

UNIVERSITY OF DERBY

**An Experimental Validation and Test Platform
development for Simulation and Control of
quadcopters.**

Ajay K Joseph

Doctor of Philosophy

2022

This research aims to develop a robust control system for quadcopters by addressing key objectives: creating an analytical mathematical model, extracting physical parameter values, developing a rapid 3D visualization platform on MATLAB, and implementing a robust controller with data and sensor fusion. Quadcopters are a type of agile, multi-rotor unmanned aerial system (UAS) that utilize four rotors for lift, offering a cost-effective and reliable alternative to traditional helicopters.

The research successfully develops an analytical mathematical model, initially analyzed through graphical representation on MATLAB/Simulink software. The non-linear equations were linearized and represented in a state-space model, simplifying the model and enabling the deployment of simpler control and filtering models. To test the control system's robustness, Gaussian white noise was added to the system and filtered using various techniques. Extracting physical parameters using off-the-shelf instruments can be expensive, so alternative techniques and equipment were investigated to obtain the best outcome. After acquiring the parameters, they were integrated into the mathematical model.

To simplify system validation, the research highlights the benefits of a rapid prototyping 3D model for analysis. Various approaches were adopted to identify anomalies within the system and build a robust control system before deployment onto hardware. Initially, an Atmega328P microcontroller was used for in-flight testing, communicating between ground stations (MATLAB/Simulink) in real-time via Bluetooth. Later, a more powerful Raspberry Pi 3 B+ was utilized as an onboard computer. A robust wireless communication system, considering noise filtering and error-checking methods, was successfully implemented using both Atmega328P and RPi 3B+. This included Bluetooth wireless communication and later, WiFi for wireless communication and integration of sensors such as a camera for visual feedback.

In conclusion, this research primarily focuses on quadcopter simulation and performance evaluation, providing a solid foundation for obtaining physical properties, control models, and sensor filtering. The developed control system demonstrates potential for improving quadcopter reliability and performance, with the analytical mathematical model, rapid 3D visualization, and robust wireless communication system aiding in the development and validation of quadcopter control systems.

ACKNOWLEDGEMENT

First and foremost, I would like to express my deepest gratitude to my supervisor, Dr Amar Bousbaine, for his continual encouragement, support and persistent patience. His advice and assistance provided me with the necessary tools to complete this dissertation. His enthusiasm and knowledge of the subject greatly inspired me to pursue this research. I have been fortunate to have received some valuable feedback on some of the ideas presented in this dissertation.

A debt of gratitude is owed to my best friends Noel David, Dr Vibin Ambukan, and Abdul Kadar for their support, encouragement, generous guidance and insightful comments and suggestions. Their advice and encouragement have been invaluable to me over the years, especially during the COVID-19 pandemic.

My deep appreciation goes to my dear parents, Mr Joseph Kurian and Mrs Josepheena Joseph, for their prayers and encouragement during my studies. Special thanks go to my sisters, Miss Anu Joseph and Amala Joseph, for their love, and encouragement. Also, my cousin Prince Paulose, Praveen Paulose and aunt Gracy Paulose.

I would like to thank all my other friends for all the help and morale boost provided during the writing-up. I would like to thank Dr William Dudeney for allowing me to use the equipment for testing and validation. Finally, I would like to thank Dr Tim Poyi for all his help in the lab and for understanding the model.

Table of Contents

ABSTRACT.....	ii
ACKNOWLEDGEMENT	iii
LIST OF FIGURES	xi
LIST OF TABLES	xviii
ABBREVIATIONS	xix
NOMENCLATURE	xx
Chapter 1 - Introduction.....	1-1
1.1 Unmanned Aerial Vehicles (UAVs)	1-2
1.2 The miniature quadrotor helicopter.....	1-5
1.3 Problem statement.....	1-6
1.4 Aim	1-7
1.5 Main objectives.....	1-7
1.6 Contributions.....	1-7
1.7 Literature Review.....	1-9
1.7.1 History of quadcopter.....	1-9
1.7.2 Previous works on the quadcopters.....	1-13
1.7.3 Thesis outline	1-2
1.8 Applications of miniature quadrotor UAVs.....	1-23
1.8.1 Military Applications	1-23
1.8.2 Surveillance.....	1-24
1.8.3 Racing	1-25
1.8.4 Wildfire detection	1-25
1.8.5 Photography	1-26
1.8.6 Research.....	1-26
1.8.7 Agricultural and Industrial applications.....	1-26
1.8.8 Disaster Management/ Search and Rescue	1-26
1.8.9 License and regulations of quadcopters within the UK	1-27

1.8.10 Open Category	1–28
1.9 Anatomy of the Quad-rotor Helicopter	1–31
1.9.1 Frame	1–32
1.9.2 Motors and Propellers	1–33
1.9.3 Battery	1–33
1.9.4 Sensors	1–33
1.9.5 Flight Control Board	1–34
1.10 Chapter Summary	1–34
Chapter 2 Basic Concept and Rigid Body Dynamics	2–36
2.1 Basic concepts of the quadrotor	2–37
2.1.1 Yaw	2–39
2.1.2 Pitch	2–40
2.1.3 Roll	2–40
2.1.4 Throttle – Vertical Motion	2–41
2.2 Coordinate frames	2–41
2.3 Euler Angles	2–43
2.4 Rotation about intermediate axes	2–45
2.5 Kinematics of a quadcopter	2–47
2.6 The inner and outer loop	2–48
2.7 Summary	2–51
Chapter 3 – Analytical and Dynamical Model of Quadcopter	3–53
3.1 Quad-rotor Modelling Assumptions	3–54
3.2 The dynamic equation for translation motion	3–55
3.3 The dynamic equation for rotational motion	3–59
3.3.1 Mass moment of inertia	3–60
3.4 Newton-Euler equation	3–61
3.5 Identification of forces and moments of the quadcopter	3–62
3.5.1 Gravitational force	3–63

3.5.2 Gyroscopic effect on a quadcopter.....	3-63
3.5.3 Control moment/torque	3-66
3.6 Quadcopter State Variable definition.....	3-67
3.7 Aerodynamic properties of the propeller blade.....	3-69
3.8 Actuator Dynamics (DC-motor)	3-81
3.8.1 Voltage and angular velocity	3-81
3.8.2 Voltage and thrust of the motor	3-83
Control system overview	3-84
3.8.3 Kalman filter	3-84
3.8.4 PID controller.....	3-86
3.9 Chapter Summary	3-88
Chapter 4 – Controller Design Methodology and Experimental Extraction of the Physical Parameters	4-89
4.1 Linearisation and State-space (SS) Representation for Quadcopter	4-90
4.1.1 Z-axis movement - Altitude	4-95
4.1.2 Yaw movement - Directional.....	4-95
4.1.3 Rotation along Y axis - Latitudinal.....	4-95
4.1.4 Rotation along X axis - Longitudinal.....	4-95
4.1.5 Controllability and Observability of the systems.....	4-95
4.2 Flight control Algorithm	4-96
4.2.1 Control Technique.....	4-97
4.2.2 Outer loop	4-97
4.2.3 Inner loop	4-97
4.2.4 Throttle controller	4-97
4.2.5 Roll and Pitch controller	4-98
4.2.6 Yaw controller	4-98
4.3 Kalman Filter Algorithm.....	4-99
4.3.1 Altitude	4-99

4.3.2 Directional.....	4–100
4.3.3 Latitudinal	4–100
4.3.4 Longitudinal.....	4–101
4.4 Identification of the physical parameters	4–102
4.4.1 Moment of Inertia	4–102
4.4.2 Motor Torque – Load cell	4–107
4.4.3 Parameter extraction procedure.....	4–110
4.4.4 Empirical measurement of motor inductance	4–116
4.4.5 Parameters implemented in the simulation	4–121
4.5 Summary	4–122
Chapter 5 - Simulation of Quadcopter Model in MATLAB/Simulink	5–124
5.1 MATLAB/Simulink Software.....	5–125
5.2 Model Implementation in MATLAB/Simulink	5–126
5.2.1 Actuator Subsystem	5–126
5.2.2 Roll Subsystem	5–127
5.2.3 Pitch Subsystem	5–128
5.2.4 Yaw Subsystem.....	5–128
5.2.5 X-Motion Subsystem	5–129
5.2.6 Y-Motion Subsystem	5–130
5.2.7 Z-Motion Subsystem.....	5–130
5.3 Mathematical Subsystem Model.....	5–131
5.4 Motor Mixer.....	5–131
5.5 The Open Loop Model.....	5–132
5.6 Controller	5–133
5.7 Kalman filter modelling	5–134
5.8 Running the Simulation	5–136
5.8.1 Noise	5–137
5.8.2 PWM output.....	5–139

5.8.3 Throttle (Vertical Motion)	5-140
5.8.4 Roll and Pitch.....	5-141
5.8.5 Yaw	5-142
5.9 Summary	5-144
Chapter 6 – 3D trajectory generation and Computer-Aided Design Modelling of Quadcopter	6-146
6.1 Quadrotor Dynamics	6-147
6.2 Quadrotor Controller.....	6-147
6.3 Simulink Model	6-147
6.4 Main MATLAB Script.....	6-148
6.5 The AnimationQuad function	6-149
6.6 Quadcopter 3D modelling.....	6-149
6.7 Standard Tessellation Language File	6-150
6.7.1 ASCII or Text format.....	6-151
6.7.2 Binary Format	6-152
6.7.3 STL file results.....	6-153
6.8 Wavefront OBJ file	6-154
6.8.1 Wavefront OBJ implementation	6-155
6.8.2 Wavefront OBJ results	6-159
6.9 3D animations	6-160
6.9.1 3D Quadcopter model	6-160
6.9.2 Animation Results.....	6-161
6.10 Summary	6-164
Chapter 7 - Sensor Fusion and Wireless Communication Systems	7-165
7.1 Sensors and communication systems.....	7-166
7.1.1 The basic principle of sensors	7-166
7.1.2 Communication Systems.....	7-167
7.2 Atmega328p setup procedure.....	7-169
7.2.1 Bluetooth (HC-05)	7-169

7.2.2 ESC PWM frequency control	7-171
7.2.3 ESC calibration	7-177
7.2.4 Ultrasonic modules	7-178
7.2.5 Error checking and UART communication	7-183
7.2.1 IMU.....	7-188
7.2.2 Sensor fusion.....	7-190
7.3 Raspberry Pi 3 (RPI 3).....	7-197
7.3.1 Sobel – Edge detection.....	7-198
7.3.2 Harris & Stephen Operator – Corner Detection.....	7-200
7.4 Summary	7-201
Chapter 8 - Conclusions and further work	8-203
8.1 Research Objective	8-204
8.2 Research summary and conclusions	8-204
8.2.1 Mathematical modelling of the quadcopter.....	8-204
8.3 Findings and contribution	8-206
8.4 Novelty.....	8-211
8.5 General Discussion	8-206
8.6 Future research recommendation	8-212
8.7 Publications.....	8-212
Reference	8-214
APPENDIX A - Civil Aviation Authority Rules and Regulations	8-226
APPENDIX B – Read Wavefront OBJ File.....	8-227
APPENDIX C – Read STL file.....	8-229
APPENDIX D - Final visualisation code.....	8-231
APPENDIX E – Bifilar Pendulum Test Results	8-245
APPENDIX F - Atmega328P	8-247
APPENDIX G - Complimentary filter.....	8-256
APPENDIX H - UDP code	8-260

