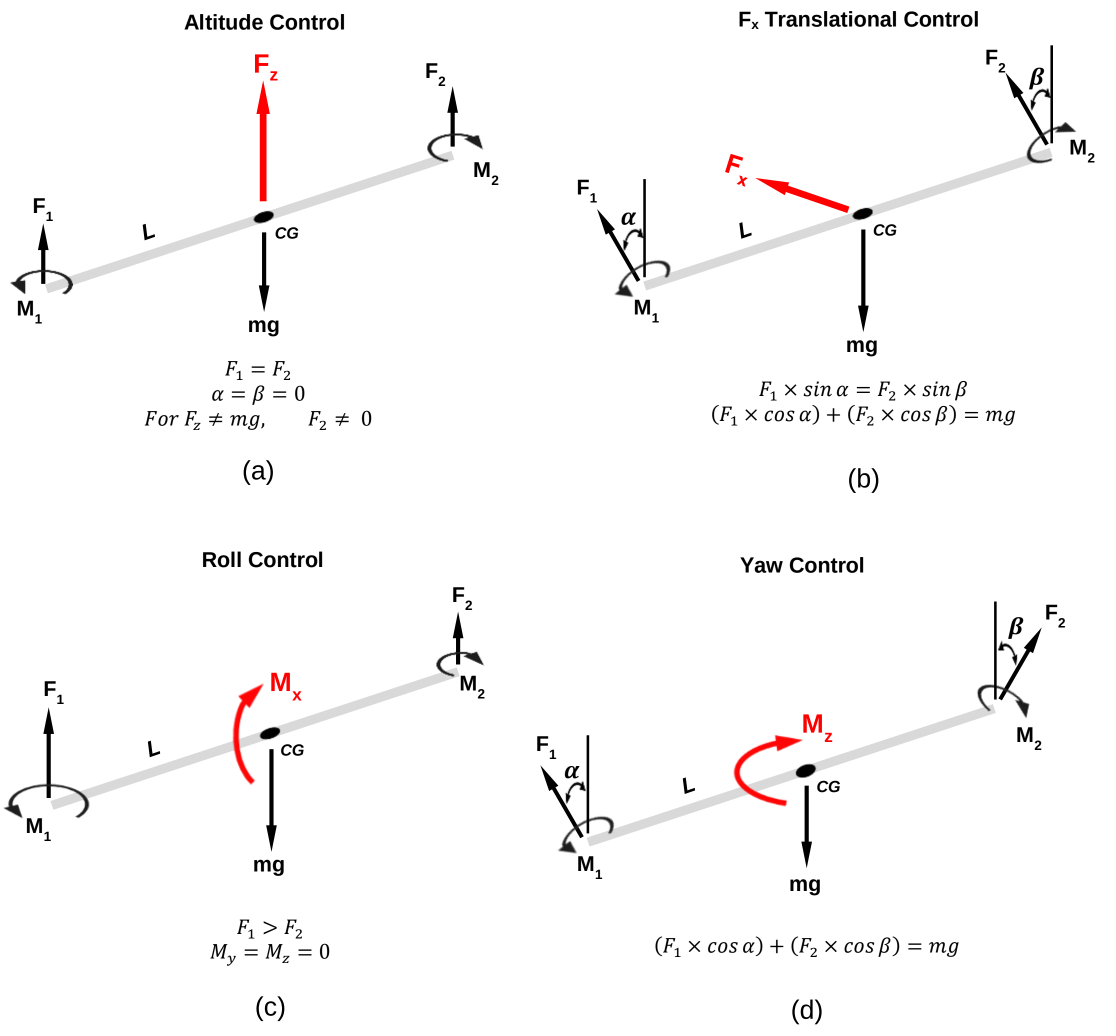
2.4.3.1 Control strategy – variable pitch

Although translational motion on Y_B axis is dependent on the roll angle, translational motion on X_B is independent of the pitch angle. The tilt angles α and β control translational motion on X_B, platform pitch and yaw. The motion control of this concept is split into altitude, translation on X_B axis, roll and yaw control.

To create a change in the generated thrust, the pitch of the propeller is varied while keeping the rotational speed constant. An altitude control shown in Figure 2.10(a) is attained by increasing or decreasing the propeller pitch such that; the thrust developed by both front propellers is equal and that there is no moment about the yaw and pitch. F_x axis translation control (Figure 2.10(b)) is by tilting the propellers around the pitch axis in the same direction by the tilt angles. Roll control is attained by varying the pitch of the propellers (Figure 2.10(c)) while yaw control is attained by tilting the rotational axis of the propellers (Figure 2.10(d)) in opposite directions.

2.4.3.2 Control strategy – fixed pitch variable speed

The control of fixed pitch variable speed concept is achieved with the same strategy presented above. The difference between both concepts is in the method of controlling thrust generation. The first variation of this concept involves changing the propeller pitch while the second relies on changing the rotational speed to control thrust generation. The strategy illustrated below (Figure 2.10) also applies to this concept.



$$F_1 = F_2$$
$$\alpha = \beta = 0$$
$$For\ F_z \neq mg, \qquad F_2 \neq 0$$

(a)

$$F_1 \times \sin\alpha = F_2 \times \sin\beta$$
$$(F_1 \times \cos\alpha) + (F_2 \times \cos\beta) = mg$$

(b)

$$F_1 > F_2$$
$$M_y = M_z = 0$$

(c)

$$(F_1 \times \cos\alpha) + (F_2 \times \cos\beta) = mg$$

(d)

Figure 2.10: Proposed attitude control for the third concept (applies to both concept variations)

2.5 Propulsion Design Analysis

The design analysis of the concepts is based on a set of relevant considerations (Table 2.3) which are ranked on a 3 tier (low, medium and high) priority system. A weight of 1 is assigned to low priority and 3 to high priority considerations. A ranking of 1 to 4 is assigned based on how well each concept satisfies the design considerations, 1 being the poorest solution and 4 the best solution. The rankings are factored with the design consideration weights and summed up for each concept. The concept with the highest total is considered the best solution.

As previously stated in section 2.3 the ranking and weight assigned in this analysis is based purely on the author's interpretation of the project background, objectives and design requirements.

2.5.1 Design Considerations

The high priority design considerations for this development are system stability and controllable DOF. Stability refers to the ability to recover the system from perturbations or, to maintain an even state in all possible modes of operation. Controllable DOF consideration accounts for the number of pure and independent degrees of control attainable for each concept.

Adaptability and ease of manufacture are the medium priority considerations. Adaptability of each concept refers to the simplicity of alteration required to implement the concept as it is presented on existing and future Guardian airframe. Ease of manufacture speaks of the extent to which the concept can be easily implemented.

Configuration symmetry and the relative estimation of the cost for each concept based on the main propulsion components and mechanisms are low priority design considerations.

Table 2.3: Propulsion configuration selection considerations

Design Consideration	Priority	Weighting
Adaptability	Medium	2
Configuration symmetry	Low	1
Controllable DOF	High	3
Cost	Low	1
Ease of manufacture	Medium	2
Stability	High	3

2.5.2 Configuration Comparison

The rankings of each concept for the design considerations and the total score for the concepts are as tabulated below (Table 2.4) based on the deliberations of the merits and demerits of the implementation of the concepts. The total achievable score is 48. The variable speed variation of the twin prop concept is the optimal solution with a score of 44.

Table 2.4: Decision analysis matrix of the propulsion configurations

Design Consideration	**Weight**	**Propulsion Configuration Concepts**			
		Tri-copter Fixed prop	**Tri-copter Tilting prop**	**Twin prop Variable Pitch**	**Twin prop Variable Speed**
Adaptability	***2***	1	2	3	4
Configuration symmetry	***1***	1	2	4	3
Controllable DOF	***3***	1	2	4	3
Cost	***1***	2	1	3	4
Ease of manufacture	***2***	3	1	2	4
Stability	***3***	1	2	3	4
	Score	17	21	38	44

CHAPTER THREE: CONTROL ALGORITHM

3.1 Introduction – Stabilization Augmented Control Algorithm

This chapter covers the development of a stability augmented control system for the VTOL UAV. It includes the development of a control strategy and a system controller for attitude correction. The stability augmented control algorithm includes a command allocation and control strategy based on the propulsion dynamics of the VTOL configuration developed in chapter 2 of this work.

The control and stabilisation algorithm is a set of mathematical models defining the relations of the linear and angular motion of the vehicle to moments and forces generated by a combination of propulsion hardware that are actuated based on the attitude adjusted control command. These set of equations provide for the control command allocation, attitude estimation, attitude correction, system propulsion dynamics, and for simulation purposes, the rigid body dynamics of the vehicle.

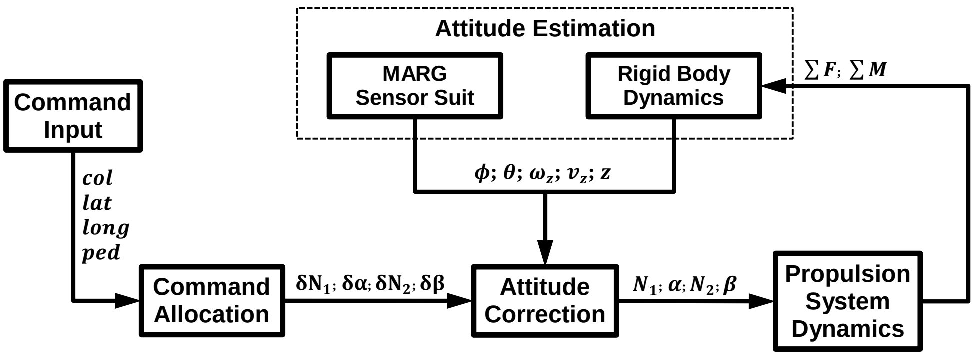

Attitude Estimation
MARG Sensor Suit
Rigid Body Dynamics
$\sum F;\ \sum M$
Command Input
col
lat
$long$
ped
$\phi;\ \theta;\ \omega_z;\ v_z;\ z$
Command Allocation
$\delta N_1;\ \delta\alpha;\ \delta N_2;\ \delta\beta$
Attitude Correction
$N_1;\ \alpha;\ N_2;\ \beta$
Propulsion System Dynamics

Figure 3.1: Block diagram of the stabilisation augmented control algorithm

The rigid body equations comprise the force and moment equations (Equation 1.6 and Equation 1.7) as well as the kinematic equations of rotation and translation (Equation 1.8 and Equation 1.9) discussed in section 1.5.3.2 (Rigid body Dynamics) of this dissertation. The propulsion system and rigid body dynamics equations provide for the propulsion and attitude of the vehicle and are useful in verifying the control algorithm in a simulator software.

3.2 Control Commands

The control commands follow those of a typical helicopter and are; collective, lateral, longitudinal and pedal (Watkinson, 2004 pp. 19-20). The collective command controls the rotational speed of the propellers collectively at the same rate, independent of their position. When a collective command input is made, all the rotational speeds change equally and change the thrust generated by the propellers. In hover flight mode, this will create an ascent or descent; while in forward motion, it will produce acceleration in the direction of motion together with a slight ascent.

The lateral command changes the rotational speed of the propellers inversely, also independent of their position. A lateral command reduces the rotational speed of one propeller while increasing the other and maintaining the generated collective thrust. This results in a rolling moment. This will cause sideways movement in hover flight mode and will induce a turn in forward motion.

The longitudinal command controls the direction and rate of the angle of tilt of the propellers. When a longitudinal command input is made, the propellers are tilted equally, in the same direction. This results in creating a component of the collective thrust on an axis parallel to the X_B (longitudinal) axis and introduces a pitching moment in the manner shown below (Figure 3.2). This consequently leads to a lateral motion and a body pitch. The pitching moment is directly dependent on the magnitude of the eccentricity (e_z) of the tilt axis to the BCS origin, the pitching moment approaches zero as tilt angle or e_z approaches zero.

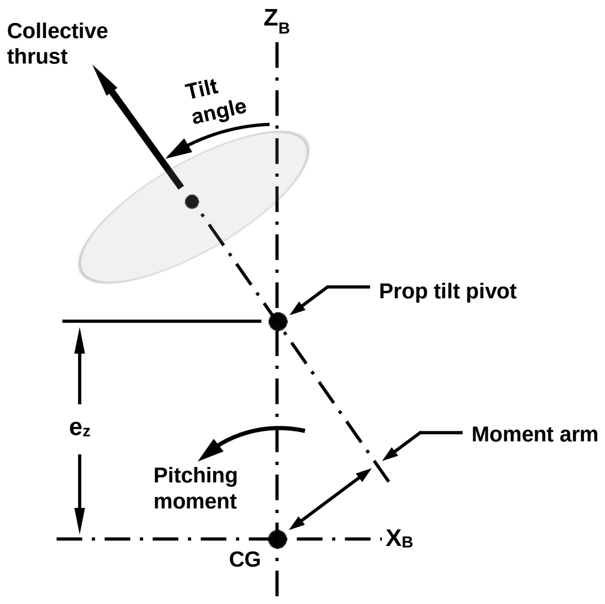


Figure 3.2: The secondary effect of the longitudinal control command

The pedal command changes the tilt angles of the propellers in either direction at the same rate. This results in a yawing moment, which in hover flight mode is used to control the yaw rate and in forward flight is used to control the sideslip angle.

Table 3.1 presents the control commands of the VTOL platform, the control hardware actuated by each command and their effects in hover and forward flight conditions.

Table 3.1: Control commands and their effects

Control Command	Hardware Actuated	Flight Condition Effects	
		Hover	Forward
Collective	Controls the rotational speed of both propellers (same speed)	Creates ascent or descent	Acceleration in the direction of motion
Lateral	Changes the rotational speed of both propellers (inversely)	Causes sideways movement	Used to induce a turn
Longitudinal	Controls the angle of tilt of the propellers (same direction and rate)	Initiates forward or backward motion	Used to regulate body pitch
Pedal	Changes the angle of tilt of the propellers (different direction same rate)	Controls the yaw rate	Used to adjust the sideslip angle

3.3 Control Allocation

The control commands are allocated to the control hardware in the manner shown in Equation 3.1. The collective, lateral, longitudinal and pedal commands ($col, lat, long$ and ped) are weighed by the command factors ($k_{col}, k_{lat}, k_{long}$ and k_{ped}) and summed up for each control hardware.

$$\delta N_1 = k_{col} \cdot col + k_{lat} \cdot lat$$
$$\delta\alpha = -k_{long} \cdot long + k_{ped} \cdot ped$$
$$\delta N_2 = k_{col} \cdot col - k_{lat} \cdot lat$$
$$\delta\beta = -k_{long} \cdot long - k_{ped} \cdot ped$$

Equation 3.1

Where δN_1 and δN_2 symbolise the command rotational speed of the propellers, $\delta\alpha$ and $\delta\beta$ represent the command propeller tilt angles.

As shown in table Table 3.1, the collective and lateral command controls the rotational speed of the propellers while the longitudinal and pedal command control the tilt angle of the propellers. Therefore, the command rotational speed is made up of the weighed collective and lateral commands. Similarly, the command propeller tilt angle is made up of the weighed longitudinal and lateral command.

The command factors ($k_{col}, k_{lat}, k_{long}$ and k_{ped}) are dependent on the propulsion hardware and the platform geometry and should therefore be tuned/determined for each VTOL platform.

3.4 Attitude Correction

3.4.1 Altitude Control

Altitude control is maintained with an altitude stabilise or altitude hold controller. Both utilise a parallel proportional derivative (PD) controller for altitude control correction. The altitude stabiliser controls the rate of altitude change while the altitude hold controller stabilises the altitude.

The altitude stabiliser utilises the rate of altitude displacement (a process variable); hence the error signal ($\boldsymbol{e(t)}$) in Equation 1.5 is set as the translational velocity (v_z) along the Z_B axis. The altitude hold controller is based on the altitude displacement and hence the error signal is set as the altitude displacement ($\delta z = z_s - z_t$), where z_s is the setpoint (desired altitude) and z_t is the system output (current altitude).

3.4.2 Attitude Control

Pitch and roll is controlled by a proportional derivative (PD) controller of the parallel form while the yaw controller is a proportional (P) controller. The choice of a PD or P controller is due to the destabilising effect of the Integral term arising from the sudden changes in signal error or system process that occur in this application.

The pitch and roll controller are based on the pitch and roll angles hence, the error signal for these controllers (pitch and roll controllers) is set as the pitch (ϕ) or roll (θ) angle respectively. The yaw controller is based on the yaw rate, therefore the error signal is set as the rotational velocities (ω_z) about the Z_B axis.

3.4.3 Control Command Adjustment

The actuation commands to the control hardware are adjusted with the altitude and attitude controller outputs (PD_{alt}, PD_{pitch}, PD_{roll} and PD_{yaw}) in the manner shown in Equation 3.2.

$$N_1 = \delta N_1 + PD_{alt} + PD_{roll} - PD_{yaw}$$
$$\alpha = \delta\alpha + PD_{pitch} + PD_{yaw}$$
$$N_2 = \delta N_2 + PD_{alt} - PD_{roll} - PD_{yaw}$$
$$\beta = \delta\beta + PD_{pitch} - PD_{yaw}$$

Equation 3.2

Where N_1 and N_2 represents the adjusted propeller rotational speeds while α and β represent the adjusted propeller tilts.

3.5 Propulsion System Dynamics

The force and moment equations that define the motion dynamics of the twin prop variable speed VTOL (Figure 2.10, on page 32) are:

$$F_x = F_1 \times sin\, \boldsymbol{\alpha} + F_2 \times sin\, \boldsymbol{\beta}$$
$$F_y = 0$$
$$F_z = F_1 \times cos\, \boldsymbol{\alpha} + F_2 \times cos\, \boldsymbol{\beta}$$

Equation 3.3

$$M_x = L\big((F_1 \times cos\, \boldsymbol{\alpha}) - (F_2 \times cos\, \boldsymbol{\beta})\big)$$
$$M_y = e_z\big((F_1 \times sin\, \boldsymbol{\alpha}) + (F_2 \times sin\, \boldsymbol{\beta})\big)$$
$$M_z = L\big((F_1 \times sin\, \boldsymbol{\alpha}) - (F_2 \times sin\, \boldsymbol{\beta})\big)$$

NB: (See Figure 3.2, on page 36)

Equation 3.4

F_x, F_y, F_z, M_x, M_y and M_z are the components of the accelerating force and moment as defined in section 1.5.3.2 (Rigid Bogy Dynamics). e_z (shown in Figure 3.2) is the eccentricity of the propeller tilt axis and the origin of the BCS projected on the Z_B axis. F_1 and F_2 are the thrust generated by the propellers, α and β are the propeller tilt angles. The thrust developed by the propeller is related to their rotational speed:

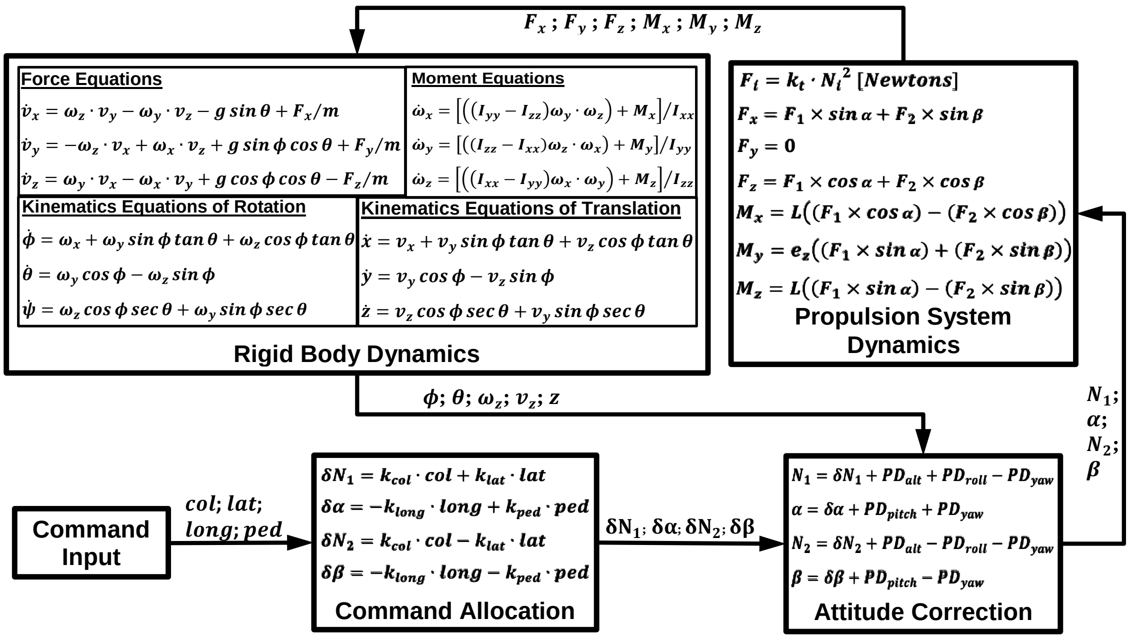
$$F_i = K_t \cdot {N_i}^2\ [Newtons]$$

Equation 3.5

K_t is the thrust coefficient and N_i *[RPM]* is the rotational speed of the propeller.

3.6 Summary

This chapter outlined a control algorithm and hardware suitable for use on micro UAVs. A summary of the flow of this algorithm is shown below.



$F_x\,; F_y\,; F_z\,; M_x\,; M_y\,; M_z$
Force Equations
$\dot{v}_x = \omega_z \cdot v_y - \omega_y \cdot v_z - g\, sin\, \theta + F_x/m$
$\dot{v}_y = -\omega_z \cdot v_x + \omega_x \cdot v_z + g\, sin\, \phi\, cos\, \theta + F_y/m$
$\dot{v}_z = \omega_y \cdot v_x - \omega_x \cdot v_y + g\, cos\, \phi\, cos\, \theta - F_z/m$
Moment Equations
$\dot{\omega}_x = \big[\big((I_{yy} - I_{zz})\omega_y \cdot \omega_z\big) + M_x\big]/I_{xx}$
$\dot{\omega}_y = \big[\big((I_{zz} - I_{xx})\omega_z \cdot \omega_x\big) + M_y\big]/I_{yy}$
$\dot{\omega}_z = \big[\big((I_{xx} - I_{yy})\omega_x \cdot \omega_y\big) + M_z\big]/I_{zz}$
Kinematics Equations of Rotation
$\dot{\phi} = \omega_x + \omega_y\, sin\, \phi\, tan\, \theta + \omega_z\, cos\, \phi\, tan\, \theta$
$\dot{\theta} = \omega_y\, cos\, \phi - \omega_z\, sin\, \phi$
$\dot{\psi} = \omega_z\, cos\, \phi\, sec\, \theta + \omega_y\, sin\, \phi\, sec\, \theta$
Kinematics Equations of Translation
$\dot{x} = v_x + v_y\, sin\, \phi\, tan\, \theta + v_z\, cos\, \phi\, tan\, \theta$
$\dot{y} = v_y\, cos\, \phi - v_z\, sin\, \phi$
$\dot{z} = v_z\, cos\, \phi\, sec\, \theta + v_y\, sin\, \phi\, sec\, \theta$
Rigid Body Dynamics
$\boldsymbol{F_i = k_t \cdot {N_i}^2\ [Newtons]}$
$\boldsymbol{F_x = F_1 \times sin\, \alpha + F_2 \times sin\, \beta}$
$\boldsymbol{F_y = 0}$
$\boldsymbol{F_z = F_1 \times cos\, \alpha + F_2 \times cos\, \beta}$
$\boldsymbol{M_x = L\big((F_1 \times cos\, \alpha) - (F_2 \times cos\, \beta)\big)}$
$\boldsymbol{M_y = e_z\big((F_1 \times sin\, \alpha) + (F_2 \times sin\, \beta)\big)}$
$\boldsymbol{M_z = L\big((F_1 \times sin\, \alpha) - (F_2 \times sin\, \beta)\big)}$
Propulsion System Dynamics
$\phi; \theta; \omega_z; v_z; z$
$N_1; \alpha; N_2; \beta$
Command Input
$col; lat; long; ped$
$\delta N_1 = k_{col} \cdot col + k_{lat} \cdot lat$
$\delta\alpha = -k_{long} \cdot long + k_{ped} \cdot ped$
$\delta N_2 = k_{col} \cdot col - k_{lat} \cdot lat$
$\delta\beta = -k_{long} \cdot long - k_{ped} \cdot ped$
Command Allocation
$\delta N_1; \delta\alpha; \delta N_2; \delta\beta$
$N_1 = \delta N_1 + PD_{alt} + PD_{roll} - PD_{yaw}$
$\alpha = \delta\alpha + PD_{pitch} + PD_{yaw}$
$N_2 = \delta N_2 + PD_{alt} - PD_{roll} - PD_{yaw}$
$\beta = \delta\beta + PD_{pitch} - PD_{yaw}$
Attitude Correction

Figure 3.3: Summary of the stabilisation augmented control algorithm

CHAPTER FOUR: PROTOTYPE CONTROL HARDWARE

4.1 Design Requirements

UAVs need specific hardware and software in order to attain stability and perform tasks efficiently. The driving requirements behind the development of a control system for UAVs are capability (functionality), Lightweight and the data communication bandwidth.

Possible hardware concerns for the control system to which extra attention was given in both the selection and implementation of the components are:

1. Processing power and capability of the flight microcontroller
2. Communication data bandwidth of the IMU sensors
3. Functionality and accuracy of the inertial measurement unit
4. Refresh frequency of the propulsion hardware and controllers
5. Size and weight of the control and stabilisation hardware

4.2 The Embedded Control Platform

A typical stability augmented control hardware (layout shown in Figure 4.1) for robotic systems (small UAVs included) has a microcontroller (MCU) or microprocessor (MPU) which decodes the command input, reads the sensor data and estimates attitude, adds the response of the estimated attitude to the control command and outputs the control signals to the propulsion hardware.

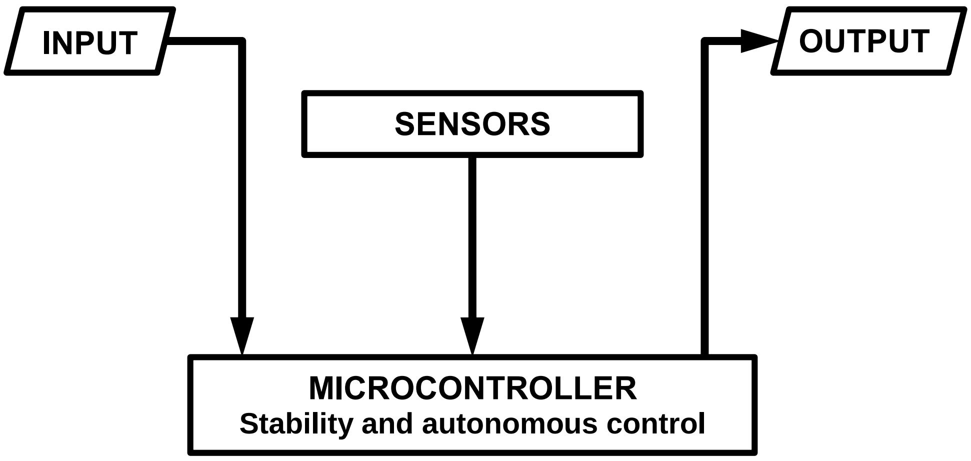


Figure 4.1: Typical layout of a stability augmented control system

All commercially available radio control (RC) units transmit and receive signals as pulse modulations, the control channels are multiplexed into pulse position modulation (PPM) signal, which is then transmitted to a receiver. The receiver de-multiplexes the PPM signal into separate channels and feeds the pulse width modulation (PWM) signals to the propulsion hardware.

To tap into the wide range of available RC control units for the remote piloting of the UAV, the on-board electronic system has to be able to decode the PPM or the PWM signals from the RC receiver. The technique for decoding the pulse modulated signals involves listening for and recording the duration of the high or low bit of the pulses, which implies that the MCU has to actually pause and monitor the pulse channel in order to decode the signal. Table 4.1 shows the duration required to decode 7 channel PWM signals from an RC receiver (JR PROPO RD731) using an 8-bit MCU (ATmega328P clocked to 16MHz) and a 32bit MCU (PIC32MX795F512L on the CHIPKIT MAX32 development board clocked to 80MHz). The table shows the total decode duration for all signals as approximately 76ms on both platforms thus indicating that the duration is independent of the hardware architecture of the microcontroller but dependent on the RC equipment and number of channels.

Table 4.1: PWM signal decode duration

Embedded Architecture	**Number of RC channels**						
	1	2	3	4	5	6	7
	Cumulative decode duration [us]						
8bit Microcontroller	15536	26930	29287	39918	51305	70962	75675
32bit Microcontroller	15355	26730	29082	39683	51055	70665	75370

It was established in section 3.2 that four control commands (altitude, translation on X_B axis, roll and yaw) are required for complete control of the UAV. Hence, four channels are required for its remote piloting, adding another channel for setup and mode switch purposes implies that 52ms of the computing time of the embedding system will be spent in decoding the control command. As these microcontrollers feature single core processors incapable of multi-threading, the maximum attainable control output refresh frequency given the above considerations will be approximately 19Hz. This output rate is inadequate in attaining stability in a rotorcraft system.

To overcome the above limitation, a system with a secondary MCU dedicated to decoding command input signals and a main MCU for attitude estimation and control calculations is proposed. Figure 4.2 shows the architectural overview of the proposed control hardware, where a main MCU handles the sensor input and fusion, as well the stabilization and control calculations; outcomes of which are sent out as either digital or analogue signals. A secondary MCU is provided for decoding input signals from an on-board flight mission controller or remote control signals from a ground control station or pilot. The input can be in the form of either digital or analogue.

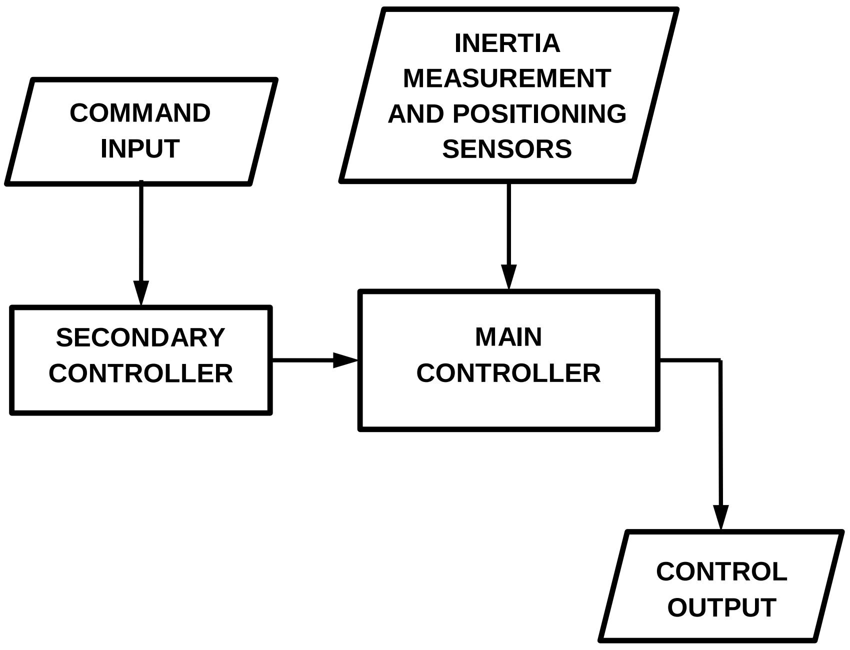


Figure 4.2: Architectural overview of the proposed control system

On the prototype embedded control system, the main controller (flight MCU) is a microcontroller (PIC32MX795F512L on the CHIPKIT's MAX32 development board) featuring a 32-bit MIPS processor core running at 80MHz, has 512K bytes of flash program memory and 128K bytes of SRAM data memory. The secondary (input) MCU (ATmega328P) has an 8-bit MIPS core running at 16MHz, 32K bytes of flash program memory, 2K bytes of SRAM data memory and 1K bytes of EEPROM.

An inertia measurement unit of 3-axis accelerometers, 3-axis angular rate sensors, 3-axis magnetometer and a pressure sensor is used for attitude estimation. The input and output pins, sensors and the input MCU are placed on a printed circuit board (shown in Figure 4.3) which plugs onto the CHIPKIT's MAX32 development board (Figure 4.4).

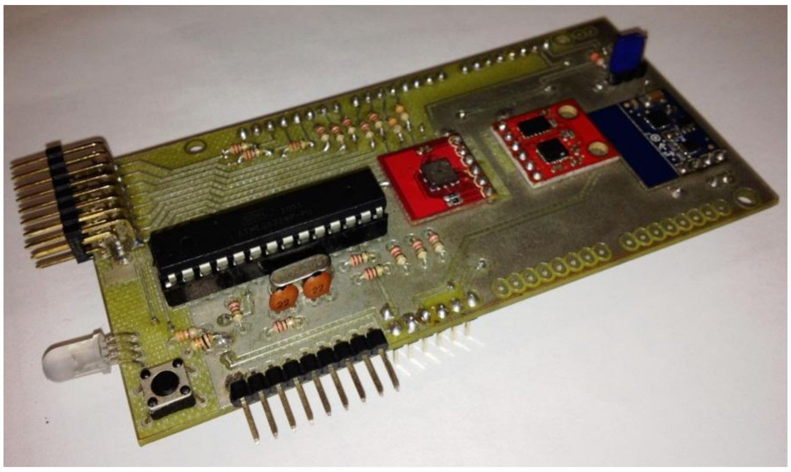


Figure 4.3: Top PCB stack of the embedded control system prototype

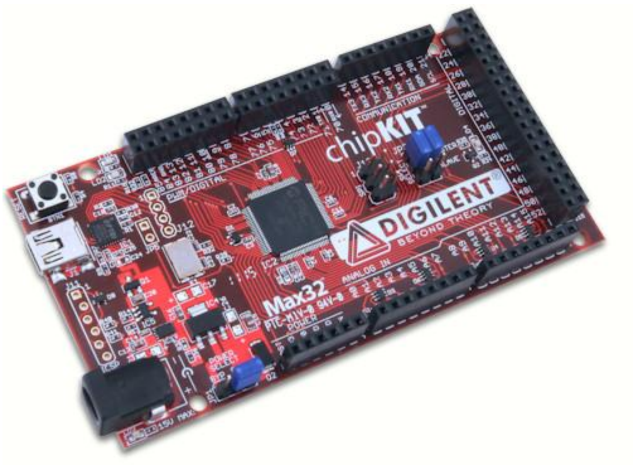


Figure 4.4: CHIPKIT's MAX32 development board

The flight microcontroller is connected to the inertial measurement unit and receives data regarding the attitude of the VTOL platform. The input MCU and sensors are placed on the inter integrated circuit (I2C) bus of the flight MCU with the flight MCU acting as the I2C master. Data transfer between the microcontrollers is setup to occur after a request for signal from the main microcontroller and an acknowledgement of new data from the secondary microcontroller is communicated between the microcontrollers. This facilitates seamless transfer of signals without compromising the integrity of the data.

4.3 Control Software

4.3.1 Secondary controller

The primary function of the software in this microcontroller is to receive and decode input, and then send the signals when requested to the main controller. However, a signal calibration routine is included in the code to improve the reliability of the commands (see Figure 4.5). This serves the dual purpose of sustaining the data integrity and a safety check for unintended hardware control as signals are not sent to the propulsion hardware if signal calibration is incomplete. Other safety features included in the routine are visual alerts and arm state check. Hardware arm state check disarms the control system (saves no signal to the output buffer) until the arm command is given and held for the pre-set duration.

The program also features an output buffer, which is used to store the decoded signals. These are refreshed at a rate of 20Hz and written to the output buffer by the active loop. The signals from the output buffer are transmitted to the flight controller on the receipt of a transmit signal request.

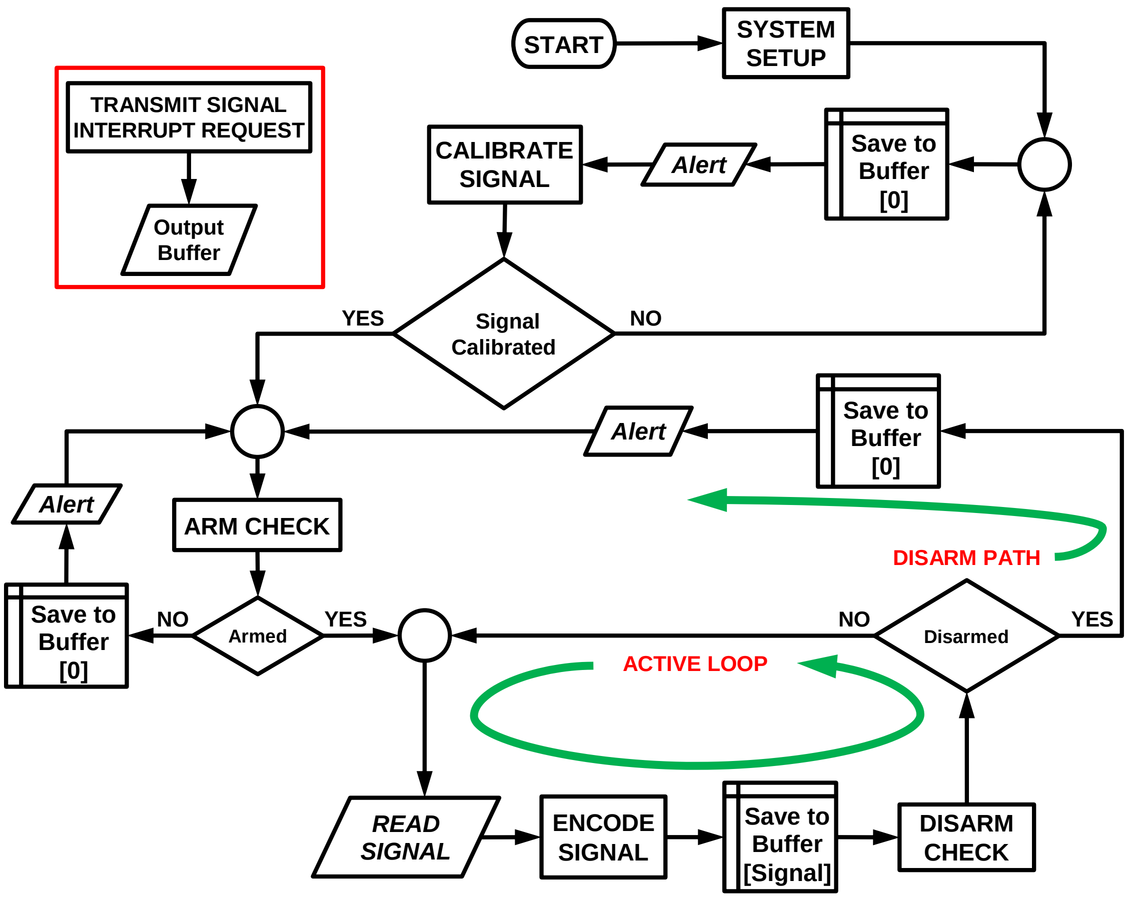


Figure 4.5: Process flow diagram of the secondary controller software

Appendix C1 contains the programs and routines written in C that outline the steps within the processes in this controller software.

4.3.2 Main controller

The software design for this controller is focused on eliminating redundancies while maintaining the data (control command and acquired raw sensor readings) transmission integrity and optimising command output refresh frequency.

An overview of the control and stabilisation process is shown in Figure 4.6. The process path starts by acquiring the commands, reads the raw sensor data from the attitude sensors, estimates the attitude of the system, and finally computes the output correcting for the estimated attitude

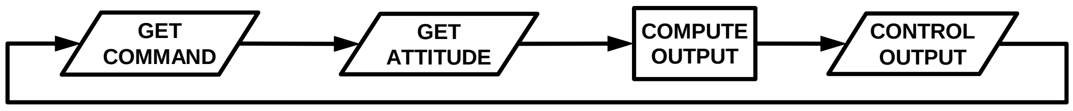


Figure 4.6: The control and stabilisation process overview

The control and stabilisation process is implemented in the form as a multi path process. The paths are arranged in three tiers of low, medium and high frequency. The low frequency path executes all the control and stabilization events. While the medium and high frequency paths execute selected events that present or can output new set of data during each loop, the main consideration driving the choice of implementing this multi-path process is the output refresh rate of the sensors and the secondary controller; hence each path is executed at a frequency corresponding to the refresh rate of the events within it.

Following in the development pattern of the secondary controller, the main controller software also features a system arm / dis-arm check and visual alerts that form part of the system safety processes. Figure 4.7 shows the complete program flow chart of the main controller software

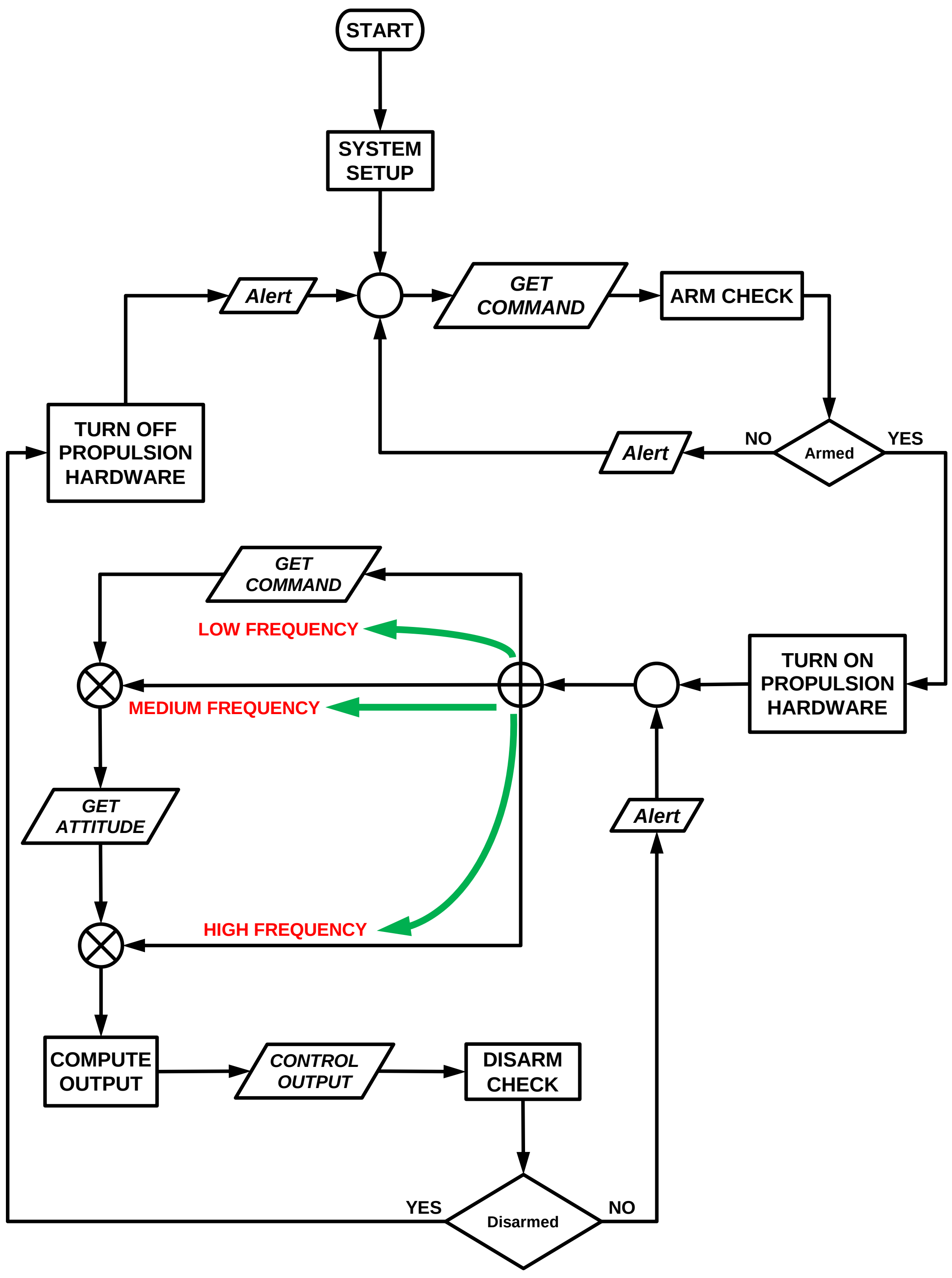


Figure 4.7: Process flow diagram of the main controller software

The programs and routines that outline the steps within the processes in this controller software are listed and defined in appendix C2.

4.4 Summary

This chapter outlined a control hardware suitable for use on micro UAVs. The hardware optimisation implemented is based on the use of a dedicated microcontroller (the secondary controller) for decoding input signals; this enables the inputs to the platform to be in the form of the computationally expensive PWM signal without compromising on the frequency of the stabilization loop. This hardware optimisation also frees up computational resources on the main processor.

Various sensor fusion and filtering techniques can be implemented on the prototype control hardware, these can be the extended Kalman filter (Kalman, 1960) and an optimised gradient descent algorithm (Madgwick, et al., 2011)).

Table 4.2 shows a comparison of the cost and functionality of the developed prototype to commercially available low-end (considering price) UAV control and stabilization hardware. The cost for the developed system is an estimated selling cost, which accounts for the cost of the microcontrollers, sensors, other components, as well as the manufacturing cost and a profit margin but excludes the development cost.

Table 4.2: Comparison of control hardware

	UAV Control Hardware			
	Developed Hardware	**APM II**	**DJI Wookong**	**Flight-Ctrl ME**
Price (R)	1800	2500	11000	4200
Sensor Package				
Altitude	Yes	Yes	Yes	Yes
Angular rate	Yes	Yes	Yes	Yes
GPS	Requires additional hardware	Requires additional hardware	Requires additional hardware	Requires expansion board
Gravity	Yes	Yes	Yes	Yes
Magnetic	Yes	Yes	Yes	Yes
Functionality				
Stabilization	Yes	Yes	Yes	Yes
Autonomous control	Requires GPS sensor	Yes	No	Requires expansion board
Output update frequency	500 Hz	100 Hz	50 Hz	450 Hz

CHAPTER FIVE: VALIDATION

5.1 Validation of Control Algorithm

This section provides a validation of the suitability and functionality of the stabilisation augmented control algorithm using a Simulink model of the motion dynamics of the VTOL platform. The chapter concludes with the discussion of the numerical simulation results of the behaviour of the twin propeller variable speed VTOL propulsion configuration for various command inputs and initial conditions.

The numerical simulations presented in this section were done through MATLAB Simulink using the ordinary differential equation3 (Bogacki-Shampine) solver and a fixed step size of 0.05 seconds. The mathematical relations that make up the Simulink model (Figure 5.1) are Equation 1.6, Equation 1.7 and Equation 1.8 for the rigid body dynamics, Equation 3.1 for command allocation, Equation 3.2 for attitude correction and Equation 3.3, Equation 3.4 and Equation 3.5 for propulsion system dynamics. Refer to Appendix D for Simulink model of all the mathematical relations.