APPENDIX I - Load cell	8-262
APPENDIX J - Motor parameter extraction table	8-263

LIST OF FIGURES

Figure 1-1 - Fixed and Rotary wing aircraft (Ajay Joseph 2017)	1-4
Figure 1-2 - Bréguet- Richet Gyroplane (Anon n.d.-o)	1-9
Figure 1-3 - Dr George De Bothezat helicopter (Anon n.d.-aj)	1-10
Figure 1-4 - Convertawings Model a Quadrotor (Anon n.d.-t).....	1-11
Figure 1-5 - Curtiss-Wright VZ-7 (Anon n.d.-w)	1-12
Figure 1-6 - Literature review roadmap	1-13
Figure 1-7 - Military application quadcopter(Anon n.d.-bj)	1-24
Figure 1-8 - Racing drone(Anon n.d.-af)	1-25
Figure 1-9 - Flight categories.....	1-28
Figure 1-10 – Summary of drone laws in the UK (Anon n.d.-bb).....	1-28
Figure 1-11 - Open – (Subcategory A1)	1-28
Figure 1-12 - Open (Subcategory - A2).....	1-29
Figure 1-13 - Open (Subcategory - A3)	1-30
Figure 1-14 - Anatomy of a quadcopter (Ajay Joseph 2017).....	1-32
Figure 1-15 - Types of frame configuration (x and +)	1-33
Figure 2-1 - Left turn	2-37
Figure 2-2 - Forward movement	2-38
Figure 2-3 - Yaw rotation	2-38
Figure 2-4 - Overall block diagram.....	2-39
Figure 2-5 - Demonstration of Yaw manoeuvre along Z-axis	2-39
Figure 2-6 - Demonstration of Pitch manoeuvre along the Y-axis	2-40
Figure 2-7 - Demonstration of Roll manoeuvre along the X-axis	2-41
Figure 2-8 - Coordinate systems (Yumpu.com n.d.) (William W.Warren J).....	2-42
Figure 2-9 - Earth and body frame.....	2-43
Figure 2-10 - X-axis rotation	2-44
Figure 2-11 - Y-axis rotation	2-44
Figure 2-12 - Z-axis rotation.....	2-44
Figure 2-13 - Inertial frame of reference	2-45
Figure 2-14 - Body frame of reference	2-45
Figure 2-15 - MATLAB default Euler angle sequence (Anon n.d.-s)	2-46
Figure 2-16 - Frame Re to R1	2-46
Figure 2-17 - Frame R1 to R2	2-47

Figure 2-18 - Frame R2 to Rb	2-47
Figure 2-19 - Body axis angular rate to Euler rates	2-48
Figure 2-20 - Inner and outer loop controller block diagram.....	2-49
Figure 2-21 - Conventional yaw angle +ve rotation	2-49
Figure 3-1 - Plant/System model block diagram	3-55
Figure 3-2 - Velocity expressed in the body frame.....	3-57
Figure 3-3 - Body frame components	3-58
Figure 3-4 - Moment of Inertia in body frame.....	3-60
Figure 3-5 - Gyro effect - propeller properties(Ajay Joseph 2017)	3-65
Figure 3-6 - Thrust(Ajay Joseph 2017).....	3-65
Figure 3-7 - Control inputs/moment/torque (Ajay Joseph 2017).....	3-67
Figure 3-8 – Airfoil (Ajay Joseph 2017).....	3-70
Figure 3-9 - Propeller blade (Ajay Joseph 2017).....	3-72
Figure 3-10 – Determine differential thrust (Ajay Joseph 2017).....	3-73
Figure 3-11 - Laminar and turbulent flow (Ajay Joseph 2017)	3-74
Figure 3-12 - Airfoil differential width (Ajay Joseph 2017).....	3-75
Figure 3-13 - Drag coefficients of various shapes (Gustafsson 2009).....	3-75
Figure 3-14 - Differential air resistance moment (Ajay Joseph 2017).....	3-77
Figure 3-15 - DC motor model (Poyi 2014)	3-81
Figure 3-16 - Flow diagram of KF (Stevens and Lewis 2003)	3-86
Figure 3-17 - Block diagram of PID controller (Srinivasan K 2016)	3-87
Figure 4-1 – Chapter 4 Overview	4-89
Figure 4-2 - Flight controller block diagram (Ajay Joseph 2017)	4-96
Figure 4-3 - X and Y controller design	4-97
Figure 4-4 - Throttle controller block	4-98
Figure 4-5 - Roll and Pitch controller block	4-98
Figure 4-6 - Yaw controller block.....	4-99
Figure 4-7 - KF block diagram	4-99
Figure 4-8 - Simulink Package Parrot Mambo	4-102
Figure 4-9 - Moment of inertia of a rod	4-103
Figure 4-10 - Moment of inertia of point mass	4-103
Figure 4-11 - Quad moment of inertia	4-103
Figure 4-12 - Moment of inertia of quad equivalent (I_{xx} , I_{yy} , I_{zz}).....	4-103
Figure 4-13 - Bifilar Pendulum.....	4-104
Figure 4-14 - Derivation of angles.....	4-104

Figure 4-15 - Derivation of displacement	4-104
Figure 4-16 - Bifilar pendulum forces	4-105
Figure 4-17 - Bifilar pendulum test Parrot Mambo	4-107
Figure 4-18 - Bifilar Pendulum custom quad.....	4-107
Figure 4-19 - Load cell circuit diagram (Anon n.d.-c).....	4-108
Figure 4-20 - Load cell measurement	4-109
Figure 4-21 - Thrust measurement platform setup.....	4-110
Figure 4-22- Kv constant	4-112
Figure 4-23 - Back emf constant Ke	4-113
Figure 4-24 - Thrust and Voltage squared relationship	4-114
Figure 4-25 - Relationship between thrust and squared angular velocity	4-114
Figure 4-26 - Voltage vs Angular velocity squared.....	4-115
Figure 4-27 - PWM vs Voltage input.....	4-116
Figure 4-28 - LCR meter readings	4-116
Figure 4-29 - Method 1 - Equivalent circuit diagram(Anon n.d.-i)	4-117
Figure 4-30 – Signal generator impedance - i/p.....	4-117
Figure 4-31 - Signal generator impedance - with load.....	4-118
Figure 4-32 - Signal generator impedance	4-118
Figure 4-33 - Inductance at higher frequencies - Method 1 (Half input V)	4-119
Figure 4-34 - Inductance measurement - Tektronix circuit diagram(Anon n.d.-q).....	4-120
Figure 4-35 - Tektronix method result.....	4-120
Figure 5-1 - Quadcopter configurations.....	5-126
Figure 5-2 - Block diagram representing the architecture of the system - Plant subsystem	5-126
Figure 5-3 - Actuator subsystem.....	5-127
Figure 5-4 - Roll subsystem	5-127
Figure 5-5 - Pitch subsystem.....	5-128
Figure 5-6 - Yaw subsystem	5-129
Figure 5-7 - Linear acceleration along x-axis	5-129
Figure 5-8 - Linear acceleration along y-axis	5-130
Figure 5-9 - Vertical acceleration along the z-axis	5-131
Figure 5-10 - Mathematical model subsystem	5-131
Figure 5-11 - Motor Mixer.....	5-132
Figure 5-12 - Block diagram representing the architecture of the system - Controller and Plant... 5-132	
Figure 5-13 - Open-loop model	5-133
Figure 5-14 - Block diagram representing the architecture of the system - Controller subsystem. 5-133	

Figure 5-15 - Inner and outer loop	5-134
Figure 5-16 - Kalman filter block parameters.....	5-135
Figure 5-17-Pitch, Roll, Yaw, Altitude - KF Simulink block diagram.....	5-136
Figure 5-18 - Block diagram representing the architecture of the system - Trajectory subsystem.....	5-136
Figure 5-19 - Open loop trajectory planner	5-137
Figure 5-20 - Signal generator	5-137
Figure 5-21 - Joystick	5-137
Figure 5-22 - Noise block set to plant model.....	5-138
Figure 5-23 - Gaussian white noise generator block.....	5-138
Figure 5-24 - Measurement noise in KF	5-139
Figure 5-25 - Block diagram representing the architecture of the system - PWM O/P.....	5-139
Figure 5-26 - PWM block set.....	5-140
Figure 5-27 - Z displacement with noise	5-140
Figure 5-28 - Attitude - KF Q and R matrix	5-141
Figure 5-29 - Roll and pitch command	5-141
Figure 5-30 - Roll and Pitch KF.....	5-142
Figure 5-31 - Roll and Pitch - Q and R matrices.....	5-142
Figure 5-32 - Yaw command	5-143
Figure 5-33 - Yaw KF response.....	5-143
Figure 5-34 - Yaw Q and R matrices	5-144
Figure 5-35 - Complete simulation model	5-144
Figure 6-1 - Workspace for 3D visualisation.....	6-147
Figure 6-2 - Sim() command.....	6-148
Figure 6-3 - AnimationQuad function	6-149
Figure 6-4: I/P parameter to AnimationQuad function – displacement, angles & variable arg.....	6-149
Figure 6-5 - Parrot Mambo STL	6-151
Figure 6-6 - Custom Quad STL	6-151
Figure 6-7 - STL ASCII representation	6-152
Figure 6-8 - STL Binary format.....	6-153
Figure 6-9 - stlread() corrupt file	6-153
Figure 6-10 - Geometric vertices	6-155
Figure 6-11- Texture vertices.....	6-155
Figure 6-12 - Face with vertex position & texture coordinate	6-155
Figure 6-13: Object file.....	6-156
Figure 6-14: Importing and refining vertices	6-156

Figure 6-15: Correction and Scaling	6-156
Figure 6-16: Rotational matrix and degrees to radian conversion	6-157
Figure 6-17: Patch function	6-158
Figure 6-18: Movie in M file and Getframe function	6-158
Figure 6-19: End - Displacement Trajectory	6-158
Figure 6-20 - wavefront obj error	6-159
Figure 6-21 - Time taken readObj() and read_wobj functions	6-160
Figure 6-22 - Wavefront OBJ result	6-160
Figure 6-23 - Windows 3D builder model	6-161
Figure 6-24 - STL view.....	6-161
Figure 6-25 - OBJ format.....	6-161
Figure 6-26 - Altitude command.....	6-162
Figure 6-27 - Quad X directional movement	6-162
Figure 6-28 - Quad Y directional movement	6-163
Figure 6-29 - Graphical representation of overshoot	6-163
Figure 6-30 - Signal builder commands.....	6-164
Figure 6-31 - Quad rectangular trajectory using all commands.....	6-164
Figure 7-1 - Sensor Fusion and Wireless Communication - Overall architecture	7-165
Figure 7-3 - Ultrasonic sensor SRF05.....	7-166
Figure 7-4 - Ultrasonic sensor SRF05 timing diagram	7-166
Figure 7-5 – Atmega328P BT connection result	7-170
Figure 7-6 - Difference in PWM frequencies	7-171
Figure 7-7 - I/O Registers	7-173
Figure 7-8 - TC1 Control Register A	7-174
Figure 7-9 - WGM and Pre-scaler.....	7-174
Figure 7-10 - Thrust Measurement	7-176
Figure 7-11 - ESC calibration sequence	7-178
Figure 7-12 - KF implementation on SRF05	7-179
Figure 7-13 - Ultrasonic SRF50 results with KF	7-180
Figure 7-14 - I2c for MAXSonar Ultrasonic	7-181
Figure 7-15 - Serial Configuration Simulink	7-183
Figure 7-16 - Corrupt data	7-184
Figure 7-17 - Expected incoming data test	7-184
Figure 7-18 - Noise captured using the oscilloscope	7-185
Figure 7-19 - IMU wiring diagram	7-188