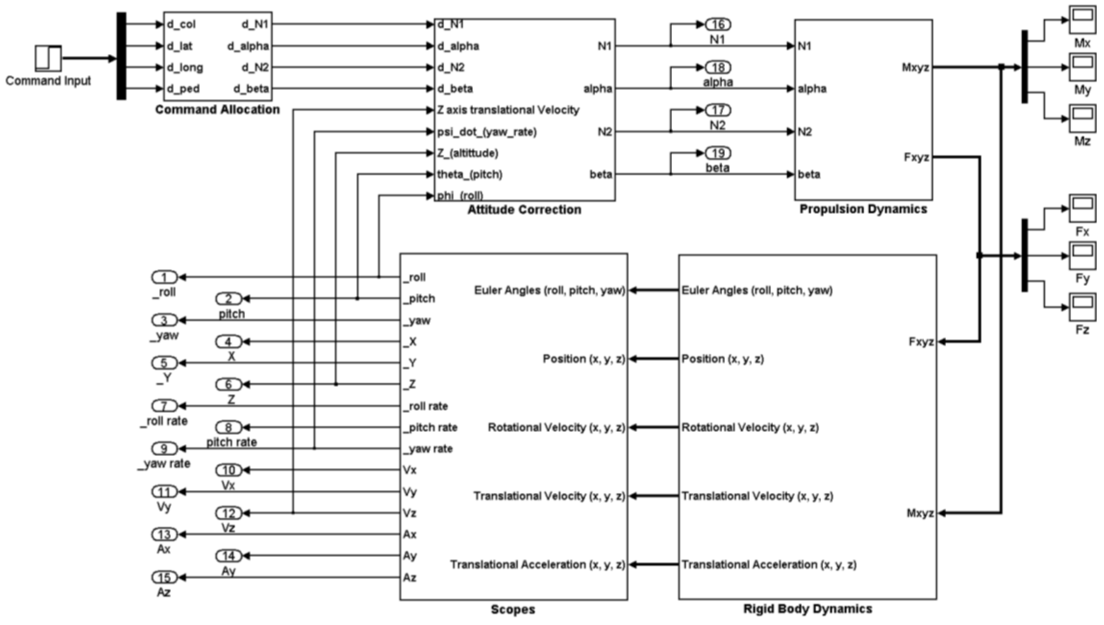


Figure 5.1: VTOL Platform System Level Model in MATLAB Simulink

5.1.1 Simulation parameters

Five Nonlinear simulations that observe the altitude, altitude rate, roll, pitch and yaw rate responses to destabilizations was conducted for 5 seconds each.

The thrust coefficient (*K_t*) and a torque coefficient (*K_m*) are determined from the propulsion characterisation experiment (see Appendix A) for a 2500 watts brushless dc motor (Turnigy C6374-200) and a 3 bladed 22 inch by 10 inch propeller. The inertia tensors and physical quantities L and e_z are generated or measured from the geometrical representation of the Guardian II airframe in a solid modelling software package (SolidWorks 2010). The quantities are tabulated in Table 5.1.

Table 5.1: Variable speed tilt-rotor parameter

Parameter	Unit	Value
mass	**kg**	**12**
I_{xx}	**Kg.m^3**	**9.68**
I_{yy}	**Kg.m^3**	**0.48**
I_{zz}	**Kg.m^3**	**9.92**
L	**m**	**1.3**
e_z	**m**	**0.1**
K_t		**4.3854e-006**
K_{col}		46
K_{lat}		**3**
K_{long}		**0.12**
K_{ped}		**0.08**

The MATLAB Simulink numerical model is setup such that each simulation is used to tune the gains of the corresponding controller. Hence, the gains where first determined and set for all controllers before executing the simulations. The control command, initial conditions and controller gain values for each simulation are shown in Table 5.2.

Table 5.2: Simulation parameters

Control Commands				
Type	**Value [%]**	**Duration [s]**	**Start time**	**End time**
Collective	**80**	5	0	5

Simulation Conditions			
Simulation	**Initial Conditions**	**Symbol**	**Value**
Altitude stabilizer	Translational velocity along the Z_B axis	V_z	-5m/s
	Translational acceleration along the Z_B axis	A_z	-9.81m/s^2
Altitude hold	Altitude displacement from set-point	δZ	0.5m
Roll stabilisation	Roll angle	ϕ	5^o
Pitch stabilisation	Pitch angle	θ	10^o
	Translational velocity along the X_B axis	V_x	5m/s
Yaw stabilisation	Yaw rate	ω_z	45^o/s

Controller Gains				
Controller	**K_P**	**K_I**	**K_D**	**N**
Altitude stabilize	905	0	0	0
Altitude hold	4680	0	1452	21
Roll	3285	0	830.36	20
Pitch	-21.2	0	-15.575	19.4
Yaw	-25	0	0	0

Figure 5.2 to Figure 5.5 below shows the initial conditions for the validation simulations.

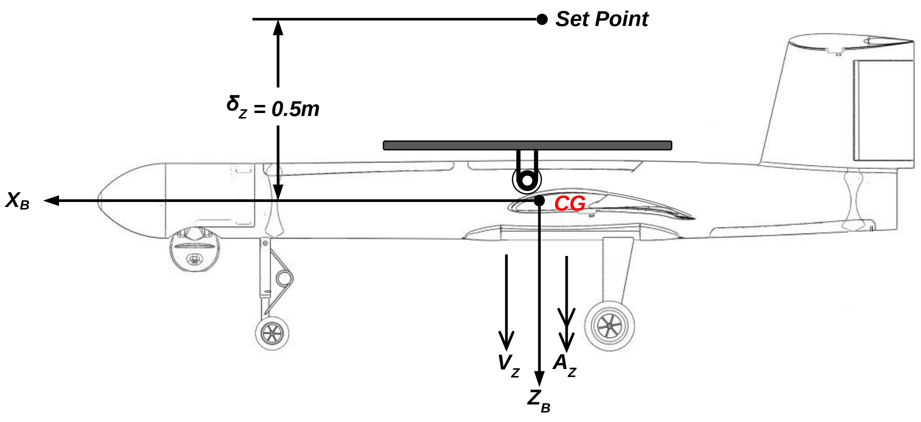


Figure 5.2: Initial conditions for altitude stabilise and altitude hold simulation

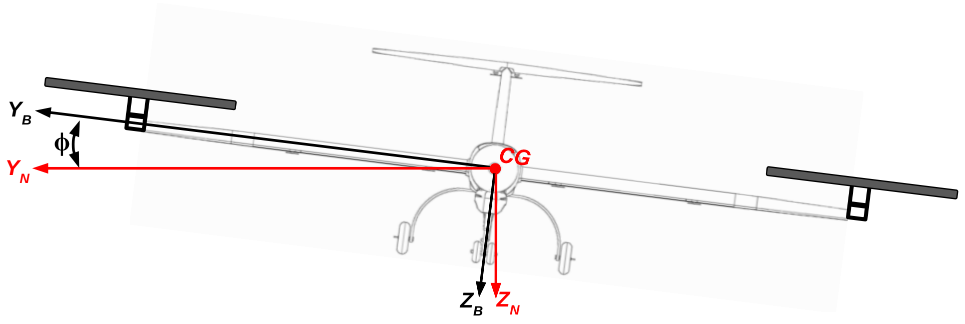


Figure 5.3: Initial condition for roll stabilisation simulation

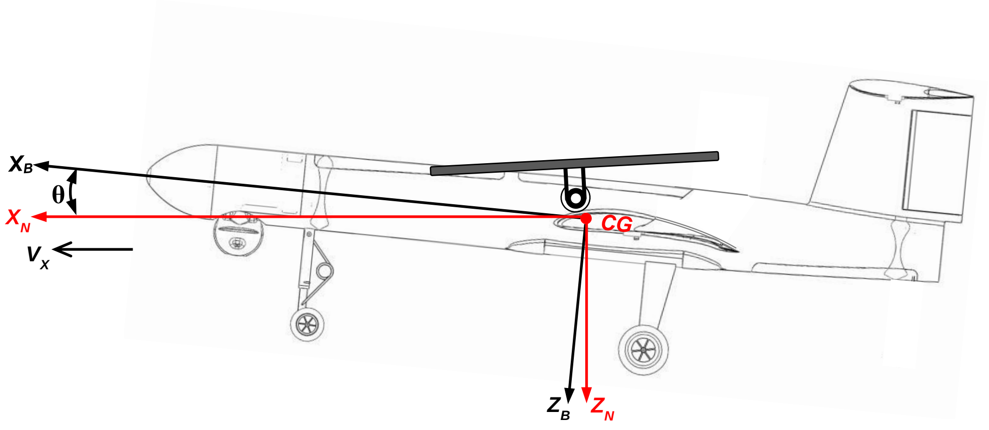


Figure 5.4: Initial conditions for pitch stabilise simulation

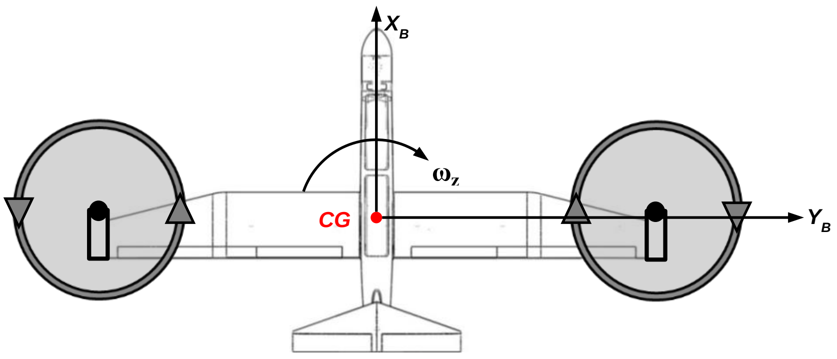
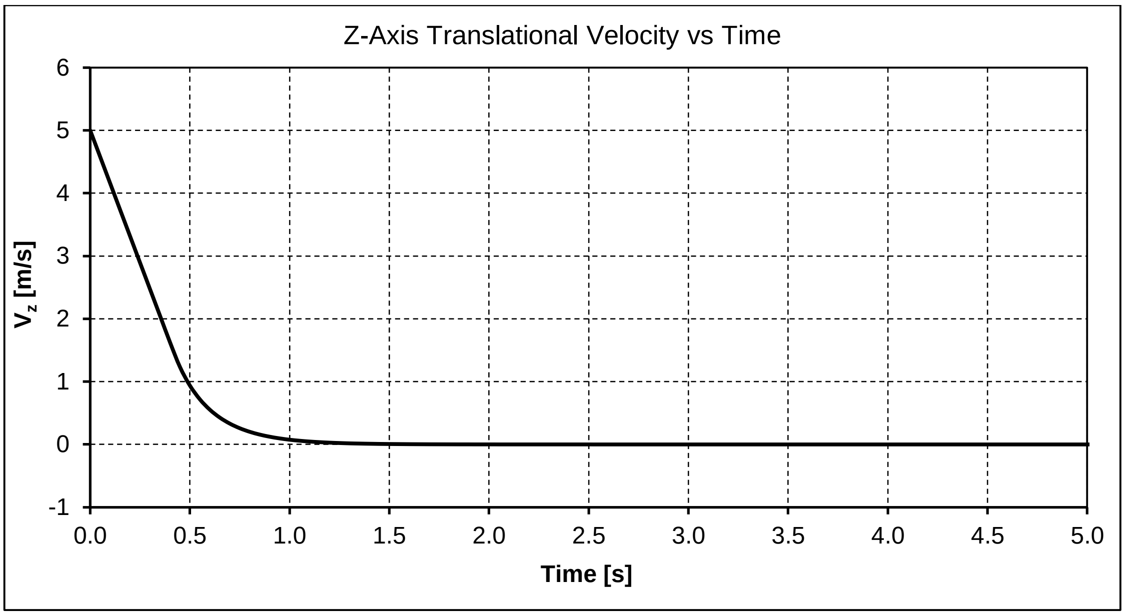


Figure 5.5: Initial condition for yaw rate stabilisation simulation

5.1.2 Simulation results

The altitude stabilize simulation observes the altitude rate. The initial simulations conditions are set to mimic a free fall at a rate of 5m/s under a gravitational pull of 9.81m/s.

Figure 5.6 below shows the altitude stabilize controller to be able to recover and stabilize the vehicle from the simulated free fall in less than 2 seconds.



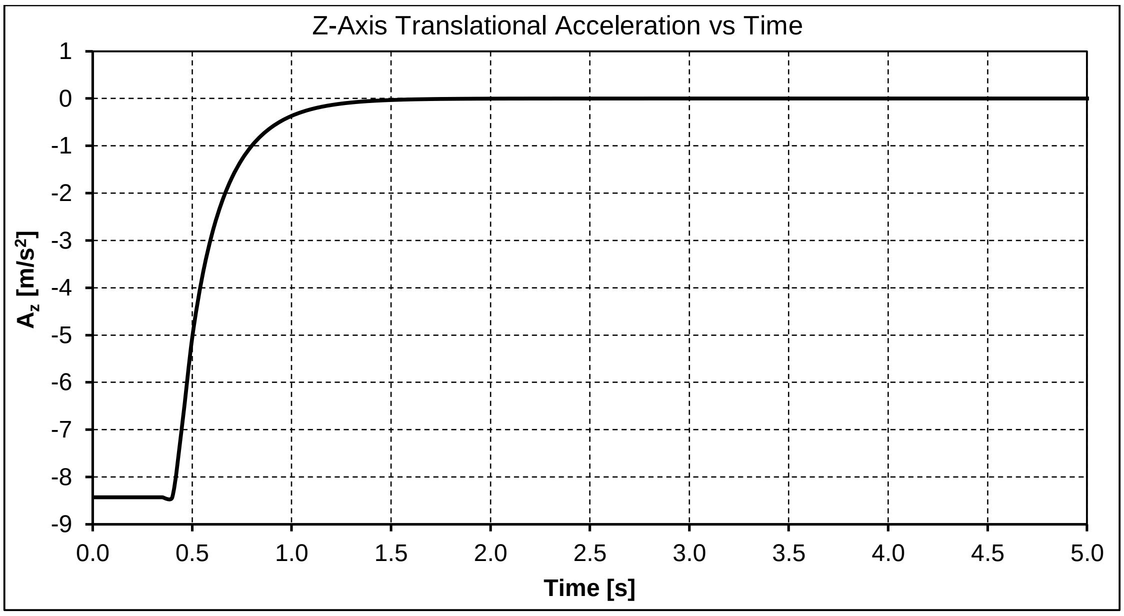


Figure 5.6: Vertical velocity and acceleration response to altitude-rate simulation

In the altitude hold simulation, the altitude is observed to stabilize (as seen in Figure 5.7) in less than 2 seconds from a destabilization of 0.5m.

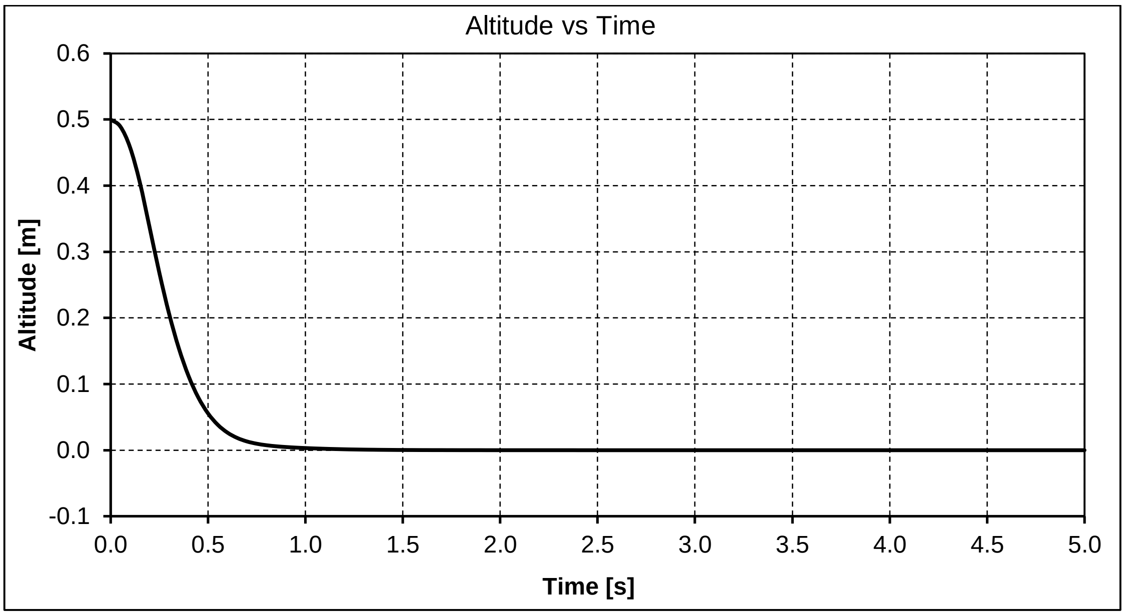


Figure 5.7: Altitude response to altitude-hold simulation

The roll-stabilize simulation results (Figure 5.8 and Figure 5.9) show the platform to stabilize in from a 5^o roll angle destabilization.

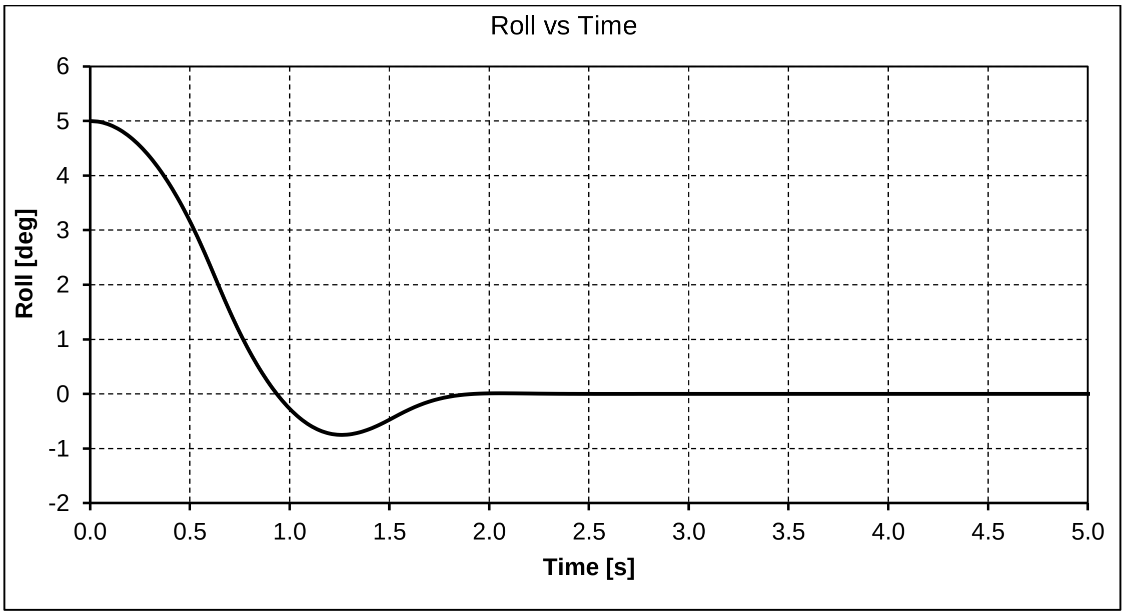
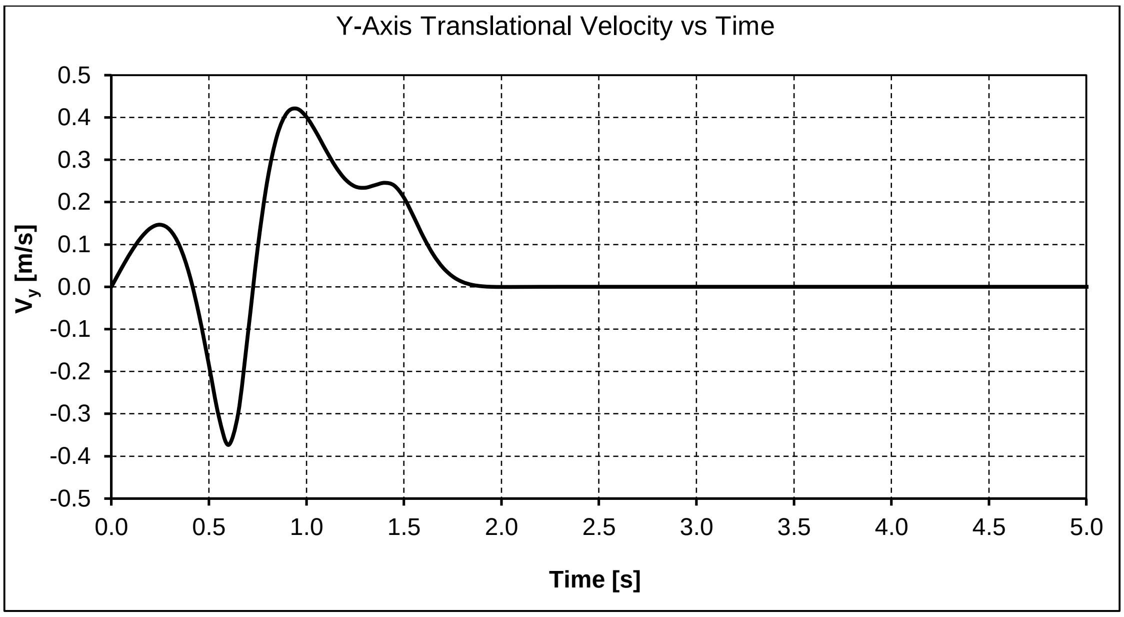


Figure 5.8: Roll angle response to roll stabilize simulation

The translation on Y_B axis and the platform altitude is affected by the roll angle; hence a destabilization in the vehicle roll direction will bring about destabilization in both axes. Figure 5.9 below shows the roll and altitude stabilize controller to be able to stabilize the translational velocity on the Y_B axis and the altitude destabilizations that is due to the roll destabilization.



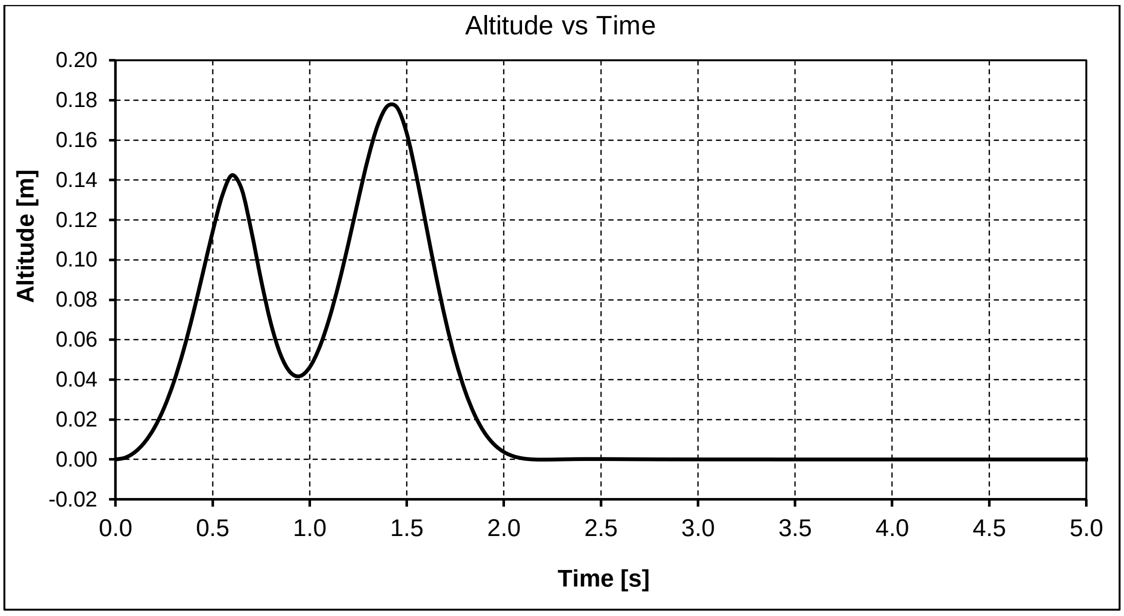
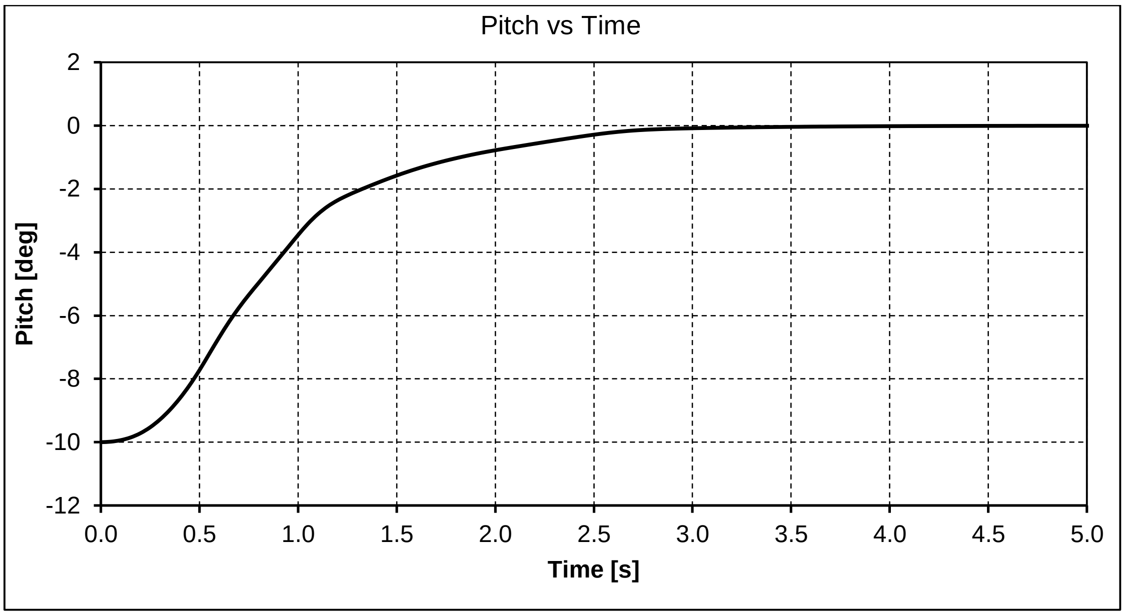


Figure 5.9: Responses and corrected propeller speed command to roll simulation

The pitch stabilise simulation results (Figure 5.10) shows the controller is able to stabilise the system from -10° destabilization in 3.5 seconds. As the pitch of the platform is coupled to translation in the X axis, a destabilisation in pitch will lead to a translational in the axis. This translation is also shown to stabilise with the pitch stabilisation.

The pitch destabilisation also creates a destabilisation in the altitude as the propulsion system thrust vertical axis component diminishes with the pitch increment. Figure 5.11 shows the control algorithm is still able to maintain stability of the altitude during pitch stabilisation.



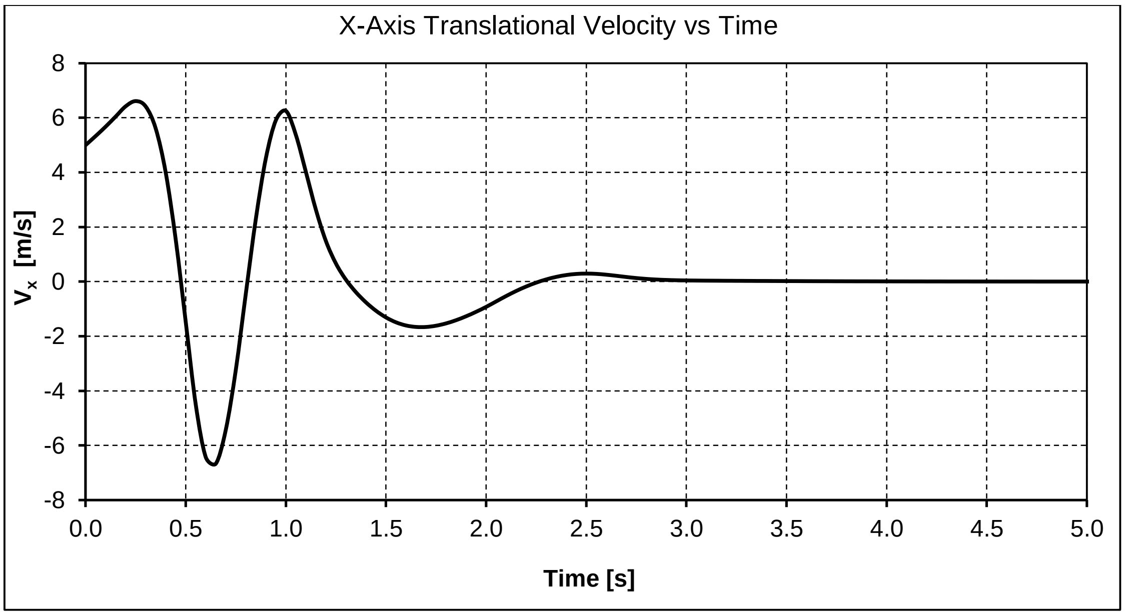
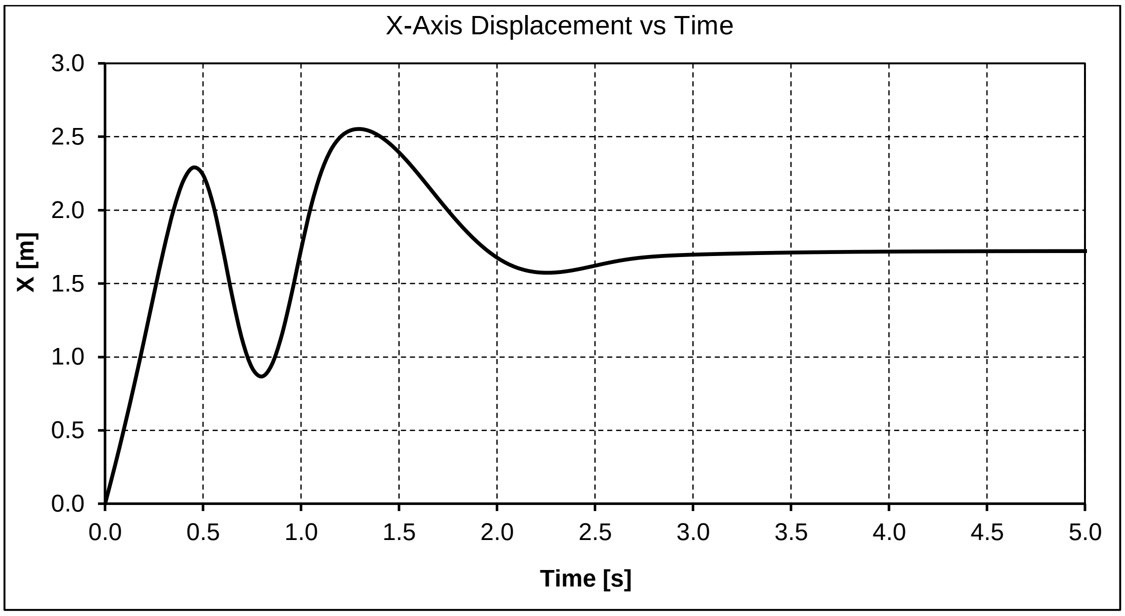


Figure 5.10: Pitch and x axis translational velocity response to pitch stabilise simulation



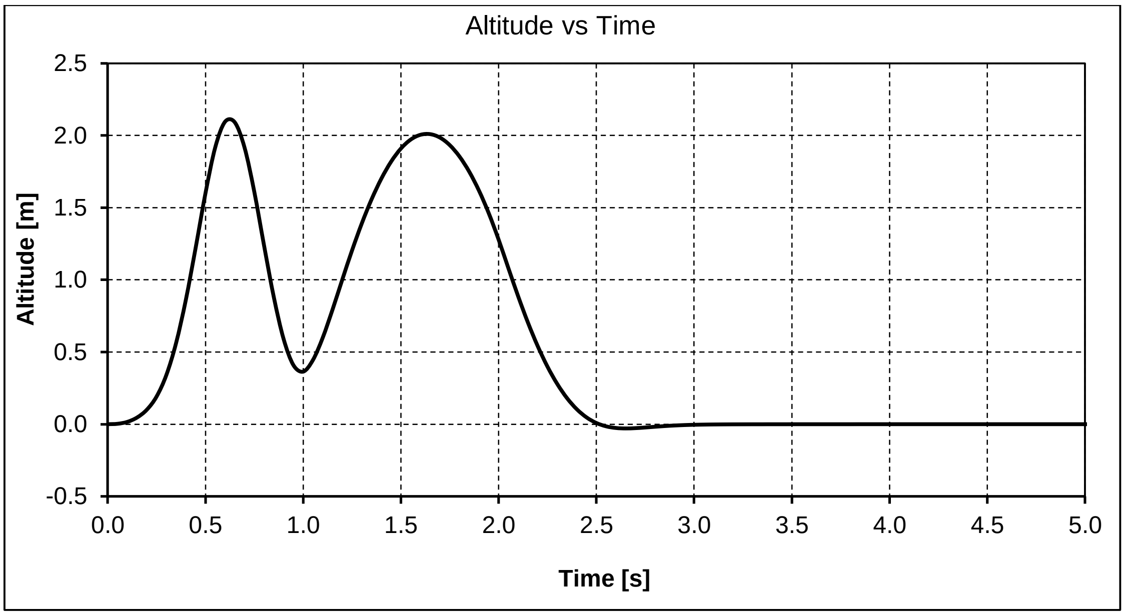
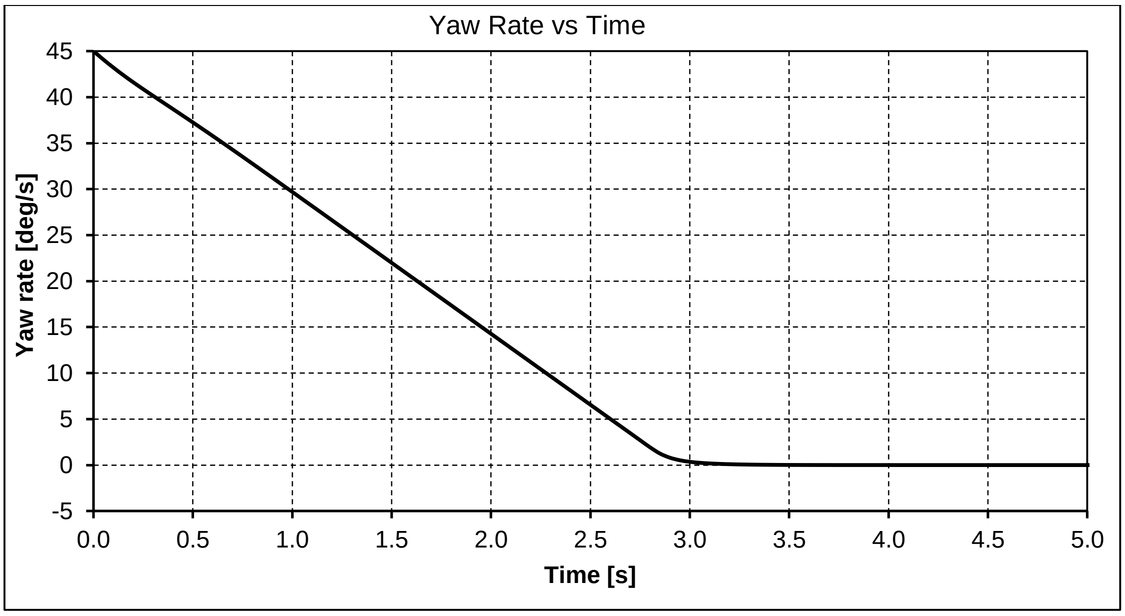


Figure 5.11: X axis displacement and altitude response to pitch stabilise simulation

The results of the yaw rate simulation results (Figure 5.12) show that the system is able to recover from a yaw rate destabilisation of 45° per second in less than 3.5 seconds while still maintaining altitude stability.



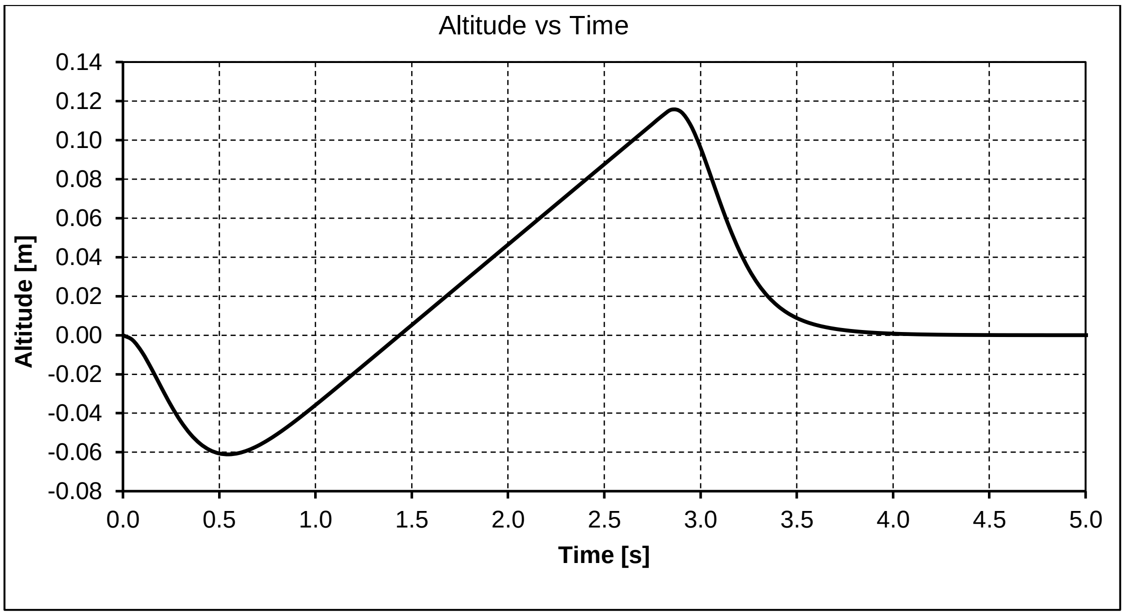


Figure 5.12: Yaw rate and altitude response to yaw rate stabilise simulation

These results show that the developed control algorithm and the proposed stabilisation routine will be able to provide stable control of the system. The results also proved the algorithm to be capable of handling multi-mode testability.

This section has successfully provided a validation of the proposed VTOL control algorithm using a Simulink simulation model. In addition to this validation, this section has provided, in the form of the simulation model, an important tool for further development and modification of VTOL UAV systems.

5.2 Validation of Control Hardware Prototype

This section provides a validation of the prototype control hardware presented in section 4.2 and the control software as discussed in section 4.3 of this work. The suitability and functionality of these systems are demonstrated using a scaled prototype of the variable speed tilt-prop VTOL propulsion configuration.

The scaled prototype shown below (Figure 5.13) has two servo arms which are used to demonstrate the tilt direction of the propellers. Each arm has an amber LED, these LEDs show the rotational speeds of the propellers. The brightness the LEDs indicate the rotational speed command that is sent to the electric motor controller. A high rotational speed command increases the brightness of the corresponding LED.

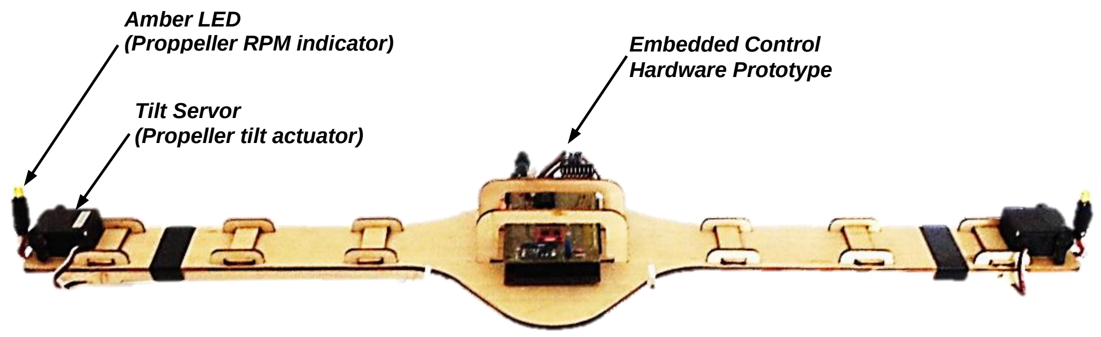


Figure 5.13: Scaled model for control hardware validation

Although the control actuators used in the scaled prototype is dissimilar (in terms of size and type for the LED) to the actuators which are to be used in the full scale model, the control signals sent to these representative hardware is similar in form and magnitude to that required by an actual model. Hence this control prototype is a true representative of the actual control system.

The validation exercise compares the behaviour (state) of the actuators on the scaled prototype to an expected actuated behaviour when subjected to a known set of destabilization. Four cases of this destabilisation are investigated and the outcomes presented.

5.2.1 Case 1: 0° roll and pitch destabilisation

When the variable speed tilt-prop VTOL platform is in a state of 0° roll and pitch destabilisation, the platform is said to be in attitude stability. Therefore, rotational speed of the propellers will be equal and the tilt direction of the propellers will be the same direction and parallel with the yaw axis (Z-axis or $\mathbf{Z_B}$) of the platform.

The figure below shows that both LEDs have the same intensity and the tilt of the servo arms are the same and close to the yaw axis of the prototype. Hence the response of the prototype corresponds to the anticipated response.

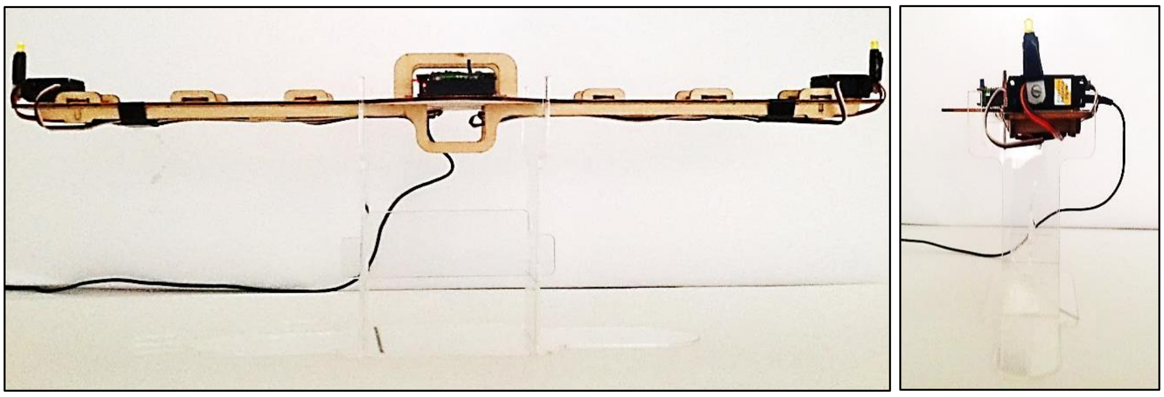


Figure 5.14: Physical model behaviour under 0° roll and pitch destabilisation

5.2.2 Case 2: ±15° roll and 0° pitch destabilisation

When the VTOL platform is in a state of ±15° roll and 0° pitch destabilisation, the control system will stabilise the platform by increasing the rotational speed of the propeller in the direction of the roll (lower propeller). As there is still no pitch destabilisation, the tilt direction of the propellers will still be in the same direction and parallel with the yaw axis (Z-axis or $\mathbf{Z_B}$) of the platform.

The figure below shows the lower LED shining with a higher intensity and the tilt of the servo arms still the same and close to the yaw axis of the prototype. Hence the response of the prototype corresponds to the anticipated response.

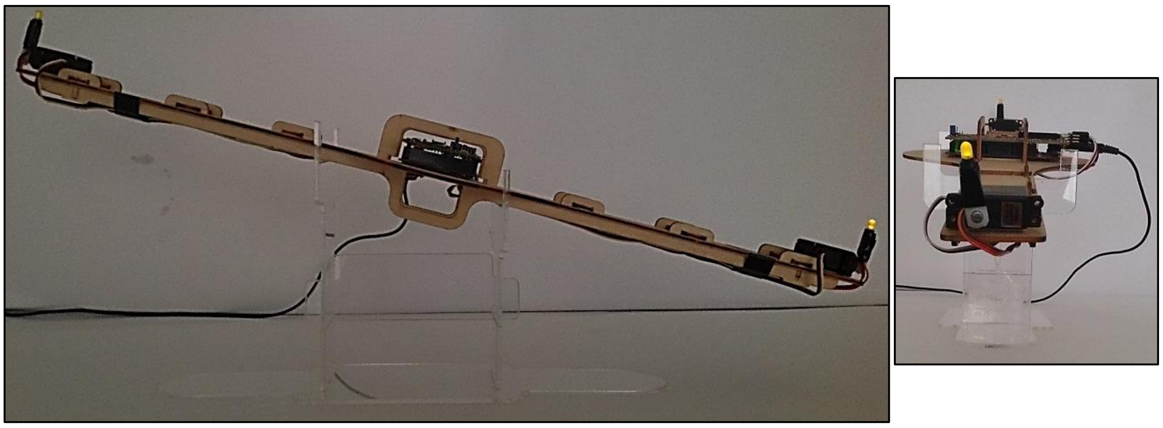


Figure 5.15: Physical model behaviour under ±15° roll and 0° pitch destabilisation

5.2.3 Case 3: 0º roll and ±15º pitch destabilisation

In the case of a pitch-up or pitch-down destabilisation, the control system will stabilise the vehicle by tilting the propellers in a direction opposite to the pitch. The reaction to rolldestabilisation is as shown above hence a 0º roll would mean the propellers will have same rotational speed.

The tilt of the servo arms is seen in the figure below to be tilted away from the Z_B axis in the direction opposite to the pitch. The LEDs are also observed to have the same intensity. Hence the response of the prototype corresponds to the anticipated response.

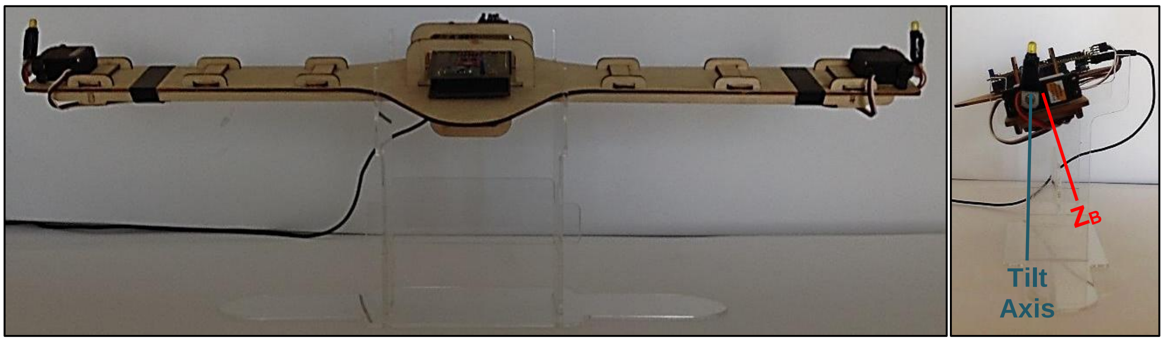


Figure 5.16: Physical model behaviour under 0º roll and ±15º pitch destabilisation

5.2.4 Case 4: ±15º roll and ±15º pitch destabilisation

This case is a combination of case 2 and case 3 above. Therefore to stabilize the vehicle, the control system will increase the rotational speed of the propeller in the direction of the roll while tilting the propellers in a direction opposite to the pitch.

In the Figure below, the lower LED is observed to be shining with a higher intensity and the servo arms are tilted away from the Z_B axis in the direction opposite to the pitch. Hence the response of the prototype corresponds to the anticipated response.

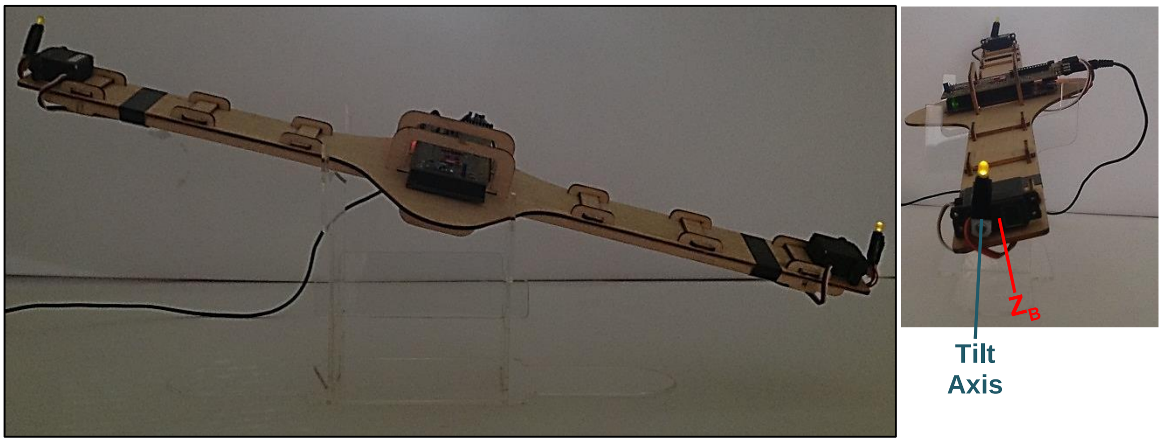


Figure 5.17: Physical model behaviour under ±15º roll and ±15º pitch destabilisation

CHAPTER SIX: CONCLUSION

6.1 Conclusion

This project presents the successful development of a small VTOL UAV system that is to be integrated into AMTL's Guardian surveillance UAV and will serve as a framework for further development of small VTOL UAV systems.

6.1.1 Propulsion characterisation and configuration

To further the development of this VTOL system, it was required that a suitable set of propulsion system be identified and characterised. Thus a propulsion characterisation experiment was designed to measure the thrust and torque developed by a set of propulsion components in the form of an internal combustion engine or an electric motor driving a propeller.

This characterisation equipment was designed for an internal combustion engine or an electric motor of up to a 7.5kW rated power and a propeller of up to 27 inches in diameter, and can measure up to 200N of thrust and 150Nm of torque from the propulsion system.

The characterisation experiment results identified that the propulsion set of Turnigy C6374 – 200 brushless out runner electric motor driving a 22 x 10 inch three bladed propeller will provide approximately 79N (8kg) of thrust at 80% throttle (4250rpm). Therefore, two of this propulsion set would satisfy the platform requirement of 12kg MTOM. These results also showed that the turning moment developed by the spinning propellers about their axis of rotation will be insufficient in maintaining yaw control for this platform.

After a comparison of VTOL methods based on objectives and design requirements of this project, the tilt-rotor was seen as the most suitable VTOL method. Four tilt-rotor propulsion configuration concepts were then proposed, these concepts were also compared for suitability to design objectives. The comparison showed a variable speed twin prop concept as the optimal propulsion configuration solution.

6.1.2 Control algorithm and hardware

As the development of VTOL systems entails the development of a control sub system to achieve stability, a stability augmented control algorithm was developed for the chosen propulsion configuration. This control algorithm, as discussed in the first section of Chapter 3, are a set of equations that provide for the control command allocation (Equation 3.1), attitude correction (Equation 3.2), system propulsion dynamics (Equation 3.3, Equation 3.4 and Equation 3.5), and for simulation purposes the rigid body dynamics (Equation 1.6, Equation 1.7, Equation 1.8 and Equation 1.9) of the vehicle.

Chapter 4 proposed a prototype control hardware and software for the VTOL platform. The hardware uses a dedicated microcontroller for decoding command input therefore freeing up computational resources on the main processor.

The control hardware also features an inertia measurement unit of 3-axis accelerometers, 3-axis angular rate sensors, 3-axis magnetometer and a pressure sensor. These are used for attitude estimation. The main controller reads the sensor data and estimates attitude, adds the response of the estimated attitude to the control command and outputs the control signals to the propulsion hardware.

6.1.3 System Validation

The first section of Chapter 5 provides a validation of the suitability and functionality of the stabilisation augmented control algorithm using a Simulink model of the motion dynamics of the VTOL platform. This validation was executed using five nonlinear simulations that observe the altitude, altitude rate, roll, pitch and yaw-rate responses to destabilizations conducted for 5 seconds each. The results of these simulations showed the developed control algorithm and the proposed stabilisation routine to be able to provide stable control of the system. The results also proved the algorithm to be capable of handling multi-mode testability.

The second section provides demonstration of the prototype control hardware and the control software using a scaled prototype of the variable speed tilt-prop VTOL propulsion configuration.

6.2 Recommendation

Although the stability of the platform was proved through the validation simulation results, the time to stability of the system can be enhanced if a more capable propulsion set is used. Hence the characterisation experiment needs to be executed for wider range of propulsion set. In addition to the above, work still has to be done in providing a positioning stability algorithm for the platform. These enhancements should then be incorporated into the simulation model for further investigations in improving the time to stability.

Without a fully capable control hardware and software the developed platform cannot exist in the physical form. Therefore, further work must still be done towards developing a control hardware and software.

REFERENCES

Acuity Technologies Inc. 2011. AT-10 VTOL Hybrid Tactical UAS. *acuitytx.* [Online] Acuity Technologies Inc., 2011. [Cited: 15 July 2011.] http://www.acuitytx.com/pages/AT-10.htm.

airforce-technology.com. 2011. Predator RQ-1 / MQ-1 / MQ-9 Reaper - Unmanned Aerial Vehicle (UAV), USA. *airforce-technology.com.* [Online] Net Resources International, 2011. [Cited: 24 August 2011.] http://www.airforce-technology.com/projects/predator-uav/.

Allen, William H. 1965. *Dictionary of Technical Terms for Aerospace Use. NASA SP-7.* Washington, D.C. : NASA, 1965. p. 57. 1965NASSP...7,,,,,A.

American Institute of Aeronautics and Astronautics, Inc. 2011. 2011 Worldwide UAV Roundup. *American Institute of Aeronautics and Astronautics, Inc.* [Online] March 2011. http://www.aiaa.org/images/PDF/WilsonChart.pdf.

Aurora Flight Sciences Corporation. 2009. Excalibur. *Aurora Flight Sciences.* [Online] Aurora Flight Sciences Corporation, 2009. [Cited: 20 July 2011.] http://www.aurora.aero/TacticalSystems/Excalibur.aspx.

Australian Defence Force. 2002. *Fundamentals of Australian Air Power.* s.l. : Aerospace Centre, 2002. Vol. IV.

Bang, Hyochoong, Tahk, Min-Jea and Choi, Hyung-Don. 2003. Large angle attitude control of spacecraft with actuator saturation. *Control Engineering Practice.* September 2003, Vol. 11, 9, pp. 989-997.

Bell Helicopter Textron INC. 2005. Eagle Eye Pocket Guide. *bellhelicopter.* [Online] June 2005. [Cited: 12 July 2011.] http://epic.org/privacy/surveillance/spotlight/0805/eagle.pdf.

Bell Helicopter Textron Inc. 2011. The Bell-Boeing V-22. *Bell Helicopter: A Textron Company.* [Online] Bell Helicopter Textron Inc., 2011. [Cited: 25 July 2011.] http://www.bellhelicopter.com/en_US/Military/Bell-BoeingV-22/1291148374339.html#/?tab=features-tab.

Clark, Robert. 2011. AT-10 Electric/HF Hybrid VTOL UAS. *acuitytx.* [Online] 2011. [Cited: 14 July 2011.] http://www.acuitytx.com/pdf/Acuity%20Technologies%20AT-10%20Brief.pdf.

Cordoba, Mario Andres. 2007. Attitude and heading refernce system I-AHRS for the EFIGENIA autonomous unmanned aerial vehicles UAV based on MEMS sensor and a neural network strategy for attitude estimation. June 2007, pp. 1- 8.

Crane, Dale. 1997. *Dictionary of Aeronautical Terms.* 3rd ed. Newcastle : Aviation Supplies & Academics, 1997. p. 224. ISBN 1-56027-287-2.

Ehrhard, Thomas P. 2010. *Air Force UAVs: The Secret History.* 1st ed. s.l. : Mitchell Institute Press, 2010. p. 6.

Fabiani, P, et al. 2007. Autonomous flight and navigation of VTOL UAVs: from autonomy demonstrations to out-of-sight flights. *Aerospace Science and Technology.* 2007, Vol. 11, pp. 183-193.

GH Craft. 2009. QTW-UAS FS4. *GH Craft.* [Online] 2009. http://www.ghcraft.com/QTW/pdf/081001_QTW_FS4e.pdf.

Ghosh, Arun K. 2013. *INTRODUCTION TO CONTROL SYSTEMS.* Delhi : PHI Learning Private Limited, 2013. ISBN-978-81-203-4820-2.

Gravity based online calibration for monolithic triaxial accelerometers' gain and offset drift. **Wu, Z.C., Wang, Z.F. and Ge, Y. 2002.** 2002. Intelligent Control and Automation, 2002. Proceedings of the 4th World Congress on. Vol. 3, pp. 2171-2175. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1021471&isnumber=21964. doi: 10.1109/WCICA.2002.1021471.

Guerrero-Castellanos, J.F., et al. 2011. Bounded attitude control of rigid bodies: Real-time experimentation to a quadrotor mini-helicopter. *Control Engineering Practice.* August 2011, Vol. 19, 8, pp. 790 - 797.

Guided System Technologies Inc. 2009. SiCX 300V. *guidedsys.* [Online] Guided System Technologies Inc, 2009. [Cited: 20 July 2011.] http://www.guidedsys.com/VTOL/sicx-300v-uav.php.

Hirschberg, Mike. 1999. Bell Designs Are Accelerating at Full Tilt. *AHS International: The Vertical Flight Society.* [Online] Winter 1999. [Cited: 15 July 2011.] http://www.vtol.org/vertiflite/.

Israel Aerospace Industries Ltd. 2002. MINI PANTHER FIXED WING VTOL MINI UAS. *IAI (Israel Aerospace Industries).* [Online] Israel Aerospace Industries Ltd., 2002. [Cited: 15 July 2011.] http://www.iai.co.il/35673-41637-en/BusinessAreas_UnmannedAirSystems_PantherFamily.aspx?btl=1.

Julier, Simon J. and Uhlmann, Je®rey K. 1997. A new extension of the Kalman filter to nonlinear systems. *International Symposium on Aerospace/Defense Sensing, Simulation and Controls 3.* 1997, p. 1.

Kalman, R. E. 1960. A new approach to linear filtering and prediction problems. *Transaction of the ASME, Series D, Journal of Basic Engineering.* March 1960, Vol. 82, pp. 35-45.

Layton, Otis Franklin. 2007. *Manned/unmanned V.T.O.L. flight vehicle. 20070246601* Bonney Lake, WA, US, 25 October 2007. Hardware.

Looney, Mark. 2010. A simple calibration for MEMS gyroscopes. *EDN Europe.* [Online] 01 July 2010. [Cited: 11 January 2012.] http://www.edn-europe.com/asimplecalibrationformemsgyroscopes+article+4204+Europe.html.

Madgwick, Sebastian O.H., Harrison, Andrew J.L and Vaidyanathan, Ravi. 2011. Estimation of IMU and MARG orientation using a gradient descent algorithm. *2011 IEEE International Conference on Rehabilitation Robotics.* June 29, 2011-July 1 2011, pp. 1 - 7.

Mahony, Robert, Hamel, Tarek and Pflimlin, Jean-Michel. 2008. Nonlinear Complementary Filters on the Special Orthogonal Group. [ed.] P J Antsaklis. *IEEE Transactions on Automatic Control.* June 2008, Vol. 53, 5, pp. 1203 - 1218. (http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4608934&tag=1).

Marcus UAV Corp. 2010. Devil Ray UAV Airframe. *MarcusUAV.* [Online] Marcus UAV Corp, 2010. [Cited: 24th August 2011.] http://www.marcusuav.com/devilrayairframe.htm.

Marcus UAV Corp. 2010. Zephyr UAV Airframe. *MarcusUAV.* [Online] Marcus UAV Corp, 2010. [Cited: 18 July 2011.] http://www.marcusuav.com/zephyrairframe.htm.

Mellinger, Daniel, Shomin, Michael and Kumar, Vijay. 2010. Control of Quadrotors for Robust Perching and Landing. *International Powered Lift Conference.* October 2010.

Ministry of Defence, UK. 2008. Memorandum from the Ministry of Defence, in Commons Select Commitee on Defence - ISTAR 01. London : Hansard, 2008.

Peng, Kemao, et al. 2007. Design and Implementation of a Fully Autonomous Flight Control System for a UAV Helicopter. *Control Conference.* 2007, July 26 2007-June 31 2007, pp. 662 - 667.

Raymer, Daniel. 2006. *Aircraft Design: A Conceptual Approach.* 4th ed. Reston : American Institute of Aeronautics, Inc.,, 2006. pp. 661, 662. ISBN 1-56347-829-3.

Riddles, Mornay. 2011. *Interview with Researcher.* Cape Town, 9 February 2011.

Royal Air Force Centre for Air Power Studies. 2009. *British Air and Space Power Doctrine, AP 3000.* 2009. p. 14. ISBN 978-0-9558189-7-2.

Schafroth, D., et al. 2010. Modeling, system identification and robust control of a coaxial micro helicopter. *Control Engineering Practice.* July 2010, Vol. 18, 7, pp. 700 - 711.

Sergio, Salazar-Cruz, Rogelio, Lozano and Juan, Escareño. 2009. Stabilization and Nonlinear Control for a novel Trirotor Mini-aircraft. *Control Engineering Practice.* August 2009, Vol. 17, 8, pp. 886 - 894.

Stevens, Brian L. and Lewis, Frank L. 1992. *Aircraft Control and Simulation.* illustrated. New York : John Wiley & Sons, Inc., 1992. p. 81. ISBN 0471613975.

STMicroelectronics. 2010. Using LSM303DLH for a tilt compensated electronic compass. *STMicroelectronics Web site.* [Online] 1, 02 August 2010. [Cited: 28 December 2011.] http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/APPLICATION_NOTE/CD00269797.pdf. Doc ID 17353 Rev 1.

Stone, H. and Wong, K.C. 1997. Preliminary Design of a Tandem-Wing Tail-Sitter UAV Using Multi-Disciplinary Design Optimisation. *International Aerospace Congress.* February 1997, pp. 707 - 720.

Sung, Kyung Hong. 2003. Fuzzy logic based closed-loop strapdown attitude system for unmanned aerial vehicle (UAV). *Sensors and Actuators A: Physical.* 15 October 2003, Vol. 107, 2, pp. 109-118. (http://www.sciencedirect.com/science/article/pii/S0924424703003534).

Taylor, John W. R. and Munson, Kenneth. 1977. *Jane's Pocket Book of Remotely Piloted Vehicles.* 1st ed. New York : Collier Books, 1977. p. 28. ISBN 0-02080-640-X.

United States Department of Defense (US DoD). April 6, 2009. *FY2009–2034 Unmanned Systems Integrated Roadmap.* Washington DC : US DoD (AT&L), April 6, 2009.

Universal Wing. 2009. "Venturer": Universal Wing's UAV. *Universalwing.* [Online] Universal Wing, 2009. [Cited: 24 August 2011.] http://www.universalwing.com/technology/our-uav.

University of New South Wales. 2008. *Design Development of a Maritime Patrol UAV.* Austrailian Defence Force Academy, School of Aerospace, Civil and Mechanical Engineering. Canberra : s.n., 2008. p. 4.

Watkinson, John. 2004. *The Art of the Helicopter.* Oxford : Elsevier Butterworth-Heinemann, 2004. pp. 19-20. ISBN 0 7506 5715 4.

Westphal, Louis C. 2001. *HANDBOOK OF CONTROL SYSTEMS ENGINEERING.* Dordrecht : Kluwer Academic Publishers, 2001. 2001044307.

Wilson, J.R. 2011. UAV Roundup 2011. *AEROSPACE AMERICA.* MARCH, 2011.

Yoo, Dong-Wan, et al. 2010. Dynamic Modeling and Stabilization Techniques for Tri-Rotor Unmanned Aerial Vehicles. *International Journal of Aeronautical & Space Science.* 11, 2010, Vol. 3, pp. 167 - 174.

APPENDIX A: PROPULSION CHARACTERISATION RESULTS

A1: Test Bench Calibration

Table A.1: Test bench calibration results

Applied Load		Gauge Reading	Calculated Load (Linear Fit)	Calculated Load (2nd order PolyFit)	Variance (Linear Fit)	Variance (Poly Fit)
[kg]	[N]	[N]	[N]	[N]	[N]	[N]
0	0	0	2.16	1.19	-2.16	-1.19
2	19.62	16.11	19.61	19.11	0.01	0.51
4	39.24	33.64	38.61	38.67	0.63	0.57
6	58.86	51.9	58.39	59.12	0.47	-0.26
8	78.48	69.85	77.84	79.28	0.64	-0.80
10	98.1	88.2	97.72	99.95	0.38	-1.85
12	117.72	106.4	117.43	120.53	0.29	-2.81
14	137.34	124.4	136.94	140.94	0.40	-3.60
16	156.96	142.4	156.44	161.42	0.52	-4.46
18	176.58	160.66	176.22	182.26	0.36	-5.68
20	196.2	179.22	196.33	203.51	-0.13	-7.31
22	215.82	197.52	216.15	224.53	-0.33	-8.71
24	235.44	215.62	235.76	245.39	-0.32	-9.95
26	255.06	234.14	255.83	266.80	-0.77	-11.74

Table A.2: Curve fit parameters

Linear fit	
a	1.0834
b	2.1611
R^2	0.9999
2nd order Polynomial	
a	0.0001
b	1.111
c	1.1865

From the variance between the actual load and the load calculated from the results of the curves fitted to the calibration data, it is seen that the linear fit provides a better fit to the data.

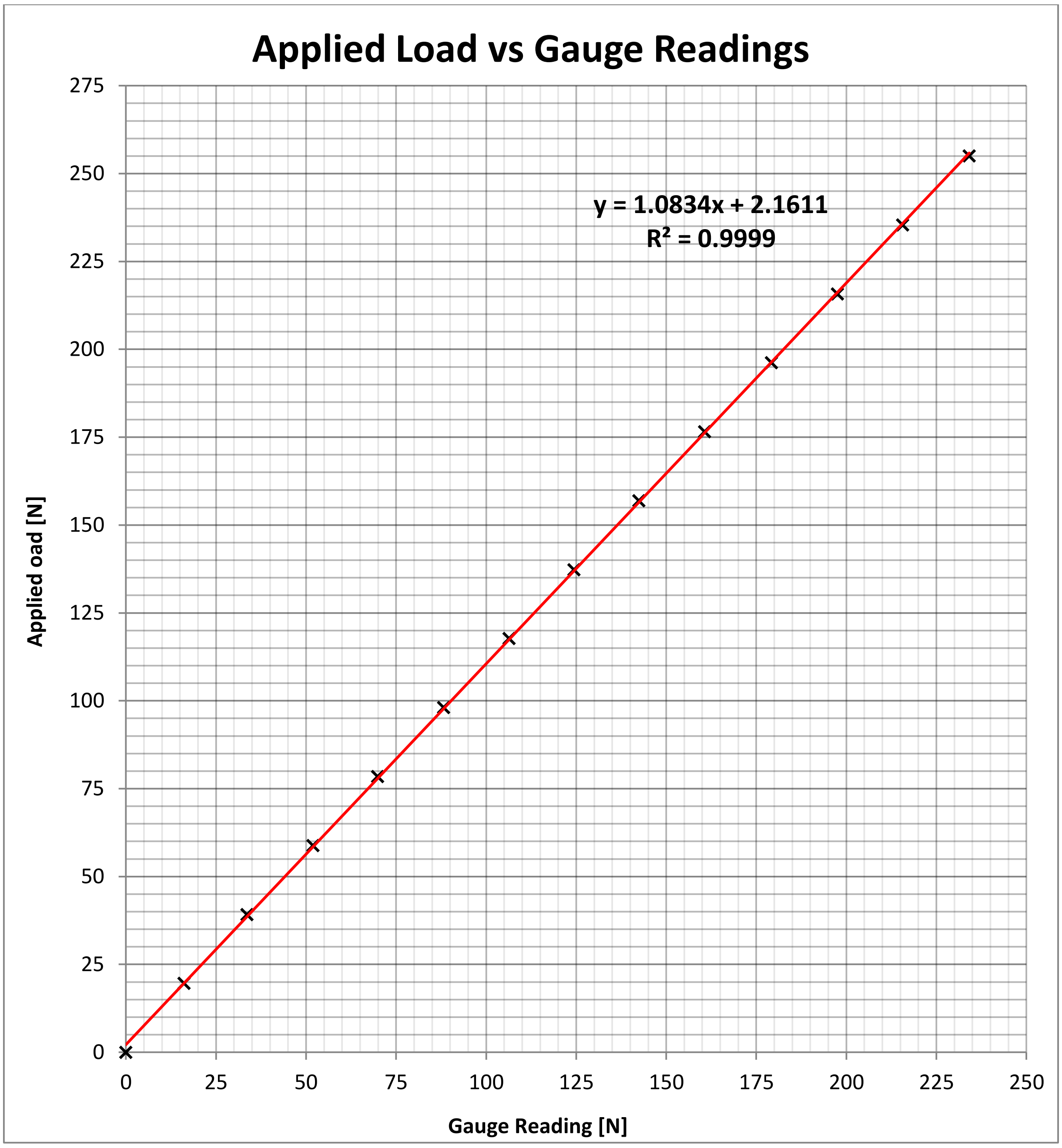


Figure A.1: Test bench calibration results

The MATLAB script used for the generation of the plot and mathematical relations from the experimental data is shown below.

```
%---------------------------------   START   ------------------------------
function createfigure(X1, Y1)

%  Clear exiting data
clear all

%  Load Test Bench Calibration data
load BenchCalib

% Create figure
testbenchcalibration = figure;

% Create axes
axes1 = axes('Parent',testbenchcalibration);
hold(axes1,'all');

% Create scatter
scatter1 = scatter(GaugeReading,ActualLoad,'MarkerEdgeColor',[0 0
0],'Marker','x','DisplayName',' Gauge Reading vs Actual Load');

% Create xlabel
xlabel('Gauge Reading (N)','FontWeight','bold');

% Create ylabel
ylabel('Actual Load (N)','FontWeight','bold');

% Create title
title('Test Bench Calibration','FontWeight','bold','FontSize',12);

% Get xdata from plot
xdata1 = get(scatter1, 'xdata');
% Get ydata from plot
ydata1 = get(scatter1, 'ydata');
% Make sure data are column vectors
xdata1 = xdata1(:);
ydata1 = ydata1(:);

% Remove NaN values and warn
nanMask1 = isnan(xdata1(:)) | isnan(ydata1(:));
if any(nanMask1)
    warning('GenerateMFile:IgnoringNaNs', ...
        'Data points with NaN coordinates will be ignored.');
    xdata1(nanMask1) = [];
    ydata1(nanMask1) = [];
end

% Find x values for plotting the fit based on xlim
axesLimits1 = xlim(axes1);
xplot1 = linspace(axesLimits1(1), axesLimits1(2));

% Preallocate for "Show equations" coefficients
coeffs1 = cell(1,1);

% Find coefficients for polynomial (order = 1)
fitResults1 = polyfit(xdata1, ydata1, 1);


% Evaluate polynomial
yplot1 = polyval(fitResults1, xplot1);

% Save type of fit for "Show equations"
```

```
fittypesArray1(1) = 2;

% Save coefficients for "Show Equation"
coeffs1{1} = fitResults1;

% Plot the fit
fitLine1 = plot(xplot1,yplot1,'DisplayName','   linear','Parent',axes1,...
    'Tag','linear',...
    'Color',[1 0 0]);

% Set new line in proper position
setLineOrder(axes1, fitLine1, scatter1);

% "Show equations" was selected
showEquations(fittypesArray1, coeffs1, 5, axes1);

% Create legend
legend1 = legend(axes1,'show');
set(legend1,'Location','SouthEast');

%-------------------------------------------------------------------------%

function setLineOrder(axesh1, newLine1, associatedLine1)

% Get the axes children
hChildren = get(axesh1,'Children');
% Remove the new line
hChildren(hChildren==newLine1) = [];
% Get the index to the associatedLine
lineIndex = find(hChildren==associatedLine1);
% Reorder lines so the new line appears with associated data
hNewChildren = [hChildren(1:lineIndex-
1);newLine1;hChildren(lineIndex:end)];
% Set the children:
set(axesh1,'Children',hNewChildren);

%-------------------------------------------------------------------------%

function showEquations(fittypes1, coeffs1, digits1, axesh1)

n = length(fittypes1);
txt = cell(length(n + 1) ,1);
txt{1,:} = ' ';
for i = 1:n
    txt{i + 1,:} =
getEquationString(fittypes1(i),coeffs1{i},digits1,axesh1);
end
text(.05,.95,txt,'parent',axesh1, ...
    'verticalalignment','top','units','normalized');

%-------------------------------------------------------------------------%





function [s1] = getEquationString(fittype1, coeffs1, digits1, axesh1)

if isequal(fittype1, 0)
    s1 = 'Cubic spline interpolant';
```

```
elseif isequal(fittype1, 1)
    s1 = 'Shape-preserving interpolant';
else
    op = '+-';
    format1 = ['%s %0.',num2str(digits1),'g*x^{%s} %s'];
    format2 = ['%s %0.',num2str(digits1),'g'];
    xl = get(axesh1, 'xlim');
    fit =  fittype1 - 1;
    s1 = sprintf('y =');
    th = text(xl*[.95;.05],1,s1,'parent',axesh1, 'vis','off');
    if abs(coeffs1(1) < 0)
        s1 = [s1 ' -'];
    end
    for i = 1:fit
        sl = length(s1);
        if ~isequal(coeffs1(i),0) % if exactly zero, skip it
            s1 = sprintf(format1,s1,abs(coeffs1(i)),num2str(fit+1-i),
op((coeffs1(i+1)<0)+1));
        end
        if (i==fit) && ~isequal(coeffs1(i),0)
            s1(end-5:end-2) = []; % change x^1 to x.
        end
        set(th,'string',s1);
        et = get(th,'extent');
        if et(1)+et(3) > xl(2)
            s1 = [s1(1:sl) sprintf('\n     ') s1(sl+1:end)];
        end
    end
    if ~isequal(coeffs1(fit+1),0)
        sl = length(s1);
        s1 = sprintf(format2,s1,abs(coeffs1(fit+1)));
        set(th,'string',s1);
        et = get(th,'extent');
        if et(1)+et(3) > xl(2)
            s1 = [s1(1:sl) sprintf('\n     ') s1(sl+1:end)];
        end
    end
    delete(th);
    % Delete last "+"
    if isequal(s1(end),'+')
        s1(end-1:end) = []; % There is always a space before the +.
    end
    if length(s1) == 3
        s1 = sprintf(format2,s1,0);
    end
end
%-------------------------------  END  -----------------------------------
```

A2: Propulsion Characterisation Results – (Experiment 1)

Table A.3: Thrust data log (Turnigy C6374 – 200 & 22 x 10 inch 3 bladed propeller)

Reading 1				
Speed	**Force**	**Voltage**	**Current**	**E Pwr**
RPM	***N***	***Volts***	***Amps***	***Watts***
0	0	41.25	0	0
1782	11.4	40.94	4	163.76
2391	22	40.63	7.8	316.914
2931	34.3	40.2	13.1	526.62
3411	46	39.65	19.8	785.07
3857	57.3	38.95	28.3	1102.29
4251	71	38.29	38.9	1489.48
4577	81	37.56	49.3	1851.71
4834	91	36.6	60.1	2199.66
5018	101.7	35.78	73.3	2622.67
Time	14:12:16			
Date	15/06/2011			

Reading 2				
Speed	**Force**	**Voltage**	**Current**	**E Pwr**
RPM	***N***	***Volts***	***Amps***	***Watts***
0	0	39.61	0	0
2331	19.3	39.2	7	274.4
2862	31	38.9	12	466.8
3342	43	38.45	18.5	711.325
3771	55	37.9	26.9	1019.51
4165	67	37.65	36.6	1377.99
4491	78	36.67	47.1	1727.16
4748	90	35.88	59.1	2120.51
5022	100	35.18	70.9	2494.26
5228	109	34.54	83.6	2887.54
Time	14:23:27			
Date	15/06/2011			

Table A.3: Continued

Reading 3				
Speed	**Force**	**Voltage**	**Current**	**E Pwr**
RPM	***N***	***Volts***	***Amps***	***Watts***
0	0	38.22	0	0
1645	9.15	37.94	3.7	140.378
2228	18	37.63	7.1	267.173
2742	28.05	37.26	11.8	439.668
3188	40	36.72	17.9	657.288
3617	50.7	36.15	25.4	918.21
3985	61.05	35.5	34.4	1221.2
4302	74	34.83	44.7	1556.9
4602	88	34.2	55.2	1887.84
4868	94	33.43	70.8	2366.84
5057	104	33.06	80.2	2651.41
Time	09:48:04			
Date	23/06/2011			

Table A.4: Torque data log (Turnigy C6374 – 200 & 22 x 10 inch 3 bladed propeller)

Reading 1				
Speed	**Torque**	**Voltage**	**Current**	**E Pwr**
RPM	***Nm***	***Volts***	***Amps***	***Watts***
0	0	41.79	0	0
1088	0.15	41.72	1.5	62.58
1808	0.45	41.49	4.7	195.003
2425	0.8	41.2	8	329.6
2982	1.1	40.78	13.1	534.218
3453	1.6	40.13	21.9	878.847
3908	2.15	39.38	31.6	1244.41
4268	2.5	38.75	39.1	1515.13
4594	3.15	37.92	49.5	1877.04
4868	3.5	36.97	64	2366.08
5108	3.85	36.13	73.9	2670.01
5288	4.15	35.2	86.7	3051.84
Time	01:45:25 PM			
Date	23/06/2011			

Table A.4: Continued

Reading 2				
Speed	**Torque**	**Voltage**	**Current**	**E Pwr**
RPM	***Nm***	***Volts***	***Amps***	***Watts***
0	0	39.4	0	0
1028	0.1	39.31	1.8	70.758
1705	0.35	39.13	3.8	148.694
2297	0.65	38.85	7.2	279.72
2819	1	38.45	12.1	465.245
3291	1.4	37.97	18.4	698.648
3737	1.95	37.36	26.6	993.776
4114	2.3	36.59	38.3	1401.4
4422	2.75	36.02	45.9	1653.32
4731	3.2	35.33	57.3	2024.41
4971	3.55	34.56	69.6	2405.38
5177	4	33.95	81.9	2780.51
Time				
Date	23/06/2011			

Reading 3				
Speed	**Torque**	**Voltage**	**Current**	**E Pwr**
RPM	***Nm***	***Volts***	***Amps***	***Watts***
0	0	38.24	0	0
994	0.15	38.15	1.7	64.855
1645	0.35	37.95	3.6	136.62
2228	0.6	37.65	7	263.55
2742	1	37.25	11.8	439.55
3188	1.35	36.7	18.4	675.28
3617	1.8	36.17	25.7	929.569
3985	2.25	35.54	34.5	1226.13
4337	2.7	34.72	47.5	1649.2
4594	3.15	34.22	55.4	1895.79
4868	3.5	33.6	67.8	2278.08
5091	3.85	33.08	79.5	2629.86
Time				
Date	23/06/2011			

A3: Experiment 1 Data Analysis Output

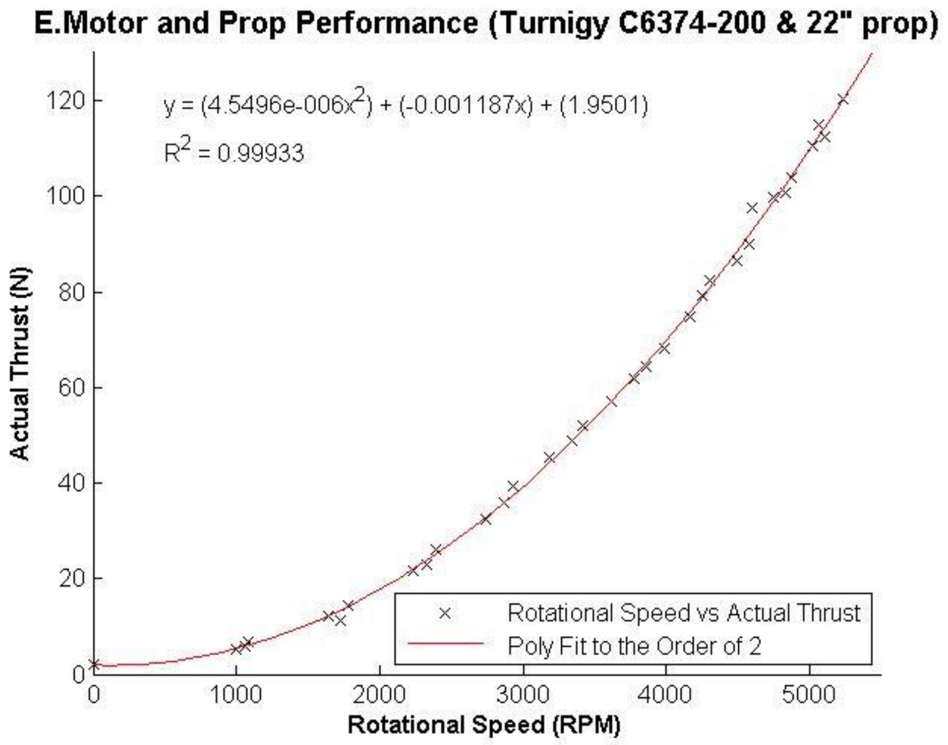


Figure A.2: Thrust vs rotational speed (Turnigy C6374 – 200 & 22" prop)

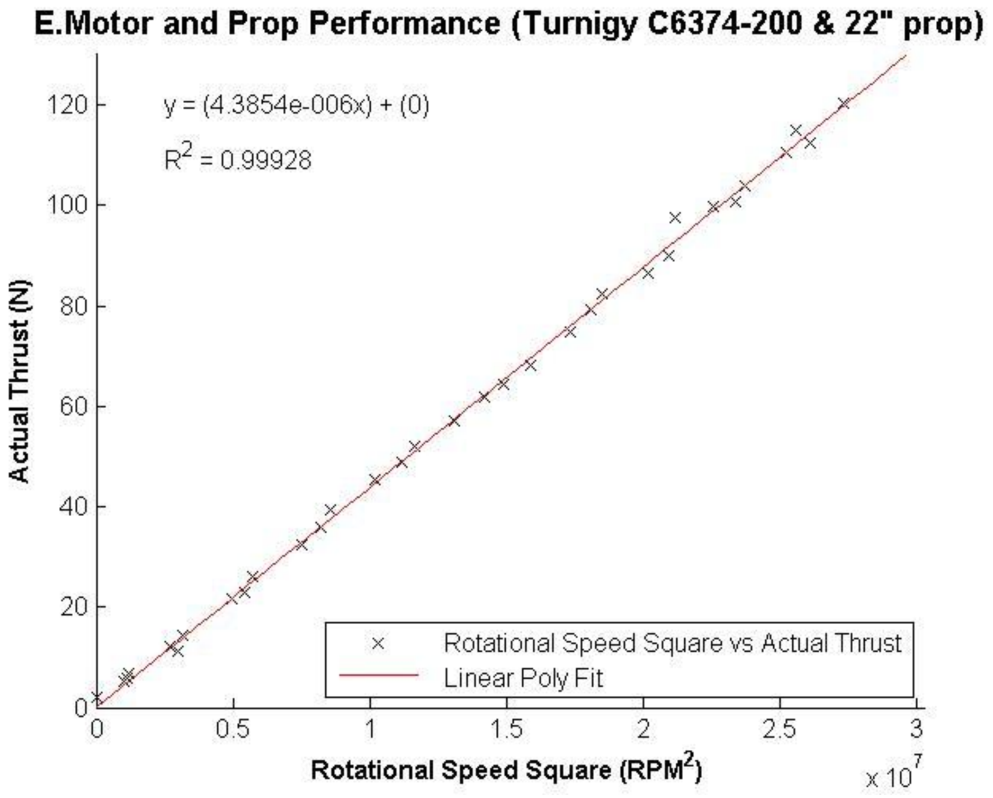


Figure A.3: Thrust vs square of rotational speed (Turnigy C6374 – 200 & 22" prop)

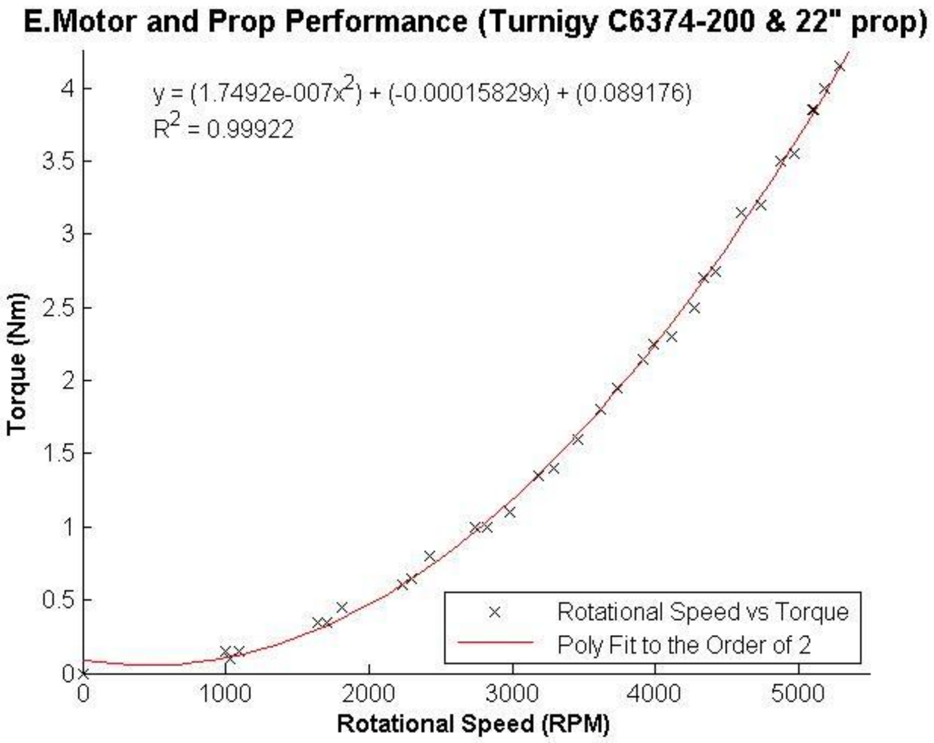


Figure A.4: Torque vs rotational speed (Turnigy C6374 – 200 & 22" prop)

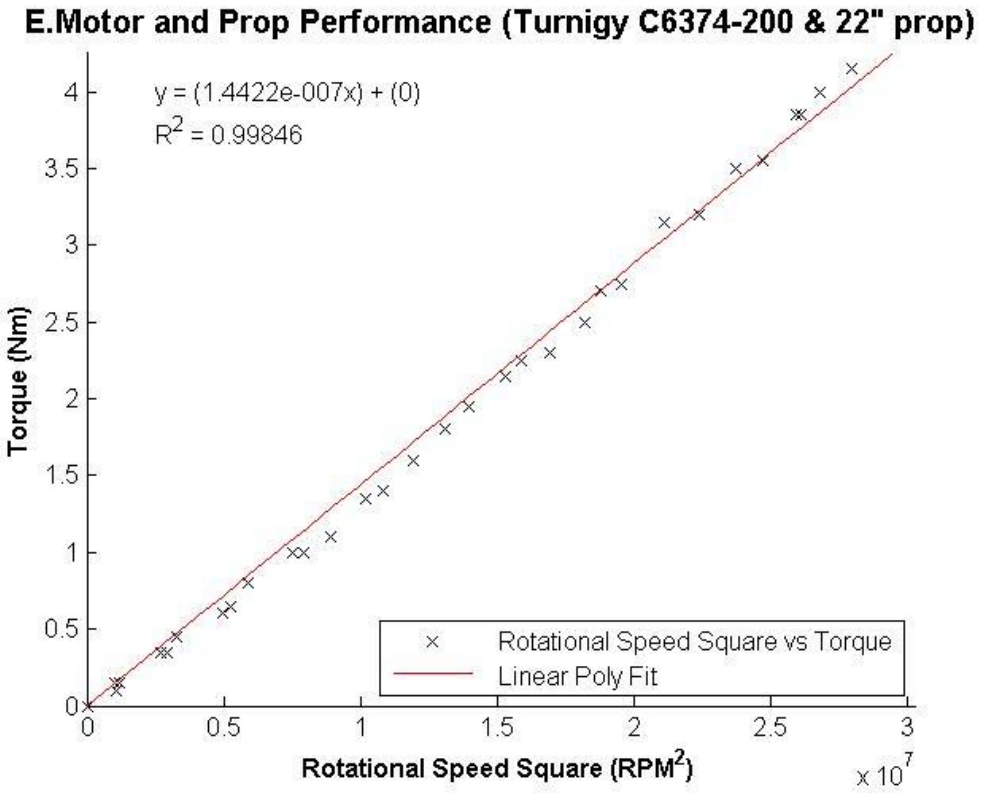


Figure A.5: Torque vs square of rotational speed (Turnigy C6374 – 200 & 22" prop)