Figure 7-20 - IMU MPU6050 Block diagram	7-189
Figure 7-21 - Accelerometer moving average	7-192
Figure 7-22 - Accelerometer block diagram demo	7-193
Figure 7-23 - Gyroscopic drift	7-193
Figure 7-24 - Complimentary filter gain 0.4 and 0.1	7-194
Figure 7-25 - Complimentary filter true angle	7-195
Figure 7-26 - KF model	7-197
Figure 7-27 - KF result	7-197
Figure 7-28 – Lane detection test area	7-198
Figure 7-29 - Sobel-X operation (Anon 2020b).....	7-199
Figure 7-30 - Edge detection model(Horak and Zalud 2015)	7-200
Figure 7-31 - Edge detection result.....	7-200
Figure 7-32 - Harris Operator - Corner detection(Horak and Zalud 2015).....	7-201
Figure 7-33 - Corner detection result.....	7-201
Figure 7-34 - Complete model.....	7-201
Figure 0-1 - Saving to the Array format	8-231
Figure 0-2 - Setting up viewing angle and step size	8-232
Figure 0-3 - stlreadAJ() renamed.....	8-232
Figure 0-4: For loop for displacement and angle	8-232

LIST OF TABLES

Table 1-1 - WiFi History (Anon n.d.-k:21).....	1–15
Table 1-2 - Drone Regulations and Classifications.....	1–31
Table 2-1 - Rotational Matrix (Rx, Ry, Rz).....	2–44
Table 2-2 - Intermediate angular velocity vector.....	2–50
Table 3-1 - State Variables	3–69
Table 3-2 -PID controller response characteristics	3–87
Table 4-1 - Moment of Inertia results Parrot Mambo	4–107
Table 4-2 - Thrust measurements from the datasheet	4–110
Table 4-3 - BLDC initial measurements.....	4–111
Table 4-4 - Table of physical parameters.....	4–122
Table 5-1 - Throttle - PID and KF response	5–141
Table 5-2 - Pitch and Roll PID response.....	5–142
Table 5-3 - Yaw PID KF response.....	5–143
Table 7-1 - FTDI - HC05 Connection.....	7–169
Table 7-2 - AT commands	7–170
Table 7-3 - Timer Default Settings	7–171
Table 7-4 - Pre-Scaler and Timers	7–175

ABBREVIATIONS

Abbreviation	Description
UAV	Unmanned Aerial Vehicle
CAA	Civil Aviation Authority
UAS	Unmanned Aircraft System
VLOS	Visual Line of Sight
CofC	Certificate of Competency
RPi	Raspberry Pi
KF	Kalman Filter
DoF	Degree of freedom
EKF	Extended Kalman Filter
UKF	Unscented Kalman Filter
LQR	Linear Quadratic Regulator
CF	Complimentary Filter
MPC	Model Predictive Controller
PID	Proportional Integral Derivative
PD	Proportional Derivative
PI	Proportional Integral
IMU	Inertial Measurement Unit
PWM	Pulse Width Modulation
RC	Radio Controller
RTI	Real-Time Interface
VTOL	Vertical Take-off and Landing
DCM	Direction Cosine Matrix
RSSI	Received Signal Strength Indicator

NOMENCLATURE

Symbol	Name	Description	Unit
g	g	Acceleration due to gravity	m/s^2
ρ	Rho	Air density	kgm^{-3}
ι	Iota	The density of a material	kgm^{-3}
A_c		Cross-sectional area	m^2
L		Length of the rod/quadcopter arm	m
Z		Motor resistance	Ω
ω_m		Motor angular velocity	$rads/s$
k_e		Motor Back emf constant	$V \cdot \frac{s}{rad}$
$\dot{u}, \dot{v}, \dot{w}$		Angular accelerations in the body frame	m/s^2
V_p		Induced air velocity due to stream tube effect	m/s
dT		Differential thrust force within the airfoil	N
dL		Differential lift force	N
m		Mass	Kg
$R_x(\phi)$		Rotation around X-axis	$rads/s$
$R_y(\theta)$		Rotation around Y-axis	$rads/s$
$R_z(\psi)$		Rotation around Z-axis	$rads/s$
I_{xx}		Rotational inertia along the x-axis	Nms^{-2}
$\vec{\Lambda}_{GR}^B$		Gyroscopic force moment vector	Nm
p, q, r		Angular velocity in the body frame	$rads/s$
$\dot{\Omega}$		Motor angular acceleration	$rads/s^2$
Ω	Omega	Motor angular velocity	$rads/s$
R_{Θ}		Combined rotational matrix sequence	
$\vec{\omega}$	Omega	Angular velocity vector	$rads/s$
T_{Θ}		Transfer matrix	
A		State matrix	
B		Input matrix	
C		Measurement Matrix	
D		Input feed-forward Matrix	
O_E		Earth frame origin	
O_B		Body frame origin (centre of mass)	
$\ddot{X}, \ddot{Y}, \ddot{Z}$		Acceleration along the X, Y, and Z axes	m/s^2
$\dot{\phi}, \dot{\theta}, \dot{\psi}$		Euler angle rates in inertial frame	$rads/s$
F		Force	N
$\hat{i}, \hat{j}, \hat{k}$		Unit vectors	1
$\vec{v}$		Velocity vector	m/s
$\vec{\Gamma}^E$		Translation motion	m
F_x, F_y, F_z		Forces with respect to the body frame	N
D		Drag coefficient	
$\vec{H}^B$		Angular moment vector	N
I^B		The mass moment inertia matrix	kgm^2
$\vec{\Lambda}_{IN}^B$		Control input vector	

J_{TP}		Mass moment of inertia of propeller	kgm^2
$\vec{H}_M$		Total angular momentum of propeller	kgm^2/s
$\vec{H}_1, \vec{H}_2, \vec{H}_3, \vec{H}_4$		Angular momentum vector per propeller	kgm^2/s
M_{GRM}		The gyroscopic effect produced by motors	
$\vec{M}_{GR}$		Total gyroscopic moment of motors	$N.m$
ϕ	Phi	Tilt angle	rad
θ	Theta	Tilt angle	rad
Ψ	Psi	Title angle	rad
V_L		Inductor voltage	V
V_R		Resistor voltage	V
V		The input voltage to the motor	V

Chapter 1 - Introduction

This chapter will focus on providing a basic overview of Unmanned Aerial Vehicles (UAVs). It will centre on the multiple divisions of UAVs and then highlight the many different elements that make up a UAV and the aspects of each component. This chapter searches and evaluates the available literature about aerial robotics or quadcopters. It documents state-of-the-art with respect to the quadcopter subject and identifies areas where further research is required. This chapter is a study conducted to contribute to understanding a particular issue and identify knowledge gaps. It starts by looking at the history of quadcopters with a systematic and extensive search for relevant sources of information, for example, through a search of databases (e.g., Google Scholar) or by consulting expert sources. This chapter aims to collate and synthesise all the existing evidence on a subject. Finally, the applications of how miniature quadrotors can be used in various situations will be discussed.

1.1 Thesis outline

The thesis has been tailored to 8 distinctive but systematically and logically assigned chapters.

Chapter 1 describes the advantages of the quadcopter, and where they are applicable and highlights the anatomy of a quadcopter with commonly used sensors and components. It also describes the legal requirements to fly an unmanned aerial vehicle in the year 2021. This chapter also provides a literature review by initially giving history and background knowledge on quadcopters. The paper synthesizes several control techniques and various approaches taken by previous researchers in this field of study. The chapter also justifies the reasons for adopting certain techniques being deployed in this research.

Chapter 2 demonstrates the working principles of rigid body movement and coordinate frame. This includes deriving rotational matrices for rotation and translation of a rigid body using the right-hand rule. It shows the importance of maintaining a sequence of rotations to formulate unique rotational and translation matrices.

Chapter 3 focuses on developing a dynamic analytical model using Newton-Euler equations. This is achieved by 6 DoF of a rigid body in three-dimensional space. In other words, 6 DoF represents the exact number of axes that a quadcopter can move in three-dimensional space. It combines motor, propeller and control theory to develop a faithful mathematical representation of the quadcopter.

Chapter 4 focuses on linearising the model in state-space (SS) representation to be able to quickly develop a control system. The chapter demonstrates how to evaluate various constants/hardware parameters such as the moment of inertia, inductance of motor and torque/thrust characteristics through empirical experiments.

Chapter 5 demonstrates the implementation of the dynamic analytical model using MATLAB/Simulink. It helps to build the mathematical equations derived in previous chapters to several subsystems in MATLAB/Simulink to analyse the performance. The chapter ensures that the model's performance is satisfactory using open and closed-loop tests. Initial verification is carried out by inspecting the results in graphical representation.

Chapter 6 describes manipulating the results from the previous chapter to visualise the results in a 3D environment. It uses rapid prototyping methods using commonly used 3D file format for visual analysis of the quadcopter model. The dynamic analytical model derived from previous chapters is corroborated to validate the graphical and visual representation results.

Chapter 7 integrates real-world sensor data with the analytical MATLAB/Simulink model to evaluate the quadcopter's performance. It demonstrates two different approaches – first, MATLAB/Simulink is a ground station to send and receive control commands through a robust wirelessly communication network, and second, the onboard computer executes commands independently and relays feedback through live stream using UDP protocol.

Chapter 8 delivers the research summary, recommendations for future research and conclusion of this thesis.

1.2 Unmanned Aerial Vehicles (UAVs)

Unmanned aerial vehicles (UAVs), also known as drones, are flying machines flying without a pilot. In other words, an aircraft operated without an onboard human pilot. A ground control station typically controls them. The flight is controlled either autonomously by onboard computers or by remote control from the ground. UAVs are often preferred over manned aircraft because they can offer greater flexibility, endurance, and cost-effectiveness. They can also be used in missions considered too "dull, dirty or dangerous" for manned aircraft. UAVs can also be controlled independently or follow a pre-programmed flight path. UAVs are generally classified according to their weight, altitude range and endurance. The two main types of UAVs are fixed-wing and rotor wings. Their functions can be utilised in both a commercial environment and for military purposes(Pretorius and Boje, 2014a).

Fixed-Wing UAVs have wings and a tail, which make them similar to an aeroplane. Fixed-wing drones move due to forward movement combined with an aerodynamic structure, allowing lift to be generated resulting in flight. They are best suited for long-range missions as they can carry more fuel and endurance. They can also land on a runway. Fixed-wing UAVs can fly at very high altitudes, as they have a fixed-wing configuration with horizontal and vertical stabiliser surfaces. They can fly at very high speeds, as they have a streamlined body design. They are capable of carrying heavier payloads as compared to rotary-wing UAVs. This type of UAV's primary disadvantage is that they cannot hover for a long time, as they rely on air currents for flying. These require a runway to land and take off. The other limitation of fixed-wing drones is the inability to fly at lower speeds. This has pushed engineers to focus their energy toward rotor-wing aircraft. This research will focus primarily on rotor-wing aircraft, especially the quadcopter variety('Drone Types: Multi-Rotor vs Fixed-Wing vs Single Rotor vs Hybrid VTOL', 2016).

The rotary-wing aircraft is designed for short missions where the payload is lighter, and the endurance is lower. They have the capability to hover in one place for a long time, which makes them ideal for surveillance purposes or target acquisition missions. They are best suited for various tasks such as surveillance, target acquisition, communication relay, target spotting, combat support, casualty evacuation (CASEVAC), and convoy escort duties. Rotor-wing UAVs have the added benefit of moving much more dynamically, even in restricted spaces. They are also capable of hovering, vertical lift-off and landing without the need for launching mechanisms. However, limitations do arise in the form of these particular drones lacking the ability to carry heavy payloads while also having heavy energy consumption compared to their fixed-wing counterparts (Basset, Tremolet and Lefebvre, 2014).

Figure 1-1 below represents a fixed-wing aircraft and a multi-copter UAV.

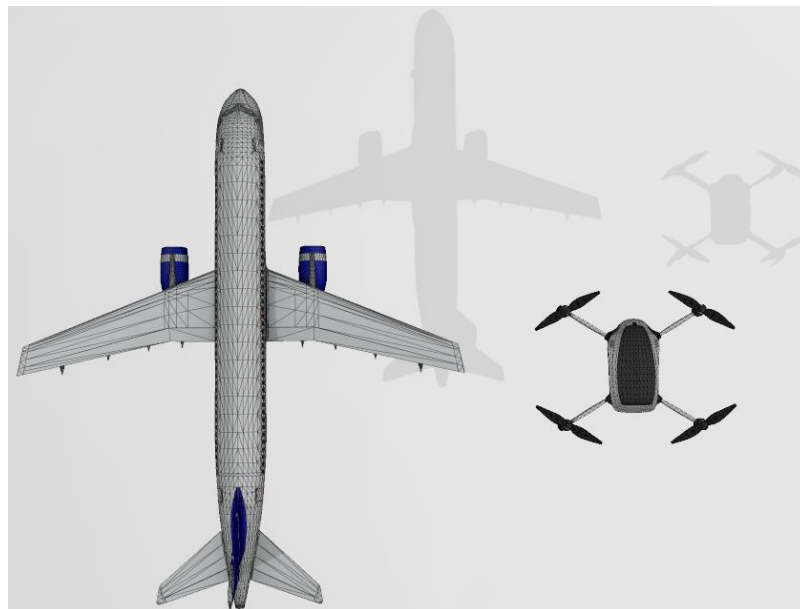


Figure 1-1 - Fixed and Rotary wing aircraft (Ajay Joseph, 2017)

Depending on the type of rotary-wing aircraft, they may have the main rotor, tail rotor, and wings to provide lift or their motors distributed symmetrically. If the rotary-wing aircraft is a UAV, then they are generally known as a multirotor or multi-copter UAV. The most common type of UAV is a multi-copter that uses GPS-based navigation systems and autopilot technology to fly without any human intervention. The flight plan is programmed into the system before take-off, and once it takes off, it will follow its pre-programmed route until it reaches its destination point. It then lands itself automatically on a predetermined spot using GPS coordinates or radio control signals from an operator on the ground. The best example is Amazon Prime Air. In 2020, Amazon won FAA (Federal Aviation Authority) approval for a

Prime Air drone delivery fleet. Amazon can deliver goods in 30 minutes or less using drones that follow a delivery route planned by the software in their computer systems(Palmer, 2020).

Multirotor UAVs can be classified by the number of rotors and their configuration. The following configurations are used in multirotor aircraft: A helicopter has two main rotors, each with one or more blades. A rotor blade is an airfoil like the wing on a plane but tilted forward instead of up and down. As the rotor turns, air flowing over the blade creates lift. The air flowing under the blade creates a downward force creating a lift force (MSFC, 2015). Tricopter - An arrangement where three identical sets of two counter-rotating propellers are mounted on each arm of a Y-shaped frame. Quadcopter - Four rotors arranged in a square or rectangle, typically with two in front and two at the rear of the frame. Hex copter - Six rotors arranged in a circle around the vehicle. And finally, Octocopter – Eight rotors (X8) positioned in a circular form (or an octagonal frame) around the vehicle. Quadcopter - The most common arrangement for multirotor aircraft is four rotors, each mounted at 90° to its neighbours. This configuration is known as quadcopters (quadrotors) or X4s when referring to more than one unit used. In recent years, there has been substantial growth in the market for research on UAV applications for multiple purposes. UAVs have been foreshadowed to be a significant aspect of aeronautics in the coming future.

1.3 The miniature quadrotor helicopter

The development of quadrotor unmanned aerial vehicles has been favourable in rectifying conventional aircraft's original issues, such as vertical lift-off. By utilising pairs of counter-spinning rotors, you can negate the need for a tail rotor blade as is used in the average helicopter model (Pretorius and Boje, 2014b).

As is the case of helicopters, the propellers would want to rotate in one direction whilst the engine would naturally need to turn in the opposite direction, demonstrating Newton's 3rd law of motion. However, in the case of quadrotors, the pairs of counter-rotating blades that rotate in opposition to the other pair allow the aircraft to essentially negate all torque, which enables the craft to stay aloft whilst having no uncontrollable rotation. Recent developments in electrical engineering have created more significant components to become smaller yet more powerful. This, in turn, has allowed the size of quadcopters and other drones to become more miniaturised in conjunction with the evolution of the developing technology (Poyi, 2014).

1.4 Problem statement

As a quadcopter is a dynamically unstable vehicle, it relies on various control techniques to function in a stable manner. These control techniques are used to keep the vehicle stable in a hover and provide control authority during flight. In order to achieve a stable hover, the quadcopter requires a control system known as an autopilot. This autopilot must be able to autonomously control the vehicle in all conditions, including in the event of a failure in any of the vehicle's control systems. This in itself is a control system in order to maintain the vehicle in a stable hover in the event of a failure of any of the vehicle's components. The autopilot must maintain stability and control of the vehicle in order to ensure the safety of the vehicle and everything around it during the flight.

The stability and control of a quadcopter are achieved through a variety of control systems and algorithms. Quadcopters rely on their control systems to determine the position of each of their rotors, which are used to control the vehicle through the use of commands. This data is used to establish the vehicle's current attitude, which is used to ensure stability. A quadcopter's attitude is defined as its configuration around the vertical axis, which is the axis that runs right through the centre of the vehicle. The vehicle's attitude can be defined in either in the body frame or in the inertial (Earth) frame. The vehicle's attitude is used to determine the vehicle's position around the vertical axis, which is used to ensure stability through the vehicle's control systems.

Positions are determined relative to the vehicle's reference frame. The reference frame is defined as the frame established through the use of the vehicle's control systems. This reference frame ensures that the vehicle's attitude is expressed in the same frame used by all of the control modules.

The attitude expressed in the vehicle frame is equal to the body's attitude plus the vehicle's roll and pitch angles, which are defined as the vehicle's angular displacement around each of its axes.

All the above needs to be considered for a stable flight. The problem starts with creating an accurate mathematical representation of the quadcopter and then extracting an approximate hardware parameter to be used in the mathematical equations. Next to consider is the frame of references to be used for the derived mathematical model. Once these are fulfilled, a control system can be developed for the stability of the quadcopter.

This research aims to overcome all the pitfalls mentioned above by making a robust model for validation. The purpose of this research also includes - designing and implementing the quadcopter model through various low-cost experiments in order to obtain accurate readings.

1.5 Aim

The project aims to build a quadcopter from the ground up with low-cost components, extract hardware parameters through various experimentation, and further develop a controller and verify the performance of the system using a rapid 3D prototype for visual feedback. This research aims to demonstrate a quick and easy method to build a quadcopter from scratch so that new sensors and algorithms (e.g., camera for computer vision, new flight controllers, filters etc) can be easily tried and tested on a real quadcopter.

1.6 Main objectives

To complete the research project, the following objectives had to be achieved:

- a) Investigate the modelling technique and control theory that will be applied to the quadcopter based on previous work.
- b) Linearise the dynamic model and create a state-space (SS) representation.
- c) Investigate and experiment with different methods to evaluate the aerodynamic and hardware parameters of the quadcopter.
- d) Simulate and verify the results on MATLAB/Simulink.
- e) Develop a rapid prototype 3D model using the popular CAD format for verification and visualisation.
- f) Develop a quadcopter controller model incorporating data fusion/sensor fusion and error correction for a GPS-denied environment. This includes various filters and incorporating a camera for computer vision.
- g) Preliminary tests include deploying the model on the hardware using Atmega328p with Bluetooth communication and basic sensors.
- h) Finally, RPi will be used to add more computationally resourceful sensors for further optimisation and development.

1.7 Contributions

The following is a summary of the unique contributions to knowledge described in this thesis:

3D cad modelling is a powerful tool for visualising results in multiple dimensions. It allows for the creation of 3D models of objects, which can be used to quickly and

accurately visualise results in multiple dimensions. This makes it easier to identify patterns and trends in data, as well as to identify areas of improvement. Additionally, 3D cad modelling can be used to create prototypes of products, which can be used to test and refine designs before they are put into production. Firstly, popular 3D CAD format OBJ and STL files can be used for the visualisation of the developed model on MATLAB. This research diagnoses and troubleshoots outdated OBJ and STL add-ons and compares its difference. The main advantage of this becomes evident during rapid prototyping. Any 3D CAD software can be used to quickly draw up a model and import on into MATLAB/Simulink environment for visual feedback.

Secondly, setting Atmega328p microcontroller output frequencies to 1Khz and 4 output pins for motor calibration. During hardware testing, it was noticed all four motors were not entering calibration mode (audio feedback). Upon investigation, it was noticed the frequency of two pins was 1Khz by default and the other pins were set at 500Hz. This had to be changed using Atmega328p's registers. There are two methods to change PWM on an Atmega328p – Software PWM and Hardware PWM. The SoftwarePWM can be done using a library. However, since this library was fairly new, it contained errors. The Hardware PWM approach was in its infancy. This approach was not used on a low-cost quadcopter at the beginning of this research.

Generally, researchers tend to take approximate values of hardware. This is mainly because measuring instruments such as LCR meters, thrust measurement units etc are expensive. In this research, some of this expensive equipment was used and the results were compared against model approximation and alternative methods to extract hardware parameters.