The MATLAB script used for the generation of the plot and mathematical relations from the experimental data is shown below.

```
%----------------------------------   START   ------------------------------
%THRUST ---------------------------------------------------------------------

%  Clear all existing variables
clear all;

%  Load data
load Dataset01

%  Define Test Bench Clibration Linear fit coefficients
tstbnchlinfit.m=1.0834;
tstbnchlinfit.b=2.1611;

%  Clibrate thrust
thrust.calibthrust=(tstbnchlinfit.m*thrust.thrust)+tstbnchlinfit.b;

%------------------------------  1  ---------------------------------------

% Create figure
thrust.performance_charaterisation = figure;

% Create axes
thrust.axes1 = axes('Parent',thrust.performance_charaterisation);
hold(thrust.axes1,'all');

% Create plot
thrust.plot1=plot(thrust.speed,thrust.calibthrust,'x','MarkerEdgeColor',[0,
0,0],'DisplayName','   Rotational Speed vs Actual Thrust');
    % Create X Label
    xlabel('Rotational Speed (RPM)', 'FontWeight','Bold');
    % Create Y Label
    ylabel('Actual Thrust (N)', 'FontWeight','Bold');
    % Create title
    title('E.Motor and Prop Performance (Turnigy C6374-200 & 22"
prop)','FontWeight','bold','FontSize',12);
    % set axis limti
    axis([0 5500 0 130]);
    % Show grid
    % grid on;

% Polyfit of thrust.speed and thrust.calibthrust to the second order
thrust.polyfit=polyfit(thrust.speed,thrust.calibthrust,2);
    % Evaluating the polyfit and saving output
    thrust.output=polyval(thrust.polyfit,thrust.speed);

% Calculating the R squared value for linear poly fit 1
thrust.Correlation= corrcoef(thrust.calibthrust, thrust.output);
thrust.R2=thrust.Correlation(2);

% Generate fit data for plot
    % X data
    thrust.X=linspace(0,5500,50);
    % Y data
    thrust.Y=polyval(thrust.polyfit,thrust.X);

% Plot the fit
```

```
thrust.fitLine1 = plot(thrust.X,thrust.Y,'DisplayName','   Poly Fit to the
Order of 2','Parent',thrust.axes1,...
    'Tag','Poly Fit to the Order of 2',...
    'Color',[1 0 0]);
    % Create legend
    legend('show');
    set(legend,'Location','SouthEast');
    % Show equation
    text(500,120,['y = (',num2str(thrust.polyfit(1)),'x^2) +
(',num2str(thrust.polyfit(2)),'x) + (',num2str(thrust.polyfit(3)),')']);
    % Show R2
    text(500,110,['R^2 = ',num2str(thrust.R2)]);

 % Turn off axes hold
 hold(thrust.axes1,'off');

%---------------------------------  2  ------------------------------------

%  Square Rotational Speed
thrust.speedsqr=thrust.speed.^2;

% Create figure
thrust.performance_charaterisation1 = figure;

% Create axes
thrust.axes2 = axes('Parent',thrust.performance_charaterisation1);
hold(thrust.axes2,'all');

% Create plot
thrust.plot2=plot(thrust.speedsqr,thrust.calibthrust,'x','MarkerEdgeColor',
[0,0,0],'DisplayName','   Rotational Speed Square vs Actual Thrust');
    % Create X Label
    xlabel('Rotational Speed Square (RPM^2)', 'FontWeight','Bold');
    % Create Y Label
    ylabel('Actual Thrust (N)', 'FontWeight','Bold');
    % Create title
    title('E.Motor and Prop Performance (Turnigy C6374-200 & 22"
prop)','FontWeight','bold','FontSize',12);
    % set axis limti
    axis([0 5500^2 0 130]);
    % Show grid
    % grid on;

% linear polyfit of thrust.speedsqr and thrust.calibthrust
thrust.linpolyfit1=polyfit(thrust.speedsqr,thrust.calibthrust,1);
% Evaluating the linear polyfit and saving output
thrust.output1=polyval(thrust.linpolyfit1,thrust.speedsqr);
% Calculating the R squared value for linear poly fit 1
thrust.Correlation1 = corrcoef(thrust.calibthrust, thrust.output1);
thrust.R2_1=thrust.Correlation1(2);

% Generate fit data for plot
    % X data
    thrust.X1=linspace(0,5500^2,50);
    % Y data
    thrust.Y1=polyval(thrust.linpolyfit1,thrust.X1);

% Plot the fit
thrust.fitLine2 = plot(thrust.X1,thrust.Y1,'DisplayName','   Linear Poly
Fit','Parent',thrust.axes2,...
    'Tag','Linear Poly Fit',...
    'Color',[1 0 0]);
    % Create legend
    legend('show');
    set(legend,'Location','SouthEast');
    % Show equation
    text(2500000,120,['y = (',num2str(thrust.linpolyfit1(1)),'x) +
(',num2str(thrust.linpolyfit1(2)),')']);
    % Show R2
    text(2500000,110,['R^2 = ',num2str(thrust.R2_1)]);

 % Turn off axes hold
 hold(thrust.axes2,'off');

%--------------------------------  3  -------------------------------------

% Create figure
thrust.performance_charaterisation2 = figure;

% Create axes
thrust.axes3 = axes('Parent',thrust.performance_charaterisation2);
hold(thrust.axes3,'all');

% Create plot
thrust.plot3=plot(thrust.speedsqr,thrust.calibthrust,'x','MarkerEdgeColor',
[0,0,0],'DisplayName','   Rotational Speed Square vs Actual Thrust');
    % Create X Label
    xlabel('Rotational Speed Square (RPM^2)', 'FontWeight','Bold');
    % Create Y Label
    ylabel('Actual Thrust (N)', 'FontWeight','Bold');
    % Create title
    title('E.Motor and Prop Performance (Turnigy C6374-200 & 22"
prop)','FontWeight','bold','FontSize',12);
    % set axis limti
    axis([0 5500^2 0 130]);
    % Show grid
    % grid on;

% linear polyfit of thrust.speedsqr and thrust.calibthrust with intercept
% at zero
thrust.slope=thrust.speedsqr\thrust.calibthrust;
thrust.linpolyfit2=[thrust.slope,0];
% Evaluating the linear polyfit  with intercept at zero and saving output
thrust.output2=polyval(thrust.linpolyfit2,thrust.speedsqr);
% Calculating the R squared value for linear poly fit 2
thrust.Correlation2 = corrcoef(thrust.calibthrust, thrust.output2);
thrust.R2_2=thrust.Correlation2(2);

% Generate fit data for plot
    % X data
    thrust.X1=linspace(0,5500^2,50);
    % Y data
    thrust.Y2=polyval(thrust.linpolyfit2,thrust.X1);

% Plot the fit
thrust.fitLine2 = plot(thrust.X1,thrust.Y2,'DisplayName','   Linear Poly
Fit','Parent',thrust.axes3,...
    'Tag','Linear Poly Fit',...
    'Color',[1 0 0]);
    % Create legend
    legend('show');
    set(legend,'Location','SouthEast');
    % Show equation
    text(2500000,120,['y = (',num2str(thrust.linpolyfit2(1)),'x) +
(',num2str(thrust.linpolyfit2(2)),')']);
    % Show R2
    text(2500000,110,['R^2 = ',num2str(thrust.R2_2)]);

 % Turn off axes hold
 hold(thrust.axes2,'off');


  %TORQUE -----------------------------------------------------------------
%-------------------------------   1  -------------------------------------

% Create figure
torque.performance_charaterisation = figure;

% Create axes
torque.axes1 = axes('Parent',torque.performance_charaterisation);
hold(torque.axes1,'all');

% Create plot
torque.plot1=plot(torque.speed,torque.torque,'x','MarkerEdgeColor',[0,0,0],
'DisplayName','   Rotational Speed vs Torque');
    % Create X Label
    xlabel('Rotational Speed (RPM)', 'FontWeight','Bold');
    % Create Y Label
    ylabel('Torque (Nm)', 'FontWeight','Bold');
    % Create title
    title('E.Motor and Prop Performance (Turnigy C6374-200 & 22"
prop)','FontWeight','bold','FontSize',12);
    % set axis limti
    axis([0 5500 0 4.25]);
    % Show grid
    % grid on;

% Polyfit of torque.speed and torque.torque to the second order
torque.polyfit=polyfit(torque.speed,torque.torque,2);
    % Evaluating the polyfit and saving output
    torque.output=polyval(torque.polyfit,torque.speed);

% Calculating the R squared value for linear poly fit 1
torque.Correlation= corrcoef(torque.torque, torque.output);
torque.R2=torque.Correlation(2);

% Generate fit data for plot
    % X data
    torque.X=linspace(0,5500,50);
    % Y data
    torque.Y=polyval(torque.polyfit,torque.X);

% Plot the fit
torque.fitLine1 = plot(torque.X,torque.Y,'DisplayName','   Poly Fit to the
Order of 2','Parent',torque.axes1,...
    'Tag','Poly Fit to the Order of 2',...
    'Color',[1 0 0]);
    % Create legend
    legend('show');
    set(legend,'Location','SouthEast');
    % Show equation
    text(500,4,['y = (',num2str(torque.polyfit(1)),'x^2) +
(',num2str(torque.polyfit(2)),'x) + (',num2str(torque.polyfit(3)),')']);
    % Show R2
    text(500,3.75,['R^2 = ',num2str(torque.R2)]);

 % Turn off axes hold
 hold(torque.axes1,'off');

%-----------------------------------  2  -------------------------------------

%  Square Rotational Speed
torque.speedsqr=torque.speed.^2;

% Create figure
torque.performance_charaterisation1 = figure;

% Create axes
torque.axes2 = axes('Parent',torque.performance_charaterisation1);
hold(torque.axes2,'all');

% Create plot
torque.plot2=plot(torque.speedsqr,torque.torque,'x','MarkerEdgeColor',[0,0,
0],'DisplayName','   Rotational Speed Square vs Torque');
    % Create X Label
    xlabel('Rotational Speed Square (RPM^2)', 'FontWeight','Bold');
    % Create Y Label
    ylabel('Torque (Nm)', 'FontWeight','Bold');
    % Create title
    title('E.Motor and Prop Performance (Turnigy C6374-200 & 22"
prop)','FontWeight','bold','FontSize',12);
    % set axis limti
    axis([0 5500^2 0 4.25]);
    % Show grid
    % grid on;

% linear polyfit of torque.speedsqr and torque.torque
torque.linpolyfit1=polyfit(torque.speedsqr,torque.torque,1);
% Evaluating the linear polyfit and saving output
torque.output1=polyval(torque.linpolyfit1,torque.speedsqr);
% Calculating the R squared value for linear poly fit 1
torque.Correlation1 = corrcoef(torque.torque, torque.output1);
torque.R2_1=torque.Correlation1(2);

% Generate fit data for plot
    % X data
    torque.X1=linspace(0,5500^2,50);
    % Y data
    torque.Y1=polyval(torque.linpolyfit1,torque.X1);

% Plot the fit
torque.fitLine2 = plot(torque.X1,torque.Y1,'DisplayName','   Linear Poly
Fit','Parent',torque.axes2,...
    'Tag','Linear Poly Fit',...
    'Color',[1 0 0]);
    % Create legend
    legend('show');
    set(legend,'Location','SouthEast');
    % Show equation
    text(2500000,4,['y = (',num2str(torque.linpolyfit1(1)),'x) +
(',num2str(torque.linpolyfit1(2)),')']);
    % Show R2
    text(2500000,3.75,['R^2 = ',num2str(torque.R2_1)]);

 % Turn off axes hold
 hold(torque.axes2,'off');

%---------------------------------  3  -------------------------------------

% Create figure
torque.performance_charaterisation2 = figure;

% Create axes
torque.axes3 = axes('Parent',torque.performance_charaterisation2);
hold(torque.axes3,'all');

% Create plot
torque.plot3=plot(torque.speedsqr,torque.torque,'x','MarkerEdgeColor',[0,0,
0],'DisplayName','   Rotational Speed Square vs Torque');
    % Create X Label
    xlabel('Rotational Speed Square (RPM^2)', 'FontWeight','Bold');
    % Create Y Label
    ylabel('Torque (Nm)', 'FontWeight','Bold');
    % Create title
    title('E.Motor and Prop Performance (Turnigy C6374-200 & 22"
prop)','FontWeight','bold','FontSize',12);
    % set axis limti
    axis([0 5500^2 0 4.25]);
    % Show grid
    % grid on;

% linear polyfit of torque.speedsqr and torque.torque with intercept
% at zero
torque.slope=torque.speedsqr\torque.torque;
torque.linpolyfit2=[torque.slope,0];
% Evaluating the linear polyfit  with intercept at zero and saving output
torque.output2=polyval(torque.linpolyfit2,torque.speedsqr);
% Calculating the R squared value for linear poly fit 2
torque.Correlation2 = corrcoef(torque.torque, torque.output2);
torque.R2_2=torque.Correlation2(2);

% Generate fit data for plot
    % X data
    torque.X1=linspace(0,5500^2,50);
    % Y data
    torque.Y2=polyval(torque.linpolyfit2,torque.X1);

% Plot the fit
torque.fitLine2 = plot(torque.X1,torque.Y2,'DisplayName','   Linear Poly
Fit','Parent',torque.axes3,...
    'Tag','Linear Poly Fit',...
    'Color',[1 0 0]);
    % Create legend
    legend('show');
    set(legend,'Location','SouthEast');
    % Show equation
    text(2500000,4,['y = (',num2str(torque.linpolyfit2(1)),'x) +
(',num2str(torque.linpolyfit2(2)),')']);
    % Show R2
    text(2500000,3.75,['R^2 = ',num2str(torque.R2_2)]);

 % Turn off axes hold
 hold(torque.axes2,'off');

%-------------------------------  END  -----------------------------------
```

A4: Propulsion Characterisation Results – (Experiment 2)

Table A.5: Thrust data log (Turnigy C637 – 200 + 20 x 10 inch 3 bladed propeller)

Pulse Width	**Reading 1**				
	Speed	**Force**	**Voltage**	**Current**	**E Pwr**
uS	***RPM***	***N***	***Volts***	***Amps***	***Watts***
1200	0	0	41.8	0	0
1250	1114	1.05	41.68	1.4	58.352
1300	1834	6.1	41.52	3.4	141.168
1350	2476	13.4	41.29	6.5	268.385
1400	3059	21.5	40.9	11.1	453.99
1450	3582	31.5	40.35	17.9	722.265
1500	4088	43.5	39.79	24.7	982.813
1550	4491	53.6	39.08	33.2	1297.46
1600	4851	65.2	39.31	42	1651.02
1650	5194	76	37.59	52.7	1980.99
1700	5468	87.3	36.7	63.8	2341.46
1750	5708	94.5	35.78	74.2	2654.88
Time	03:08:58 PM				
Date	24/06/2011				
Pulse Width	**Reading 2**				
	Speed	**Force**	**Voltage**	**Current**	**E Pwr**
uS	***RPM***	***N***	***Volts***	***Amps***	***Watts***
1200	0	0	39.61	0	0
1250	1071	1.6	39.5	1.3	51.35
1300	1748	5.9	39.33	3.1	121.923
1350	2365	12.7	39.06	6.1	238.266
1400	2914	21	38.65	10.5	405.825
1450	3428	30.8	38.2	15.5	592.1
1500	3908	40.5	37.63	22.2	835.386
1550	4302	51	36.95	30.6	1130.67
1600	4680	61.4	36.29	40	1451.6
1650	5005	72	35.68	48.6	1734.05
1700	5288	81.3	34.95	59.5	2079.53
1750	5545	89.7	34.31	70.4	2415.42
Time	10:46:44 AM				
Date	28/06/2011				

Table A.5: Continued

Pulse Width	Reading 3				
	Speed	Force	Voltage	Current	E Pwr
uS	*RPM*	*N*	*Volts*	*Amps*	*Watts*
1200	0	0	38.27	0	0
1250	1011	2	38.2	1.3	49.66
1300	1680	6.8	38.06	3.1	117.986
1350	2280	12.5	37.87	5.9	223.433
1400	2828	21	37.58	9.6	360.768
1450	3333	29.6	37.2	15	558
1500	3813	37.05	36.74	21.7	797.258
1550	4217	48.5	36.22	29.1	1054
1600	4594	57.3	35.63	37.9	1350.38
1650	4937	69	35.08	47.1	1652.27
1700	5228	79	34.47	57.9	1995.81
1750	5485	88	33.88	68.3	2314
Time	11:32:59 AM				
Date	28/06/2011				

Table A.6: Torque data log (Turnigy C637 – 200 + 20 x 10 inch 3 bladed propeller)

Pulse Width	Reading 1				
	Speed	Torque	Voltage	Current	E Pwr
uS	*RPM*	*Nm*	*Volts*	*Amps*	*Watts*
1200	0	0	41.91	0	0
1250	1148	0.1	41.81	1.4	58.534
1300	1851	0.35	41.66	3.3	137.478
1350	2493	0.6	41.41	6.5	269.165
1400	3085	0.95	41.06	10.8	443.448
1450	3625	1.3	40.61	17.1	694.431
1500	4122	1.7	40.02	24.7	988.494
1550	4542	2.15	39.29	32.7	1284.78
1600	4902	2.5	38.54	42.2	1626.39
1650	5228	2.9	37.79	52.4	1980.2
1700	5537	3.25	37.02	63.9	2365.58
1750	5742	3.8	36.09	75.4	2721.19
Time	01:57:21 PM				
Date	28/06/2011				

Table A.6: Continued

Pulse Width	Reading 2				
	Speed	Torque	Voltage	Current	E Pwr
uS	*RPM*	*Nm*	*Volts*	*Amps*	*Watts*
1200	0	0	39.72	0	0
1250	1131	0.1	39.69	1.1	43.659
1300	1773	0.25	39.61	2.8	110.908
1350	2400	0.55	39.42	5.8	228.636
1400	2948	0.8	39.15	9.8	383.67
1450	3480	1.15	38.83	15.4	597.982
1500	3977	1.65	38.42	22.8	875.976
1550	4388	1.95	37.86	30.9	1169.87
1600	4765	2.35	37.31	39.6	1477.48
1650	5091	2.75	36.59	49.5	1811.21
1700	5400	3.05	35.97	60.8	2186.98
1750	5640	3.5	35.26	72.4	2552.82
Time	02:02:00 PM				
Date	28/06/2011				
Pulse Width	**Reading 3**				
	Speed	**Torque**	**Voltage**	**Current**	**E Pwr**
uS	*RPM*	*Nm*	*Volts*	*Amps*	*Watts*
1200	0	0	38.56	0	0
1250	1122	0.1	38.52	1	38.52
1300	1731	0.25	38.44	2.6	99.944
1350	2348	0.5	38.29	5.4	206.766
1400	2880	0.75	38.06	9.2	350.152
1450	3394	1.1	37.77	14.8	558.996
1500	3891	1.45	37.38	21.6	807.408
1550	4302	1.85	36.95	29.7	1097.42
1600	4680	2.25	36.45	38.4	1399.68
1650	5022	2.65	35.92	48.9	1756.49
1700	5322	2.95	35.34	59.3	2095.66
1750	5588	3.3	34.76	71.5	2485.34
Time	02:10:09 PM				
Date	28/06/2011				

A5: Experiment 2 Data Analysis Output

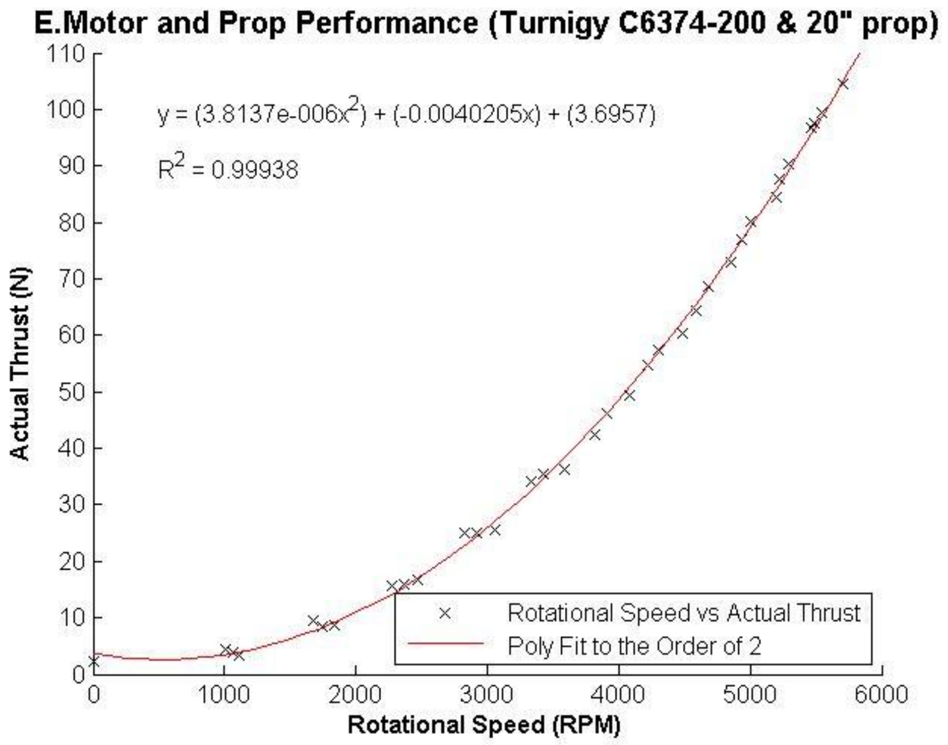


Figure A.6: Thrust vs rotational speed (Turnigy C6374 – 200 & 20” prop)

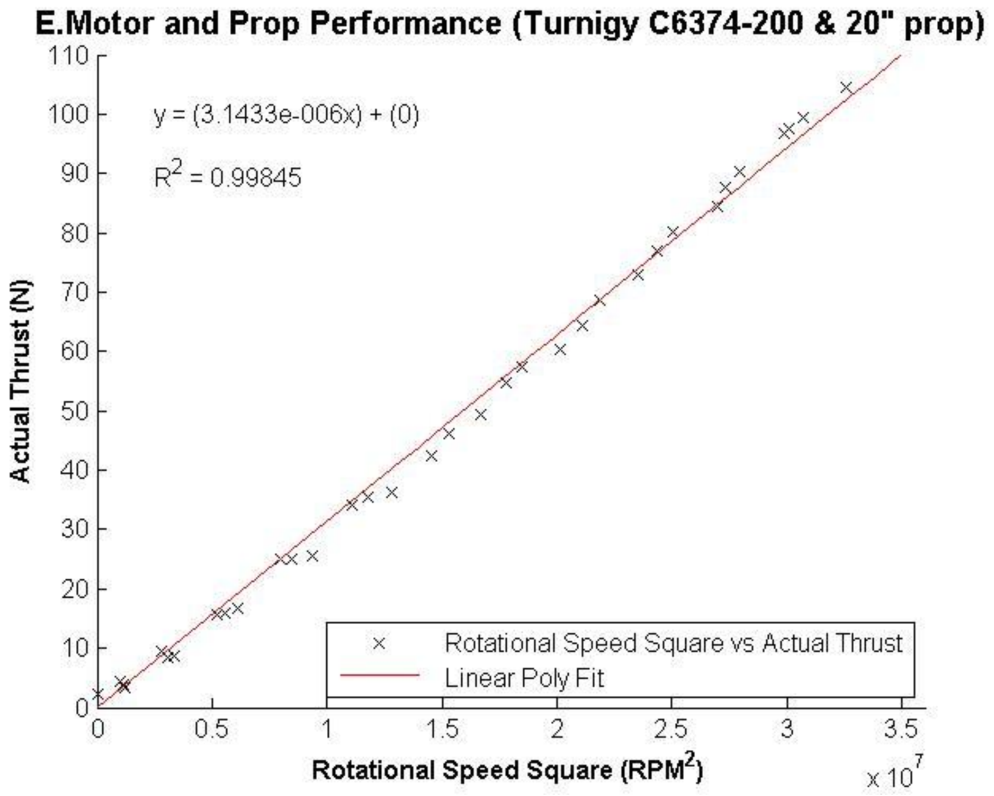


Figure A.7: Thrust vs square of rotational speed (Turnigy C6374 – 200 & 20” prop)

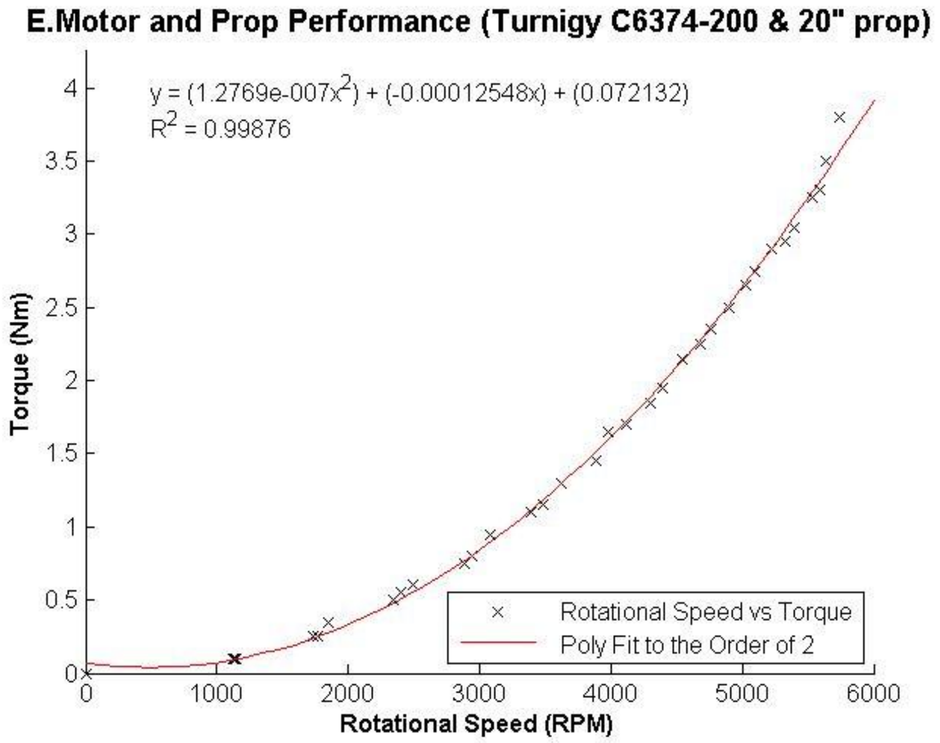


Figure A.8: Torque vs rotational speed (Turnigy C6374 – 200 & 20'' prop)

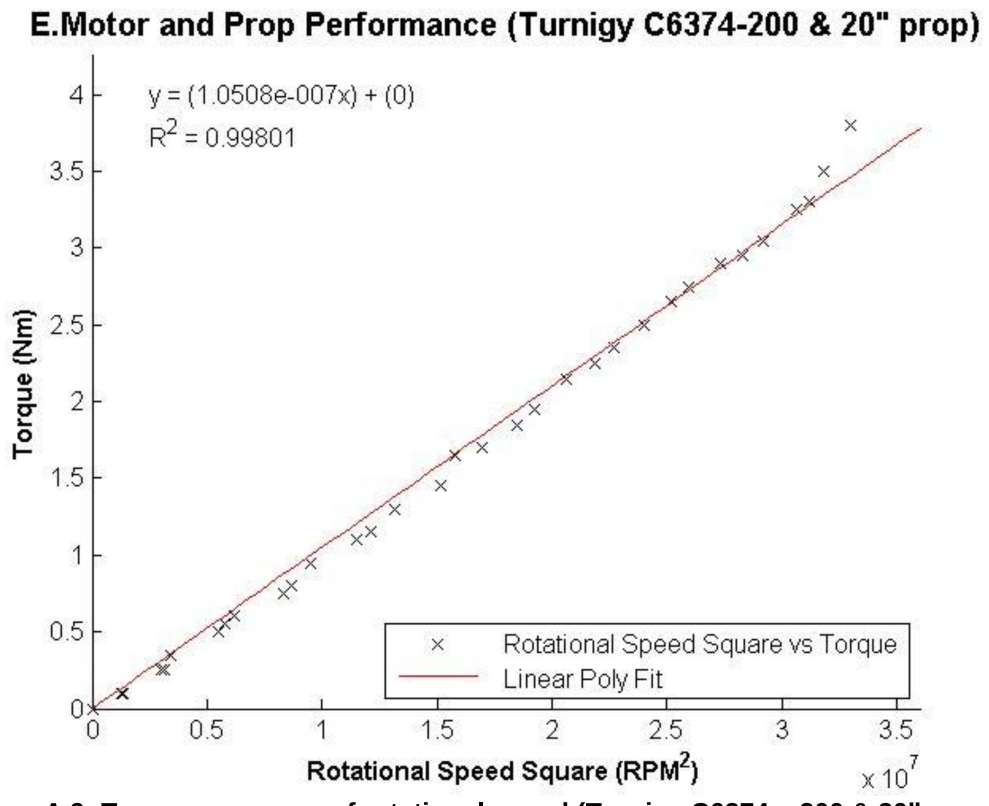


Figure A.9: Torque vs square of rotational speed (Turnigy C6374 – 200 & 20'' prop)

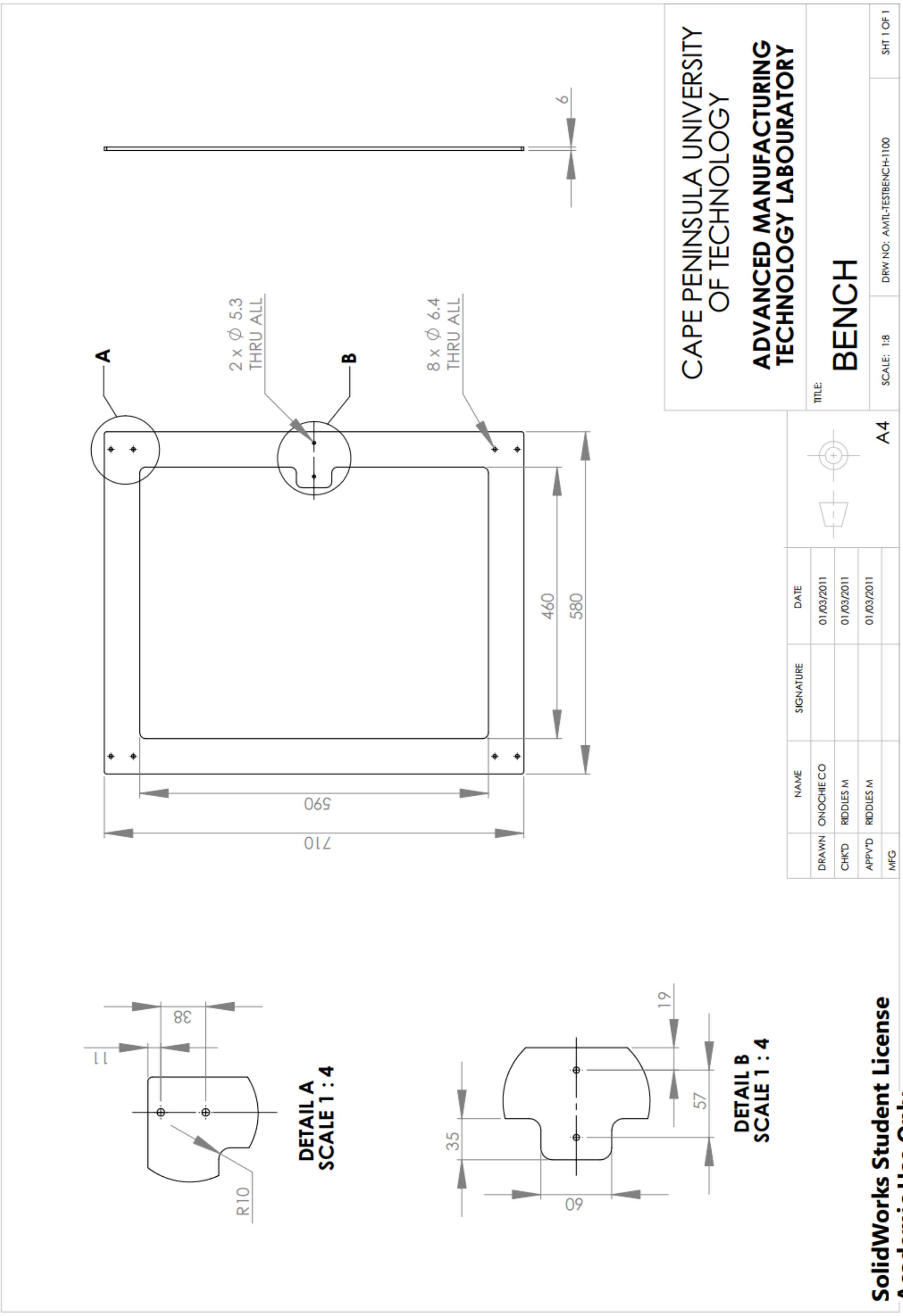
The MATLAB script used for the generation of the plot and mathematical relations from the experimental data is shown below.

```
%----------------------------------   START   ------------------------------
%THRUST ---------------------------------------------------------------------
%  Clear all existing variables
clear all;

%  Load data
load Dataset02

%  Define Test Bench Clibration Linear fit coefficients
tstbnchlinfit.m=1.0834;
tstbnchlinfit.b=2.1611;

%  Clibrate thrust
thrust.calibthrust=(tstbnchlinfit.m*thrust.thrust)+tstbnchlinfit.b;

%------------------------------------  1  ----------------------------------

% Create figure
thrust.performance_charaterisation = figure;

% Create axes
thrust.axes1 = axes('Parent',thrust.performance_charaterisation);
hold(thrust.axes1,'all');

% Create plot
thrust.plot1=plot(thrust.speed,thrust.calibthrust,'x','MarkerEdgeColor',[0,
0,0],'DisplayName','   Rotational Speed vs Actual Thrust');
    % Create X Label
    xlabel('Rotational Speed (RPM)', 'FontWeight','Bold');
    % Create Y Label
    ylabel('Actual Thrust (N)', 'FontWeight','Bold');
    % Create Title
    title('E.Motor and Prop Performance (Turnigy C6374-200 & 20"
prop)','FontWeight','bold','FontSize',12);
    % set axis limti
    axis([0 6000 0 110]);
    % Show grid
    % grid on;

% Polyfit of thrust.speed and thrust.calibthrust to the second order
thrust.polyfit=polyfit(thrust.speed,thrust.calibthrust,2);
    % Evaluating the polyfit and saving output
    thrust.output=polyval(thrust.polyfit,thrust.speed);

% Calculating the R squared value for linear poly fit 1
thrust.Correlation= corrcoef(thrust.calibthrust, thrust.output);
thrust.R2=thrust.Correlation(2);

% Generate fit data for plot
    % X data
    thrust.X=linspace(0,6000,50);
    % Y data
    thrust.Y=polyval(thrust.polyfit,thrust.X);
```

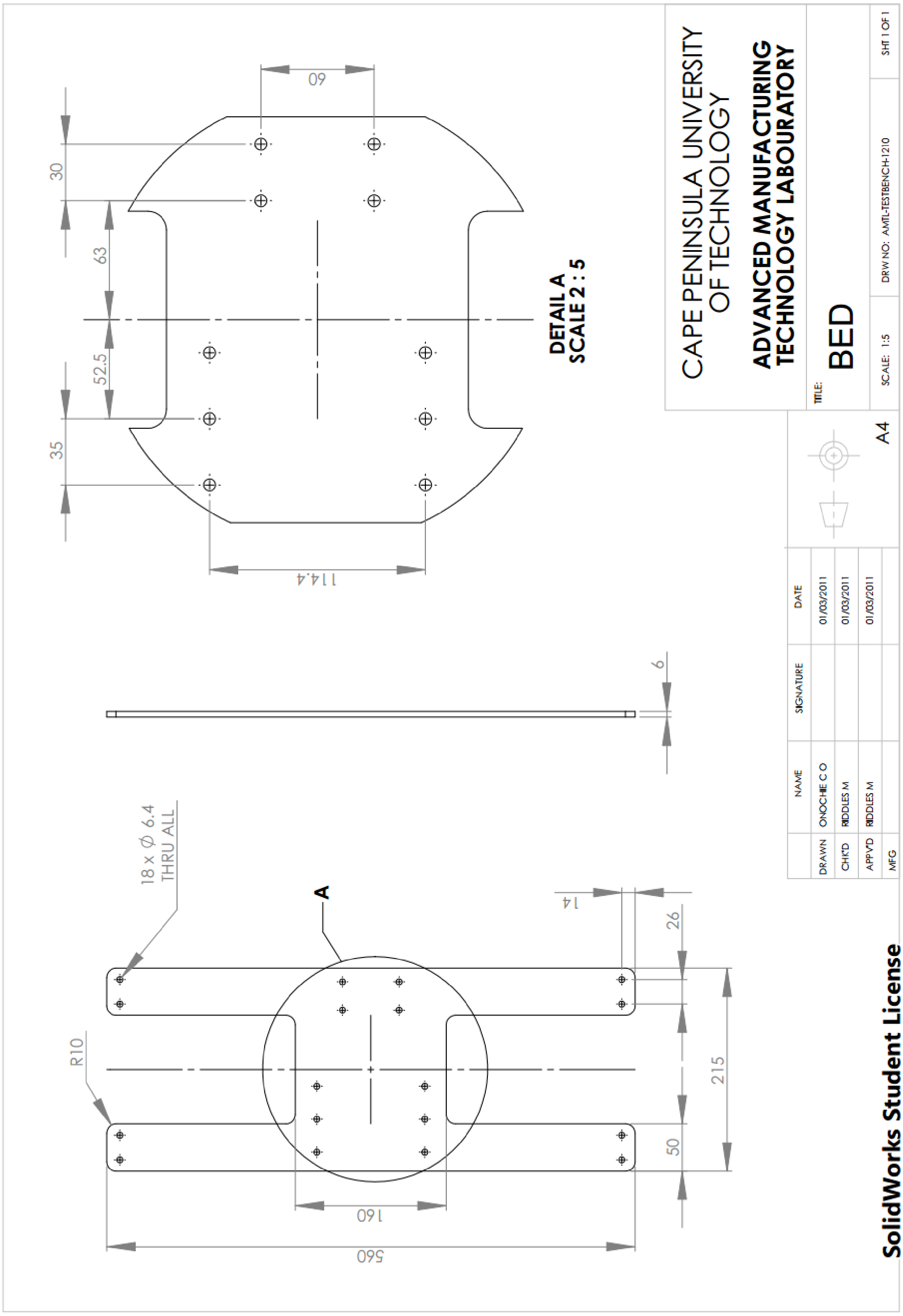
```
% Plot the fit
thrust.fitLine1 = plot(thrust.X,thrust.Y,'DisplayName','   Poly Fit to the
Order of 2','Parent',thrust.axes1,...
    'Tag','Poly Fit to the Order of 2',...
    'Color',[1 0 0]);
    % Create legend
    legend('show');
    set(legend,'Location','SouthEast');
    % Show equation
    text(500,100,['y = (',num2str(thrust.polyfit(1)),'x^2) +
(',num2str(thrust.polyfit(2)),'x) + (',num2str(thrust.polyfit(3)),')']);
    % Show R2
    text(500,90,['R^2 = ',num2str(thrust.R2)]);

 % Turn off axes hold
 hold(thrust.axes1,'off');

%-------------------------------------  2  --------------------------------

%  Square Rotational Speed
thrust.speedsqr=thrust.speed.^2;

% Create figure
thrust.performance_charaterisation1 = figure;

% Create axes
thrust.axes2 = axes('Parent',thrust.performance_charaterisation1);
hold(thrust.axes2,'all');

% Create plot
thrust.plot2=plot(thrust.speedsqr,thrust.calibthrust,'x','MarkerEdgeColor',
[0,0,0],'DisplayName','   Rotational Speed Square vs Actual Thrust');
    % Create X Label
    xlabel('Rotational Speed Square (RPM^2)', 'FontWeight','Bold');
    % Create Y Label
    ylabel('Actual Thrust (N)', 'FontWeight','Bold');
    % Create Title
    title('E.Motor and Prop Performance (Turnigy C6374-200 & 20"
prop)','FontWeight','bold','FontSize',12);
    % set axis limti
    axis([0 6000^2 0 110]);
    % Show grid
    % grid on;

% linear polyfit of thrust.speedsqr and thrust.calibthrust
thrust.linpolyfit1=polyfit(thrust.speedsqr,thrust.calibthrust,1);
% Evaluating the linear polyfit and saving output
thrust.output1=polyval(thrust.linpolyfit1,thrust.speedsqr);
% Calculating the R squared value for linear poly fit 1
thrust.Correlation1 = corrcoef(thrust.calibthrust, thrust.output1);
thrust.R2_1=thrust.Correlation1(2);

% Generate fit data for plot
    % X data
    thrust.X1=linspace(0,6000^2,50);
    % Y data
    thrust.Y1=polyval(thrust.linpolyfit1,thrust.X1);

% Plot the fit
thrust.fitLine2 = plot(thrust.X1,thrust.Y1,'DisplayName','   Linear Poly
Fit','Parent',thrust.axes2,...
    'Tag','Linear Poly Fit',...
    'Color',[1 0 0]);
    % Create legend
    legend('show');
    set(legend,'Location','SouthEast');
    % Show equation
    text(2500000,100,['y = (',num2str(thrust.linpolyfit1(1)),'x) +
(',num2str(thrust.linpolyfit1(2)),')']);
    % Show R2
    text(2500000,90,['R^2 = ',num2str(thrust.R2_1)]);

 % Turn off axes hold
 hold(thrust.axes2,'off');

%-------------------------------------  3  --------------------------------

% Create figure
thrust.performance_charaterisation2 = figure;

% Create axes
thrust.axes3 = axes('Parent',thrust.performance_charaterisation2);
hold(thrust.axes3,'all');

% Create plot
thrust.plot3=plot(thrust.speedsqr,thrust.calibthrust,'x','MarkerEdgeColor',
[0,0,0],'DisplayName','   Rotational Speed Square vs Actual Thrust');
    % Create X Label
    xlabel('Rotational Speed Square (RPM^2)', 'FontWeight','Bold');
    % Create Y Label
    ylabel('Actual Thrust (N)', 'FontWeight','Bold');
    % Create Title
    title('E.Motor and Prop Performance (Turnigy C6374-200 & 20"
prop)','FontWeight','bold','FontSize',12);
    % set axis limti
    axis([0 6000^2 0 110]);
    % Show grid
    % grid on;

% linear polyfit of thrust.speedsqr and thrust.calibthrust with intercept
% at zero
thrust.slope=thrust.speedsqr\thrust.calibthrust;
thrust.linpolyfit2=[thrust.slope,0];
% Evaluating the linear polyfit  with intercept at zero and saving output
thrust.output2=polyval(thrust.linpolyfit2,thrust.speedsqr);
% Calculating the R squared value for linear poly fit 2
thrust.Correlation2 = corrcoef(thrust.calibthrust, thrust.output2);
thrust.R2_2=thrust.Correlation2(2);

% Generate fit data for plot
    % X data
    thrust.X1=linspace(0,6000^2,50);
    % Y data
    thrust.Y2=polyval(thrust.linpolyfit2,thrust.X1);

% Plot the fit
thrust.fitLine2 = plot(thrust.X1,thrust.Y2,'DisplayName','   Linear Poly
Fit','Parent',thrust.axes3,...
    'Tag','Linear Poly Fit',...
    'Color',[1 0 0]);
    % Create legend
    legend('show');
    set(legend,'Location','SouthEast');
    % Show equation
    text(2500000,100,['y = (',num2str(thrust.linpolyfit2(1)),'x) +
(',num2str(thrust.linpolyfit2(2)),')']);
    % Show R2
    text(2500000,90,['R^2 = ',num2str(thrust.R2_2)]);

 % Turn off axes hold
 hold(thrust.axes2,'off');

 %TORQUE  ----------------------------------------------------------------
 %------------------------------------  1  --------------------------------

% Create figure
torque.performance_charaterisation = figure;

% Create axes
torque.axes1 = axes('Parent',torque.performance_charaterisation);
hold(torque.axes1,'all');

% Create plot
torque.plot1=plot(torque.speed,torque.torque,'x','MarkerEdgeColor',[0,0,0],
'DisplayName','   Rotational Speed vs Torque');
    % Create X Label
    xlabel('Rotational Speed (RPM)', 'FontWeight','Bold');
    % Create Y Label
    ylabel('Torque (Nm)', 'FontWeight','Bold');
    % Create Title
    title('E.Motor and Prop Performance (Turnigy C6374-200 & 20"
prop)','FontWeight','bold','FontSize',12);
    % set axis limti
    axis([0 6000 0 4.25]);
    % Show grid
    % grid on;

% Polyfit of torque.speed and torque.torque to the second order
torque.polyfit=polyfit(torque.speed,torque.torque,2);
    % Evaluating the polyfit and saving output
    torque.output=polyval(torque.polyfit,torque.speed);

% Calculating the R squared value for linear poly fit 1
torque.Correlation= corrcoef(torque.torque, torque.output);
torque.R2=torque.Correlation(2);

% Generate fit data for plot
    % X data
    torque.X=linspace(0,6000,50);
    % Y data
    torque.Y=polyval(torque.polyfit,torque.X);

% Plot the fit
torque.fitLine1 = plot(torque.X,torque.Y,'DisplayName','   Poly Fit to the
Order of 2','Parent',torque.axes1,...
    'Tag','Poly Fit to the Order of 2',...
    'Color',[1 0 0]);
    % Create legend
    legend('show');
    set(legend,'Location','SouthEast');
    % Show equation
    text(500,4,['y = (',num2str(torque.polyfit(1)),'x^2) +
(',num2str(torque.polyfit(2)),'x) + (',num2str(torque.polyfit(3)),')']);

 % Show R2
    text(500,3.75,['R^2 = ',num2str(torque.R2)]);
 % Turn off axes hold
 hold(torque.axes1,'off');

%-------------------------------------  2  --------------------------------

%  Square Rotational Speed
torque.speedsqr=torque.speed.^2;

% Create figure
torque.performance_charaterisation1 = figure;

% Create axes
torque.axes2 = axes('Parent',torque.performance_charaterisation1);
hold(torque.axes2,'all');

% Create plot
torque.plot2=plot(torque.speedsqr,torque.torque,'x','MarkerEdgeColor',[0,0,
0],'DisplayName','   Rotational Speed Square vs Torque');
    % Create X Label
    xlabel('Rotational Speed Square (RPM^2)', 'FontWeight','Bold');
    % Create Y Label
    ylabel('Torque (Nm)', 'FontWeight','Bold');
    % Create Title
    title('E.Motor and Prop Performance (Turnigy C6374-200 & 20"
prop)','FontWeight','bold','FontSize',12);
    % set axis limti
    axis([0 6000^2 0 4.25]);
    % Show grid
    % grid on;

% linear polyfit of torque.speedsqr and torque.torque
torque.linpolyfit1=polyfit(torque.speedsqr,torque.torque,1);
% Evaluating the linear polyfit and saving output
torque.output1=polyval(torque.linpolyfit1,torque.speedsqr);
% Calculating the R squared value for linear poly fit 1
torque.Correlation1 = corrcoef(torque.torque, torque.output1);
torque.R2_1=torque.Correlation1(2);

% Generate fit data for plot
    % X data
    torque.X1=linspace(0,6000^2,50);
    % Y data
    torque.Y1=polyval(torque.linpolyfit1,torque.X1);

% Plot the fit
torque.fitLine2 = plot(torque.X1,torque.Y1,'DisplayName','   Linear Poly
Fit','Parent',torque.axes2,...
    'Tag','Linear Poly Fit',...
    'Color',[1 0 0]);
    % Create legend
    legend('show');
    set(legend,'Location','SouthEast');
    % Show equation
    text(2500000,4,['y = (',num2str(torque.linpolyfit1(1)),'x) +
(',num2str(torque.linpolyfit1(2)),')']);
    % Show R2
    text(2500000,3.75,['R^2 = ',num2str(torque.R2_1)]);

 % Turn off axes hold
 hold(torque.axes2,'off');
```

```
%-------------------------------------  3  --------------------------------

% Create figure
torque.performance_charaterisation2 = figure;

% Create axes
torque.axes3 = axes('Parent',torque.performance_charaterisation2);
hold(torque.axes3,'all');

% Create plot
torque.plot3=plot(torque.speedsqr,torque.torque,'x','MarkerEdgeColor',[0,0,
0],'DisplayName','   Rotational Speed Square vs Torque');
    % Create X Label
    xlabel('Rotational Speed Square (RPM^2)', 'FontWeight','Bold');
    % Create Y Label
    ylabel('Torque (Nm)', 'FontWeight','Bold');
    % Create Title
    title('E.Motor and Prop Performance (Turnigy C6374-200 & 20"
prop)','FontWeight','bold','FontSize',12);
    % set axis limti
    axis([0 6000^2 0 4.25]);
    % Show grid
    % grid on;

% linear polyfit of torque.speedsqr and torque.torque with intercept
% at zero
torque.slope=torque.speedsqr\torque.torque;
torque.linpolyfit2=[torque.slope,0];
% Evaluating the linear polyfit  with intercept at zero and saving output
torque.output2=polyval(torque.linpolyfit2,torque.speedsqr);
% Calculating the R squared value for linear poly fit 2
torque.Correlation2 = corrcoef(torque.torque, torque.output2);
torque.R2_2=torque.Correlation2(2);

% Generate fit data for plot
    % X data
    torque.X1=linspace(0,6000^2,50);
    % Y data
    torque.Y2=polyval(torque.linpolyfit2,torque.X1);

% Plot the fit
torque.fitLine2 = plot(torque.X1,torque.Y2,'DisplayName','   Linear Poly
Fit','Parent',torque.axes3,...
    'Tag','Linear Poly Fit',...
    'Color',[1 0 0]);
    % Create legend
    legend('show');
    set(legend,'Location','SouthEast');
    % Show equation
    text(2500000,4,['y = (',num2str(torque.linpolyfit2(1)),'x) +
(',num2str(torque.linpolyfit2(2)),')']);
    % Show R2
    text(2500000,3.75,['R^2 = ',num2str(torque.R2_2)]);

 % Turn off axes hold
 hold(torque.axes2,'off');

%----------------------------------  END  ---------------------------------
```

APPENDIX B: THRUST TORQUE TEST-BENCH MANUFACTURING DRAWING



6
A
B
2 x ∅ 5.3 THRU ALL
8 x ∅ 6.4 THRU ALL
460
580
590
710
38
11
R10
DETAIL A SCALE 1 : 4
19
57
35
60
DETAIL B SCALE 1 : 4
CAPE PENINSULA UNIVERSITY OF TECHNOLOGY
ADVANCED MANUFACTURING TECHNOLOGY LABOURATORY
TITLE:
BENCH
SCALE: 1:8
DRW NO: AMTL-TESTBENCH-1100
SHT 1 OF 1
A4
NAME
SIGNATURE
DATE
DRAWN ONOCHIE CO 01/03/2011
CHK'D RIDDLES M 01/03/2011
APPV'D RIDDLES M 01/03/2011
MFG
SolidWorks Student License Academic Use Only



CAPE PENINSULA UNIVERSITY OF TECHNOLOGY

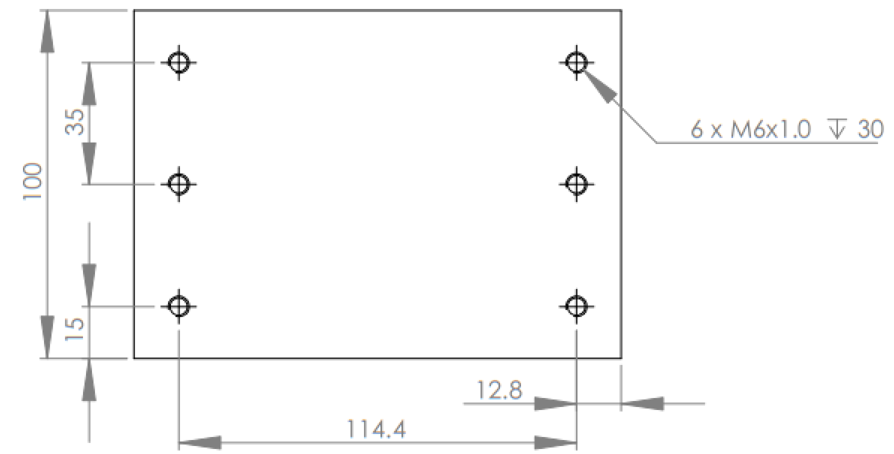
ADVANCED MANUFACTURING TECHNOLOGY LABOURATORY

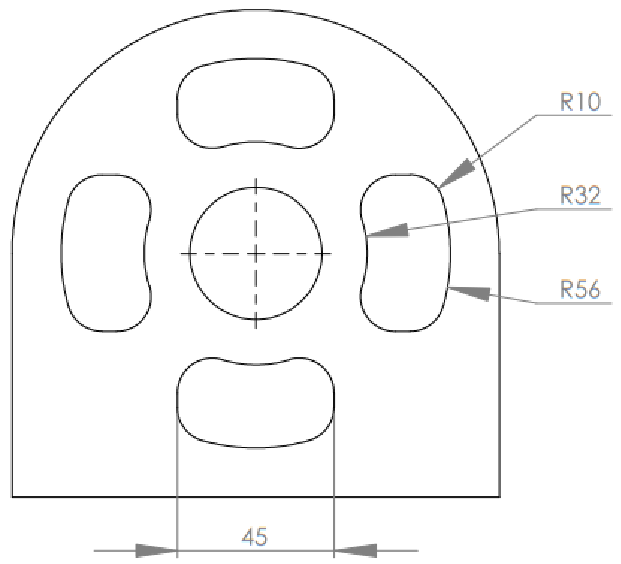
TITLE: BED

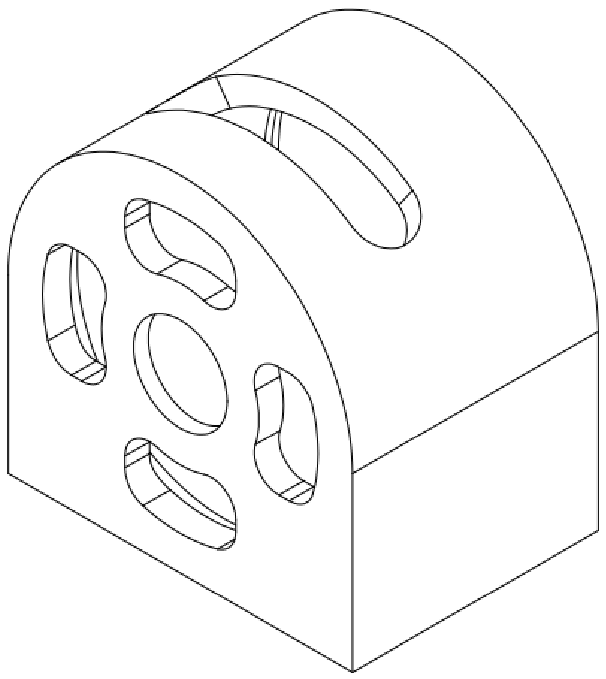
	NAME	SIGNATURE	DATE
DRAWN	ONOCHIE C O		01/03/2011
CHK'D	RIDDLES M		01/03/2011
APPV'D	RIDDLES M		01/03/2011
MFG			

A4

SCALE: 1:5 DRW NO: AMTL-TESTBENCH-1210 SHT 1 OF 1







ISOMETRIC VIEW

	NAME	SIGNATURE	DATE
DRAWN	ONOCHIE CO		01/03/2011
CHK'D	RIDDLES M		01/03/2011
APPV'D	RIDDLES M		01/03/2011
MFG			

A4

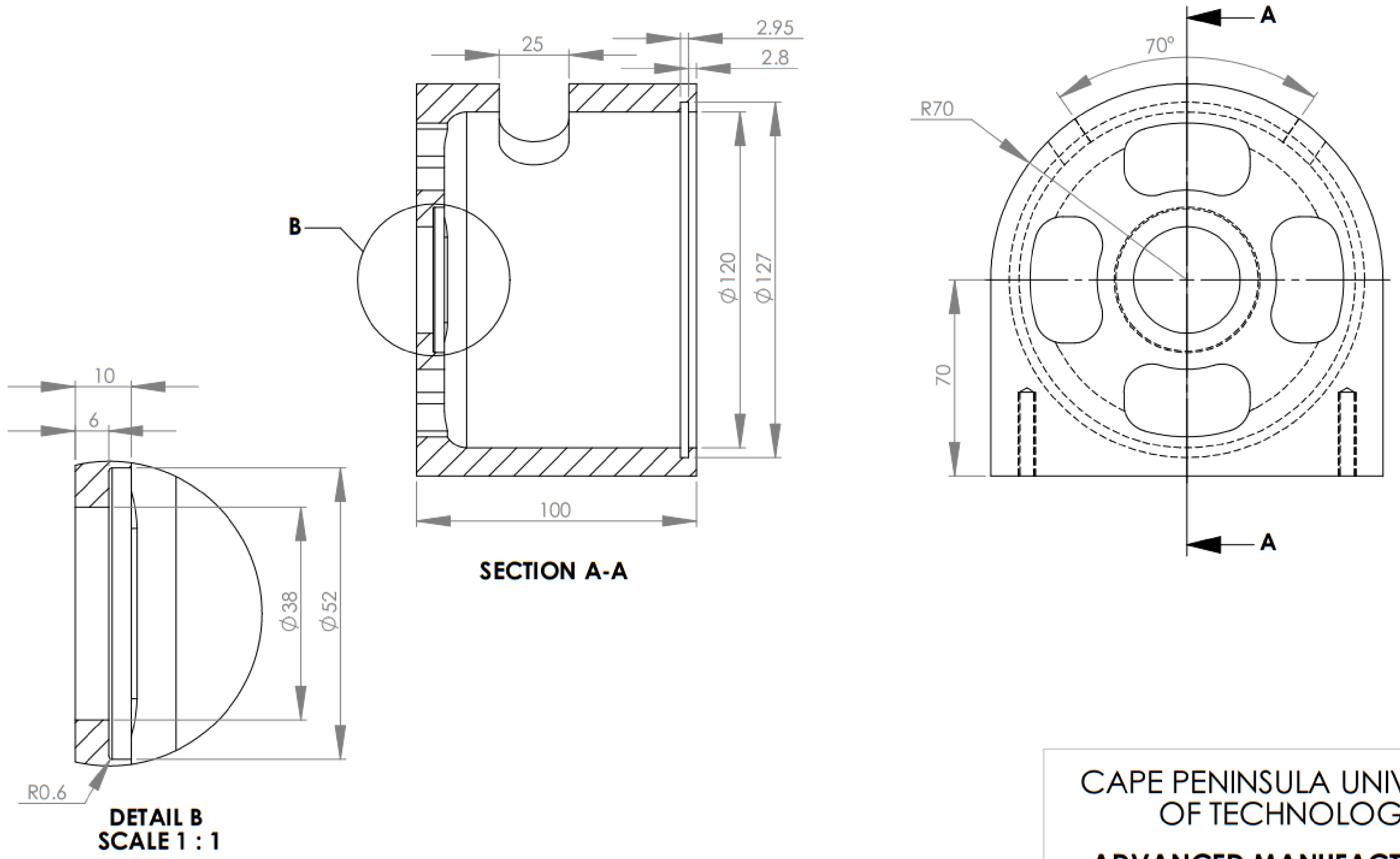
CAPE PENINSULA UNIVERSITY OF TECHNOLOGY

ADVANCED MANUFACTURING TECHNOLOGY LABOURATORY

TITLE:

MOTOR HOUSING

SCALE: 1:2 | DRW NO: AMTL-TESTBENCH-1221 | SHT 1 OF 2



	NAME	SIGNATURE	DATE
DRAWN	ONOCHIE CO		01/03/2011
CHK'D	RIDDLES M		01/03/2011
APPV'D	RIDDLES M		01/03/2011
MFG			

A4

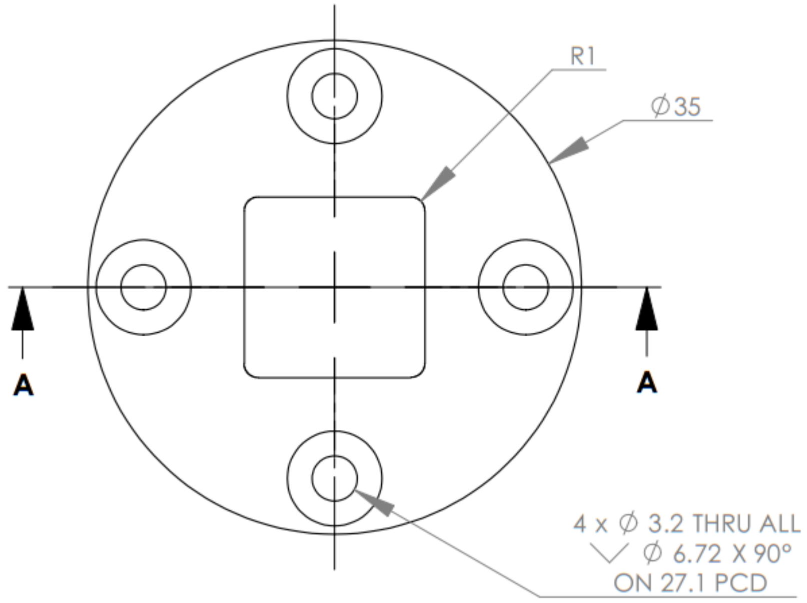
CAPE PENINSULA UNIVERSITY OF TECHNOLOGY

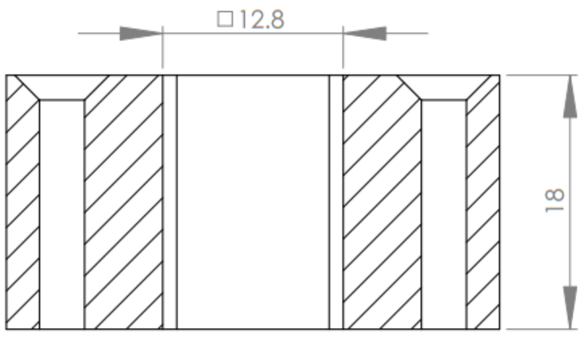
ADVANCED MANUFACTURING TECHNOLOGY LABOURATORY

TITLE:

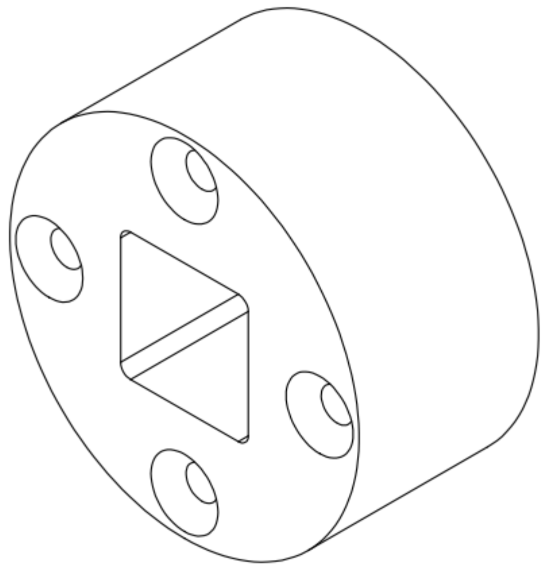
MOTOR HOUSING

SCALE: 1:2 | DRW NO: AMTL-TESTBENCH-1221 | SHT 2 OF 2





SECTION A-A



ISOMETRIC VIEW

	NAME	SIGNATURE	DATE
DRAWN	ONOCHIE CO		01/03/2011
CHK'D	RIDDLES M		01/03/2011
APPV'D	RIDDLES M		01/03/2011
MFG			

A4

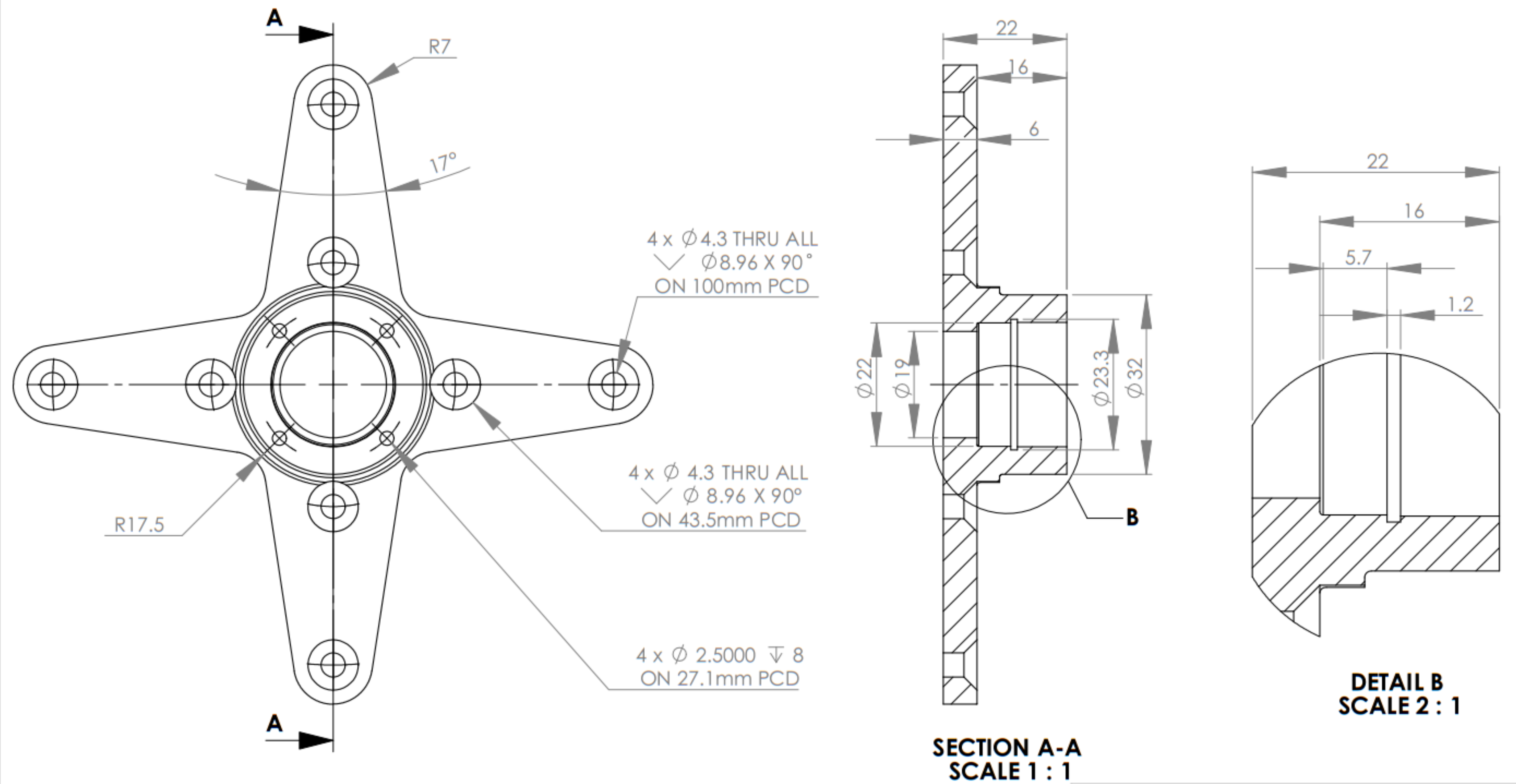
CAPE PENINSULA UNIVERSITY OF TECHNOLOGY

ADVANCED MANUFACTURING TECHNOLOGY LABOURATORY

TITLE:

SOCKET

SCALE: 2:1 | DRW NO: AMTL-TESTBENCH-1250 | SHT 1 OF 1



SECTION A-A
SCALE 1 : 1

DETAIL B
SCALE 2 : 1

CAPE PENINSULA UNIVERSITY OF TECHNOLOGY

ADVANCED MANUFACTURING TECHNOLOGY LABOURATORY

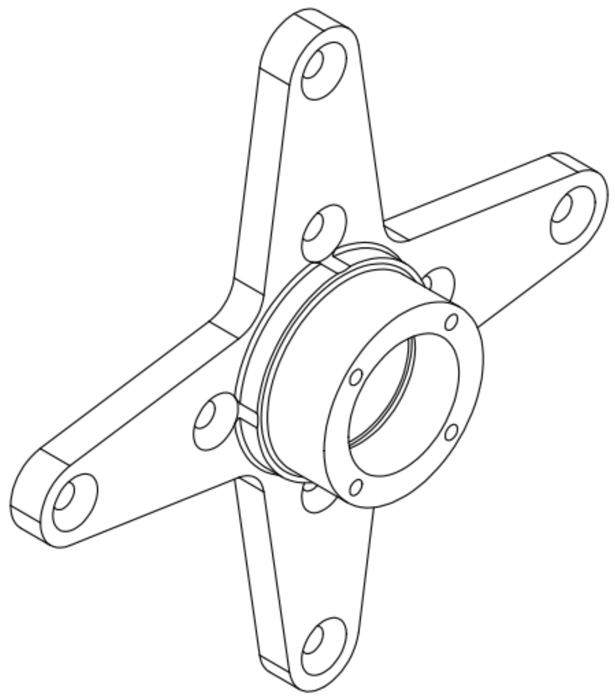
TITLE:

THRUST PLATE

	NAME	SIGNATURE	DATE
DRAWN	ONOCHIE CO		01/03/2011
CHK'D	RIDDLES M		01/03/2011
APPV'D	RIDDLES M		01/03/2011
MFG			

A4

SCALE: 1:1	DRW NO: AMTL-TESTBENCH-1223	SHT 1 OF 2


ISOMETRIC VIEW

	NAME	SIGNATURE	DATE
DRAWN	ONOCHIE CO		01/03/2011
CHK'D	RIDDLES M		01/03/2011
APPV'D	RIDDLES M		01/03/2011
MFG			

A4

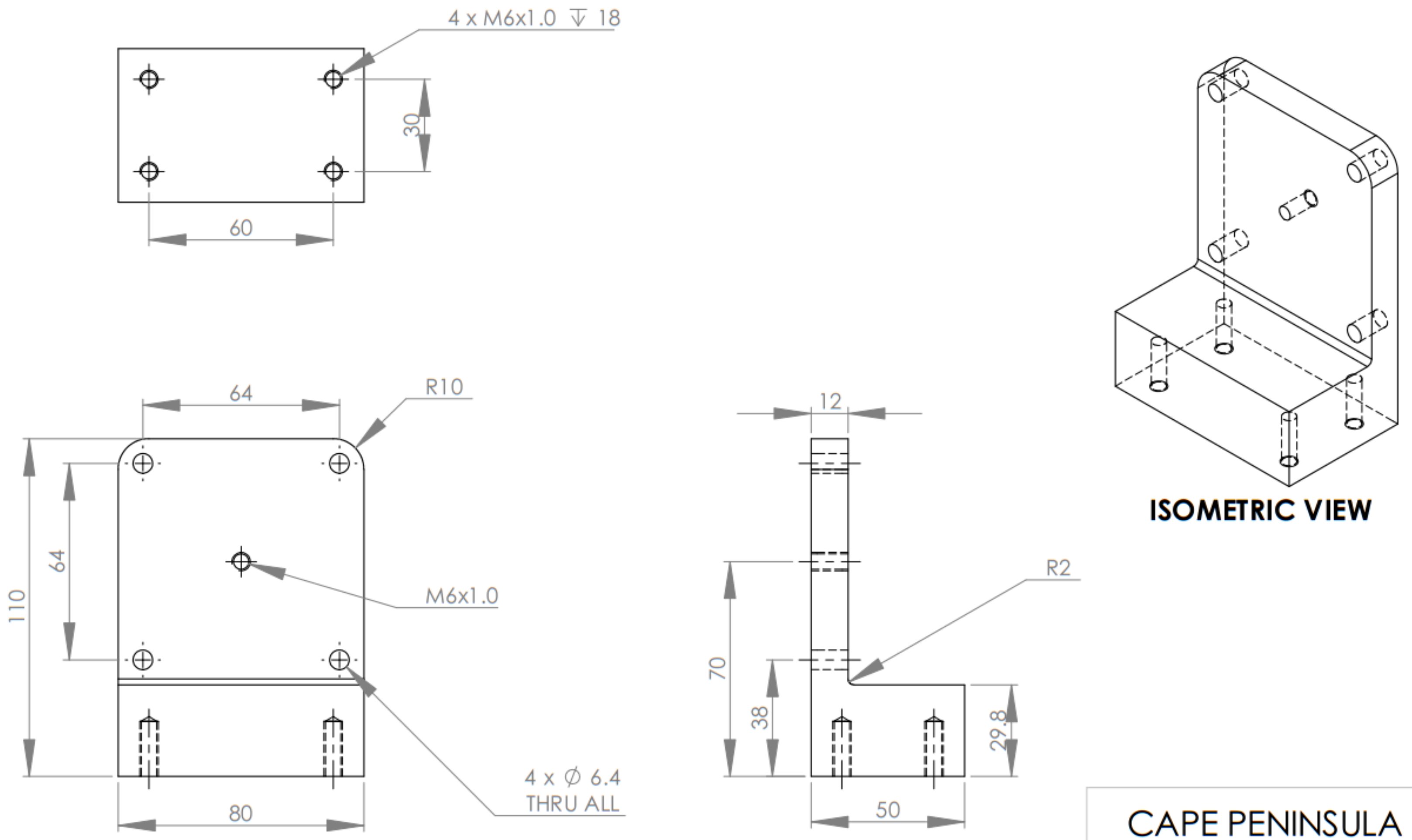
CAPE PENINSULA UNIVERSITY OF TECHNOLOGY

ADVANCED MANUFACTURING TECHNOLOGY LABOURATORY

TITLE:

THRUST PLATE

SCALE: 1:1 | DRW NO: AMTL-TESTBENCH-1223 | SHT 2 OF 2



	NAME	SIGNATURE	DATE
DRAWN	ONOCHIE CO		01/03/2011
CHK'D	RIDDLES M		01/03/2011
APPV'D	RIDDLES M		01/03/2011
MFG			

A4

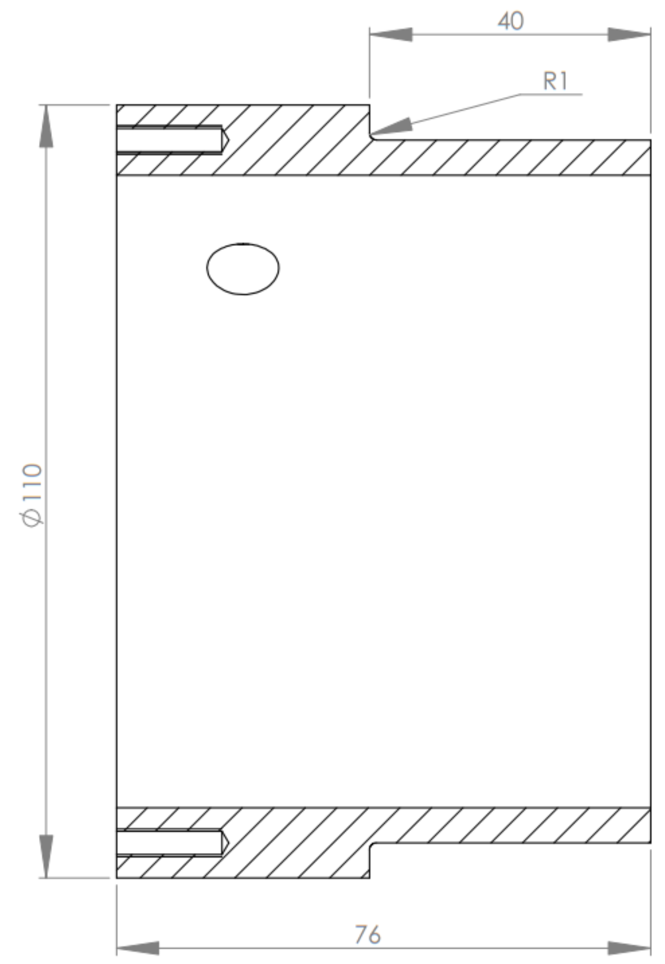
CAPE PENINSULA UNIVERSITY OF TECHNOLOGY

ADVANCED MANUFACTURING TECHNOLOGY LABOURATORY

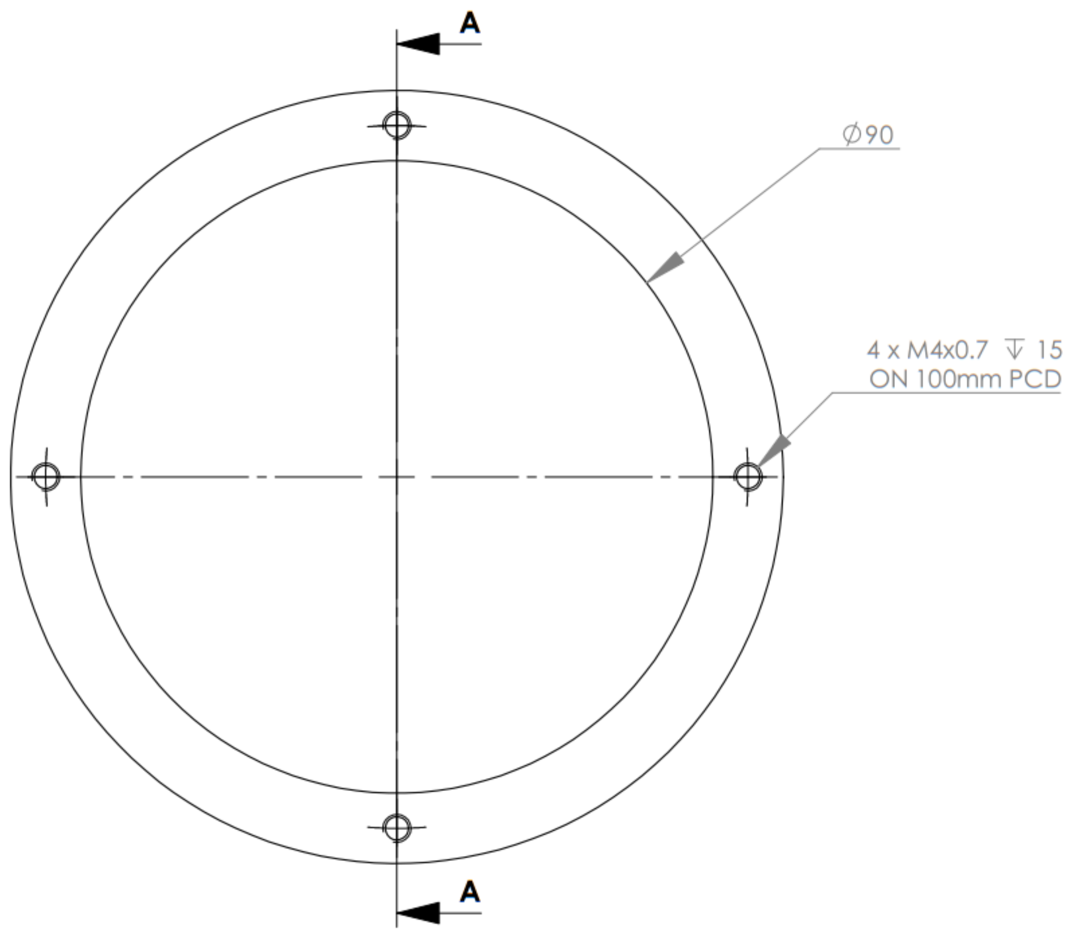
TITLE:

TORQUE GUAGE MOUNT

SCALE: 1:2 | DRW NO: AMTL-TESTBENCH-1230 | SHT 1 OF 1



SECTION A-A



CAPE PENINSULA UNIVERSITY OF TECHNOLOGY

ADVANCED MANUFACTURING TECHNOLOGY LABOURATORY

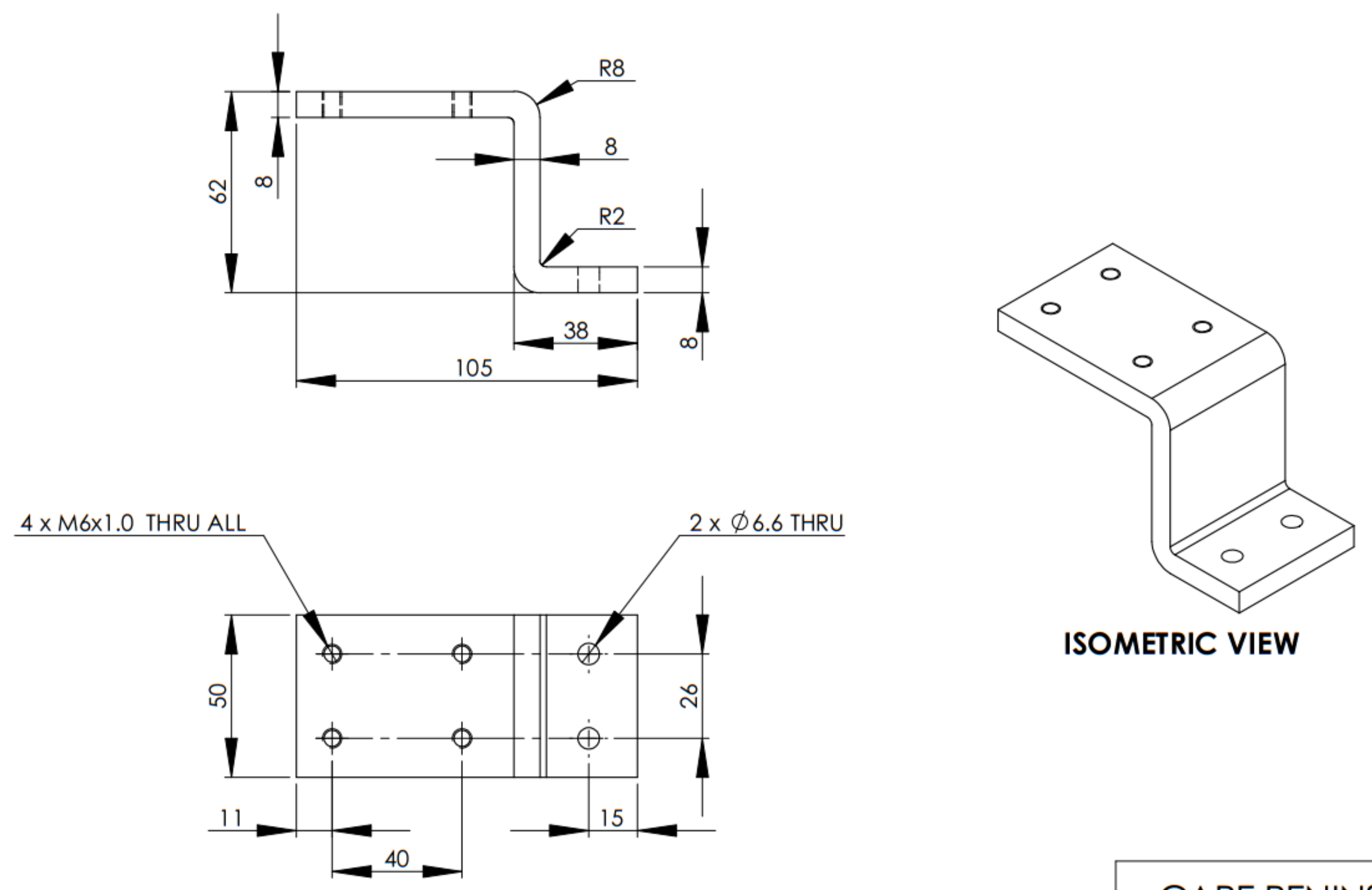
TITLE:

TORQUE SLEEVE

	NAME	SIGNATURE	DATE
DRAWN	ONOCHIE CO		01/03/2011
CHK'D	RIDDLES M		01/03/2011
APPV'D	RIDDLES M		01/03/2011
MFG			

A4

SCALE: 1:1 | DRW NO: AMTL-TESTBENCH-1222 | SHT 1 OF 1



CAPE PENINSULA UNIVERSITY OF TECHNOLOGY

ADVANCED MANUFACTURING TECHNOLOGY LABOURATORY

TITLE: Z PROFILE

	NAME	SIGNATURE	DATE
DRAWN	ONOCHIE CO		01/03/2011
CHK'D	RIDDLES M		01/03/2011
APPV'D	RIDDLES M		01/03/2011
MFG			

A4

SCALE: 1:2	DRW NO: AMTL-TESTBENCH-1240	SHT 1 OF 1

APPENDIX C: CONTROL AND STABILISATION SOFTWARE

C1: Secondary MCU Software

The secondary MCU software is written in C using the Arduino Integrated Development Environment (IDE). The Arduino IDE is used due to its simplicity and the vast amount of libraries available on it.

This software is made up of six Arduino IDE (.pde) files. The RC_Pulse_Decoder.pde file contains the setup and main program sequence, the Input.pde file contains the routine for RC signal input and decoding, the Output.pde file contain the signal encode and other output related functions. The Debug.pde and Alerts.pde contains the routine for communicating to a serial monitor for debugging purposes and routine for visual alerts. Finally, the Setup.pde file contains the routine for calibrating the RC signal.

The contents of these files are presented below.

RC_Pulse_Decoder.pde

```
/*
To be loaded on an ATMega328 with the arduino UNO Bootloader

Reads pulse signals from an RC receiver and sends it to a MAX32 over I2C interface using
the arduino TWI/I2C library by Nicholas Zambetti 2006.

Part of the sketch provides means to print the received data and other variables for
debugging. Change the value of the "debug" variable to enable or disable debugging, ange the
value of the "DEBUGGING" varialble
to switch between debugging modes
 0 = program location; 1 = input debugging mode;
 2 = output debugging mode; 3 = actual output mode; 4 = RC setup

 ___________________________*CONNECTIONS*_______________________________
 __ATMega328________________________|_______________RC Reciever_____________
 digital pin 2                      |                thro
 digital pin 4                      |                aile
 digital pin 7                      |                elev
 digital pin 8                      |                rudd
 digital pin 9                      |             gear / RC_S
 digital pin 12                     |                aux1
 digital pin 10                     |                aux2
 ___________________________________|______________________________________
 digital pin 3                      |             Status led (red)
 ___________________________________|______________________________________
 __ATMega328________________________|_________________MAX32_______________
 A4 SDA                             |               20 SDA
 A5 SCL                             |               21 SCL
 GND                                |                GND
 ___________________________________|______________________________________

 Written by Cyprian Onochie, on 28th December 2011 */
```

```
#include <Wire.h>

// Alert Variables
//----------------------------------------------------------
int led = 3;                // define digital pin for alert led
int k = 0;                  // alert counter
int p = 0;                  // rc alert counter
int l = LOW;                // alert led state
//----------------------------------------------------------


// Input Variables
//----------------------------------------------------------
// define digital input pins for RC signal input
int ch1 = 2;
int ch2 = 4;
int ch3 = 7;
int ch4 = 8;
int ch5 = 9;
int ch6 = 12;
//int ch7 = 10;

// define integer arrays and variables to store pulse values
int _thro = 1100;
int thro[10];
int _aile = 1500;
int aile[10];
int _elev = 1500;
int elev[10];
int _rudd = 1500;
int rudd[10];
int aux1 = 1100;
int RC_S = 0;

// min and max range for the pulse signals in uS
int _thro_min = 1200;
int _thro_max = 1200;
int _aile_min = 1200;
int _aile_max = 1200;
int _elev_min = 1200;
int _elev_max = 1200;
int _rudd_min = 1200;
int _rudd_max = 1200;
int aux1_min = 1200;
int aux1_max = 1200;
// device arm  counter
int i = 0;
unsigned int j = 0;
// disarm counter
int _j = 0;
boolean arm = false;
//----------------------------------------------------------


// Output Variables
//----------------------------------------------------------
// define the 8 bit data array for wire send
// the length of this array must reflect the number of channels to be transmitted
byte rc[10] = {
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
//----------------------------------------------------------
```

```
//  RC signal setup Variables
//-----------------------------------------------------------
// RC setup counter, toggled off
int r = 0;
boolean signal_setup = false;
boolean setup_mode = false;
//-----------------------------------------------------------


// Debugging Variables
//-----------------------------------------------------------
// variable to store RC input read function duration
int rc_read_time =

// used to select debugging modes. 1, 2 and 3 = on; 0 = off
int DEBUGGING = 0;

// used to enable debugging, true = on; false = off
boolean debug = false;
//-----------------------------------------------------------


void setup() {
// initialize serial communications at 115200bps when in debugging mode
if (debug == true){
Serial.begin(115200);
  }
  // Uncomment to set I2C frequency to 400kHz
  //#define TWI_FREQ 400000L

  // join i2c bus with address #10
  Wire.begin(10);

  // register I2C event
  Wire.onRequest(requestSignal);

  // set digital pin 3 as output
  pinMode(led,OUTPUT);
}

void loop() {
  // check for debug mode status                                    DEBUGGING
  checkDebugMode();

  // checks for the arm signal and arm
  checkArm();

  // perform RC signal range setup when enabled
  setup_RC();

  // print selected data to serial monitor
  printSignal();

  // check for device arm, arming should take 4 seconds
  while (i > 4000)  {
    // read the pulse values from RC reciever and store the signals in the buff array.
    //get the running average of the signals and constrain between 1100 and 1900
    getSignal();

    // creat an array of 8 bit data to be sent over the I2C interface
    // must follow call to get signal
    encodeSignal();
    // checks for the disarm signal and disarm
    checkDisArm();

    // check for debug mode status                                                    DEBUGGING
    checkDebugMode();

    // print rc read the results and read duration to the serial monitor:            DEBUGGING
    if (debug == true){
      Serial.print("Main loop while   ");              // where am I?
    }
    printSignal();
  }

  // print rc read the results and read duration to the serial monitor:              DEBUGGING
  if (debug == true){
    Serial.print("Main loop   ");              // where am I?
  }
}

// interrupt function that executes whenever data is requested by master
// this function is registered as an event, see void setup()
void requestSignal()
{
  Wire.write(rc,9);       // respond with message of bytes as expected by master
}
```

Input.pde:

```
/*********************Input Signal related functions**********************************/
void checkArm(){
  checkDebugMode();
  getSignal();
  encodeSignal();

  // check for arm only when RC signal setup is done
  if (signal_setup == true){

    // checks for the arm signal and increaments counter to greater than 4 seconds
    while ((_thro < (_thro_min + 90)) && (_rudd > (_rudd_max - 90)) && (i < 4200)){

      // turn off alert led when its on
      if (l == HIGH){
        digitalWrite(led, LOW);
        l = LOW;
      }

      //reset the disarm counter when disarmed
      j = 0;
      getSignal();
      checkDebugMode();

/* Get and encode RC signal, print results and counter duration to the serial monitor when
debugging is on                                                              DEBUGGING */
      if (debug == true){
        Serial.print("checkArm   ");                          // where am i?
      }
      printSignal();

// output no values
output0();
      if (DEBUGGING == 0){                                    // DEBUGGING off
        /* get duration number determined from the time taken to loop to here for this case*/
         i = i + 1;
        delay(400);
      }
      else {                                                  // DEBUGGING on
        i = i + 110;
      }

    if (i > 4200){
      i = 4500;
      // arm device
      j = 0;
      arm = true;
    }
    else {
      i = 0;
    }
  }

  // device arming alert
  deviceArming_alert();

  // device disarmed alert
  deviceDisarmed_alert();
}


void checkDisArm()
```

```
{
  checkDebugMode();

  getSignal();

  // checks for the disarm signal and increaments counter to greater than 4 seconds
  while ((_thro < (_thro_min + 80)) && (_rudd < (_rudd_min + 80)) && (j < 4200)){
 /* Get and encode RC signal, print results and counter duration to the serial monitor when
debugging is on                                                                 DEBUGGIN */
    if (debug == true){
      Serial.print("checkDisArm   ");                              // where am I?
    }
    else if (DEBUGGING == 1){
      Serial.println(j);
    }
    printSignal();
    // increament counter
          if (DEBUGGING == 0){                                     // DEBUGGING off
         /* get duration number determined from the time taken to loop to here for this case */
j = j + 1;
      delay(400);
}
    else {                 // DEBUGGING on
      j = j + 110;

    }

    getSignal();
    encodeSignal();
  }
  //disarms the device if disarm signal is held for 4 seconds
  if (j > 4200){
    // disarm device
    j = 4500;
    i = 0;
    arm = false;
    // output no values
output0();
  }
  else {
    j = 0;
  }
  // device arming alert
  deviceDisarming_alert();
}
```

Output.pde

```
/**********************Output related functions******************************/

void getSignal()
{
  /* variable to store RC input read function duration, equate to elapsed time
                                                                        DEBUGGING */
  if (DEBUGGING == 1){
    rc_read_time = millis();
  }

    /* reads the pulse values from RC receiver stores the signals in the buff array get the
    running average of the signals and constrain it between 1100 and 1900  */
    thro[0] =  pulseIn(ch1, HIGH);

    // used to provide the average of 10 consecctive readings
    thro[9] = thro[8];              thro[8] = thro[7];              thro[7] = thro[6];
    thro[6] = thro[5];              thro[5] = thro[4];              thro[4] = thro[3];
    thro[3] = thro[2];              thro[2] = thro[1];              thro[1] = thro[0];
    _thro = thro[0] + thro[1] + thro[2] + thro[3] + thro[4] + thro[5] + thro[6] +   thro[7] + thro[8] +
    thro[9];
    _thro = _thro / 10;
    _thro = constrain(_thro, _thro_min, _thro_max);

    aile[0] =  pulseIn(ch2, HIGH);

    //  used to provide the moving average over a range 10 readings
    aile[9] = aile[8];              aile[8] = aile[7];              aile[7] = aile[6];
    aile[6] = aile[5];              aile[5] = aile[4];              aile[4] = aile[3];
    aile[3] = aile[2];              aile[2] = aile[1];              aile[1] = aile[0];
    _aile = aile[0] + aile[1] + aile[2] + aile[3] + aile[4] + aile[5] + aile[6] + aile[7] + aile[8] + aile[9];
    _aile = _aile / 10;
    _aile = constrain(_aile, _aile_min, _aile_max);

    elev[0] =  pulseIn(ch3, HIGH);

    //  used to provide the moving average over a range 10 readings
    elev[9] = elev[8];              elev[8] = elev[7];              elev[7] = elev[6];
    elev[6] = elev[5];              elev[5] = elev[4];              elev[4] = elev[3];
    elev[3] = elev[2];              elev[2] = elev[1];              elev[1] = elev[0];
    _elev= elev[0] + elev[1] + elev[2] + elev[3] + elev[4] + elev[5] + elev[6] + elev[7] + elev[8] +
    elev[9];
    _elev = _elev / 10;
    _elev = constrain(_elev, _elev_min, _elev_max);

    rudd[0] =  pulseIn(ch4, HIGH);

    // used to provide the moving average over a range 10 readings
    rudd[9] = rudd[8];      rudd[8] = rudd[7];      rudd[7] = rudd[6];
    rudd[6] = rudd[5];      rudd[5] = rudd[4];      rudd[4] = rudd[3];
    rudd[3] = rudd[2];      rudd[2] = rudd[1];      rudd[1] = rudd[0];
    _rudd = rudd[0] + rudd[1] + rudd[2] + rudd[3] + rudd[4] + rudd[5] + rudd[6] + rudd[7] +
    rudd[8] + rudd[9];
    _rudd = _rudd / 10;
    _rudd = constrain(_rudd, _rudd_min, _rudd_max);

  //int gear[0] =  pulseIn(ch5, LOW);
  aux1 = pulseIn(ch6, HIGH);
  aux1 =  constrain(aux1, aux1_min, aux1_max);
  //int aux2[0] =  pulseIn(ch7, HIGH);

  // store actual RC input read time                                      DEBUGGING
  if (DEBUGGING == 1){
    rc_read_time = millis() - rc_read_time;
  }
}

void output0()
{
  // outputs zero to the array of 8 bit data sent over the I2C interface
  if (setup_mode == false){
    rc[0] = 4;                        // high byte
    rc[1] = 176;                                          // low byte
    rc[2] = 4;                        // high byte
    rc[3] = 176;                                          // low byte
    rc[4] = 4;                        // high byte
    rc[5] = 176;                                          // low byte
    rc[6] = 4;                        // high byte
    rc[7] = 176;                                          // low byte
    rc[8] = 80;
  }
  else if (setup_mode == true){
    rc[0] = _thro >> 8;               // high byte
    rc[1] = _thro;                                        // low byte
    rc[2] = _aile >> 8;               // high byte
    rc[3] = _aile;                                        // low byte
    rc[4] = _elev >> 8;               // high byte
    rc[5] = _elev;                                        // low byte
    rc[6] = _rudd >> 8;               // high byte
    rc[7] = _rudd;                                         // low byte
    rc[8] = 80;
  }
}

void encodeSignal()
{
  if (arm == false){
    output0();
  }
  else if (arm == true){
    // creat an array of 8 bit data to be sent over the I2C interface
    rc[0] = _thro >> 8;               // high byte
    rc[1] = _thro;                                        // low byte
    rc[2] = _aile >> 8;               // high byte
    rc[3] = _aile;                                        // low byte
    rc[4] = _elev >> 8;               // high byte
    rc[5] = _elev;                                        // low byte
    rc[6] = _rudd >> 8;               // high byte
    rc[7] = _rudd;                                        // low byte
    /*rc[10] = gear >> 8;
      rc[11] = gear;*/
    rc[8] = map (aux1, aux1_min, aux1_max, 90, 255);
    //rc[6] = map (aux2, 1090, 1900, 0, 255);
    /*rc[12] = aux2 >> 8;
      rc[13] = aux2;*/
  }
```