Full control of each propeller is required for a robust quadcopter model. Atmega microcontroller's PWM timers frequencies are different for different pins. This research demonstrates how to manipulate timer registers to ensure all four motors are running at the correct frequency. The contribution to this research is the implementation of Hardware PWM to set frequencies as desired on an Atmega328p microcontroller.

1.8 Literature Review

1.8.1 History of quadcopter

1.8.1.1 *Bréguet- Richet Gyroplane No.1 (1907)*

On 19th September 1907, Gyroplane No.1 is known to be the first helicopter to have risen vertically from the ground, with men assisting its lift-off at the edge of each of four arms supporting rotors. Since Gyroplane No.1 was assisted by men, it wasn't given credit for being the first helicopter to make a free flight even though the men assisting its lift-off didn't contribute anything towards the lifting power of the rotors. Gyroplane No.1 was built by Louis Breguet and Jacques Breguet in association with Professor Charles Richet (Figure 2-1). The aircraft consisted of rectangular central chassis of steel tubing to support the pilot and the power plant. At the end of each arm, there was a 4-blade biplane rotor. The drawback of Gyroplane No.1 was it was neither steerable nor controllable when airborne (*Bréguet-Richet Gyroplane No.1 helicopter - development history, photos, technical data*, 2010).

Specification of Gyroplane:

Crew: **1**, engine: gasoline engine rated at **29kW**, rotor diameter: **8.0m**, take-off weight: **578kg**, empty weight: **340kg**.

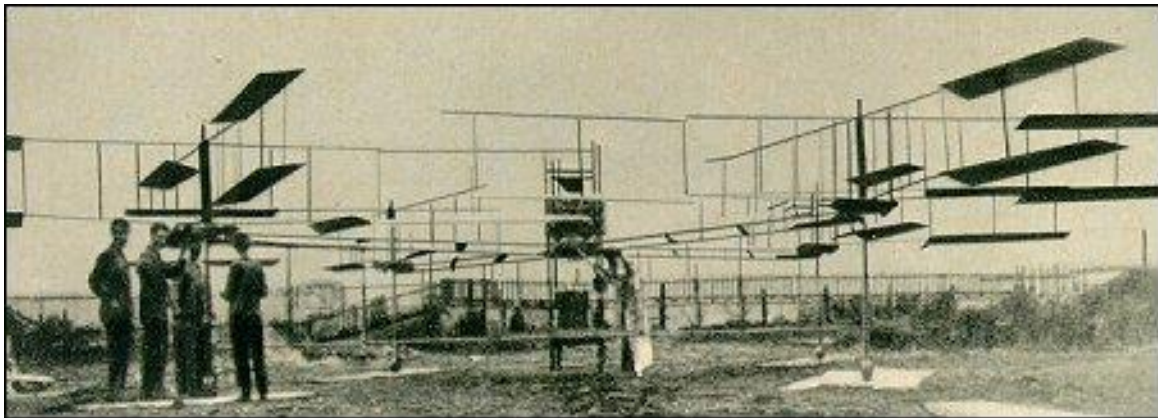


Figure 1-2 - Bréguet- Richet Gyroplane (2010)

1.8.1.2 *Oehmichen (1920)*

In 1920, French engineer Etienne Oehmichen was a staff of the French car company Peugeot. Oechminchen built a quadrotor helicopter that resembled De Bothezat's quadrotor but had eight additional rotors for control, stability, and propulsion. In May 1924, Oehmichen tested a new quadrotor called Oehmichen No.2. It managed to stay airborne for 7 minutes and 40 seconds

and flew 1km at an average speed of 4.9mph. Oehmichen No.2 had eight propellers and four rotors, all driven by a single engine. Federation Aeronautique Internationale (FAI) awarded Oehmichen for demonstrating the ability to complete a circular course with a quadrotor. Oehmichen was one of the first to conduct laboratory studies on the rotor's aerodynamic performance in forward flight and axial flight (*Etienne Oehmichen Biography - Tourism Châlons*, 2011).

1.8.1.3 *Dr George De Bothezat helicopter (1922)*

In January 1921, Dr George De Bothezat and Ivan Jerome were awarded a US Army Air Corporation contract to develop a vertical flying machine. It had an "X" shaped structure and weighed 1678kg. It had an 8.1m diameter six-blade rotor. In order to control yaw and thrust, two small propellers with variable pitch were used at the end of the lateral arms. At the junction of the frames, a tiny lifting rotor was installed, which was situated just above the 180hp Le Rhone radial engine, but later it was deemed unnecessary and removed. The aircraft made its first take-off in October 1922 (Figure 2-2). The aircraft made around 100 flights with an upgraded rotary engine – 220hp Bentley BR-2. Though the contract required the aircraft to hover at 100m, the highest it managed to achieve was about 5m. Despite spending \$200,000, the vehicle was underpowered, sophisticated, unreliable, and unresponsive. In the late 1930s, the Russian scientist De Bothezat built a single seater aircraft, a coaxial helicopter which was even less successful (*Encyclopedia - De Bothezat helicopter - rotary-engined aircraft. The de*, 2020).

Specification for Bothezat helicopter:

Engine: 1 x Rhone rated at **135kW**, rotor diameter: **8.08m**, length and width: **19.8m**, height: **3.05m**, take-off weight: **1700kg**

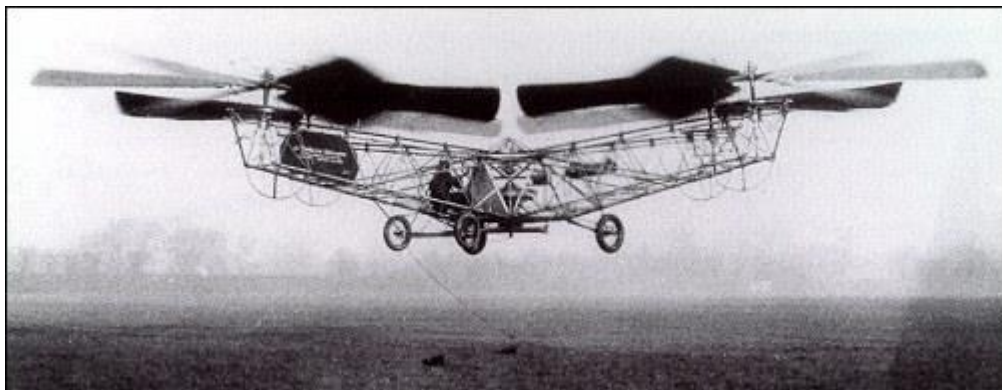


Figure 1-3 - Dr George De Bothezat helicopter (2020)

1.8.1.4 *Convertawings Model a Quadrotor (1956)*

This unique helicopter was intended to be the prototype for a line of much larger civil and military quadrotor helicopters. The design featured two engines driving four rotors through a system of v belts. (Figure 2-3) No tail rotor was needed, and control was obtained by varying the thrust between rotors. Flown successfully many times in the mid-1950s, this helicopter proved the quadrotor design, and it was also the first four-rotor helicopter to demonstrate successful forward flight (*Convertawings Model 'A' Quadrotor at the Cradle of Aviation Museum*, 2020). Due to a lack of orders for commercial or military versions, however, the project was terminated. Convert a wing proposed a Model E that would have a maximum weight of 42,000 lb. (19 t) with a payload of 10,900 lb. (4.9 t) over 300 miles and at up to 173 mph (278 km/h).

Specifications: Rotor Diameter – **19'4"**, Length – **26"**, Engines: Two 90 HP Continental, Top Speed – **80 mph**, Weight – **2200 lbs**



Figure 1-4 - Convertawings Model a Quadrotor (Convertawings Model 'A' Quadrotor at the Cradle of Aviation Museum, 2020)

1.8.1.5 *Curtiss-Wright VZ-7 (1958)*

The Curtiss-Wright VZ-7 was a VTOL aircraft designed by the Curtiss-Wright company for the US Army. The VZ-7 was controlled by changing the thrust of each of the four propellers (Figure 2-4). US Army was given two prototypes of this VTOL aircraft in mid-1958. The pilot's seat, fuel tanks, and flight controls were all located in the fuselage of the VZ-7. The propellers

were fixed, unshrouded, on both sides of the fuselage (the aircraft initially had shrouds, but these were later removed). There was a total of four propellers. The thrust of each propeller was changed to power the VZ-7. The flying platform was easy to control and fly (*Curtiss-Wright VZ-7*, 2015).

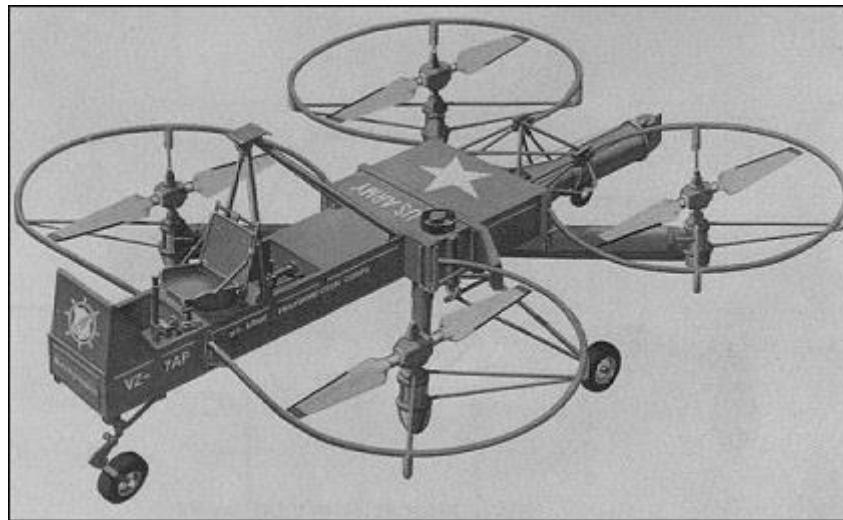


Figure 1-5 - Curtiss-Wright VZ-7 (2015)

1.8.1.6 *Consumer UAV market*

After 1958 there wasn't any significant advancement in quadcopter technology. Some of the technologies/devices were invented and commercialised decades later. BLDC motors were initially called "DC motors with solid state commutation" by T.G Wilson and P.H. Trickey in the 1960s (*Brushless Vs. Brushed Motors*, 2015). Intel Corporation developed a 4-bit microcontroller known as i4004 in 1971 which was bought by BUSICOM, a Japanese company to use in calculators (*History of Microcontrollers | Toshiba Electronic Devices & Storage Corporation | Europe(EMEA)*, 2020). Microelectromechanical systems (MEMS) were initially proposed in the 1960s, however, it wasn't commercialised until the 1980s (*MEMS*, 2017). Akira Yoshino developed the Li-on battery in 1985. Factors like mass production, boost in information technology, and miniaturisation of components amalgamation of these devices played a crucial part in the development of quadcopters.

In 2006 the FAA allowed drones to be flown in civilian airspace following Hurricane Katrina. Predator drones were able to detect the heat signatures of a human being using a thermal camera. Meanwhile, companies like DJI, 3DR, Parrot etc., were repurposing military UAV technology in quadcopters.

Advancement in electronics around 2005-2010 catapulted mass production of cheap, lightweight sensors and flight controllers. This paved the way for Parrot, a French drone manufacturer, to unveil AR drones in 2010 (*A Not-So-Short History of Unmanned Aerial Vehicles (UAV)*, 2020).

In 2013 DJI released its iconic Phantom series with a modern camera system. During the same time, major companies like Uber, Amazon, UPS, Google, FedEx etc., realised the advantages of the drone delivery system.

1.8.2 Previous works on the quadcopters

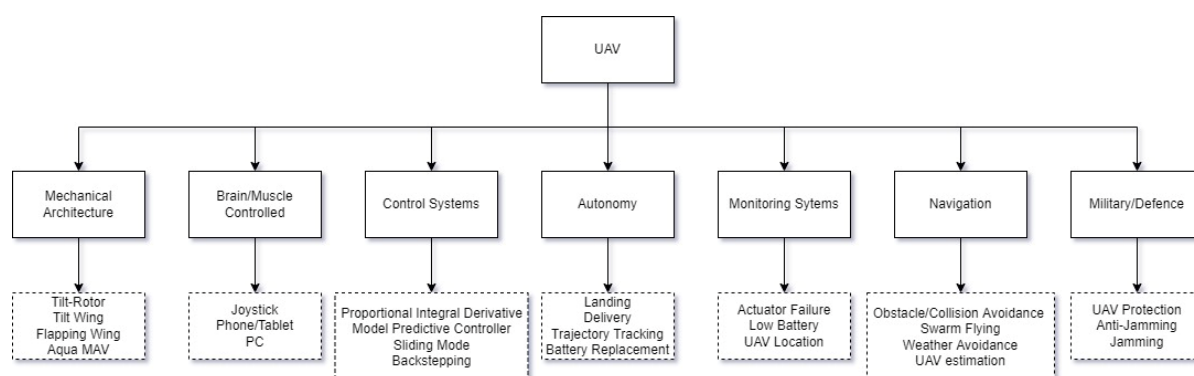


Figure 1-6 - Literature review roadmap

Figure 1-6 above gives a roadmap to this section. Some areas will be covered in detail which isn't highlighted in Figure 1-6 above such as feedback systems, Viccon cameras, different control systems, animation-related research and other sensors. This literature review excludes delving deep into individual subtopics, especially in areas which are less relevant to this research such as mechanical architecture, brain/muscle control, autonomy etc.

The quadcopter industry was in its infancy for many years and hadn't yet become widely popular. Since the early 1900s, there haven't been significant changes in the quadcopter. Researchers have attempted various methods to improve performance throughout the years.

Recently, there have been advancements in quadcopter UAV control technology. The number of initiatives addressing this issue has sharply and dramatically increased. The majority of these endeavours rely on readily available toys (off the shelf), such as the Draganflyer (2014), which was later modified to have more tangible correspondence capabilities. Between 1999 and 2001, mesicopter development started. It intended to investigate the viability of a quadcopter on a centimetre scale (Kroo *et al.*, 2011). The project's primary use was the placement of a massive number of smaller size vehicles that provided climatic and meteorological data over enormous regions. Another noteworthy project is Starmac Hoffmann *et al.* (2004), which focuses on the

display of multi-specialist control of quadcopters weighing about 1 kg. However, none of these frameworks was predicated on a defensible and legitimate method of configuration progression.

Even though there are now many distinct mechanical designs for UAVs, choosing the best UAV to use for a certain purpose can be challenging. The user's individual needs might be met by examining some of the typical categories and elaborating on their attributes. As a result, it is advantageous to examine the performance of widely available architectural designs so that the user may make decisions based on the benefits and drawbacks of each design. The primary feature that sets apart various UAV types depends on the mission capabilities and the operational goals.

UAVs are being created to be controlled by Wi-Fi, whereas historically model aircraft and boats were controlled using RC (radio control). This makes it possible to control the UAV's flight from any device with Wi-Fi capabilities, such as a tablet or smartphone running the necessary program. UAV user forums claim that radio control (RC) is the best way for controlling drones because Wi-Fi control only has a range of about 50 meters whereas RC can provide ranges of up to 18 kilometres (conditions dependent) (*DJI - Official Website*, 2016; *Parrot*, 2016). Additionally, they favour the joystick's ergonomics and tactile feel to the on-screen buttons found on a tablet or smartphone (*FORUM*, 2015). However, Wi-Fi control does offer the opportunity to build drone Wi-Fi networks, allowing users to video stream, swarm technology etc.

In 1995, there were 16 million internet users; in 2019, there were 4.4 billion (or 56.8% of the world's population) (*Internet Growth Statistics 1995 to 2022 - the Global Village Online*, 2022a). In comparison to 35% in 2006, 82% of adults in Great Britain were online daily (or nearly daily) by 2016. The development of Wi-Fi, which has made it feasible for computers, cellphones, and tablets to connect wirelessly to the internet, has contributed to this expansion (*The Economist*, 2017). Wi-Fi has been at the centre of the IoT explosion in addition to providing internet connectivity, as a variety of WiFi devices with built-in sensors have been widely available, allowing things like thermostats, ovens, and refrigerators to be networked and controlled over the internet (Watterson, 2016). A single wireless standard was first realized historically in the middle of the 1980s in order to allow the connecting of network equipment from various vendors. To create a standard that would be accepted by all manufacturers, an Institute of Electrical and Electronic Engineers (IEEE) group led by engineer Victor Hayes was

established in 1988 (‘Victor Hayes | IEEE Computer Society’, no date). On the 2.4GHz frequency band, the standard permitted data transfer rates of one or two megabits per second (Mbit/s) (*The Economist*, 2017). The original 802.11 standards soon underwent modifications. Table 1-1 below describes a few of these more prevalent adjustments (admin, 2017).

Table 1-1 - WiFi History (All About the New Wi-Fi 6 Standard - 802.11ax Explained (Tutorial), no date, p. 21)

Standard Designation	Release Year	Frequency Band	Data Rate (Max)	Approx. Range (Indoor)	Comments
802.11a	1999	5 GHz	54 Mbps	115 ft	Never used commercially
802.11b	1999	2.4 GHz	11 Mbps	115 ft	Actual first useable standard
802.11g	2003	2.4 GHz	54 Mbps	125 ft	
802.11n	2009	2.4 GHz and 5 GHz	600 Mbps	230 ft	First dual-band standard
802.11ac	2013	5 GHz	3.47 Gbps	120-150 ft	All AC routers include also a 2.4Ghz radio (802.11n)
802.11ax	Est. 2019	2.4 GHz and 5 GHz concurrently	14 Gbps	Better than AC	The successor of AC (to be released in 2019)
802.11ad	2012	60 GHz	6.7 Gbps	11 ft	Not widely used
802.11ay	Est.2020	60 GHz	20 Gbps	33 ft	Successor of AD

The manufacturers have benefited from WiFi technology by enabling flight commands to be sent to quadcopters by mobile phones, and tablets. As a result, there is no longer a need for the drone maker to provide a radio transmitter, which lowers expenses. These UAVs are produced by several different companies, including Parrott (Minidrone Spider, AR2), Yuneec (Mantis Q), DJI (Mavic 2 Zoom, Mavic 2 Pro), Skydio (Skydio R1), and Skydio (*Parrot*, 2016) (*Skydio 2+TM and X2TM – Skydio Inc.*, no date) (*Home*, no date) (*DJI - Official Website*, 2016). The ability to build a Wi-Fi drone network is made possible by using Wi-Fi as the method of communication to control UAVs (Harrington, 2019).

The research (Hou *et al.*, 2022) draws attention to fixed-wing aircraft with tailless using interval type-2 fuzzy logic control system (IT2-FLS). The fault-tolerant attitude tracking control problem for a tailless flying wing aircraft is examined in this paper in the context of an external disturbance, unclear parameters, and time-varying actuator failures. An RNN (recurrent neural network) is utilized to solve the general convex nonlinear programming

version of the constrained control allocation issue in order to arrive at the desired control torque allocation optimal solution. A concurrent analysis of the tailless aircraft system's closed-loop stability demonstrates that all of its signals are essentially fixed-time stable. To show how the suggested fixed-time IT2 fuzzy fault-tolerant control scheme and the constraint control allocation scheme are superior to each other, two representative contrast simulations have been shown.

The brain-piloting drone interface has been a project for some years for the British tech company Ultra Electronics, which claims to have finally advanced the idea to the prototype stage. Despite the futuristic and exciting sound of controlling a UAV with simple command thoughts, a demonstration video of the method may not have the same impact on viewers' hearts as the mind-elevated drone. In fact, it resembles someone who has been gazing intently at a computer screen for a prolonged period (Crumley, 2021). Generally, muscle-controlled or brain-controlled UAVs are widely used by hobbyists. This research takes the concept to another new level. Since a few years ago, Ultra Electronics has shown the technology and used it to manage a variety of electrical devices, including high-tech home appliances and defence equipment. It is unknown whether the military or police services have chosen the brain-controlling prototype at this stage of its development for practical usage - on UAVs or anything else. However, it is safe to predict that as the business develops more, it will prove to be genuinely amazing. Although a lot of details on how this system is implemented are scarce. The study suggests they have created tags/icons, which have a signal within them and human eyes respond to these signals in a particular manner. This signal is processed by observing the visual cortex of the brain and turning these signals into a command. Interestingly, the laptop doesn't include a webcam that can gaze at the user. Through the brain sensor worn on the head, the human brain is what instructs the UAV (*New drone technology unveiled, your mind can now instruct a drone to fly* | WION, 2021).

In the study (Yıldırım, Çabuk and Bakırcıoğlu, 2019) Octocopter and Dodecacopter are put through five different working conditions for comparisons, including no disturbance, periodic disturbance, and non-periodic disturbance. The root mean square of the location errors of the Octocopter and Dodecacopter systems increased by 69.7% and 47.6%, respectively, in the simulations when the amplitude was increased by 100% under the periodic disrupting effect. Similar to periodic disturbances, both systems showed an increase of 13% and 7% for non-periodic disturbances. The Octocopter system is partially more stable without a disturbing effect, but as the disturbing effect rises, the Dodecacopter system flies more steadily than the

octocopter system. According to the drone system's trajectory results, the typical PID controller performs satisfactorily but eventually experiences high-level disruptions.

The paper by Kumar *et al.* (2020) states that quaternion state feedback computes UAV motor control instructions without the gimbal lock problem of Euler angle-based controllers. Without trigonometric parameters, the quaternion implementation simplifies state estimation. The UAV attitude controller and control allocation use the quadcopter dynamic model and state space. Taylor series expansion based on hovering determined system control allocation. Quaternion feedback attitude controllers worked well for reference attitude tracking. An external position controller was used to control the inner quaternion feedback loop for way-point missions.