_Debug.pde

```
/*********************Debugging functions*********************************/
void checkDebugMode(){
  // check for debug on
  if (debug == true){
    // listen for debugging mode switch
    while (Serial.available()){
      DEBUGGING = Serial.read() - '0';
    }
  }
  else {
    DEBUGGING = 0;
  }
}

void printSignal()
{
  // print rc read results and read duration to the serial monitor:                    DEBUGGING
  if (DEBUGGING == 1){
    getSignal();
    Serial.print("thro = " );
    Serial.print(_thro);
    Serial.print("   elev = " );
    Serial.print(_elev);
    Serial.print("   aile = " );
    Serial.print(_aile);
    Serial.print("   rudd = " );
    Serial.print(_rudd);
    //Serial.print(" gear = " );
    //Serial.print(gear);
    Serial.print("   aux1 = " );
    Serial.println(aux1);
    //Serial.print(" aux2 = " );
    //Serial.println(aux2);

    Serial.print("rc read time = " );
    Serial.print(rc_read_time);
    Serial.println(" ms");
    delay(5);
  }
  else if (DEBUGGING == 2){
   getSignal();
   Serial.print("thro = " );
   Serial.print(pulseIn(ch1, HIGH));
   Serial.print("   elev = " );
   Serial.print(pulseIn(ch3, HIGH));
   Serial.print("   aile = " );
   Serial.print(pulseIn(ch2, HIGH));
   Serial.print("   rudd = " );
   Serial.print(pulseIn(ch4, HIGH));
   //Serial.print(" gear = " );
   //Serial.print(gear);
   Serial.print("   aux1 = " );
   Serial.println(pulseIn(ch6, HIGH));
   //Serial.print(" aux2 = " );
   //Serial.println(aux2);

   Serial.print("rc read time = " );
   Serial.print(rc_read_time);
   Serial.println(" ms");
   delay(5);
  }
  else if (DEBUGGING == 3){
    Serial.print("rc[0] = " );
    Serial.print(rc[0], DEC);
    Serial.print("   rc[1] = " );
    Serial.print(rc[1], DEC);
    Serial.print("   rc[2] = " );
    Serial.print(rc[2], DEC);
    Serial.print("   rc[3] = " );
    Serial.print(rc[3], DEC);
    Serial.print("   rc[4] = " );
    Serial.println(rc[4], DEC);

    Serial.print("i = " );
    Serial.println(i);
    Serial.print("j = " );
    Serial.println(j);
    delay(5);
  }
  else if (DEBUGGING == 4){
    Serial.print("thro min = " );
    Serial.print(_thro_min);
    Serial.print("   thro max = " );
    Serial.print(_thro_max);
    Serial.print("   elev min = " );
    Serial.print(_elev_min);
    Serial.print("   elev max = " );
    Serial.print(_elev_max);
    Serial.print("   aile min = " );
    Serial.print(_aile_min);
    Serial.print("   aile max = " );
    Serial.print(_aile_max);
    Serial.print("   rudd min = " );
    Serial.print(_rudd_min);
    Serial.print("   rudd max = " );
    Serial.print(_rudd_max);
    //Serial.print(" gear = " );
    //Serial.print(gear);
    Serial.print("   aux1 min = " );
    Serial.print(aux1_min);
    Serial.print("   aux1 max = " );
    Serial.print(aux1_max);
    Serial.print("   RC_S = " );
    Serial.println(RC_S);
    delay(5);
  }
}
```

Alerts.pde

```
/**********************Alert functions*********************************/
// device armed alert
void deviceArming_alert(){
  if (i > 4200){
    digitalWrite(led,HIGH);          l = HIGH;               delay(1000);
    digitalWrite(led,LOW);           l = LOW;                delay(200);
    digitalWrite(led,HIGH);          l = HIGH;               delay(1000);
    digitalWrite(led,LOW);           l = LOW;                delay(200);
  }
}


// device disarmed alert
void deviceDisarming_alert(){
  if ((j == 4500) && (i == 0)){
    digitalWrite(led,HIGH);          l = HIGH;               delay(100);
    digitalWrite(led,LOW);           l = LOW;                delay(100);
    digitalWrite(led,HIGH);          l = HIGH;               delay(100);
    digitalWrite(led,LOW);           l = LOW;                delay(200);
    digitalWrite(led,HIGH);          l = HIGH;               delay(100);
    digitalWrite(led,LOW);           l = LOW;                delay(100);
    digitalWrite(led,HIGH);          l = HIGH;               delay(100);
    digitalWrite(led,LOW);           l = LOW;
  }
}


void deviceDisarmed_alert(){
  if (signal_setup == false){
    // turn on led
    digitalWrite(led, HIGH);         l = HIGH;
  }

  // device disarmed alert blink without delay
  else if (signal_setup == true){
    k++;
    switch (k){
    case 1:
      digitalWrite(led, LOW);        l = LOW;                break;
    case 10:
      digitalWrite(led, HIGH);       l = HIGH;               break;
    case 15:
      digitalWrite(led, LOW);        l = LOW;                break;
    case 90:
      digitalWrite(led, HIGH);       l = HIGH;               break;
    case 95:
      digitalWrite(led, LOW);        l = LOW;                break;
    }
    if (k >= 170){
      k = 0;
    }
  }
}
```

Setup.pde

```
/*********************Setup functions*********************************/
void setup_RC() {
 // state of gear channel
 RC_S = pulseIn(ch5, HIGH);

 // RC signal setup for live use, activated with the gear channel
 while (RC_S > 1600){
   setup_mode = true;
   getSignal();
   encodeSignal();

   if (signal_setup == false){

     // turn on led
     digitalWrite(led, HIGH);
     l = HIGH;
   }
   int throttle = pulseIn(ch1, HIGH);

   // get the minimum pulse signals
   _thro_min = min(_thro_min, throttle);

  // get the maximum pulse signals
   _thro_max = max(_thro_max, throttle);

   int aileron = pulseIn(ch2, HIGH);

   // get the minimum pulse signals
   _aile_min = min(_aile_min, aileron);

   // get the maximum pulse signals
   _aile_max = max(_aile_max, aileron);

   int elevator = pulseIn(ch3, HIGH);

   // get the minimum pulse signals
   _elev_min = min(_elev_min, elevator);

   // get the maximum pulse signals
   _elev_max = max(_elev_max, elevator);

   int rudder = pulseIn(ch4, HIGH);

   // get the minimum pulse signals
   _rudd_min = min(_rudd_min, rudder);

   // get the maximum pulse signals
   _rudd_max = max(_rudd_max, rudder);

   int auxillary1 = pulseIn(ch6, HIGH);

   // get the minimum pulse signals
   aux1_min = min(aux1_min, auxillary1);

   // get the maximum pulse signals
   aux1_max = max(aux1_max, auxillary1);

   if (((_thro_max - _thro_min) > 900) && ((_elev_max - _elev_min) > 900) &&
     ((_aile_max - _aile_min) > 900) && ((_rudd_max - _rudd_min) > 900) &&
     ((aux1_max - aux1_min) > 900)){
     // switch RC signal setup on
```

```
      signal_setup = true;
    }

    if (signal_setup == true){

      // turn off led
      digitalWrite(led, LOW);
      l = LOW;
    }

    checkDebugMode();

    if ((debug == true) && (signal_setup == true)){
      Serial.print("setup_RC_True   ");              // where am i?
    }
    else if ((debug == true) && (signal_setup == false)){
      Serial.print("setup_RC_False   ");              // where am i?
    }
    printSignal();

    RC_S = pulseIn(ch5, HIGH);
  }

  setup_mode = false;
}
```

C2: Main MCU Software

The main MCU software is written in C also using the Arduino Integrated Development Environment (IDE).

This software is made up of twelve Arduino IDE (.pde) files. The MAX32.pde file contains the setup and main program sequence.

The Input.pde file contains the routine for RC signal from the secondary MCU and the _Setup.pde file contains the calibration routine for all the sensors.

The I2C.pde file contains the I^{2}C scripts used for communication over the I^{2}C bus; the ADXL345.pde, HMC5883L.pde and ITG3200.pde define the routines for setup and communication to the MEMS accelerometer, magnetometer and the angular rate sensor.

The sensor fusion algorithm attitude estimation based on the extended Kalman filter is written in IMU_k.pde while IMU_q.pde contains the gradient descent based IMU sensor fusion algorithm.

The Output.pde file contain the signal encode and other output related functions. The Debug.pde and Alerts.pde contains the routine for communicating to a serial monitor for debugging purposes and routine for visual alerts. Finally, the Setup.pde file contains the routine for calibrating the RC signal.

The contents of these files are presented below.

MAX32.pde

```
/*                  INSTRUCTION NOTE
 To be loaded on MAX32

 Reads RC output from the ATMega328 over the I2C interface
 using the TWI/I2C library by Nicholas Zambetti 2006 and the Servo ver2
 library by Michael Margolis 2009.

 Part of the sketch provides means to print the received
 data and other variables for debugging.
 Simply change the value of the "DEBUGGING" variable
 (0 = debugging off; 1 = input debugging mode;
 2 = output debugging mode; 3 = imu debugging mode
 3 = imu raw readings debugging mode;
 4 = imu attitude debugging mode;)

 ________________*CONNECTIONS*_______________________________
 ____ATMega328__________|___________MAX32____________________
 A4 SDA                 |            20 SDA
 A5 SCL                 |            21 SCL
 GND                    |             GND
 _______________________|____________________________________
    RC ARRAY____________|__________RC Receiver CHANNEL_______
 RC[0]                  |                  d_col
 RC[1]                  |                  d_lat
 RC[2]                  |                  d_long
 RC[3]                  |                  d_ped
 RC[4]                  |                  d_mode
 RC[5]                  |                  RC_ch5
 RC[6]                  |                  RC_ch7
 _______________________|_____________________________________
     MAX32______________|______VTOL Propulsion Components______
 digital pin 6          |    Left Electric Motor (e_motor1)
 digital pin 7          |    Right Electric Motor (e_motor2)
 digital pin 8          |  Left rotor tilt servo (servo_tilt1)
 digital pin 9          |  Right rotor tilt servo (servo_tilt2)
 digital pin 10         |                   aux1
 digital pin 11         |                   aux2
 digital pin 12         |                   aux3
 digital pin 13         |          External Status LED
 _______________________|____________________________________
 digital pin 4          |          Status led (green)
 digital pin 5          |          Status led (blue)
 _______________________|____________________________________

 Written by Cyprian Onochie  18 December 2011
 */


#include <Wire.h>
#include <Servo.h>
#include <math.h>


// ADXL345 Constants
//------------------------------------------------------------
// ADXL345 ACC address
#define ACCELEROMETER  0x53
// number of bytes to read each time (two bytes for each axis)
#define A_TO_READ      6
// power control register
#define A_POWER_CTL    0x2D
// data format register
#define A_DATA_FORMAT  0x31
//------------------------------------------------------------


// HMC5883L Constants
//------------------------------------------------------------
// HMC5883L 7bit address
#define COMPASS_ADDRESS       0x1E
// Configuration Register A
#define COMPASS_ConfigRegA    0x00
```

```
// Configuration Register B
#define COMPASS_ConfigRegB    0x01
// Mode Register
#define COMPASS_ModeRegister  0x02
// number of bytes to read each time (two bytes for each axis)
#define C_TO_READ         6
// default gain value
// +-1.3 Ga
#define magGain               0x20
// ModeRegister valid modes
#define ContinuousConversion  0x00
#define SingleConversion      0x01
//------------------------------------------------------------


// ITG3200 Constants
//------------------------------------------------------------
// gyro address, binary = 11101001 when AD0 is connected to Vcc
#define GYRO                 0x68
#define G_SMPLRT_DIV         0x15
#define G_DLPF_FS   0x16
#define G_INT_CFG   0x17
#define G_PWR_MGM   0x3E
// number of bytes to read each time (two bytes for each axis)
#define G_TO_READ   6
//------------------------------------------------------------


// BMP085 Constants
//------------------------------------------------------------
// 0x77 default I2C address
#define BAROMETER       0x77
#define BUFFER_SIZE     3
// Oversampling Setting
const unsigned char OSS = 0;

/* ---- Registers ---- */
#define CAL_AC1           0xAA  // R   Calibration data (16 bits)
#define CAL_AC2           0xAC  // R   Calibration data (16 bits)
#define CAL_AC3           0xAE  // R   Calibration data (16 bits)
#define CAL_AC4           0xB0  // R   Calibration data (16 bits)
#define CAL_AC5           0xB2  // R   Calibration data (16 bits)
#define CAL_AC6           0xB4  // R   Calibration data (16 bits)
#define CAL_B1            0xB6  // R   Calibration data (16 bits)
#define CAL_B2            0xB8  // R   Calibration data (16 bits)
#define CAL_MB            0xBA  // R   Calibration data (16 bits)
#define CAL_MC            0xBC  // R   Calibration data (16 bits)
#define CAL_MD            0xBE  // R   Calibration data (16 bits)
#define CONTROL           0xF4  // W   Control register
#define CONTROL_OUTPUT    0xF6  // R   Output registers 0xF6=MSB, 0xF7=LSB,
0xF8=XLSB

/************************************/
/*    REGISTERS PARAMETERS          */
/************************************/
// BMP085 Modes
// oversampling=0, internalsamples=1, maxconvtimepressure=4.5ms, avgcurrent=3uA,
// RMSnoise_hPA=0.06, RMSnoise_m=0.5
#define MODE_ULTRA_LOW_POWER    0
// oversampling=1, internalsamples=2, maxconvtimepressure=7.5ms, avgcurrent=5uA,
// RMSnoise_hPA=0.05, RMSnoise_m=0.4
#define MODE_STANDARD           1
// oversampling=2, internalsamples=4, maxconvtimepressure=13.5ms, avgcurrent=7uA,
// RMSnoise_hPA=0.04, RMSnoise_m=0.3
#define MODE_HIGHRES            2
// oversampling=3, internalsamples=8, maxconvtimepressure=25.5ms,
avgcurrent=12uA, RMSnoise_hPA=0.03, RMSnoise_m=0.25
#define MODE_ULTRA_HIGHRES      3
// "Sampling rate can be increased to 128 samples per second (standard mode) for
// dynamic measurement, .n this case it is sufficient to measure temperature only
// once per second and to use this value for all pressure measurements during
period."
// (from BMP085 data-sheet Rev1.2 page 10).
// Control register
#define READ_TEMPERATURE        0x2E
```

```
#define READ_PRESSURE           0x34
// Mean Sea Level Pressure = 1013.25 hPA (1hPa = 100Pa = 1mbar)
#define MSLP                    101325
//-----------------------------------------------------------


// IMU Constants and variables
//-----------------------------------------------------------
// proportional gain governs rate of convergence to accelerometer/magnetometer
#define Kp 10.0f
// integral gain governs rate of convergence of gyroscope biases
#define Ki 0.05f

// half the sample period in seconds
float halfT = 0;
uint32_t now = 0;
uint32_t lastUpdate = 0;

float  q0 = 1.0;
float  q1 = 0.0;
float  q2 = 0.0;
float  q3 = 0.0;
//-----------------------------------------------------------


//BMP085 Variables
//-----------------------------------------------------------
// Calibration values
int ac1 = 0;
int ac2 = 0;
int ac3 = 0;
unsigned int ac4 = 0;
unsigned int ac5 = 0;
unsigned int ac6 = 0;
int b1 = 0;
int b2 = 0;
int mb = 0;
int mc = 0;
int md = 0;
// b5 is calculated in GetTemperature(...), this variable is also used in
Altitude(...)
// so ...Temperature(...) must be called before ...Pressure(...).
long b5 = 0;
float temperature_Centigrade = 0;
int32_t altitude_MilliMetre = 0;
//-----------------------------------------------------------


// Alert Variables
//-----------------------------------------------------------
int led_green = 4;           // define digital pin for alert led (normal mode)
int led_blue = 5;                     // define digital pin for alert led
(debugging mode)
int k = 0;                   // alert counter
int l = 0;                   // led state
//-----------------------------------------------------------


// IMU Variables
//-----------------------------------------------------------
float accel[3];                       // Raw accelerometer readings (x, y, z)
float gyro[3];                        // Raw gyro readings (x, y, z) in rad/s
float compass[3];                     // Raw magnetometer reading (x, y, z)
float pitch = 0, roll = 0, yaw = 0;   // Estimations of Euler angles in degrees
// Prior estimations of Euler angles in degrees
float pitch_old = 0, roll_old = 0, yaw_old = 0;

//-----------------------------------------------------------


// Input Variables
//-----------------------------------------------------------
// Variables for RC command input
int collective = 1200;
int d_col_min = 1200;
int d_col_max = 1200;
int lateral = 1200;
int d_lat_min = 1200;
int d_lat_max = 1200;
int longitudinal = 1200;
int d_long_min = 1200;
int d_long_max = 1200;
int pedal = 1200;
int d_ped_min = 1200;
int d_ped_max = 1200;
// Constants for control command factors
const float k_col = 1;
const float k_lat = 0.25;
const float k_long = 0.5;
const float k_ped = 0.25;
int d_mode = 0;
int d_mode_min = 0;
int d_mode_max = 250;
// used to arm or disarm propulsion components
boolean armed = false;
// to increment last command input update cycle
byte gc = 0;
// 8 bit data array to serve as I2C communication receive buffer
byte rc[11] = {
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
//------------------------------------------------------------


// Output Variables
//------------------------------------------------------------
// attitude update sample period in milliseconds
int fullT = 0;
// define electric motor and servo objects for the VTOL Propulsion Components
const int e_motor1 = 6;
const int e_motor2 = 7;
Servo servo_tilt1;
Servo servo_tilt2;
// variables for propulsion system positioning control
int motor1, motor2, tilt1, tilt2;
// minimum, middle and maximum pwm(uS) for the servos range
const int s_min = 1000;
const int s_mid = 1500;
const int s_max = 2000;
// minimum and maximum range for the electric motor pwm(uS)
const int m_min = 0;
const int m_max = 255;
// PD controller factors
const int Kp_alt = 0;
const int Td_alt = 0;
const int Kp_roll = 9;
const int Td_roll = 2;
const int Kp_pitch = -15;
const int Td_pitch = -11;
const int Kp_yaw = -100;
const int Td_yaw = 0;

int PD_alt, PD_roll, PD_pitch, PD_yaw;
//-------------------------------------------------------------


// Sensor Calibration
//------------------------------------------------------------
// Sensor calibration scale and offset values
#define ACCEL_X_OFFSET ((ACCEL_X_MIN + ACCEL_X_MAX) / 2.0f)
#define ACCEL_Y_OFFSET ((ACCEL_Y_MIN + ACCEL_Y_MAX) / 2.0f)
#define ACCEL_Z_OFFSET ((ACCEL_Z_MIN + ACCEL_Z_MAX) / 2.0f)
#define ACCEL_X_SCALE (GRAVITY / (ACCEL_X_MAX - ACCEL_X_OFFSET))
#define ACCEL_Y_SCALE (GRAVITY / (ACCEL_Y_MAX - ACCEL_Y_OFFSET))
#define ACCEL_Z_SCALE (GRAVITY / (ACCEL_Z_MAX - ACCEL_Z_OFFSET))

#define COMPASS_X_OFFSET ((COMPASS_X_MIN + COMPASS_X_MAX) / 2.0f)
#define COMPASS_Y_OFFSET ((COMPASS_Y_MIN + COMPASS_Y_MAX) / 2.0f)
#define COMPASS_Z_OFFSET ((COMPASS_Z_MIN + COMPASS_Z_MAX) / 2.0f)
#define COMPASS_X_SCALE (100.0f / (COMPASS_X_MAX - COMPASS_X_OFFSET))
#define COMPASS_Y_SCALE (100.0f / (COMPASS_Y_MAX - COMPASS_Y_OFFSET))
```

```
#define COMPASS_Z_SCALE (100.0f / (COMPASS_Z_MAX - COMPASS_Z_OFFSET))

// Accelerometer Put MIN/MAX and OFFSET readings here
// "accel x,y,z (min/max) = X_MIN/X_MAX  Y_MIN/Y_MAX  Z_MIN/Z_MAX"
#define ACCEL_X_MIN ((float) -271)
#define ACCEL_X_MAX ((float) 277)
#define ACCEL_Y_MIN ((float) -311)
#define ACCEL_Y_MAX ((float) 290)
#define ACCEL_Z_MIN ((float) -270)
#define ACCEL_Z_MAX ((float) 290)

// Magnetometer
// "compass x,y,z (min/max) = X_MIN/X_MAX  Y_MIN/Y_MAX  Z_MIN/Z_MAX"
#define COMPASS_X_MIN ((float) -212)
#define COMPASS_X_MAX ((float) 253)
#define COMPASS_Z_MIN ((float) -354)
#define COMPASS_Z_MAX ((float) 93)
#define COMPASS_Y_MIN ((float) -344)
#define COMPASS_Y_MAX ((float) 100)

// Gyroscope
// "gyro x,y,z (current/average) = .../OFFSET_X  .../OFFSET_Y  .../OFFSET_Z
const float GYRO_AVERAGE_OFFSET_X = 0.01;
const float GYRO_AVERAGE_OFFSET_Y = -0.09;
const float GYRO_AVERAGE_OFFSET_Z = -0.02;

// "1G reference" used for accelerometer calibration
#define GRAVITY 256.0f
// turn on sensor calibration mode
boolean CalibrationMode = false;
//------------------------------------------------------------

// Debugging Variables
//------------------------------------------------------------
String BOLDOFF = "\033[0m";
String BOLD = "\033[1m";
int Total_Duration = 0;
// variable to store I2C read function duration
int wire_read_time = 0;
// variable to store servo write function duration
int servo_write_time = 0;
// variable to store attitude determination function duration
int attitude_get_time = 0;
// includes the duration of I2C communication to IMU devices
int debug_print_time = 0;
int old_debugging_mode = 0;
// 0 = debugging off; 1 = input debugging mode; 2 = output debugging mode;
// 3 = imu raw readings debugging mode; 4 = imu attitude debugging mode;
int DEBUGGING = 0;
boolean caption_Print = false;
int SerialSpeed;
// used to enable debugging, DEBUG_TRUE = on; DEBUG_FALSE = off
#define DEBUG_FALSE
//------------------------------------------------------------


// Task Path Constants and Variables
//------------------------------------------------------------
const uint32_t LOW_FREQUENCY_PATH_DURATION_MICROSECONDS     = 100000;
const uint32_t MEDIUM_FREQUENCY_PATH_DURATION_MICROSECONDS  = 10000;
const uint32_t HIGH_FREQUENCY_PATH_DURATION_MICROSECONDS    = 2500;

uint32_t Low_Frequency_Path_Timer_MicroSeconds        = 0;
boolean Run_Low_Frequency_Tasks = true;
uint32_t Medium_Frequency_Path_Timer_MicroSeconds     = 0;
boolean Run_Medium_Frequency_Tasks = true;
uint32_t High_Frequency_Path_Timer_MicroSeconds       = 0;
boolean Run_High_Frequency_Tasks = true;
//------------------------------------------------------------

void setup(){
  initPathTimers();
  // join i2c bus (address optional for master)
  Wire.begin();
#ifdef DEBUG_TRUE
  Serial.begin(115200);
#endif

  initAcc();                   // initializes the accelerometer
  initGyro();                  // initializes the gyroscope
  initCompass();               // initializes the magnetometer
  initBar();                   // initializes the pressure sensor
  setup_alert();               // set-up the led for alerts

  calibrateSensors();
  startUp();                   // print to serial monitor when ready after restart

  // code below is used for platform validation only
  armDisarmCheck();
  checkDebugMode();
  getCommand();
  getAttitude();
  computeOutput();
  controlOutput();
  deviceArmed_alert();
  // end of platform validation code
}

void loop(){
#ifdef DEBUG_TRUE
  Total_Duration = micros();
#endif
  // check for debug mode status
  checkDebugMode();
  updatePathTimers();

  if (Run_Low_Frequency_Tasks){
    getCommand();
    Run_Low_Frequency_Tasks = false;
  }

  if (Run_Medium_Frequency_Tasks){
    getAttitude();
    Run_Medium_Frequency_Tasks = false;
  }

  if (Run_High_Frequency_Tasks){
    computeOutput();
    controlOutput();
    // armDisarmCheck();       // Line uncommented for platform validation
    deviceArmed_alert();
    Run_High_Frequency_Tasks = false;
  }

#ifdef DEBUG_TRUE
  Total_Duration = micros() - Total_Duration;
#endif
  // print results to the serial monitor:
  debugPrint();
}
```

_Setup.pde

```
/*************** Sensor setup and calibration functions ************************/
void calibrateSensors(){
  sensorcalibStartup();

  while(CalibrationMode == true){
    if(CalibrationMode == false) break;
    calibrateAccelerometer();

    if (CalibrationMode == false) break;
    calibrateGyroscope();

    if (CalibrationMode == false) break;
    calibrateMagnetometer();

    if (CalibrationMode == false) break;
    calibrateBarometer();

    exitMode(" ", " ", " ", " ", " ", "Do you want to continue sensor 
calibration?");
  }
}

//Determine the min. and max. values for the earth gravitation on all axis.
void calibrateAccelerometer(){
  float accel_x_min=0, accel_x_max=0;
  float accel_y_min=0, accel_y_max=0;
  float accel_z_min=0, accel_z_max=0;

  exitMode("Calibrate Accelerometer",
  "Point any of the board axis straight down",
  "while carefully tilting the board a little in every direction",
  "Do the same for all axis for both direction",
  "Move the board as slow as practicable, only gravity effect is desired",
  "Continue sensor calibration?");

  Serial.println("");
  Serial.print(BOLD);
  Serial.println("Calibrating Accelerometer ...............");
  Serial.print(BOLDOFF);
  Serial.println("Press any key to continue");
  Serial.println("");
  // pause for input
  while(Serial.available() == 0){
    readAccelerometer(accel);
    accel_x_min = min(accel_x_min, accel[0]);          // get the minimum values
    accel_x_max = max(accel_x_max, accel[0]);          // get the maximum values
    accel_y_min = min(accel_y_min, accel[1]);          // get the minimum values
    accel_y_max = max(accel_y_max, accel[1]);          // get the maximum values
    accel_z_min = min(accel_z_min, accel[2]);          // get the minimum values
    accel_z_max = max(accel_z_max, accel[2]);          // get the maximum values

    if(Serial.available()){
      Serial.flush();
      Serial.print(BOLD);
      Serial.print("accel x min = ");
      Serial.print(accel_x_min);
      Serial.print("     accel x_max = ");
      Serial.println(accel_x_max);
      Serial.print("accel y_min = ");
      Serial.print(accel_y_min);
      Serial.print("     accel y_max = ");
      Serial.println(accel_y_max);
      Serial.print("accel z_min = ");
      Serial.print(accel_z_min);
      Serial.print("     accel z_max = ");
      Serial.println(accel_z_max);
      Serial.print(BOLDOFF);
      Serial.println(" ");
      break;
    }
  }
}
```

```
// determine the average the noise of the gyroscope on all three axes
void calibrateGyroscope(){
  float gyro_x_ave=0, gyro_y_ave=0, gyro_z_ave=0;
  float num_samples=0;

  exitMode("Calibrate Gyroscope", "Lay the board still on a table.",
  "Wait 10 seconds, and do not move the board.",
  " ", " ", "Continue sensor calibration?");

  Serial.println("");
  Serial.print(BOLD);
  Serial.println("Calibrating Gyroscope ...............");
  Serial.print(BOLDOFF);
  Serial.println("Press any key to continue");
  Serial.println("");
  while (Serial.available() == 0){             // pause for input
    num_samples++;                             // increament samples counter
    readGyroscope(gyro);
    gyro_x_ave = gyro_x_ave + gyro[0];         // sum up the noise
    gyro_y_ave = gyro_y_ave + gyro[1];
    gyro_z_ave = gyro_z_ave + gyro[2];
    if (Serial.available()){                   // the noise sum
      gyro_x_ave = gyro_x_ave / num_samples;
      gyro_y_ave = gyro_y_ave / num_samples;
      gyro_z_ave = gyro_z_ave / num_samples;
      Serial.flush();
      Serial.print(BOLD);
      Serial.print("gyro x_ave = ");
      Serial.println(gyro_x_ave);
      Serial.print("gyro y_ave = ");
      Serial.println(gyro_y_ave);
      Serial.print("gyro z_ave = ");
      Serial.println(gyro_z_ave);
      Serial.print(BOLDOFF);
      Serial.println(" ");
      break;
    }
  }
}

// Determine the min. and max. values for the earth magnetic field on each axis.
void calibrateMagnetometer(){
  float compass_x_min=0, compass_x_max=0;
  float compass_y_min=0, compass_y_max=0;
  float compass_z_min=0, compass_z_max=0;

  exitMode("Calibrate Magnetometer",
  "Hold the board flat like a compass with the x-axis pointing north",
  "Rotate the board around the y-axis so it starts pointing down",
  "Do the same for all axes for both directions",
  "Stay away from any source of magnetic distortions",
  "Continue sensor calibration?");

  Serial.println("");
  Serial.print(BOLD);
  Serial.println("Calibrating Magnetometer ...............");
  Serial.print(BOLDOFF);
  Serial.println("Press any key to continue");
  Serial.println("");
  while (Serial.available() == 0){                         // pause for input
    readCompass(compass);
    compass_x_min = min(compass_x_min, compass[0]);     // get the minimum values
    compass_x_max = max(compass_x_max, compass[0]);     // get the maximum values
    compass_y_min = min(compass_y_min, compass[1]);     // get the minimum values
    compass_y_max = max(compass_y_max, compass[1]);     // get the maximum values
    compass_z_min = min(compass_z_min, compass[2]);     // get the minimum values
    compass_z_max = max(compass_z_max, compass[2]);     // get the maximum values

    Serial.print("compass x_min = ");
    Serial.print(compass_x_min);
    Serial.print("     compass x_max = ");
    Serial.print(compass_x_max);
    Serial.print("     compass y_min = ");
    Serial.print(compass_y_min);
    Serial.print("     compass y_max = ");
    Serial.print(compass_y_max);
    Serial.print("     compass z_min = ");
    Serial.print(compass_z_min);
    Serial.print("     compass z_max = ");
    Serial.println(compass_z_max);
    delay(100);

    if (Serial.available()){
      Serial.flush();
      Serial.print(BOLD);
      Serial.print("compass x_min = ");
      Serial.print(compass_x_min);
      Serial.print("     compass x_max = ");
      Serial.println(compass_x_max);
      Serial.print("compass y_min = ");
      Serial.print(compass_y_min);
      Serial.print("     compass y_max = ");
      Serial.println(compass_y_max);
      Serial.print("compass z_min = ");
      Serial.print(compass_z_min);
      Serial.print("     compass z_max = ");
      Serial.println(compass_z_max);
      Serial.print(BOLDOFF);
      Serial.println(" ");
      break;
    }
  }
}

void calibrateBarometer(){
}

// Apply calibration to raw sensor readings
void compensate_sensor_errors(){
  // Compensate accelerometer error
  accel[0] = (accel[0] - ACCEL_X_OFFSET) * ACCEL_X_SCALE;
  accel[1] = (accel[1] - ACCEL_Y_OFFSET) * ACCEL_Y_SCALE;
  accel[2] = (accel[2] - ACCEL_Z_OFFSET) * ACCEL_Z_SCALE;

  // Compensate magnetometer error
  compass[0] = (compass[0] - COMPASS_X_OFFSET) * COMPASS_X_SCALE;
  compass[1] = (compass[1] - COMPASS_Z_OFFSET) * COMPASS_Z_SCALE;
  compass[2] = (compass[2] - COMPASS_Y_OFFSET) * COMPASS_Y_SCALE;

  // Compensate gyroscope error
  gyro[0] = gyro[0] - GYRO_AVERAGE_OFFSET_X;
  gyro[1] = gyro[1] - GYRO_AVERAGE_OFFSET_Y;
  gyro[2] = gyro[2] - GYRO_AVERAGE_OFFSET_Z;
}

// provides the option of continuing or exiting the calibration
// displays instruction for calibration when continuing
void exitMode(char title[255], char instruction1[255], char instruction2[255],
char instruction3[255], char instruction4[255], char message[255]){
  if(CalibrationMode == true){
    // Continue Sensor Calibration?
    while(Serial.available() == 0){
      Serial.println("");
      Serial.print("        ");
      Serial.println(message);
      Serial.println("          (Y, y) YES                  (N, n) NO");
      delay(10);
      // pause for input
      while(Serial.available() == 0){
      }

      if(Serial.available()){
        byte selection = Serial.read();
        Serial.flush();

        // compare serial read to values of Y = 86 and y = 121 in the ASCII table
        if((selection == 'Y') || (selection == 'y')){
          CalibrationMode = true;
          Serial.print(BOLD);
          Serial.println("");
          Serial.println(title);
          Serial.print(BOLDOFF);
          Serial.println(instruction1);
          Serial.println(instruction2);
          Serial.println(instruction3);
          Serial.println(instruction4);
          Serial.println("Press any key to continue");
          Serial.println("");
          // pause for input
          while(Serial.available() == 0){
          }
          if(Serial.available()){
            Serial.flush();
            break;
          }
        }
        // compare serial read to values of N = 78 and n = 110 in the ASCII table
        else if((selection == 'N') || (selection == 'n')){
          CalibrationMode = false;
          break;
        }
        else{
          Serial.println("");
          Serial.println("Enter a valid input");
        }
        Serial.flush();
      }
      Serial.flush();
    }
  }
}


void initPathTimers(){
  // initialise all timers
  Low_Frequency_Path_Timer_MicroSeconds = micros();
  Medium_Frequency_Path_Timer_MicroSeconds = micros();
  High_Frequency_Path_Timer_MicroSeconds = micros();
}


void updatePathTimers(){
  if ((micros() - Low_Frequency_Path_Timer_MicroSeconds) >=
LOW_FREQUENCY_PATH_DURATION_MICROSECONDS){
    Run_Low_Frequency_Tasks = true;
    Low_Frequency_Path_Timer_MicroSeconds = micros();
  }
  if ((micros() - Medium_Frequency_Path_Timer_MicroSeconds) >=
MEDIUM_FREQUENCY_PATH_DURATION_MICROSECONDS){
    Run_Medium_Frequency_Tasks = true;
    Medium_Frequency_Path_Timer_MicroSeconds = micros();
  }
  if ((micros() - High_Frequency_Path_Timer_MicroSeconds) >=
HIGH_FREQUENCY_PATH_DURATION_MICROSECONDS){
    Run_High_Frequency_Tasks = true;
    High_Frequency_Path_Timer_MicroSeconds = micros();
  }
}
```

Input.pde

```
/********************** Input Signal related functions  ************************/

void getCommand(){
  // variable to store serial read function duration, equate to elapsed time
#ifdef DEBUG_TRUE
  wire_read_time = micros();
#endif
  /*
   // request 9 bytes from slave device #10
   Wire.requestFrom(10, 9);
   // slave may send less than requested
   while(Wire.available() >= 9)
   {
   for (byte index = 0; index < 9; index++)
   {
   // receive a byte
   rc[index] = Wire.receive();
   }
   }
   // encode the 8 bit wire buffer to actual RC signal
   collective = rc[0] << 8 | rc[1];
   lateral = rc[2] << 8 | rc[3];
   longitudinal = rc[4] << 8 | rc[5];
   pedal = rc[6] << 8 | rc[7];
   d_mode = rc[8];

   // check the arm status
   d_mode = map (d_mode, d_mode_min, d_mode_max, 0, 3);


   // get the minimum and maximum pulse signals
   if (d_mode == 0){
   d_col_min = min(d_col_min, collective);
   d_col_max = max(d_col_max, collective);

   d_lat_min = min(d_lat_min, lateral);
   d_lat_max = max(d_lat_max, lateral);

   d_long_min = min(d_long_min, longitudinal);
   d_long_max = max(d_long_max, longitudinal);

   d_ped_min = min(d_ped_min, pedal);
   d_ped_max = max(d_ped_max, pedal);
   }

   if (d_mode != 0){
   collective = map (collective, d_col_min, d_col_max, 0, 1000);
   lateral = map (lateral, d_lat_min, d_lat_max, -400, 400);
   longitudinal = map (longitudinal,d_long_min, d_long_max, -400, 400);
   pedal = map (pedal, d_ped_min, d_ped_max, -400, 400);
   }
   */

  // actual code commented out above
  // code below is used for platform validation only
  collective = 0;
  lateral = 0;
  longitudinal = 0;
  pedal = 0;
  d_mode = 2;
  // end of platform validation code


  // store actual wire read duration
DEBUGGING
#ifdef DEBUG_TRUE
  wire_read_time = micros() - wire_read_time;
#endif
}
```

I2C.pde

```
/**********************I2C Functions*****************************/

// Write val to address register on I2C device
void writeTo(int DEVICE, byte address, byte val) {
  // start transmission to I2C device
  Wire.beginTransmission(DEVICE);
  // send register address
  Wire.send(address);
  // send value to write
  Wire.send(val);
  // end transmission
  Wire.endTransmission();
}

// read num bytes starting from address register on I2C device in to buff array
void readFrom(int DEVICE, byte address, int num, byte buff[]){
  // start transmission to ACC
  Wire.beginTransmission(DEVICE);
  // send address to read from
  Wire.send(address);
  // end transmission
  Wire.endTransmission();

  //start transmission to I2C device
  Wire.beginTransmission(DEVICE);
  // request 6 bytes from I2C device
  Wire.requestFrom(DEVICE, num);

  int i = 0;
  if (Wire.available() > 0){
    // I2C device may send less than requested (abnormal)
    while(Wire.available())
    {
      // receive a byte
      buff[i] = Wire.receive();
      i++;
    }
    // end transmission
    Wire.endTransmission();
  }
}

// Read 2 bytes from address register on I2C device
// First byte will be from 'address'
// Second byte will be from 'address'+1
int readIntFrom(int DEVICE, byte address){
  unsigned int msb, lsb;

  Wire.beginTransmission(DEVICE);
  Wire.send(address);
  Wire.endTransmission();

  Wire.requestFrom(DEVICE, 2);
  while(Wire.available()<2);
  msb = Wire.receive();
  lsb = Wire.receive();

  return (int) ((long)msb<<8 | (long)lsb);
}
```

ADXL345.pde

```
/**********************ADXL345 Accelerometer********************************/
void initAcc(){
  writeTo(ACCELEROMETER, A_POWER_CTL, 0x00);            // Turning on the ADXL345
  writeTo(ACCELEROMETER, A_POWER_CTL, 0x10);            // enable measure (pg 25
of 40)
  writeTo(ACCELEROMETER, A_POWER_CTL, 0x8);
  // the device is in +-4g range reading at full_res, right-justified mode(pg26
of 40)
  writeTo(ACCELEROMETER, A_DATA_FORMAT, 0x09);
}

void readAccelerometer(float * result){
  int regAddress = 0x32;       // first axis-acceleration-data register on the
ADXL345
  byte buff[A_TO_READ];
  // read the acceleration data from the ADXL345
  readFrom(ACCELEROMETER, regAddress, A_TO_READ, buff);

  // read axes, in 10 bit resolution, ie 2 bytes.  LSB first!!
  // x axis on accelerometer
  result[0] = (int16_t)((buff[1] << 8) | buff[0]);
  result[0] *= -1;

  // y axis of accelerometer
  result[1] = (int16_t)((buff[3] << 8) | buff[2]);
  result[1] *= -1;

  // z axis of accelerometer
  result[2] = (int16_t)((buff[5] << 8) | buff[4]);
  result[2] *= -1;
}
```

HMC5883L.pde

```
/*********************HMC5883L Magnetometer**********************************/

//initializes the Magnetometer
void initCompass(){    // SampleAveraging = 8,DataOutputRate = 15HZ,
NormalOperation
  Wire.beginTransmission(COMPASS_ADDRESS);
  Wire.send(COMPASS_ConfigRegA);
  Wire.send(0X70);
  Wire.endTransmission();
  delay(50);

  Wire.beginTransmission(COMPASS_ADDRESS);
  Wire.send(COMPASS_ConfigRegB);
  Wire.send(magGain);
  Wire.endTransmission();
  delay(50);

  Wire.beginTransmission(COMPASS_ADDRESS);
  Wire.send(COMPASS_ModeRegister);  // Set continuous mode (default to 15Hz)
  Wire.send(ContinuousConversion);  // End transmission
  Wire.endTransmission();
  delay(50);
}

void readCompass(float * result){
  /* Magnetometer HMC5883L I2C registers:      * x axis MSB = 03, x axis LSB = 04
   * z axis MSB = 05, y axis LSB = 06 * y axis MSB = 07, z axis LSB = 08 */
  int regAddress = 0x03;
  int x, y, z;
  byte buff[C_TO_READ];

  // read the gyro data from the ITG3200
  readFrom(COMPASS_ADDRESS, regAddress, C_TO_READ, buff);
  result[0] =  (int16_t)((buff[0] << 8) | buff[1]);
  result[1] = (int16_t)((buff[4] << 8) | buff[5]);
  result[2] = (int16_t)((buff[2] << 8) | buff[3]);
}
```

ITG3200.pde

```
/*********************ITG3200 Gyroscope********************************/

//initializes the gyroscope
void initGyro(){
  /* power management set to:         * clock select = PLL with Z Gyro reference
   * no reset, no sleep mode * no standby mode
   * parameter to +/- 2000 deg/sec    * 8kHz internal sample rate
   * low pass filter = 256Hz * sample rate  = 1kHz
   * no interrupt            */
  writeTo(GYRO, G_PWR_MGM, 0x03);
  writeTo(GYRO, G_SMPLRT_DIV, 0x07);
  // +/- 2000 degrees/sec, 1KHz, 1E, 19
  writeTo(GYRO, G_DLPF_FS, 0x18);
  writeTo(GYRO, G_INT_CFG, 0x00);
}

void readGyroscope(float * result){
  /* registers:                                * temp MSB = 1B, temp LSB = 1C
   * x axis MSB = 1D, x axis LSB = 1E * y axis MSB = 1F, y axis LSB = 20
   * z axis MSB = 21, z axis LSB = 22 */
  int regAddress = 0x1D;
  int temp, x, y, z;
  byte buff[G_TO_READ];

  //read the gyro data from the ITG3200
  readFrom(GYRO, regAddress, G_TO_READ, buff);

  result[1] = (int16_t)((buff[0] << 8) | buff[1]);
  result[0] = (int16_t)((buff[2] << 8) | buff[3]);
  result[2] = (int16_t)((buff[4] << 8) | buff[5]);

  // Convert reading to radian per seconds
  result[0] = result[0] / (14.375 * 180 /PI);
  result[1] = result[1] / (14.375 * 180 /PI);
  result[2] = result[2] / (14.375 * 180 /PI);
}
```

BMP085.pde

```
/*********************BMP085 Barometer********************************/
void initBar(){
  // Turning on the ADXL345
  // Stores all of the bmp085's calibration values into global variables
  // Calibration values are required to calculate temp and pressure
  // This function should be called at the beginning of the program
  ac1 = readIntFrom(BAROMETER, CAL_AC1);
  ac2 = readIntFrom(BAROMETER, CAL_AC2);
  ac3 = readIntFrom(BAROMETER, CAL_AC3);
  ac4 = readIntFrom(BAROMETER, CAL_AC4);
  ac5 = readIntFrom(BAROMETER, CAL_AC5);
  ac6 = readIntFrom(BAROMETER, CAL_AC6);
  b1 = readIntFrom(BAROMETER, CAL_B1);
  b2 = readIntFrom(BAROMETER, CAL_B2);
  mb = readIntFrom(BAROMETER, CAL_MB);
  mc = readIntFrom(BAROMETER, CAL_MC);
  md = readIntFrom(BAROMETER, CAL_MD);
  temperature_Centigrade = getTemperature() * 0.01;
}

short getTemperature(){
  // Calculate temperature. Value returned will be in units of 0.1 deg C
  unsigned int ut;
  // Write 0x2E into Register 0xF4. This requests a temperature reading
  Wire.beginTransmission(BAROMETER);
  Wire.send(CONTROL);
  Wire.send(READ_TEMPERATURE);
  Wire.endTransmission();
  delay(5);                  // Wait at least 4.5ms
  ut = readIntFrom(BAROMETER, CONTROL_OUTPUT); // two bytes from reg. 0xF6 & 0xF7
  long x1, x2;
  x1 = ((long)ut - ac6) * ac5 >> 15;
  x2 = ((long)mc << 11) / (x1 + md);
  b5 = x1 + x2;

  return ((b5 + 8)>>4);
}

long getAltitude(){
  // Calculate pressure
  // calibration values must be known
  // b5 is also required so getTemperature(...) must be called first.
  // Value returned will be pressure in units of Pa.

  unsigned char msb, lsb, xlsb;
  unsigned long up = 0;

  // Write 0x34+(OSS<<6) into register 0xF4
  // Request a pressure reading w/ oversampling setting
  Wire.beginTransmission(BAROMETER);
  Wire.send(CONTROL);
  Wire.send(READ_PRESSURE + (OSS<<6));
  Wire.endTransmission();

  // Wait for conversion, delay time dependent on OSS
  delay(2 + (3<<OSS));

  // Read register 0xF6 (MSB), 0xF7 (LSB), and 0xF8 (XLSB)
  Wire.beginTransmission(BAROMETER);
  Wire.send(CONTROL_OUTPUT);
  Wire.endTransmission();
  Wire.requestFrom(BAROMETER, 3);

  // Wait for data to become available
  while(Wire.available() < 3);
  msb = Wire.receive();
  lsb = Wire.receive();
  xlsb = Wire.receive();

  up = (((unsigned long) msb << 16) | ((unsigned long) lsb << 8)
  | (unsigned long) xlsb) >> (8-OSS);

  long x1, x2, x3, b3, b6, p;
  unsigned long b4, b7;

  b6 = b5 - 4000;
  // Calculate B3
  b6 = b5 - 4000;              // b5 is updated by calcTrueTemperature()
  x1 = (b2* (b6 * b6 >> 12)) >> 11;
  x2 = ac2 * b6 >> 11;
  x3 = x1 + x2;
  b3 = (((((long)ac1)*4 + x3)<<OSS) + 2)>>2;

  // Calculate B4
  x1 = ac3 * b6 >> 13;
  x2 = (b1 * (b6 * b6 >> 12)) >> 16;
  x3 = ((x1 + x2) + 2) >> 2;
  b4 = (ac4 * (uint32_t) (x3 + 32768)) >> 15;

  b7 = ((uint32_t)up - b3) * (50000 >> OSS);
  p = b7 < 0x80000000 ? (b7 << 1) / b4 : (b7 / b4) << 1;

  x1 = (p >> 8) * (p >> 8);
  x1 = (x1 * 3038) >> 16;
  x2 = (-7357 * p) >> 16;
  p += (x1 + x2 + 3791)>>4;

  return p;
}
```

Alerts.pde

```
/**********************Alert functions***********************************/

void setup_alert(){
  pinMode(led_green,OUTPUT);                // set digital pin 4 as output
  pinMode(led_blue,OUTPUT);                 // set digital pin 5 as output

  digitalWrite(led_green, LOW);             // turn off led
  digitalWrite(led_blue, LOW);
  l = LOW;
}

void deviceArmed_alert(){
  // device Armed alert blink without delay
  if (d_mode != 0 && debug == false){
    k++;
    switch (k){
    case 1:
      digitalWrite(led_green, HIGH);
      l = HIGH;
      break;
    case 101:
      digitalWrite(led_green, LOW);
      l = LOW;
      break;
    case 8101:
      digitalWrite(led_green, HIGH);
      l = HIGH;
      break;
    case 8201:
      digitalWrite(led_green, LOW);
      l = LOW;
      break;
    }
    if (k >= 16200){
      k = 0;
    }
  }
  // device Armed alert blink without delay
  // when debugging is enabled
  else if (d_mode != 0 && debug != false){
    k++;
    switch (k){
    case 1:
      digitalWrite(led_blue, HIGH);
      l = HIGH;
      break;
    case 3:
      digitalWrite(led_blue, LOW);
      l = LOW;
      break;
    case 61:
      digitalWrite(led_blue, HIGH);
      l = HIGH;
      break;
    case 63:
      digitalWrite(led_green, LOW);
      l = LOW;
      break;
    }
    if (k >= 120){
      k = 0;
    }
  }
  if (d_mode == 0){
    digitalWrite(led_green, LOW);
    digitalWrite(led_blue, LOW);
  }
}
```

IMU_K.pde

```
/**********************IMU Extended Kalman functions **************************/

void getAtitude(){
  // variable to store attitude determination function duration, equate to
elapsed time
  if (debug == true){
    atitude_get_time = micros();
  }

  readAccelerometer(accel);           // Accelerometer data in Gs
  normalize3DVec(accel);
  readGyroscope(gyro);                              // Gyroscope in deg/sec
  getInclination();
  readCompass(compass);
  getPitchRollYaw();

  // store actual attitude determination duration
  if (debug == true){
    atitude_get_time = micros() - atitude_get_time;
  }
}

void getInclination() {
  int w = 0;
  float tmpf = 0.0;
  int currentTime, signRzGyro;

  currentTime = millis();
  interval = currentTime - lastTime;
  lastTime = currentTime;

  if (firstSample) {
    for(w=0;w<=2;w++) {
      RwEst[w] = accel[w];   //initialize with accelerometer readings
    }
  }
  else{
    //evaluate RwGyro vector
    if(abs(RwEst[2]) < 0.01) {

// Rz is too small and because it is used as reference for computing Axz, Ayz
// its error fluctuations will amplify leading to bad results
// in this case skip the gyro data and just use previous estimate

      for(w=0;w<=2;w++) {
        RwGyro[w] = RwEst[w];
      }
    }
    else {
  // get angles between projection of R on ZX/ZY plane and Z axis,
  // based on last RwEst
      for(w=0;w<=1;w++){
        tmpf = gyro[w];                       // get current gyro rate in deg/s
        tmpf *= interval / 1000.0f;           // get angle change in deg
  // get angle and convert to degrees
        Awz[w] = atan2(RwEst[w],RwEst[2]) * 180 / PI;
      Awz[w] += tmpf;               // get updated angle according to gyro movement
      }

  // estimate sign of RzGyro by looking in what qudrant the angle Axz is,
  // RzGyro is pozitive if  Axz in range -90 ..90 => cos(Awz) >= 0
      signRzGyro = ( cos(Awz[0] * PI / 180) >=0 ) ? 1 : -1;

  // reverse calculation of RwGyro from Awz angles, for formulas deductions see
  // http://starlino.com/imu_guide.html
      for(w=0;w<=1;w++){
        RwGyro[0] = sin(Awz[0] * PI / 180);
        RwGyro[0] /= sqrt( 1 + squared(cos(Awz[0] * PI / 180))
         * squared(tan(Awz[1] * PI / 180)) );
        RwGyro[1] = sin(Awz[1] * PI / 180);
        RwGyro[1] /= sqrt( 1 + squared(cos(Awz[1] * PI / 180))
         * squared(tan(Awz[0] * PI / 180)) );
      }
      RwGyro[2] = signRzGyro * sqrt(1 - squared(RwGyro[0]) - squared(RwGyro[1]));
    }

    //combine Accelerometer and gyro readings
    for(w=0;w<=2;w++) RwEst[w] = (accel[w] + wGyro * RwGyro[w]) / (1 + wGyro);

    normalize3DVec(RwEst);
  }

  firstSample = false;
}

void getPitchRollYaw(){
  pitch = Awz[0];
  roll = Awz[1];
  yaw = 0;
}

void normalize3DVec(float * vector) {
  float R;
  R = sqrt(vector[0]*vector[0] + vector[1]*vector[1] + vector[2]*vector[2]);
  vector[0] /= R;
  vector[1] /= R;
  vector[2] /= R;
}

float squared(float x){
  return x*x;
}
```

IMU_Q.pde

```
/**************** IMU Gradient Descent Algorithm functions ********************/

void getAttitude(){
  // variable to store attitude determination function duration,
  // equate to elapsed
#ifdef DEBUG_TRUE
  attitude_get_time = micros();
#endif

  readAccelerometer(accel);
  readGyroscope(gyro);
  readCompass(compass);
  AHRSupdate(gyro[0], gyro[1], gyro[2], accel[0], accel[1], accel[2],
             compass[0], compass[1], compass[2]);
  getPitchRollYaw();
  altitude_MilliMetre = getAltitude();

  // store actual attitude determination duration
#ifdef DEBUG_TRUE
  attitude_get_time = micros() - attitude_get_time;
#endif
}

void AHRSupdate(float gx, float gy, float gz, float ax, float ay, float az,
                float mx, float my, float mz) {
  float norm;
  float hx, hy, hz, bx, bz;
  float vx, vy, vz, wx, wy, wz;
  float ex, ey, ez;
  // integral quaternion elements
  float iq0, iq1, iq2, iq3;
  // scaled integral error
  float exInt = 0, eyInt = 0, ezInt = 0;


  // auxiliary variables to reduce number of repeated operations
  float q0q0 = q0*q0;
  float q0q1 = q0*q1;
  float q0q2 = q0*q2;
  float q0q3 = q0*q3;
  float q1q1 = q1*q1;
  float q1q2 = q1*q2;
  float q1q3 = q1*q3;
  float q2q2 = q2*q2;
  float q2q3 = q2*q3;
  float q3q3 = q3*q3;

  now = millis();            // determine sample period in milliseconds
  fullT = now - lastUpdate;  // determine half sample period in seconds
  halfT = fullT / 2000.0;
  lastUpdate = now;


  // normalise the measurements
  norm = sqrt(ax*ax + ay*ay + az*az);
  ax = ax / norm;
  ay = ay / norm;
  az = az / norm;
  norm = sqrt(mx*mx + my*my + mz*mz);
  mx = mx / norm;
  my = my / norm;
  mz = mz / norm;


  // compute reference direction of flux
  hx = 2*mx*(0.5 - q2q2 - q3q3) + 2*my*(q1q2 - q0q3) + 2*mz*(q1q3 + q0q2);
  hy = 2*mx*(q1q2 + q0q3) + 2*my*(0.5 - q1q1 - q3q3) + 2*mz*(q2q3 - q0q1);
  hz = 2*mx*(q1q3 - q0q2) + 2*my*(q2q3 + q0q1) + 2*mz*(0.5 - q1q1 - q2q2);
  bx = sqrt((hx*hx) + (hy*hy));
  bz = hz;

  // estimated direction of gravity and flux (v and w)
  vx = 2*(q1q3 - q0q2);
  vy = 2*(q0q1 + q2q3);
  vz = q0q0 - q1q1 - q2q2 + q3q3;
  wx = 2*bx*(0.5 - q2q2 - q3q3) + 2*bz*(q1q3 - q0q2);
  wy = 2*bx*(q1q2 - q0q3) + 2*bz*(q0q1 + q2q3);
  wz = 2*bx*(q0q2 + q1q3) + 2*bz*(0.5 - q1q1 - q2q2);

  // error is sum of cross product between reference direction of
  // fields and direction measured by sensors
  ex = (ay*vz - az*vy) + (my*wz - mz*wy);
  ey = (az*vx - ax*vz) + (mz*wx - mx*wz);
  ez = (ax*vy - ay*vx) + (mx*wy - my*wx);

  // integral error scaled integral gain
  exInt = exInt + ex*Ki;
  eyInt = eyInt + ey*Ki;
  ezInt = ezInt + ez*Ki;

  // adjusted gyroscope measurements
  gx = gx + Kp*ex + exInt;
  gy = gy + Kp*ey + eyInt;
  gz = gz + Kp*ez + ezInt;

  // integrate quaternion rate and normalise
  iq0 = (-q1*gx - q2*gy - q3*gz)*halfT;
  iq1 = (q0*gx + q2*gz - q3*gy)*halfT;
  iq2 = (q0*gy - q1*gz + q3*gx)*halfT;
  iq3 = (q0*gz + q1*gy - q2*gx)*halfT;

  q0 += iq0;
  q1 += iq1;
  q2 += iq2;
  q3 += iq3;

  // normalise quaternion
  norm = sqrt(q0*q0 + q1*q1 + q2*q2 + q3*q3);
  q0 = q0 / norm;
  q1 = q1 / norm;
  q2 = q2 / norm;
  q3 = q3 / norm;
}

void getPitchRollYaw(){
  // estimated gravity direction
  float gx, gy, gz;

  gx = 2 * (q1*q3 - q0*q2);
  gy = 2 * (q0*q1 + q2*q3);
  gz = q0*q0 - q1*q1 - q2*q2 + q3*q3;

  yaw_old = yaw;
  pitch_old = pitch;
  roll_old = roll;

  yaw = -atan2(2 * q1 * q2 - 2 * q0 * q3, 2 * q0*q0 + 2 * q1 * q1 - 1)
   * 180/M_PI;
  pitch = -atan(gx / sqrt(gy*gy + gz*gz))  * 180/M_PI;
  roll = -atan(gy / sqrt(gx*gx + gz*gz))  * 180/M_PI;
}
```

Output.pde

```
/********************** Output related functions ******************************/
void armDisarmCheck(){
  /*
   // enable the propulsion hardware when armed
   if ((d_mode != 0) && (armed == false)){
   e_motor.attach(6);
   servo_rrt.attach(7);
   servo_rrp.attach(8);
   servo_lrt.attach(9);
   servo_lrp.attach(10);
   armed = true;
   }
   else if ((d_mode == 0) && (armed == true)){
   e_motor.detach();          // disable propulsion hardware when disarmed
   servo_rrt.detach();
   servo_rrp.detach();
   servo_lrt.detach();
   servo_lrp.detach();
   armed = false;
   }   */

  // code below is used for platform validation only
  pinMode(e_motor1, OUTPUT);
  pinMode(e_motor2, OUTPUT);
  servo_tilt1.attach(8);
  servo_tilt2.attach(9);
  armed = true;
  // end of platform validation code
}

void computeOutput(){
  // control allocation
  int d_motor1 = (k_col * collective) + (k_lat * lateral);
  int d_motor2 = (k_col * collective) - (k_lat * lateral);
  int d_tilt1 = -(k_long * longitudinal) + (k_ped * pedal);
  int d_tilt2 = -(k_long * longitudinal) - (k_ped * pedal);

  // attitude correction routine. Determine PD controller for roll pitch and yaw
  PD_alt = 0;
  PD_roll = (Kp_roll * roll) + (Td_roll * (roll - roll_old)/fullT);
  PD_pitch = (Kp_pitch * pitch) + (Td_pitch * (pitch - pitch_old)/fullT);
  PD_yaw = Kp_yaw * gyro[2];

  // control command adjustment
  motor1 = d_motor1 + PD_alt + PD_roll - PD_yaw;
  motor1 = constrain(motor1, m_min, m_max);
  motor2 = d_motor2 + PD_alt - PD_roll - PD_yaw;
  motor2 = constrain(motor2, m_min, m_max);
  tilt1 = s_mid + d_tilt1 + PD_pitch + PD_yaw;
  tilt1 = constrain(tilt1, s_min, s_max);
  tilt2 = s_mid + d_tilt2 + PD_pitch - PD_yaw;
  tilt2 = constrain(tilt2, s_min, s_max);
}

void controlOutput(){
#ifdef DEBUG_TRUE            // declare variable to store servo write duration
  servo_write_time = micros();
#endif

  if (d_mode != 0){
    analogWrite(e_motor1, motor1);
    analogWrite(e_motor2, motor2);
    servo_tilt1.writeMicroseconds(tilt1);
    servo_tilt2.writeMicroseconds(tilt2);
  }

#ifdef DEBUG_TRUE            // store actual servo write duration
  servo_write_time = micros() - servo_write_time;
#endif
}
```

Debug.pde

```
/**********************Debugging functions**********************************/

// used to check for debugging mode status
void checkDebugMode(){
  // check for debug on
#ifdef DEBUG_TRUE
  // listen for debugging mode switch
  while (Serial.available()){
    DEBUGGING = Serial.read() - '0';
  }
#endif
#ifdef DEBUG_FALSE
  DEBUGGING = 0;
#endif
  if (DEBUGGING != old_debugging_mode){
    caption_Print = false;
  }
  old_debugging_mode = DEBUGGING;
}


// Use this to print the results to the serial monitor
void debugPrint(){
#ifdef DEBUG_TRUE
  debug_print_time = micros();
#endif

#ifdef DEBUG_TRUE
  if (DEBUGGING == 1){
    Serial.print("d_col = " );
    Serial.print(collective);
    Serial.print("   d_lat = " );
    Serial.print(lateral);
    Serial.print("   d_long = " );
    Serial.print(longitudinal);
    Serial.print("   d_ped = " );
    Serial.print(pedal);
    Serial.print("   d_mode = " );
    Serial.println(d_mode, DEC);

    delay(100);
  }
  else if (DEBUGGING == 2){
    Serial.print("PD_roll = " );
    Serial.print(PD_roll);
    Serial.print("   PD_pitch = " );
    Serial.print(PD_pitch);
    Serial.print("   PD_yaw = " );
    Serial.print(PD_yaw);
    Serial.print("   PD_alt = " );
    Serial.println(PD_alt);

    delay(100);
  }
  else if (DEBUGGING == 3){
    Serial.print("d_col min = " );
    Serial.print(d_col_min);
    Serial.print("   d_col max = " );
    Serial.print(d_col_max);
    Serial.print("   d_lat min = " );
    Serial.print(d_lat_min);
    Serial.print("   d_lat max = " );
    Serial.print(d_lat_max);
    Serial.print("   d_long min = " );
    Serial.print(d_long_min);
    Serial.print("   d_long max = " );
    Serial.print(d_long_max);
    Serial.print("   d_ped min = " );
    Serial.print(d_ped_min);
    Serial.print("   d_ped max = " );
    Serial.print(d_ped_max);
    Serial.print("   d_mode min = " );
    Serial.print(d_mode_min);
    Serial.print("   d_mode max = " );
    Serial.println(d_mode_max);

    delay(100);
  }
  else if (DEBUGGING == 4){
    if (caption_Print == false){
      Serial.println("Platform Control OUTPUT  " );
      Serial.print("Left Electric Motor");
      Serial.print(";");
      Serial.print("Right Electric Motor");
      Serial.print(";");
      Serial.print("Left Tilt Servo (us)");
      Serial.print(";");
      Serial.print("Right Tilt Servo (us)");
      Serial.print(";");
      Serial.println("Left Pitch Servo (us)");
      Serial.print(motor1);
      Serial.print(";");
      Serial.print(motor2);
      Serial.print(";" );
      Serial.print(tilt1);
      Serial.print(";" );
      Serial.println(tilt2);
      caption_Print = true;
    }
    else {
      Serial.print(motor1);
      Serial.print(";");
      Serial.print(motor2);
      Serial.print(";" );
      Serial.print(tilt1);
      Serial.print(";" );
      Serial.println(tilt2);
    }
    delay(100);
  }
  else if (DEBUGGING == 5) {
    Serial.print("ACC[x, y, z] " );
    Serial.print(accel[0]);
    Serial.print(",  " );
    Serial.print(accel[1]);
    Serial.print(",  " );
    Serial.print(accel[2]);
    Serial.print(",        " );
    Serial.print("GYRO[x, y, z] " );
    Serial.print(gyro[0]);
    Serial.print(",  " );
    Serial.print(gyro[1]);
    Serial.print(",  " );
    Serial.print(gyro[2]);
    Serial.print(",        " );
    Serial.print("MAG[x, y, z] " );
    Serial.print(compass[0]);
    Serial.print(",  " );
    Serial.print(compass[1]);
    Serial.print(",  " );
    Serial.println(compass[2]);
    delay(100);
  }
  else if (DEBUGGING == 6){
    if (caption_Print == false){
      Serial.println("Attitude [Euler Angles](deg),
      Altitude and Temperature (0.1 deg C)" );
      Serial.print("Pitch");
      Serial.print(";");
      Serial.print("Roll");
      Serial.print(";");
      Serial.print("Yaw");
      Serial.print(";");
      Serial.print("Altitude");
      Serial.print(";");
      Serial.println("Temperature");
      Serial.print(pitch);
      Serial.print(";" );
      Serial.print(roll);
      Serial.print(";" );
      Serial.print(yaw);
      Serial.print(";" );
      Serial.println(altitude_MilliMetre);
      caption_Print = true;
    }
    else {
      Serial.print(pitch);
      Serial.print(";" );
      Serial.print(roll);
      Serial.print(";" );
      Serial.print(yaw);
      Serial.print(";" );
      Serial.println(altitude_MilliMetre);
    }
    delay(100);
  }
  else if (DEBUGGING == 7){
    // Print device state
    if (armed == true){
      Serial.print("Armed      " );
    }
    else {
      Serial.print("Disarmed   " );
    }

    // print wire read time
    Serial.print("wire read time = " );
    Serial.print(wire_read_time);
    Serial.print(" us,   ");

    // print servo write time
    Serial.print("servo write time = " );
    Serial.print(servo_write_time);
    Serial.print(" us,   ");

    // print attitude get time
    Serial.print("atitude get time = " );
    Serial.print(attitude_get_time);
    Serial.print(" us,   ");

    // print total program loop time
    Serial.print("Total Duration = " );
    Serial.print(Total_Duration);
    Serial.println(" us");
    Serial.print("Debug Print Time = ");
    Serial.println(micros() - debug_print_time);

    delay(100);
  }
  else if (DEBUGGING == 8){
    Serial.print("Q0; Q1; Q3; Q3; Roll_n-1; Pitch_n-1;
    Yaw_n-1; Sample Time (uS)" );
    Serial.print(q0);
    Serial.print(";   " );
    Serial.print(q1);
    Serial.print(";   " );
    Serial.print(q2);
    Serial.print(";   " );
    Serial.print(q3);
    Serial.print(";   " );
    Serial.print(roll_old);
    Serial.print(";   " );
    Serial.print(pitch_old);
    Serial.print(";   " );
    Serial.print(yaw_old);
    Serial.print(";   " );
    Serial.println(fullT);
    delay(100);
  }
#endif
}
```

```
// Use this to print to tell when ready after restart
DEBUGGING
void startUp(){
#ifdef DEBUG_TRUE
  if (DEBUGGING == 0){
    Serial.println("");
    Serial.print(BOLD);
    Serial.print("Starting Up");
    Serial.print(".");
    delay(200);
    Serial.print(".");
    delay(200);
    Serial.print(".");
    delay(200);
    Serial.print(".");
    delay(200);
    Serial.print(".");
    delay(200);
    Serial.print(".");
    delay(200);
    Serial.print(".");
    delay(200);
    Serial.print(".");
    delay(200);
    Serial.print(".");
    delay(200);
    Serial.println(".");
    delay(200);
    Serial.println("**********************************************************");
    Serial.println("READY");
    Serial.println("SERIAL0 CONNECTED");
    Serial.print(BOLDOFF);
  }
#endif
}


void sensorcalibStartup(){
  if (CalibrationMode == true){
    // initialize serial communications at 115200 bps:
    Serial.begin(115200);
    Serial.println("");
    Serial.print(BOLD);
    Serial.print("Entering Sensor Calibration Mode");
    Serial.print(".");
    delay(100);
    Serial.print(".");
    delay(100);
    Serial.print(".");
    delay(100);
    Serial.print(".");
    delay(100);
    Serial.print(".");
    delay(100);
    Serial.print(".");
    delay(100);
    Serial.print(".");
    delay(100);
    Serial.print(".");
    delay(100);
    Serial.print(".");
    delay(100);
    Serial.println(".");
    delay(100);
    Serial.println("**********************************************************");
    Serial.println("READY");
    Serial.print(BOLDOFF);
  }
}
```

APPENDIX D: SIMULINK MODELS FOR ALGORITHM VALIDATION

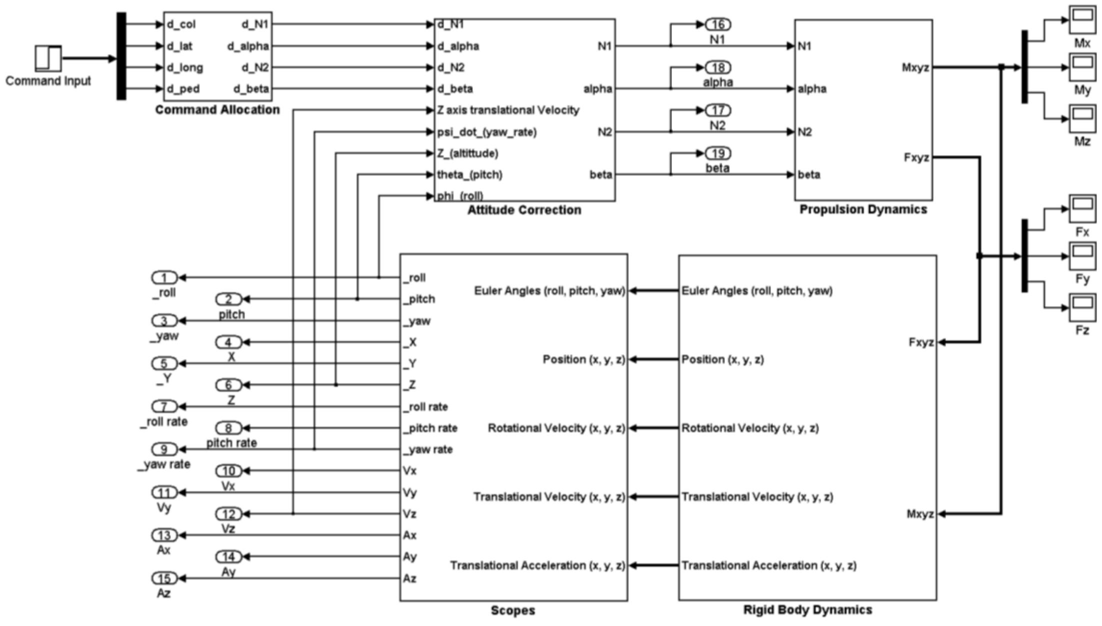


Figure D.1: VTOL Platform System Level Model in MATLAB Simulink

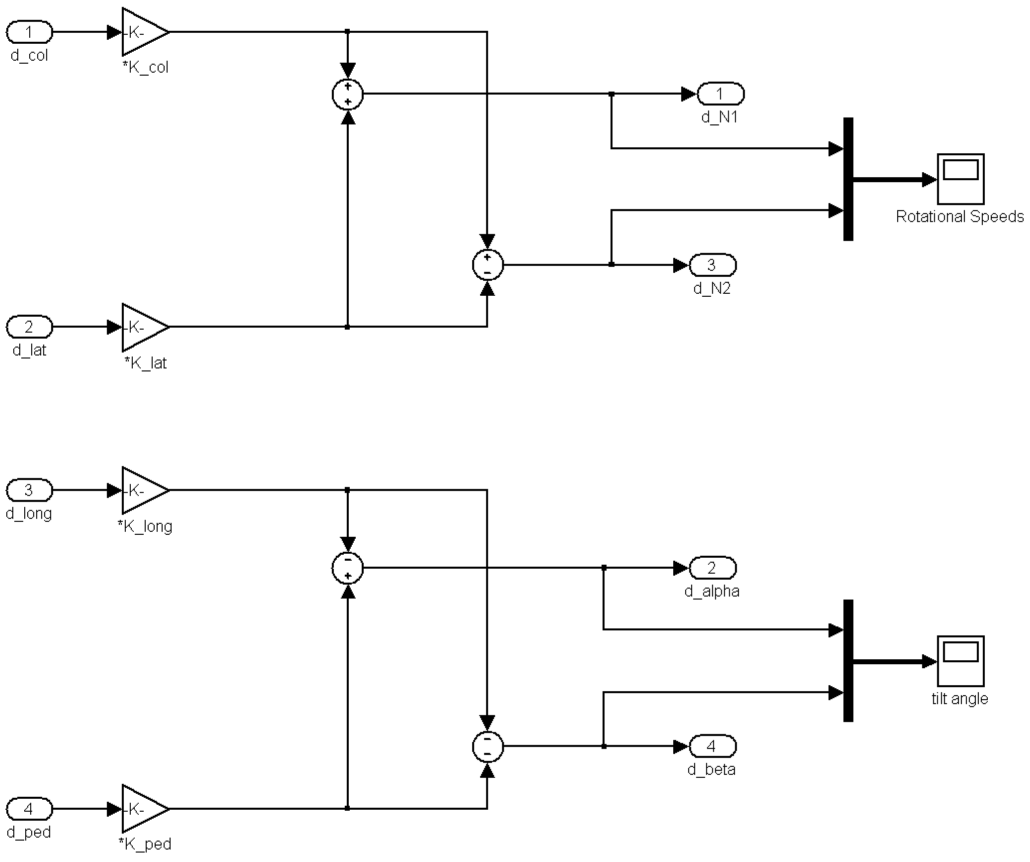


Figure D.2: Model for command allocation

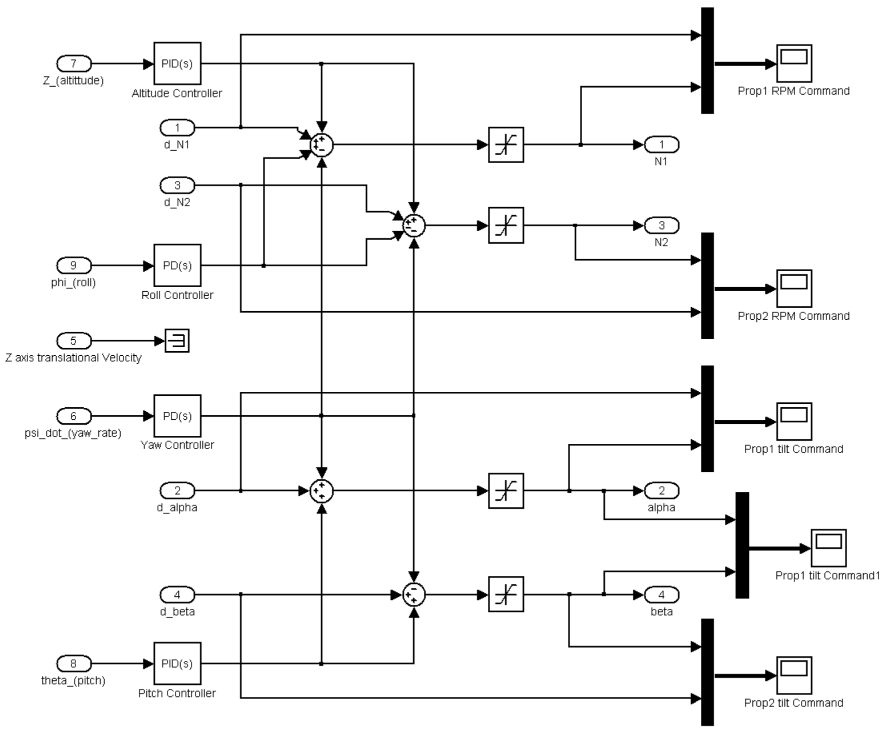


Figure D.3: Simulation model for attitude correction

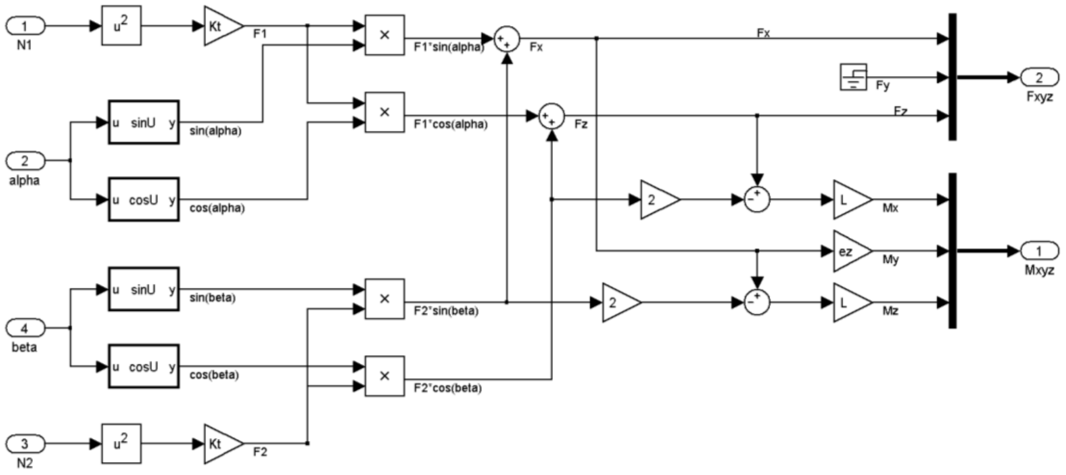


Figure D.4: Model for propulsion system dynamics

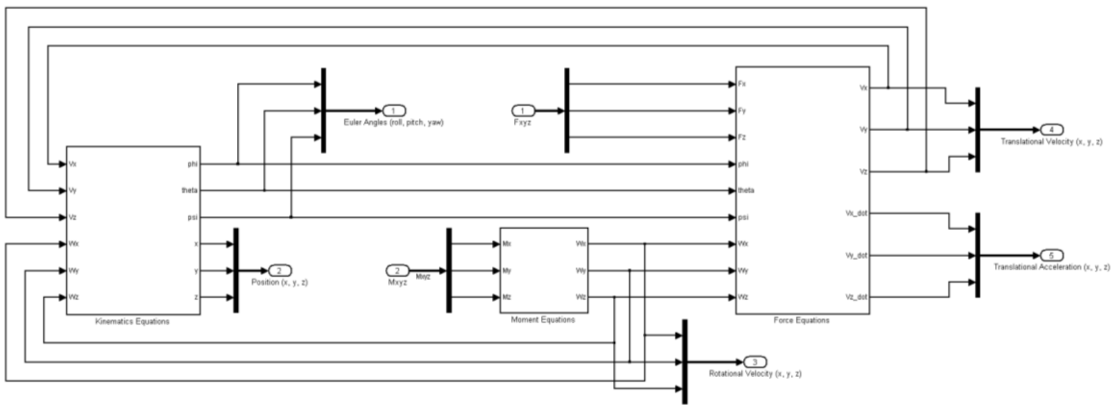


Figure D.5: Simulation Model for the rigid body dynamics

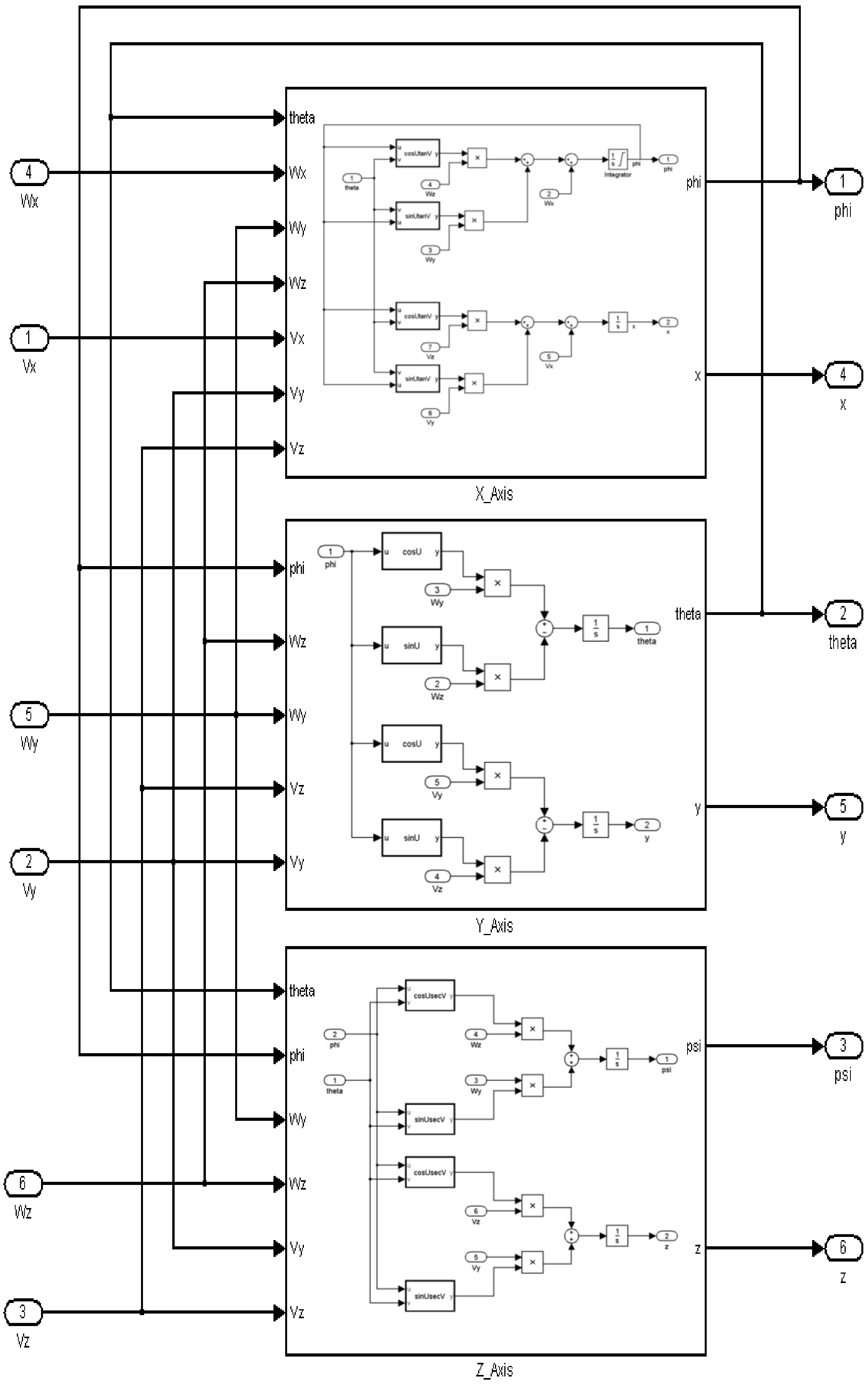


Figure D.6: Model for the kinematic equations

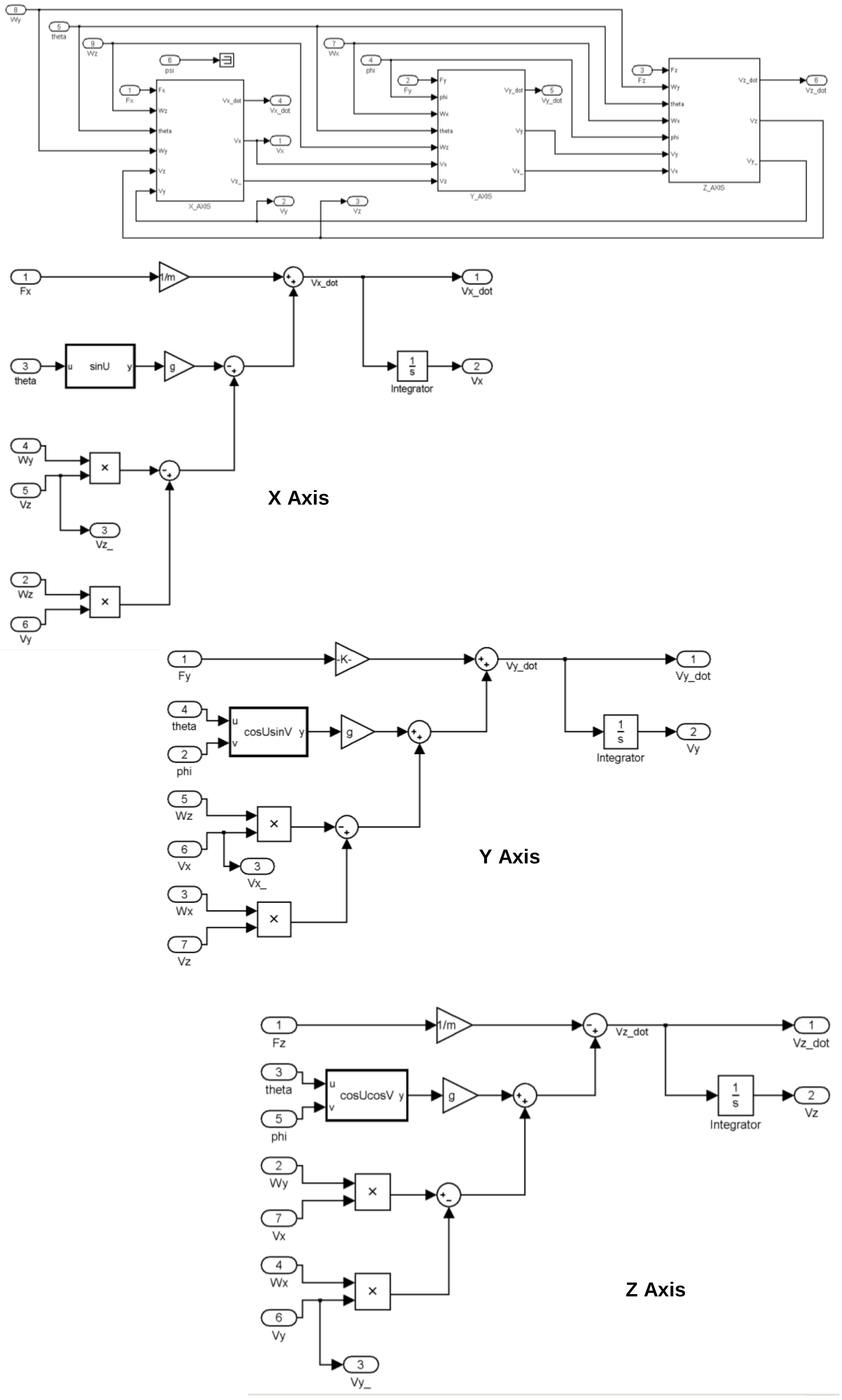


Figure D.7: Models for force equations

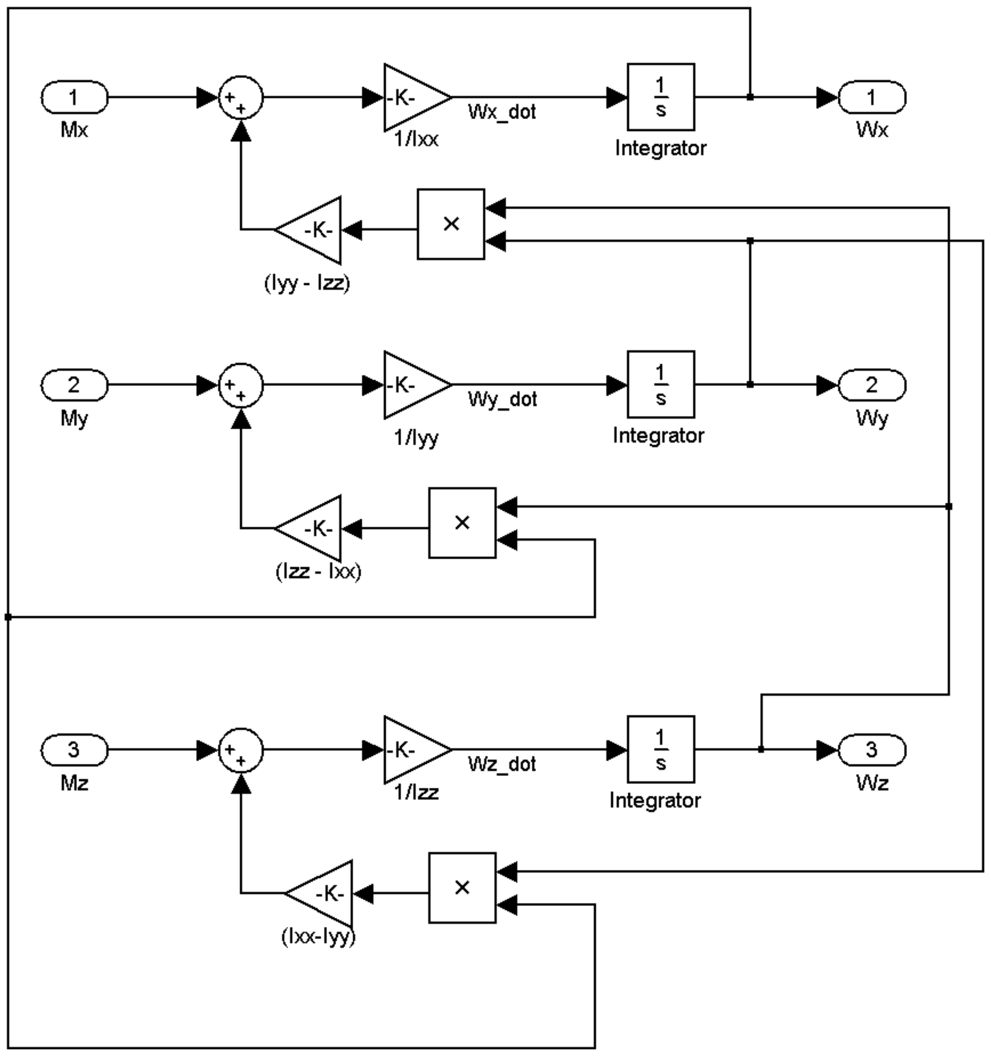


Figure D.8: Model for moment equation

APPENDIX E: SIMULINK RESULTS FOR ALGORITHM VALIDATION

E1: Altitude Stabilize

Table E.1: Altitude stabilize simulation results

Time [s]	V_z [m/s]	A_z [m/s^2]	N_1 [RPM]	N_2 [RPM]
0.00	5.0000	-8.4296	5000.0	5000.0
0.05	4.5785	-8.4296	5000.0	5000.0
0.10	4.1570	-8.4296	5000.0	5000.0
0.15	3.7356	-8.4296	5000.0	5000.0
0.20	3.3141	-8.4296	5000.0	5000.0
0.25	2.8926	-8.4296	5000.0	5000.0
0.30	2.4711	-8.4296	5000.0	5000.0
0.35	2.0496	-8.4296	5000.0	5000.0
0.40	1.6282	-8.4296	5000.0	5000.0
0.45	1.2341	-6.8859	4783.7	4783.7
0.50	0.9376	-5.0656	4515.4	4515.4
0.55	0.7179	-3.7842	4316.6	4316.6
0.60	0.5528	-2.8595	4167.2	4167.2
0.65	0.4276	-2.1796	4053.8	4053.8
0.70	0.3318	-1.6723	3967.1	3967.1
0.75	0.2581	-1.2896	3900.5	3900.5
0.80	0.2012	-0.9984	3849.0	3849.0
0.85	0.1570	-0.7752	3809.0	3809.0
0.90	0.1227	-0.6034	3778.0	3778.0
0.95	0.0960	-0.4705	3753.8	3753.8
1.00	0.0752	-0.3673	3734.9	3734.9
1.05	0.0589	-0.2872	3720.2	3720.2
1.10	0.0461	-0.2247	3708.6	3708.6
1.15	0.0362	-0.1759	3699.6	3699.6
1.20	0.0284	-0.1378	3692.5	3692.5
1.25	0.0222	-0.1080	3687.0	3687.0
1.30	0.0174	-0.0846	3682.7	3682.7
1.35	0.0137	-0.0664	3679.3	3679.3
1.40	0.0107	-0.0520	3676.6	3676.6
1.45	0.0084	-0.0408	3674.5	3674.5
1.50	0.0066	-0.0320	3672.9	3672.9
1.55	0.0052	-0.0251	3671.6	3671.6
1.60	0.0041	-0.0197	3670.6	3670.6
1.65	0.0032	-0.0155	3669.8	3669.8
1.70	0.0025	-0.0121	3669.2	3669.2
1.75	0.0020	-0.0095	3668.7	3668.7
1.80	0.0015	-0.0075	3668.3	3668.3
1.85	0.0012	-0.0059	3668.0	3668.0
1.90	0.0010	-0.0046	3667.7	3667.7
1.95	0.0007	-0.0036	3667.6	3667.6
2.00	0.0006	-0.0028	3667.4	3667.4
2.05	0.0005	-0.0022	3667.3	3667.3
2.10	0.0004	-0.0017	3667.2	3667.2
2.15	0.0003	-0.0014	3667.1	3667.1
2.20	0.0002	-0.0011	3667.1	3667.1
2.25	0.0002	-0.0008	3667.0	3667.0
2.30	0.0001	-0.0007	3667.0	3667.0
2.35	0.0001	-0.0005	3667.0	3667.0
2.40	0.0001	-0.0004	3667.0	3667.0
2.45	0.0001	-0.0003	3666.9	3666.9
2.50	0.0001	-0.0003	3666.9	3666.9
2.55	0.0000	-0.0002	3666.9	3666.9
2.60	0.0000	-0.0002	3666.9	3666.9
2.65	0.0000	-0.0001	3666.9	3666.9
2.70	0.0000	-0.0001	3666.9	3666.9
2.75	0.0000	-0.0001	3666.9	3666.9
2.80	0.0000	-0.0001	3666.9	3666.9
2.85	0.0000	0.0000	3666.9	3666.9
2.90	0.0000	0.0000	3666.9	3666.9
2.95	0.0000	0.0000	3666.9	3666.9
3.00	0.0000	0.0000	3666.9	3666.9
3.05	0.0000	0.0000	3666.9	3666.9
3.10	0.0000	0.0000	3666.9	3666.9
3.15	0.0000	0.0000	3666.9	3666.9
3.20	0.0000	0.0000	3666.9	3666.9
3.25	0.0000	0.0000	3666.9	3666.9
3.30	0.0000	0.0000	3666.9	3666.9
3.35	0.0000	0.0000	3666.9	3666.9
3.40	0.0000	0.0000	3666.9	3666.9
3.45	0.0000	0.0000	3666.9	3666.9
3.50	0.0000	0.0000	3666.9	3666.9
3.55	0.0000	0.0000	3666.9	3666.9
3.60	0.0000	0.0000	3666.9	3666.9
3.65	0.0000	0.0000	3666.9	3666.9
3.70	0.0000	0.0000	3666.9	3666.9
3.75	0.0000	0.0000	3666.9	3666.9
3.80	0.0000	0.0000	3666.9	3666.9
3.85	0.0000	0.0000	3666.9	3666.9
3.90	0.0000	0.0000	3666.9	3666.9
3.95	0.0000	0.0000	3666.9	3666.9
4.00	0.0000	0.0000	3666.9	3666.9
4.05	0.0000	0.0000	3666.9	3666.9
4.10	0.0000	0.0000	3666.9	3666.9
4.15	0.0000	0.0000	3666.9	3666.9
4.20	0.0000	0.0000	3666.9	3666.9
4.25	0.0000	0.0000	3666.9	3666.9
4.30	0.0000	0.0000	3666.9	3666.9
4.35	0.0000	0.0000	3666.9	3666.9
4.40	0.0000	0.0000	3666.9	3666.9
4.45	0.0000	0.0000	3666.9	3666.9
4.50	0.0000	0.0000	3666.9	3666.9
4.55	0.0000	0.0000	3666.9	3666.9
4.60	0.0000	0.0000	3666.9	3666.9
4.65	0.0000	0.0000	3666.9	3666.9
4.70	0.0000	0.0000	3666.9	3666.9
4.75	0.0000	0.0000	3666.9	3666.9
4.80	0.0000	0.0000	3666.9	3666.9
4.85	0.0000	0.0000	3666.9	3666.9
4.90	0.0000	0.0000	3666.9	3666.9
4.95	0.0000	0.0000	3666.9	3666.9
5.00	0.0000	0.0000	3666.9	3666.9