The following topics have recently attracted a lot of attention in drone technology: nonlinear tracking and detection of small unmanned aerial vehicles and micro-drone targets. To handle the nonlinear models, the Cubature Kalman-multi-Bernoulli filter, which uses a third-degree spherical-radical cubature rule, has recently been introduced (Barbary and ElAzeem, 2021). In terms of mathematics, the Cubature Kalman filter is more logical and precise. Additionally, a recent variational Bayesian approximation-based multi-Bernoulli filter has been provided with the estimation of fluctuation of measurement variances. The research suggests due to the unknown likelihood of detection, Cubature Kalman and variational Bayesian-Multi-Bernoulli filters are not suited for monitoring the micro-drones. As is well known, monitoring small objects was made possible by track-before-detect (TBD) techniques. In an effort to estimate the fluctuating measurement variances, a novel robust Cubature Kalman-Multi-Bernoulli filter with variational Bayesian-TBD is presented. An enhanced filter is a useful tool for addressing the estimation of the detection profile for micro-drones. To estimate a hybrid kinematic state of micro-drones, a unique implementation using a non-linear Cubature Kalman Gaussian mixture and Inverse Gamma approximation is provided. The effectiveness and reliability of the suggested algorithm are confirmed by the simulation results.

The paper by Dr Mark Petovello (2003) explains the difference between the Kalman filter and the Least-squares model. The paper explains the Least-square relies on minimising measurement residuals, or the difference between actual and predicted measurements. In contrast, the Kalman filter is generated by minimising the solution's mean-square error. The author gives a good example regarding a situation which could arise in an urban canyon. In an urban canyon, the Kalman filter predicts forward velocity if a receiver can only track two satellites. If the received turns a corner, the filter may take several seconds to detect it, causing

the trajectory and corner to overshoot. By extension, generating a least-squares estimate at every epoch—although noisier and less accurate—provides insight into the "raw" data quality, without filtering. This information can help choose a Kalman filter parameter.

Similarly, the Double Exponential Smoothing Based Prediction (DESP) can be a viable alternative to the Kalman filter for predictive tracking algorithm development. They are widely used in business and economic forecasting. The paper carries out tests and claims to produce experimental results 135 times faster than Kalman and Extended Kalman filter-based predictions with derivative-free measurement models (LaViola, 2003). This paper is mainly focused on the Virtual Reality (VR) and Augmented Reality (AR) predictive models. However, (Chief, 2016) addresses that exponential smoothing creates forecasts that lag behind the actual trend. The author's research is based on weather forecasts and proceeds to explain that the delay is a result of the process of smoothing. This technique is called "smoothing" since it disregards random variation's ups and downs. Consequently, this appears as a smoother line or curve on a graph. But avoiding random variation also enables you to observe the underlying phenomenon, which is useful when presenting data and predicting future values. Since this predictor cannot handle trends well, exponential smoothing works best for short-term, non-seasonal forecasts. Thus, cyclical or seasonal data impairs projections. If the series has a pattern, this averaging won't work. This method only works when past and future continuity is anticipated. Since it believes existing trends will continue, it's excellent for short-term forecasting. As the forecast progresses, this assumption becomes problematic.

In 2007, Bousabdallah and Siegwart (2007) modelled the aerodynamics of quadcopters by using momentum and blade element theory. To control position and attitude, they used a backstepping and integral controller. The results they obtained had a 3cm to 20cm maximum reference deviation for attitude and position. This research implements momentum and blade element theory to estimate the forces.

Hoffmann et al. (2022) address concern encountered when deviating significantly from hover flight command of vertical take-off and landing (VTOL) quadcopter. This study presents several aerodynamic effects encountered by the quadcopter whilst introducing wind disturbances at high-speed manoeuvres. The results were validated through flight tests and using a thrust measuring stand.

A variety of sliding mode controllers (Islam *et al.*, 2015) and adaptive backstepping controllers (Satici, Poonawala and Spong, 2013), (Huang *et al.*, 2010) have also been developed for

quadrotors. In Li and Sun (2022), an adaptive integral backstepping controller surpassed both a nonadaptive and a PID controller in terms of performance. According to Finite-time Adaptive Integral Backstepping Fast Terminal Sliding Mode Control Application on Quadrotor UAV, (2021), an adaptive backstepping controller, an integral backstepping fast terminal mode controller, and a nonadaptive backstepping controller all performed better than each other. In contrast to PD, backstepping, boundary layer sliding mode, and active disturbance rejection controllers, Zhang *et al.* (2019) create an adaptive fuzzy global sliding mode controller.

Another quadcopter architecture of this kind was designed and produced by Şenkul and Altuğ (2013). The test findings for these strategies, however, do not show any intrinsic behaviour regarding hovering stability. The ability of tilt-rotor quadcopters to hover while maintaining a consistent pitch and/or roll angle has recently drawn the attention of several academics. However, just a little research on these arrangements has been done.

In 2015, Powers *et al.* (2015) attempted to develop controllers in two different ways. Firstly, by linearizing the dynamic model and secondly test the controller on a non-linear dynamic model. They used a simple PID controller for position and attitude control, contrary to Bouabdallah and Siegwart's research. They showed simplified motor dynamics by taking an approximation of parameters and implementing continuous-time LQR. They have also developed nonlinear controllers. They used the controllers to test fly a nano quadcopter, given the maximum acceleration of $0.008 \frac{m}{s^2}$ and to manoeuvre in a circular path of a radius of half a meter. They recorded an error of 10cm on the horizontal plane, and the vertical plane had an error of 1cm.

The study carried out by Dong *et al.* (2013) and Bolandi *et al.* (2013) suggest using the control technique known as single input single output (SISO). This was implemented by decoupling around each axis and by linearizing the dynamical model.

The PID controller designed by Dong *et al.* (2013) used Ziegler-Nichol's technique and root locus analysis. Although Zeigler-Nichol's technique does not work in all the systems, Dong *et al.* achieved a 50cm maximum positional error, while carrying out outdoor trajectory tracking flight tests. However, no information was available regarding velocity tracking.

The objective of the work carried out by Oktay and Eraslan (2020) was to investigate the link between airspeed and the propeller thrust coefficient using a computational fluid dynamics

(CFD) analysis of a propeller from a quadcopter UAV. In order to achieve this goal, the computational domain was built with two distinct regions: the far field and the rotating zone. Incompressible and turbulent flow assumptions have been utilised in conjunction with the overset mesh approach. Analysis was carried out with the assistance of a Navier-Stokes solver (Ansys Fluent v17.2) making use of the k-Omega SST turbulence model and incorporating curvature correction. According to the findings of the research, the thrust coefficient fell as the airspeed increased, as was to be expected. In terms of the thrust coefficient, it has been discovered that CFD findings follow trends that are comparable to those reported in experimental results. In addition, it was found that the predictions of numerical analysis got closer to the findings of experiments as the airspeeds were lower. Ansys Fluent v17.2 CFD simulations have been performed at 5000 rpm rotational speed and 2.4 m/s, 3 m/s, and 3.6 m/s airspeeds on an 11x4.7cm propeller. All simulations converged with a residual order of 10^{-4} or better after a large enough number of iterations (more than 1000). The obtained thrust force values were converted to unitless thrust coefficients, and they were found to be consistent with the experimental results found in the literature by Czyz *et al.* (2021).

Ortega *et al.* (2020) suggest using the 3D CAD model of the structural components of a power plant to design an offline flight path for unmanned aerial vehicles, using inspiration from synthetic vision systems for aerial vehicles that use 3D space to give pilots a clear and intuitive way to perceive their flying environment (UAVs). This method is used because CAD models of structural elements are easy to get these days. Since the research is about checking the structure, only the surface features of the model are needed. To get the 3D model, analyse it, and work with it, first the STL file is made to get surface points and vectors that are parallel to the surface. Later, the flight path is made by adding other variables like wall offset and controlled trajectory between each outline and each successive layer. This method would help when the UAV needs to fly without a GPS or when the pilot can't see it (BLOS – Beyond Visual Line of Sight). The work was done with the help of MATLAB software and using the Gazebo simulator, which uses strong kinematic and dynamic models, and the final path was checked. MATLAB algorithms generate trajectories from CAD models in this system. The algorithm uses AM (Additive Manufacturing) slicers. AM slicers are used. Like AM, the CAD model must be converted to an STL file with only surface information. The model geometry is triangular and STL has triangular normal vectors. MATLAB sorts the STL file into matrices and slices it before calculating a trajectory. Slicers establish AM technology layer thickness. Hardware elements like camera view angle and overlap percentage were used to map the

camera feed and determine slicer height, or layer thickness. A few lines of code can calibrate the system to different values. Slice data becomes UAV trajectory. The inspection was carried out with Intel(R) R200 cameras. Hardware required 300-mm wall offset and slicer height. Scaling the model to 1x104 m and back to metres avoided MATLAB rounding difficulties.

This work carried out by Haque (2021) aims to create a fast, accurate, and resource-efficient way for UAVs to derive their present pose in CAD models. The system receives the goal position and a 3-D CAD model of the building. The UAV initially doesn't know its building location. Using a Simultaneous Localization and Mapping (SLAM) technique, the stereo camera system and IMU create a globally consistent map of its surroundings. It stores spatially associated 3D features and the map. The correspondences, generated by SLAM technology are used to construct a transformation between the SLAM map and the 3D CAD model, localising the UAV. Their approach localises the UAV in the test building in an average of 15 seconds depending on the abundance of target features in the observed data. Without a motion capture system, the results were confirmed by placing tags on the ground at strategically known places in the building and quantifying the error in the projection of the current UAV location on the ground with the tag. Localizing the UAV in CAD requires five major procedures. The first phase is pre-processing, which extracts CAD model features and computes feature descriptors. During UAV operation, the system maps its surroundings and localises within it. SLAM algorithms do this. Third, an object detection real-time CNN extracts known features from the surroundings. These features are SLAM-registered. Fourthly, feature descriptors are calculated from extracted features. Finally, feature descriptors from the CAD model and observed data are matched to compute a rigid transformation. The transformation localises the UAV in the CAD model as the SLAM map already localises it. SLAMdunk module on Parrot Bebop 2 and the ground control station make up the hardware system. The NVIDIA Jetson TK1 powers the SLAMdunk, which has a stereo camera rig, IMU, front-facing ultrasound rig, barometer, magnetometer, and IMU.

A commendable list of contributions to quadcopter research was made by Castillo et al. in 2011 (Guerrero Castellanos *et al.*, 2014) (Berna *et al.*, 2011) on real-time operating systems (RTOS) with nested saturations. A non-linear control law based on saturation function for quadcopter hovering conditions was demonstrated in this study. This was in conjunction with a previous study by Sanahuja et al. (2010). They use the saturation function in each state to stabilise n integrators in a cascade with bounded inputs. The results suggest that the quadcopter helicopter has maintained stability even during abrupt changes made to the system.

It is crucial to have uncorrupted data from sensors fed back to the system to achieve stability during flight manoeuvres. The in-house testing and verification of a system become difficult when relying on GPS systems due to their lack of reception within buildings. To cover this hindrance, researchers at the Georgia Institute of Technology and MIT adopted a VICON MX camera system for indoor multi-vehicle flight testbeds (Valenti *et al.*, 2006). The VICON MX camera system detects the vehicle's orientation and position in real-time. Thanks to Tarus software which can track reflective markers attached to the vehicle and calculate attitude and position data up to 120Hz with a 10ms delay. The data is received at the ground control station via TCP/IP packets.