E2: Altitude Hold

Table E.2: Altitude-hold simulation results

Time	Z	V_z	N_1	N_2
[s]	[m]	[m/s]	[RPM]	[RPM]
0.00	0.5000	0.0000	5000.0	5000.0
0.05	0.4895	-0.4215	5000.0	5000.0
0.10	0.4579	-0.8430	5000.0	5000.0
0.15	0.4052	-1.2644	4739.7	4739.7
0.20	0.3374	-1.4059	3669.8	3669.8
0.25	0.2688	-1.3250	3079.0	3079.0
0.30	0.2068	-1.1499	2828.5	2828.5
0.35	0.1545	-0.9452	2783.2	2783.2
0.40	0.1124	-0.7435	2850.1	2850.1
0.45	0.0799	-0.5619	2970.9	2970.9
0.50	0.0558	-0.4089	3109.7	3109.7
0.55	0.0385	-0.2871	3244.3	3244.3
0.60	0.0266	-0.1952	3362.1	3362.1
0.65	0.0186	-0.1294	3457.3	3457.3
0.70	0.0133	-0.0845	3529.3	3529.3
0.75	0.0098	-0.0554	3580.2	3580.2
0.80	0.0075	-0.0372	3614.2	3614.2
0.85	0.0060	-0.0262	3635.4	3635.4
0.90	0.0048	-0.0196	3647.9	3647.9
0.95	0.0039	-0.0155	3654.8	3654.8
1.00	0.0032	-0.0128	3658.3	3658.3
1.05	0.0027	-0.0107	3660.2	3660.2
1.10	0.0022	-0.0091	3661.1	3661.1
1.15	0.0017	-0.0076	3661.8	3661.8
1.20	0.0014	-0.0063	3662.3	3662.3
1.25	0.0011	-0.0052	3662.9	3662.9
1.30	0.0009	-0.0042	3663.5	3663.5
1.35	0.0007	-0.0033	3664.0	3664.0
1.40	0.0005	-0.0026	3664.5	3664.5
1.45	0.0004	-0.0021	3665.0	3665.0
1.50	0.0003	-0.0016	3665.4	3665.4
1.55	0.0003	-0.0013	3665.7	3665.7
1.60	0.0002	-0.0010	3666.0	3666.0
1.65	0.0002	-0.0008	3666.2	3666.2
1.70	0.0001	-0.0006	3666.3	3666.3
1.75	0.0001	-0.0005	3666.5	3666.5
1.80	0.0001	-0.0004	3666.6	3666.6
1.85	0.0001	-0.0003	3666.6	3666.6
1.90	0.0000	-0.0002	3666.7	3666.7
1.95	0.0000	-0.0002	3666.7	3666.7
2.00	0.0000	-0.0001	3666.8	3666.8
2.05	0.0000	-0.0001	3666.8	3666.8
2.10	0.0000	-0.0001	3666.8	3666.8
2.15	0.0000	-0.0001	3666.8	3666.8
2.20	0.0000	-0.0001	3666.8	3666.8
2.25	0.0000	0.0000	3666.8	3666.8
2.30	0.0000	0.0000	3666.9	3666.9
2.35	0.0000	0.0000	3666.9	3666.9
2.40	0.0000	0.0000	3666.9	3666.9
2.45	0.0000	0.0000	3666.9	3666.9
2.50	0.0000	0.0000	3666.9	3666.9
2.55	0.0000	0.0000	3666.9	3666.9
2.60	0.0000	0.0000	3666.9	3666.9
2.65	0.0000	0.0000	3666.9	3666.9
2.70	0.0000	0.0000	3666.9	3666.9
2.75	0.0000	0.0000	3666.9	3666.9
2.80	0.0000	0.0000	3666.9	3666.9
2.85	0.0000	0.0000	3666.9	3666.9
2.90	0.0000	0.0000	3666.9	3666.9
2.95	0.0000	0.0000	3666.9	3666.9
3.00	0.0000	0.0000	3666.9	3666.9
3.05	0.0000	0.0000	3666.9	3666.9
3.10	0.0000	0.0000	3666.9	3666.9
3.15	0.0000	0.0000	3666.9	3666.9
3.20	0.0000	0.0000	3666.9	3666.9
3.25	0.0000	0.0000	3666.9	3666.9
3.30	0.0000	0.0000	3666.9	3666.9
3.35	0.0000	0.0000	3666.9	3666.9
3.40	0.0000	0.0000	3666.9	3666.9
3.45	0.0000	0.0000	3666.9	3666.9
3.50	0.0000	0.0000	3666.9	3666.9
3.55	0.0000	0.0000	3666.9	3666.9
3.60	0.0000	0.0000	3666.9	3666.9
3.65	0.0000	0.0000	3666.9	3666.9
3.70	0.0000	0.0000	3666.9	3666.9
3.75	0.0000	0.0000	3666.9	3666.9
3.80	0.0000	0.0000	3666.9	3666.9
3.85	0.0000	0.0000	3666.9	3666.9
3.90	0.0000	0.0000	3666.9	3666.9
3.95	0.0000	0.0000	3666.9	3666.9
4.00	0.0000	0.0000	3666.9	3666.9
4.05	0.0000	0.0000	3666.9	3666.9
4.10	0.0000	0.0000	3666.9	3666.9
4.15	0.0000	0.0000	3666.9	3666.9
4.20	0.0000	0.0000	3666.9	3666.9
4.25	0.0000	0.0000	3666.9	3666.9
4.30	0.0000	0.0000	3666.9	3666.9
4.35	0.0000	0.0000	3666.9	3666.9
4.40	0.0000	0.0000	3666.9	3666.9
4.45	0.0000	0.0000	3666.9	3666.9
4.50	0.0000	0.0000	3666.9	3666.9
4.55	0.0000	0.0000	3666.9	3666.9
4.60	0.0000	0.0000	3666.9	3666.9
4.65	0.0000	0.0000	3666.9	3666.9
4.70	0.0000	0.0000	3666.9	3666.9
4.75	0.0000	0.0000	3666.9	3666.9
4.80	0.0000	0.0000	3666.9	3666.9
4.85	0.0000	0.0000	3666.9	3666.9
4.90	0.0000	0.0000	3666.9	3666.9
4.95	0.0000	0.0000	3666.9	3666.9
5.00	0.0000	0.0000	3666.9	3666.9

E3: Roll Stabilize

Table E.3: Roll stabilize simulation results

Time [s]	Roll [deg]	Roll rate [deg/s]	V_y [m/s]	V_z [m/s]	Y [m]	Z [m]	N_1 [RPM]	N_2 [RPM]
0.00	5.00	0.00	0.000	0.000	0.000	0.000	5000.0	0.0
0.05	4.98	-0.73	0.042	0.033	0.001	0.001	5000.0	0.0
0.10	4.93	-1.47	0.082	0.069	0.004	0.004	5000.0	0.0
0.15	4.83	-2.20	0.115	0.111	0.008	0.009	5000.0	0.0
0.20	4.71	-2.94	0.138	0.161	0.014	0.016	5000.0	0.0
0.25	4.54	-3.67	0.147	0.217	0.021	0.026	5000.0	0.0
0.30	4.34	-4.41	0.134	0.279	0.027	0.039	5000.0	0.0
0.35	4.10	-5.14	0.096	0.340	0.032	0.055	5000.0	0.0
0.40	3.82	-5.88	0.028	0.391	0.033	0.073	5000.0	0.0
0.45	3.51	-6.61	-0.068	0.419	0.031	0.094	5000.0	0.0
0.50	3.16	-7.35	-0.185	0.408	0.024	0.114	5000.0	0.0
0.55	2.78	-8.05	-0.305	0.328	0.010	0.132	5000.0	1797.2
0.60	2.36	-8.56	-0.374	0.075	-0.008	0.142	5000.0	3563.0
0.65	1.93	-8.62	-0.299	-0.303	-0.025	0.135	3612.5	4899.1
0.70	1.51	-8.10	-0.111	-0.483	-0.035	0.114	1799.4	5000.0
0.75	1.12	-7.41	0.091	-0.477	-0.034	0.090	557.3	5000.0
0.80	0.77	-6.68	0.252	-0.382	-0.025	0.068	0.0	5000.0
0.85	0.45	-5.94	0.358	-0.250	-0.009	0.052	0.0	5000.0
0.90	0.17	-5.21	0.411	-0.107	0.010	0.044	0.0	5000.0
0.95	-0.07	-4.47	0.421	0.028	0.031	0.042	0.0	5000.0
1.00	-0.27	-3.74	0.401	0.148	0.052	0.046	0.0	5000.0
1.05	-0.44	-3.00	0.365	0.247	0.071	0.056	0.0	5000.0
1.10	-0.57	-2.27	0.323	0.327	0.088	0.070	0.0	5000.0
1.15	-0.67	-1.53	0.283	0.389	0.104	0.088	464.8	5000.0
1.20	-0.73	-0.82	0.253	0.428	0.117	0.108	1059.0	5000.0
1.25	-0.75	-0.14	0.237	0.433	0.130	0.130	1718.8	5000.0
1.30	-0.74	0.47	0.234	0.387	0.142	0.151	2389.6	5000.0
1.35	-0.71	0.99	0.240	0.278	0.154	0.167	3004.9	5000.0
1.40	-0.64	1.41	0.246	0.103	0.166	0.177	3504.3	5000.0
1.45	-0.57	1.74	0.239	-0.131	0.178	0.176	3852.6	5000.0
1.50	-0.47	1.94	0.211	-0.361	0.189	0.163	4050.8	4352.9
1.55	-0.38	1.90	0.165	-0.476	0.199	0.142	4134.9	3545.8
1.60	-0.29	1.71	0.118	-0.505	0.205	0.117	4142.6	3003.5
1.65	-0.21	1.45	0.076	-0.480	0.210	0.092	4104.8	2712.1
1.70	-0.14	1.17	0.045	-0.424	0.213	0.070	4044.1	2610.3
1.75	-0.09	0.90	0.024	-0.354	0.215	0.050	3975.4	2637.8
1.80	-0.05	0.65	0.012	-0.279	0.216	0.034	3907.8	2746.2
1.85	-0.02	0.44	0.005	-0.208	0.216	0.022	3846.1	2898.3
1.90	-0.01	0.28	0.001	-0.146	0.216	0.014	3792.7	3065.6
1.95	0.00	0.15	0.000	-0.095	0.216	0.008	3748.6	3227.0
2.00	0.01	0.06	0.000	-0.057	0.216	0.004	3713.7	3368.7
2.05	0.01	0.00	0.000	-0.031	0.216	0.002	3687.5	3482.9
2.10	0.01	-0.03	0.000	-0.014	0.216	0.000	3669.0	3567.9
2.15	0.01	-0.04	0.000	-0.005	0.216	0.000	3657.2	3625.7
2.20	0.01	-0.04	0.000	0.000	0.216	0.000	3650.7	3661.2
2.25	0.00	-0.04	0.000	0.002	0.216	0.000	3648.3	3680.0
2.30	0.00	-0.03	0.000	0.002	0.216	0.000	3648.6	3687.4
2.35	0.00	-0.02	0.000	0.001	0.216	0.000	3650.8	3687.9
2.40	0.00	-0.01	0.000	0.001	0.216	0.000	3653.7	3684.9
2.45	0.00	-0.01	0.000	0.000	0.216	0.000	3656.9	3680.5
2.50	0.00	0.00	0.000	0.000	0.216	0.000	3659.8	3676.2
2.55	0.00	0.00	0.000	0.000	0.216	0.000	3662.2	3672.5
2.60	0.00	0.00	0.000	-0.001	0.216	0.000	3664.1	3669.8
2.65	0.00	0.00	0.000	-0.001	0.216	0.000	3665.5	3667.9
2.70	0.00	0.00	0.000	0.000	0.216	0.000	3666.4	3666.8

Table E.3: Continued

Time [s]	Roll [deg]	Roll rate [deg/s]	V_y [m/s]	V_z [m/s]	Y [m]	Z [m]	N_1 [RPM]	N_2 [RPM]
2.75	0.00	0.00	0.000	0.000	0.216	0.000	3666.9	3666.2
2.80	0.00	0.00	0.000	0.000	0.216	0.000	3667.2	3666.0
2.85	0.00	0.00	0.000	0.000	0.216	0.000	3667.2	3666.0
2.90	0.00	0.00	0.000	0.000	0.216	0.000	3667.2	3666.1
2.95	0.00	0.00	0.000	0.000	0.216	0.000	3667.2	3666.3
3.00	0.00	0.00	0.000	0.000	0.216	0.000	3667.1	3666.5
3.05	0.00	0.00	0.000	0.000	0.216	0.000	3667.0	3666.6
3.10	0.00	0.00	0.000	0.000	0.216	0.000	3666.9	3666.7
3.15	0.00	0.00	0.000	0.000	0.216	0.000	3666.9	3666.8
3.20	0.00	0.00	0.000	0.000	0.216	0.000	3666.9	3666.9
3.25	0.00	0.00	0.000	0.000	0.216	0.000	3666.8	3666.9
3.30	0.00	0.00	0.000	0.000	0.216	0.000	3666.8	3666.9
3.35	0.00	0.00	0.000	0.000	0.216	0.000	3666.8	3666.9
3.40	0.00	0.00	0.000	0.000	0.216	0.000	3666.8	3666.9
3.45	0.00	0.00	0.000	0.000	0.216	0.000	3666.9	3666.9
3.50	0.00	0.00	0.000	0.000	0.216	0.000	3666.9	3666.9
3.55	0.00	0.00	0.000	0.000	0.216	0.000	3666.9	3666.9
3.60	0.00	0.00	0.000	0.000	0.216	0.000	3666.9	3666.9
3.65	0.00	0.00	0.000	0.000	0.216	0.000	3666.9	3666.9
3.70	0.00	0.00	0.000	0.000	0.216	0.000	3666.9	3666.9
3.75	0.00	0.00	0.000	0.000	0.216	0.000	3666.9	3666.9
3.80	0.00	0.00	0.000	0.000	0.216	0.000	3666.9	3666.9
3.85	0.00	0.00	0.000	0.000	0.216	0.000	3666.9	3666.9
3.90	0.00	0.00	0.000	0.000	0.216	0.000	3666.9	3666.9
3.95	0.00	0.00	0.000	0.000	0.216	0.000	3666.9	3666.9
4.00	0.00	0.00	0.000	0.000	0.216	0.000	3666.9	3666.9
4.05	0.00	0.00	0.000	0.000	0.216	0.000	3666.9	3666.9
4.10	0.00	0.00	0.000	0.000	0.216	0.000	3666.9	3666.9
4.15	0.00	0.00	0.000	0.000	0.216	0.000	3666.9	3666.9
4.20	0.00	0.00	0.000	0.000	0.216	0.000	3666.9	3666.9
4.25	0.00	0.00	0.000	0.000	0.216	0.000	3666.9	3666.9
4.30	0.00	0.00	0.000	0.000	0.216	0.000	3666.9	3666.9
4.35	0.00	0.00	0.000	0.000	0.216	0.000	3666.9	3666.9
4.40	0.00	0.00	0.000	0.000	0.216	0.000	3666.9	3666.9
4.45	0.00	0.00	0.000	0.000	0.216	0.000	3666.9	3666.9
4.50	0.00	0.00	0.000	0.000	0.216	0.000	3666.9	3666.9
4.55	0.00	0.00	0.000	0.000	0.216	0.000	3666.9	3666.9
4.60	0.00	0.00	0.000	0.000	0.216	0.000	3666.9	3666.9
4.65	0.00	0.00	0.000	0.000	0.216	0.000	3666.9	3666.9
4.70	0.00	0.00	0.000	0.000	0.216	0.000	3666.9	3666.9
4.75	0.00	0.00	0.000	0.000	0.216	0.000	3666.9	3666.9
4.80	0.00	0.00	0.000	0.000	0.216	0.000	3666.9	3666.9
4.85	0.00	0.00	0.000	0.000	0.216	0.000	3666.9	3666.9
4.90	0.00	0.00	0.000	0.000	0.216	0.000	3666.9	3666.9
4.95	0.00	0.00	0.000	0.000	0.216	0.000	3666.9	3666.9
5.00	0.00	0.00	0.000	0.000	0.216	0.000	3666.9	3666.9

E4: Pitch Stabilize

Table E.4: Pitch stabilize simulation results

Time [s]	Pitch [deg]	Pitch rate [deg/s]	V_x [m/s]	V_z [m/s]	X [m]	Z [m]	N_1 [RPM]	N_2 [RPM]	alpha [deg]	beta [deg]
0.00	-10.00	0.00	5.000	0.000	0.000	0.000	3666.9	3666.9	30.00	30.00
0.05	-9.98	0.62	5.332	0.134	0.258	0.003	3750.2	3750.2	30.00	30.00
0.10	-9.94	1.30	5.677	0.406	0.531	0.016	4041.0	4041.0	30.00	30.00
0.15	-9.85	2.14	6.045	0.806	0.818	0.046	4593.7	4593.7	30.00	30.00
0.20	-9.72	3.23	6.421	1.369	1.121	0.100	5000.0	5000.0	30.00	30.00
0.25	-9.53	4.37	6.611	2.309	1.433	0.192	5000.0	5000.0	30.00	30.00
0.30	-9.28	5.51	6.413	3.623	1.736	0.341	5000.0	5000.0	30.00	30.00
0.35	-8.98	6.65	5.603	5.161	2.005	0.563	5000.0	5000.0	30.00	30.00
0.40	-8.62	7.79	3.987	6.607	2.202	0.863	5000.0	5000.0	30.00	30.00
0.45	-8.20	8.93	1.530	7.468	2.290	1.223	5000.0	5000.0	30.00	30.00
0.50	-7.72	10.07	-1.485	7.160	2.239	1.599	5000.0	5000.0	25.17	25.17
0.55	-7.20	10.51	-4.516	5.148	2.043	1.917	5000.0	5000.0	-0.05	-0.05
0.60	-6.68	10.15	-6.435	1.811	1.739	2.095	5000.0	5000.0	-16.01	-16.01
0.65	-6.20	9.38	-6.671	-1.835	1.404	2.091	5000.0	5000.0	-22.31	-22.31
0.70	-5.75	8.51	-5.425	-4.939	1.115	1.916	5000.0	5000.0	-22.13	-22.13
0.75	-5.35	7.71	-3.233	-7.081	0.925	1.609	4249.3	4249.3	-18.99	-18.99
0.80	-4.97	7.55	-0.433	-7.471	0.867	1.237	19.3	19.3	-18.62	-18.62
0.85	-4.59	7.55	2.300	-6.613	0.946	0.879	0.0	0.0	-22.57	-22.57
0.90	-4.21	7.55	4.519	-4.808	1.142	0.589	0.0	0.0	-29.16	-29.16
0.95	-3.83	7.55	5.911	-2.314	1.420	0.409	0.0	0.0	-30.00	-30.00
1.00	-3.46	7.49	6.254	0.465	1.733	0.364	2536.1	2536.1	-30.00	-30.00
1.05	-3.10	6.62	5.367	2.456	2.022	0.445	5000.0	5000.0	-30.00	-30.00
1.10	-2.80	5.48	4.011	3.582	2.249	0.599	5000.0	5000.0	-30.00	-30.00
1.15	-2.55	4.34	2.627	4.099	2.405	0.793	5000.0	5000.0	-30.00	-30.00
1.20	-2.36	3.41	1.481	4.150	2.498	1.001	5000.0	5000.0	-18.64	-18.64
1.25	-2.20	2.88	0.660	3.919	2.542	1.204	5000.0	5000.0	-9.16	-9.16
1.30	-2.07	2.64	0.066	3.552	2.553	1.391	5000.0	5000.0	-3.80	-3.80
1.35	-1.94	2.53	-0.390	3.110	2.538	1.558	5000.0	5000.0	-1.88	-1.88
1.40	-1.81	2.46	-0.762	2.616	2.505	1.701	5000.0	5000.0	-1.94	-1.94
1.45	-1.69	2.36	-1.069	2.084	2.455	1.819	5000.0	5000.0	-2.70	-2.70
1.50	-1.58	2.24	-1.313	1.526	2.393	1.909	5000.0	5000.0	-3.39	-3.39
1.55	-1.47	2.10	-1.492	0.954	2.321	1.971	5000.0	5000.0	-3.71	-3.71
1.60	-1.37	1.95	-1.607	0.377	2.242	2.005	5000.0	5000.0	-3.69	-3.69
1.65	-1.27	1.81	-1.661	-0.197	2.160	2.009	5000.0	5000.0	-3.46	-3.46
1.70	-1.19	1.68	-1.662	-0.762	2.077	1.985	5000.0	5000.0	-3.15	-3.15
1.75	-1.10	1.56	-1.616	-1.315	1.996	1.933	5000.0	5000.0	-2.84	-2.84
1.80	-1.03	1.45	-1.531	-1.854	1.919	1.854	5000.0	5000.0	-2.58	-2.58
1.85	-0.96	1.35	-1.414	-2.378	1.847	1.748	5000.0	5000.0	-2.38	-2.38
1.90	-0.89	1.26	-1.271	-2.887	1.782	1.616	5000.0	5000.0	-2.21	-2.21
1.95	-0.83	1.18	-1.107	-3.380	1.725	1.459	5000.0	5000.0	-2.06	-2.06
2.00	-0.78	1.10	-0.926	-3.859	1.676	1.278	4717.6	4717.6	-1.93	-1.93
2.05	-0.72	1.05	-0.727	-3.983	1.638	1.080	3274.8	3274.8	-1.97	-1.97
2.10	-0.67	1.02	-0.529	-3.796	1.609	0.885	2233.5	2233.5	-2.32	-2.32
2.15	-0.62	1.01	-0.345	-3.460	1.589	0.704	1600.4	1600.4	-2.93	-2.93
2.20	-0.57	1.00	-0.180	-3.056	1.578	0.541	1272.5	1272.5	-3.70	-3.70
2.25	-0.52	0.99	-0.038	-2.624	1.574	0.399	1160.0	1160.0	-4.55	-4.55
2.30	-0.47	0.98	0.080	-2.183	1.576	0.278	1206.3	1206.3	-5.42	-5.42
2.35	-0.42	0.96	0.173	-1.746	1.583	0.180	1375.6	1375.6	-6.25	-6.25
2.40	-0.38	0.94	0.239	-1.327	1.594	0.104	1640.0	1640.0	-6.99	-6.99
2.45	-0.33	0.90	0.279	-0.943	1.607	0.047	1970.8	1970.8	-7.56	-7.56
2.50	-0.29	0.84	0.293	-0.608	1.622	0.009	2335.6	2335.6	-7.86	-7.86
2.55	-0.25	0.76	0.282	-0.336	1.636	-0.015	2699.4	2699.4	-7.80	-7.80
2.60	-0.21	0.66	0.253	-0.134	1.650	-0.026	3029.4	3029.4	-7.33	-7.33
2.65	-0.18	0.55	0.212	-0.001	1.662	-0.029	3301.1	3301.1	-6.47	-6.47

Table E.4: Continued

Time [s]	Pitch [deg]	Pitch rate [deg/s]	V_x [m/s]	V_z [m/s]	X [m]	Z [m]	N_1 [RPM]	N_2 [RPM]	alpha [deg]	beta [deg]
2.70	-0.15	0.44	0.169	0.073	1.671	-0.027	3502.5	3502.5	-5.37	-5.37
2.75	-0.13	0.35	0.130	0.102	1.678	-0.023	3634.5	3634.5	-4.17	-4.17
2.80	-0.12	0.27	0.098	0.103	1.684	-0.018	3708.3	3708.3	-3.06	-3.06
2.85	-0.11	0.21	0.075	0.088	1.688	-0.013	3739.5	3739.5	-2.14	-2.14
2.90	-0.10	0.17	0.060	0.069	1.692	-0.009	3743.9	3743.9	-1.45	-1.45
2.95	-0.09	0.15	0.049	0.050	1.694	-0.006	3734.3	3734.3	-0.97	-0.97
3.00	-0.08	0.13	0.043	0.034	1.697	-0.004	3719.5	3719.5	-0.67	-0.67
3.05	-0.08	0.12	0.038	0.022	1.699	-0.003	3704.6	3704.6	-0.49	-0.49
3.10	-0.07	0.11	0.035	0.014	1.701	-0.002	3692.2	3692.2	-0.38	-0.38
3.15	-0.07	0.10	0.032	0.009	1.702	-0.001	3682.9	3682.9	-0.33	-0.33
3.20	-0.06	0.09	0.030	0.005	1.704	-0.001	3676.5	3676.5	-0.30	-0.30
3.25	-0.06	0.09	0.028	0.003	1.705	-0.001	3672.5	3672.5	-0.29	-0.29
3.30	-0.05	0.08	0.026	0.002	1.707	-0.001	3670.1	3670.1	-0.27	-0.27
3.35	-0.05	0.08	0.024	0.002	1.708	0.000	3668.8	3668.8	-0.26	-0.26
3.40	-0.04	0.07	0.022	0.001	1.709	0.000	3668.2	3668.2	-0.25	-0.25
3.45	-0.04	0.06	0.021	0.001	1.710	0.000	3667.9	3667.9	-0.24	-0.24
3.50	-0.04	0.06	0.019	0.001	1.711	0.000	3667.8	3667.8	-0.22	-0.22
3.55	-0.03	0.06	0.018	0.001	1.712	0.000	3667.7	3667.7	-0.21	-0.21
3.60	-0.03	0.05	0.016	0.001	1.713	0.000	3667.6	3667.6	-0.19	-0.19
3.65	-0.03	0.05	0.015	0.001	1.714	0.000	3667.5	3667.5	-0.18	-0.18
3.70	-0.03	0.04	0.014	0.000	1.714	0.000	3667.4	3667.4	-0.16	-0.16
3.75	-0.03	0.04	0.013	0.000	1.715	0.000	3667.3	3667.3	-0.15	-0.15
3.80	-0.02	0.04	0.011	0.000	1.716	0.000	3667.3	3667.3	-0.14	-0.14
3.85	-0.02	0.03	0.011	0.000	1.716	0.000	3667.2	3667.2	-0.13	-0.13
3.90	-0.02	0.03	0.010	0.000	1.717	0.000	3667.1	3667.1	-0.12	-0.12
3.95	-0.02	0.03	0.009	0.000	1.717	0.000	3667.1	3667.1	-0.11	-0.11
4.00	-0.02	0.03	0.008	0.000	1.718	0.000	3667.0	3667.0	-0.10	-0.10
4.05	-0.02	0.02	0.007	0.000	1.718	0.000	3667.0	3667.0	-0.09	-0.09
4.10	-0.01	0.02	0.007	0.000	1.718	0.000	3667.0	3667.0	-0.09	-0.09
4.15	-0.01	0.02	0.006	0.000	1.719	0.000	3666.9	3666.9	-0.08	-0.08
4.20	-0.01	0.02	0.006	0.000	1.719	0.000	3666.9	3666.9	-0.07	-0.07
4.25	-0.01	0.02	0.005	0.000	1.719	0.000	3666.9	3666.9	-0.07	-0.07
4.30	-0.01	0.02	0.005	0.000	1.719	0.000	3666.9	3666.9	-0.06	-0.06
4.35	-0.01	0.02	0.004	0.000	1.720	0.000	3666.9	3666.9	-0.06	-0.06
4.40	-0.01	0.01	0.004	0.000	1.720	0.000	3666.9	3666.9	-0.05	-0.05
4.45	-0.01	0.01	0.003	0.000	1.720	0.000	3666.9	3666.9	-0.05	-0.05
4.50	-0.01	0.01	0.003	0.000	1.720	0.000	3666.9	3666.9	-0.05	-0.05
4.55	-0.01	0.01	0.003	0.000	1.720	0.000	3666.9	3666.9	-0.04	-0.04
4.60	-0.01	0.01	0.002	0.000	1.720	0.000	3666.9	3666.9	-0.04	-0.04
4.65	-0.01	0.01	0.002	0.000	1.721	0.000	3666.9	3666.9	-0.04	-0.04
4.70	-0.01	0.01	0.002	0.000	1.721	0.000	3666.9	3666.9	-0.03	-0.03
4.75	-0.01	0.01	0.002	0.000	1.721	0.000	3666.9	3666.9	-0.03	-0.03
4.80	0.00	0.01	0.002	0.000	1.721	0.000	3666.9	3666.9	-0.03	-0.03
4.85	0.00	0.01	0.001	0.000	1.721	0.000	3666.9	3666.9	-0.03	-0.03
4.90	0.00	0.01	0.001	0.000	1.721	0.000	3666.9	3666.9	-0.02	-0.02
4.95	0.00	0.01	0.001	0.000	1.721	0.000	3666.9	3666.9	-0.02	-0.02
5.00	0.00	0.01	0.001	0.000	1.721	0.000	3666.9	3666.9	-0.02	-0.02

E5: Yaw Rate Stabilize

Table E.5: Yaw rate stabilize simulation results

Time [s]	Yaw rate [deg/s]	V_z [m/s]	Z [m]	N_1 [RPM]	N_2 [RPM]	alpha [deg]	beta [deg]
0.00	45.00	0.000	0.000	4791.9	4791.9	-45.00	45.00
0.05	44.08	-0.092	-0.002	4708.3	4708.3	-45.00	45.00
0.10	43.21	-0.155	-0.009	4566.5	4566.5	-45.00	45.00
0.15	42.39	-0.186	-0.017	4431.4	4431.4	-45.00	45.00
0.20	41.62	-0.189	-0.027	4327.9	4327.9	-45.00	45.00
0.25	40.87	-0.174	-0.036	4260.1	4260.1	-45.00	45.00
0.30	40.14	-0.147	-0.044	4223.3	4223.3	-45.00	45.00
0.35	39.42	-0.115	-0.051	4210.0	4210.0	-45.00	45.00
0.40	38.70	-0.082	-0.056	4212.6	4212.6	-45.00	45.00
0.45	37.98	-0.050	-0.059	4225.0	4225.0	-45.00	45.00
0.50	37.25	-0.022	-0.061	4242.5	4242.5	-45.00	45.00
0.55	36.52	0.002	-0.061	4261.7	4261.7	-45.00	45.00
0.60	35.78	0.022	-0.060	4280.5	4280.5	-45.00	45.00
0.65	35.03	0.038	-0.059	4297.5	4297.5	-45.00	45.00
0.70	34.28	0.050	-0.057	4312.2	4312.2	-45.00	45.00
0.75	33.52	0.060	-0.054	4324.4	4324.4	-45.00	45.00
0.80	32.76	0.067	-0.051	4334.1	4334.1	-45.00	45.00
0.85	32.00	0.072	-0.047	4341.7	4341.7	-45.00	45.00
0.90	31.23	0.076	-0.044	4347.4	4347.4	-45.00	45.00
0.95	30.47	0.078	-0.040	4351.6	4351.6	-45.00	45.00
1.00	29.70	0.080	-0.036	4354.6	4354.6	-45.00	45.00
1.05	28.93	0.081	-0.032	4356.8	4356.8	-45.00	45.00
1.10	28.16	0.082	-0.028	4358.2	4358.2	-45.00	45.00
1.15	27.39	0.082	-0.024	4359.2	4359.2	-45.00	45.00
1.20	26.62	0.082	-0.020	4359.9	4359.9	-45.00	45.00
1.25	25.85	0.082	-0.015	4360.3	4360.3	-45.00	45.00
1.30	25.07	0.082	-0.011	4360.5	4360.5	-45.00	45.00
1.35	24.30	0.082	-0.007	4360.6	4360.6	-45.00	45.00
1.40	23.53	0.082	-0.003	4360.7	4360.7	-45.00	45.00
1.45	22.76	0.082	0.001	4360.7	4360.7	-45.00	45.00
1.50	21.99	0.082	0.005	4360.7	4360.7	-45.00	45.00
1.55	21.22	0.082	0.009	4360.7	4360.7	-45.00	45.00
1.60	20.45	0.082	0.013	4360.7	4360.7	-45.00	45.00
1.65	19.68	0.082	0.018	4360.7	4360.7	-45.00	45.00
1.70	18.90	0.082	0.022	4360.7	4360.7	-45.00	45.00
1.75	18.13	0.082	0.026	4360.7	4360.7	-45.00	45.00
1.80	17.36	0.082	0.030	4360.7	4360.7	-45.00	45.00
1.85	16.59	0.082	0.034	4360.7	4360.7	-45.00	45.00
1.90	15.82	0.082	0.038	4360.7	4360.7	-45.00	45.00
1.95	15.05	0.082	0.042	4360.7	4360.7	-45.00	45.00
2.00	14.28	0.082	0.046	4360.7	4360.7	-45.00	45.00
2.05	13.50	0.082	0.051	4360.7	4360.7	-45.00	45.00
2.10	12.73	0.082	0.055	4360.7	4360.7	-45.00	45.00
2.15	11.96	0.082	0.059	4360.7	4360.7	-45.00	45.00
2.20	11.19	0.082	0.063	4360.7	4360.7	-45.00	45.00
2.25	10.42	0.082	0.067	4360.7	4360.7	-45.00	45.00
2.30	9.65	0.082	0.071	4360.7	4360.7	-45.00	45.00
2.35	8.88	0.082	0.075	4360.7	4360.7	-45.00	45.00
2.40	8.10	0.082	0.079	4360.7	4360.7	-45.00	45.00
2.45	7.33	0.082	0.084	4360.7	4360.7	-45.00	45.00
2.50	6.56	0.082	0.088	4360.7	4360.7	-45.00	45.00
2.55	5.79	0.082	0.092	4360.7	4360.7	-45.00	45.00
2.60	5.02	0.082	0.096	4360.7	4360.7	-45.00	45.00
2.65	4.25	0.082	0.100	4360.7	4360.7	-45.00	45.00
2.70	3.48	0.082	0.104	4360.7	4360.7	-45.00	45.00

Table E.5: Continued

Time [s]	Yaw rate [deg/s]	V_z [m/s]	Z [m]	N_1 [RPM]	N_2 [RPM]	alpha [deg]	beta [deg]
2.75	2.71	0.082	0.108	4360.7	4360.7	-45.00	45.00
2.80	1.93	0.082	0.112	4360.7	4360.7	-45.00	45.00
2.85	1.25	0.032	0.116	4333.2	4333.2	-31.32	31.32
2.90	0.80	-0.082	0.114	4221.0	4221.0	-20.04	20.04
2.95	0.53	-0.189	0.107	4044.1	4044.1	-13.13	13.13
3.00	0.36	-0.256	0.096	3859.3	3859.3	-8.89	8.89
3.05	0.25	-0.282	0.082	3707.9	3707.9	-6.22	6.22
3.10	0.18	-0.276	0.068	3604.4	3604.4	-4.45	4.45
3.15	0.13	-0.250	0.055	3546.3	3546.3	-3.23	3.23
3.20	0.09	-0.214	0.044	3523.2	3523.2	-2.36	2.36
3.25	0.07	-0.176	0.034	3523.8	3523.8	-1.73	1.73
3.30	0.05	-0.140	0.026	3537.9	3537.9	-1.27	1.27
3.35	0.04	-0.109	0.020	3558.1	3558.1	-0.92	0.92
3.40	0.03	-0.083	0.015	3579.5	3579.5	-0.67	0.67
3.45	0.02	-0.062	0.011	3599.3	3599.3	-0.49	0.49
3.50	0.01	-0.047	0.009	3616.0	3616.0	-0.35	0.35
3.55	0.01	-0.035	0.007	3629.2	3629.2	-0.25	0.25
3.60	0.01	-0.026	0.005	3639.3	3639.3	-0.18	0.18
3.65	0.01	-0.020	0.004	3646.8	3646.8	-0.13	0.13
3.70	0.00	-0.015	0.003	3652.1	3652.1	-0.09	0.09
3.75	0.00	-0.012	0.003	3655.9	3655.9	-0.07	0.07
3.80	0.00	-0.009	0.002	3658.6	3658.6	-0.05	0.05
3.85	0.00	-0.007	0.002	3660.5	3660.5	-0.03	0.03
3.90	0.00	-0.006	0.001	3661.9	3661.9	-0.02	0.02
3.95	0.00	-0.005	0.001	3662.9	3662.9	-0.02	0.02
4.00	0.00	-0.004	0.001	3663.7	3663.7	-0.01	0.01
4.05	0.00	-0.003	0.001	3664.3	3664.3	-0.01	0.01
4.10	0.00	-0.002	0.000	3664.8	3664.8	-0.01	0.01
4.15	0.00	-0.002	0.000	3665.3	3665.3	0.00	0.00
4.20	0.00	-0.001	0.000	3665.6	3665.6	0.00	0.00
4.25	0.00	-0.001	0.000	3665.8	3665.8	0.00	0.00
4.30	0.00	-0.001	0.000	3666.1	3666.1	0.00	0.00
4.35	0.00	-0.001	0.000	3666.2	3666.2	0.00	0.00
4.40	0.00	-0.001	0.000	3666.4	3666.4	0.00	0.00
4.45	0.00	0.000	0.000	3666.5	3666.5	0.00	0.00
4.50	0.00	0.000	0.000	3666.6	3666.6	0.00	0.00
4.55	0.00	0.000	0.000	3666.6	3666.6	0.00	0.00
4.60	0.00	0.000	0.000	3666.7	3666.7	0.00	0.00
4.65	0.00	0.000	0.000	3666.7	3666.7	0.00	0.00
4.70	0.00	0.000	0.000	3666.8	3666.8	0.00	0.00
4.75	0.00	0.000	0.000	3666.8	3666.8	0.00	0.00
4.80	0.00	0.000	0.000	3666.8	3666.8	0.00	0.00
4.85	0.00	0.000	0.000	3666.8	3666.8	0.00	0.00
4.90	0.00	0.000	0.000	3666.8	3666.8	0.00	0.00
4.95	0.00	0.000	0.000	3666.8	3666.8	0.00	0.00
5.00	0.00	0.000	0.000	3666.9	3666.9	0.00	0.00

APPENDIX F: PROJECT COST REPORT

Table F.1: Expense report

	Item	Supplier	Qty	Descprition	Amount (inc Vat) [R]	*Date of Purchase*
Test bench Development and Characterisation Experiment						
1	**Force Guage**	Polytest Instruments	1	Mecmesin AFG250 Force Guage (Tension and compression)	**12369.00**	*2011/01/27*
2	**Torque Guage**	Polytest Instruments	1	Mecmesin Static Torque Transducer 150Nm round	**13530.66**	*2011/01/27*
3	**Charger**	Polytest Instruments	1	Battery charger for the BFG200	**3990.00**	*2011/01/27*
4	**Batteries**	Polytest Instruments	1	Set of AAA bateries for the BFG200 force guage	**85.50**	*2011/01/27*
5	**61900 2RS FBJ Ball Bearing**	R & V Bearing Supplies cc	1	61900 2RS FBJ Ball Bearing	**25.00**	*2011/03/01*
6	**RNA 4917 Roller Bearing**	R & V Bearing Supplies cc	1	RNA 4917 Roller Bearing	**200.00**	*2011/03/01*
7	**51107 Thrust Bearing**	R & V Bearing Supplies cc	1	51107 Thrust Bearing	**25.00**	*2011/03/01*
8	**M6 X 16**	Nesco Engineering	4	M6 X 16 high tension bolts	**2.22**	*2011/03/01*
9	**M6 X 30 Capscrews**	Nesco Engineering	20	M6 X 30 Cap screws	**19.67**	*2011/03/01*
10	**Aluminium Sheet 50mm-M82**	Non Ferrous Metals	1	Aluminium Sheet 50mm-M82	**205.20**	*2011/03/01*
11	**Aluminium Sheet 100mm - 6082T6**	Non Ferrous Metals	1	Aluminium Sheet 100mm - 6082T6	**598.50**	*2011/03/01*
12	**Square tube 40 x 40 x 3**	D & E Steel	2	40 x 40 x 3 Sq. tube 6m length mild steel	**640.50**	*2011/03/24*
13	**Angle Iron 60 x 60 x 8**	D & E Steel	1	60 x 60 x 8 Angle 6m length mild steel	**90.06**	*2011/03/24*
14	**155mm dia EN8 Black 300mm long**	D & E Steel	1	155mm dia. EN8 Black 300mm long	**513.00**	*2011/03/24*
15	**Mild Steel Plate 6mm thick**	Brackenfell Staal	0.41	Mild Steel Plate 6mm thick	**315.89**	*2011/03/02*
16	**Cutting**	Brackenfell Staal	1	Cutting of Mild Steel Plate 6mm thick	**14.82**	*2011/03/02*
17	**Aluminium Sheet 50mm-M82**	Non Ferrous Metals	1	Aluminium Sheet 50mm-M82 150 x 110 x 50mm	**171.00**	*2011/03/30*
18	**internal circlip dia 110mm**	Bearing Man Group Bellville	1	internal circlip dia. 110mm 4mm thick	**12.83**	*2011/03/31*
19	**internal circlip dia 12mm**	Bearing Man Group Bellville	1	internal circlip dia. 12mm 1mm thick	**0.23**	*2011/03/31*
20	**6900 DDU Ball Bearing**	Bearing Man Group Bellville	1	6900 DDU Ball Bearing	**79.17**	*2011/03/31*
21	**SH 16 WON Linear Bearing Unit**	Bearing Man Group Bellville	2	SH 16 WON Linear Bearing Unit	**628.46**	*2011/03/31*
22	**5 Cell Lipo Batteries**	Claude Mackrill	2	5 Cell 18.5V 5500mah Lipo Batteries	**4500.00**	*2011/05/06*
23	**5 Cell Lipo Batteries**	Claude Mackrill	2	5 Cell 18.5V 500mah Lipo Batteries	**4468.00**	*2011/06/13*
24	**3 Blade 16 x 10 in Propeller, Tractor action**	Claude Mackrill	1	3 Blade 16 x 10 in Propeller, Tractor action	**799.00**	*2011/05/06*

Table F.1: Continued

	Item	Supplier	Qty	Descprition	Amount (inc Vat) [R]	*Date of Purchase*	
25	**3 Blade 20 x 10 in Propeller, Tractor action**	Claude Mackrill	1	3 Blade 20 x 10 in Propeller, Tractor action	**799.00**	*2011/05/06*	
26	**3 Blade 22 x 10 in Propeller, Tractor action**	Claude Mackrill	1	3 Blade 22 x 10 in Propeller, Tractor action	**948.00**	*2011/05/06*	
27	**Emeter 2 RDU**	Claude Mackrill	1	Hyperion Emeter 2 Remote Data Unit for electronic flight system	**1745.00**	*2011/05/16*	
28	**Temperature Probe type 1**	Claude Mackrill	1	Temperature sensor for Emeter 2 RDU	**124.00**	*2011/05/16*	
29	**Temperature Probe type 2**	Claude Mackrill	1	Temperature sensor for Emeter 2 RDU	**122.00**	*2011/05/16*	
30	**Digital Tachometer**	Claude Mackrill	1	Digital Tachometer sensor for Emeter 2 RDU	**120.00**	*2011/05/16*	***Sub Total [R]***
31	**Emeter 2 System Monitor Unit**	Claude Mackrill	1	Hyperion Emeter 2 testbench system monitor unit	**1789.00**	*2011/05/16*	***48930.71***
Electrical Setup and Connection for Test bench							
32	**Bullet Connectors 4.5mm**	Claude Mackrill	10	Gold plated 4.5mm bullet electrical connectors	**614.20**	*2011/06/01*	
33	**Bullet Connectors 6mm**	Claude Mackrill	10	Gold plated 6mm bullet electrical connectors	**635.80**	*2011/06/02*	
34	**Banana Plug Socket Inline**	Brights	4	Banana Plug Socket Inline	**22.76**	*2011/06/01*	
35	**Banana Plug Inline**	Brights	4	Banana Plug Plastic Inline	**23.16**	*2011/06/01*	
36	**Heat Shrink 1m Prepack 4.8mm**	Brights	2	Heat Shrink 1m Prepack 4.8mm white and black	**39.00**	*2011/06/01*	
37	**Heat Shrink 1m Prepack 6.4mm**	Brights	1	Heat Shrink 1m Prepack 6.4mm red	**29.50**	*2011/06/01*	
38	**Flex Speaker Cable 4mm**	Brights	1	Flex Speaker Cable 4mm red/black	**6.99**	*2011/06/01*	
39	**Flex Speaker Cable 1.5mm**	Brights	1	Flex Speaker Cable 1.5mm red/black	**3.50**	*2011/06/01*	
40	**Cable Ties**	Brights	1	Cable tie black(100XPKT) 200 x 4.6mm	**17.42**	*2011/06/06*	
41	**Solder less Banana Plug Socket**	Yebo Electronics	2	Solderless Banana Plug Socket	**12.86**	*2011/06/01*	***Sub Total [R]***
42	**Solder less Banana Plug**	Yebo Electronics	2	Solderless Banana Plug	**13.09**	*2011/06/01*	***1418.28***
Stabilization Electronics							
43	**Quad copter**	Clown Hobbies	1	The Arducopter Quadcopter for the piloting system electronics	**5199.00**	*2011/06/13*	
44	**Propeller Set 10 x 4.5**	Clown Hobbies	2	Propeller Set 10 x 4.5	**100.00**	*2011/07/23*	***Sub Total [R]*** ***5299.01***
					Total [R]		***55648.00***