Meanwhile, Stanford University used a multivehicle testbed for X4-Flyer quadcopters (Waslander *et al.*, 2005). The study shows classical linear techniques failed to provide sufficient stability, so they implemented reinforcement learning control and integral sliding mode technique in scenarios with nonlinear disturbance. Both techniques resulted in significantly improved performance and similar controller response time. The biggest drawback is that these camera setups are not very cheap.

An alternative to this is sensor filtering and sensor fusion technique. Cheap sensors pick up noise during operation, and this impedes obtaining accurate data. The complementary filter is the most straightforward sensor filtering technique to implement. The paper by Kokaj (2018) uses Fast Fourier Transform and a complimentary filter to evaluate signals from sensors on their self-balancing robot (Kokaj, Bula and Dermaku, 2018). They use a cascading model of complimentary filters to filter the signal's illogical data. The complementary filter model consists of high and low-pass filters, which are tuned individually. Another study by Andrien *et al.* (2018) on complementary filters uses the similarity-based approach in which a novel adaptation scheme based on similarities between vector and gyroscopic measurements is developed. The study describes the rationale behind this approach: over time, the estimation of attitude should depend on vector measurement. The effectiveness of this approach is tested on a non-linear quadcopter model.

A more robust approach to the sensor fusion technique is using Kalman Filter. This is a powerful tool, particularly if the system of the model is linear. The paper suggests that advanced filters like particle filter are computationally demanding and requires a powerful processor. Sebesta and Boizot (2014), Beck *et al.* (2016), and Hetenyi *et al.* (2016) conduct sensor filters and fusion using only the Kalman filter. Hetenyi *et al.* study show that attitude is

determined every 100ms by using 10Hz data from an ultrasonic sensor and accelerometer data at 100Hz.

1.9 Applications of miniature quadrotor UAVs

In the modern era, the loss of human life is incredibly frowned upon, especially if the loss of life came about from undertaking a dangerous task. The aviation industry has lost many lives during its early years; however, due to recent technological advancements, unmanned aerial vehicles are becoming more commonplace. This has allowed for tasks deemed far too risky to be efficiently completed without fearing the loss of human lives.

The quadcopter helicopter is the most popular and valuable flying machine of our age. This device has a wide range of applications in different fields like military, commercial, and civilian sectors.

1.9.1 Military Applications

In military applications, they are primarily used for scanning various objects or assessing situations from 1000 feet (Gandhi and Ghosal, 2018). The ability to vertically take off and land (VOTL) within a small area makes quadcopters especially useful in combat situations. It works effectively under different conditions so that one does not need extra help or a companion during an operation which makes them more cost-effective than other flying devices like helicopters, planes etc. PC-1 quadcopter (shown Figure 1-2 below) manufactured by Ukrspecsystems currently serves in the Armed Forces of Ukraine is a testament to the military application(no date a).



Figure 1-7 - Military application quadcopter(no date b)

In the commercial sector, the quadcopter helicopter serves various applications such as security surveillance over large areas because they are small-sized compared to other aircraft like planes making them easily portable across different places, thus enabling their efficient usage at any time.

1.9.2 Surveillance

The quadcopters have been used as surveillance tools by law enforcement agencies and private investigators alike. They are small enough to fly into tight spaces, stable enough to hover over an area without being noticed, yet responsive enough that they can quickly change position when needed. Most miniature quadcopters possess infrared cameras so that they can record footage in low-light-level environments.

Unlike human soldiers who need to manually walk or drive to keep watch on countries' borders, quadcopters can fly along the border within a much shorter time frame to recon the areas. This would especially be useful in more remote regions. These UAVs are usually fitted with high-quality cameras, which increase the efficiency at which the border is patrolled due to the drone being able to capture a much wider area than what a human can achieve on the ground whilst also being able to identify objects with high accuracy from high altitudes (Blazakis, 2006).

1.9.3 Racing

Although not all racing drones are designed to fit inside the definition of a "miniature" UAV (i.e., those weighing under 200g), there is a growing interest in them predominantly due to their speed and agility on short courses like football fields or basketball courts. Because they fly relatively low off the ground, racers must always pay attention to avoid crashing into obstacles during the competition. The best miniature quadcopter helicopters come with multiple channel controls, which allow users more control over each individual rotor blade's pitch angle than would otherwise be possible using just one or two channels; thus, making these devices ideal for competitive fliers who demand precision from their equipment while still retaining its manoeuvrability. A typical miniature drone is shown in Figure 1-3 below.



Figure 1-8 - Racing drone(Free Image on Pixabay - Drone, Reverb, Racing, Quadcopter, no date)

1.9.4 Wildfire detection

Regarding wildfire detection, quadcopters can be tasked with surveying large areas of vegetation such as forests and can reach many hard-to-reach places. They can be fitted with infrared sensors to quickly detect rising temperatures indicative of fires. Due to the quick surveillance time and ability to monitor a more expansive area whilst collecting highly accurate information about the location of a potential wildfire, a faster response time can be achieved for the fire brigade. This in turn can minimise damage to the area in question as the brigade can more quickly reach the place and resolve the issue (Kyuchukova *et al.*, 2019).

1.9.5 Photography

Miniature quadcopters have increased in demand, especially in the photography industry. They have been primarily used for aerial image capture and scanning of locations to produce 3-dimensional models. These have been found to be cheaper than using other technologies such as satellite imaging (Ruwaimana *et al.*, 2018).

Many other uses can be achieved with quadcopters, such as streaming or security on private land.

Some miniature quadcopters feature HD video cameras which allow users to record and save the footage for later enjoyment. This is a fun activity that anyone can enjoy, regardless of their skill level. Miniature copter helicopters are also available with motion sensors that automatically take still photos when the device moves in the air (most often employed as part of an aerial photography hobby).

1.9.6 Research

The use of quadcopters has also increased in terms of research capabilities. They can be used for many data acquisition missions in diverse fields such as archaeology, geology, atmospheric etc., so long as the correct data acquisition components are fitted. Due to miniature quadcopters being relatively cheap and being available in many different configurations to suit the need of the researcher, they are used widely [14].

1.9.7 Agricultural and Industrial applications

The agricultural and industrial fields have also started to incorporate more quadcopters into their work process. Such uses of quadcopters include crop spraying, factory surveillance etc. Industries can use quadcopters to keep maintenance on structures such as towers and buildings to ensure working order is maintained. Farms can also utilise quadcopters to take images of their crop and livestock so that they can adjust the variables, i.e. water, fertiliser, and pesticides, to increase crop yield (Ashour *et al.*, 2016) (Mogili and Deepak, 2018) (Veroustraete, 2015).

1.9.8 Disaster Management/ Search and Rescue

The world undergoes many natural and man-made disasters every year. Some examples include earthquakes, tsunamis or war. During these times of crisis, disaster management alongside search and rescue teams would try to work quickly to minimise the damage and save as many people as they can that could be trapped or fatally injured. The main obstacle the teams face when confronting these disasters is safely traversing the area in a limited time frame. In

an emergency scenario such as this, people will need to be found as early as possible to ensure that the chances of survival stay high (Macintyre, Barbera and Smith, 2006).

In recent years, quadcopters have been trailed to see how well they fare in search and rescue operations with auspicious results (Loughran, 2020). The unsuitability of manned aircraft for such missions have always been an issue since the dawn of aviation. A quadcopter may be an extremely effective solution as they can easily navigate through dangerous areas and can quickly recon the area to help quickly identify where trapped individuals or injured people are situated without the risk of human life. Quadcopters also have the advantage of being highly manoeuvrable, allowing them to navigate tight closed spaces created by disasters. During the flood in Kerala, India, in 2018, groups of quadcopter enthusiasts rose to the occasion towards relief operations (M, 2019).

1.9.9 License and regulations of quadcopters within the UK

Like any other aircraft, an unmanned aircraft must always be flown in a safe manner, both with respect to other aircraft in the air and people and properties on the ground (Ajay Joseph, 2017).

The Eighth Edition of the UAS regulation package was released on 31st December 2020. This edition has divided UAV drones into three categories and five classes. Figure 1-4 and Figure 1-5 represent flight categories. The three categories are Open, Specific and Certified. This section will focus only on the "Open" category since researchers, hobbyists, and some commercial users fall under this category. The delegated regulation covered in CAP2012 – Requirements for Flying In The Open Category are summarised in this section (*List of Unmanned Aircraft*, no date). Note the term "Drone" is used throughout in-cooperating different types of drones such as quadcopters, hexacopter, octocopters etc.

The drones are classified based on their physical size, weight, and speed until the end of the transitional period (1st January 2023). Any drones sold after this period should be classified into five categories (C0 to C4). Until the end of the translation period, drones will be known as "legacy" models (*UK Drone Laws: An In-depth Guide*, no date a).

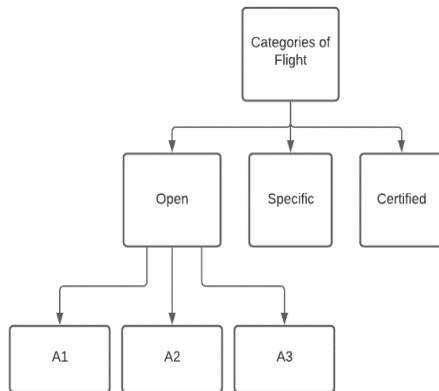


Figure 1-9 - Flight categories

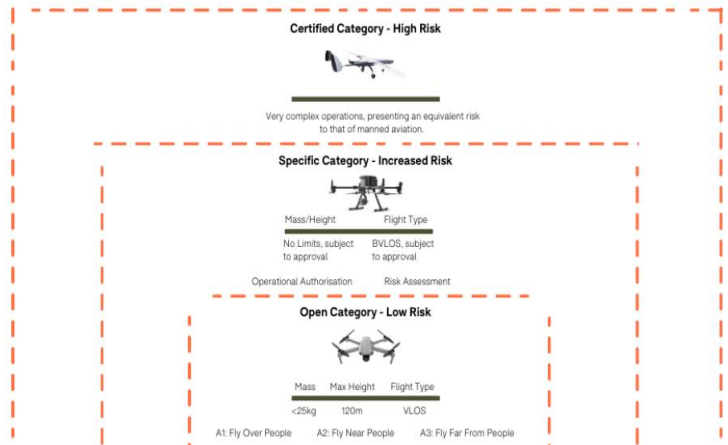


Figure 1-10 – Summary of drone laws in the UK (UK Drone Laws: An In-depth Guide, no date b)

1.9.10 Open Category

The open category consists of subcategories **A1**, **A2** and **A3**. These categories stipulate what drones can be flown alongside, and how and where they are allowed to use.

1.9.10.1 A1 – Fly Over People

The A1 category poses the least risk because of its size, speed and weight. These drones can be operated almost everywhere except over assemblies of people or restricted areas (Figure 1-6). Examples of such drones are Parrot Mambo and DJI Mavic Mini. A1 subcategory drones must be less than 250g in weight (*What are the requirements under the subcategories of the ‘open’ category?*, no date a).



Figure 1-11 - Open – (Subcategory A1)

1.9.10.2 A2 – Fly Near People

The A2 subcategory drones weigh up to 4kg, and they are C2-rated. These drones are rated to fly up to 30m. Legacy drones (currently on the market) can fly up to 50m. However, they should weigh less than 2kg (Figure 1-7).

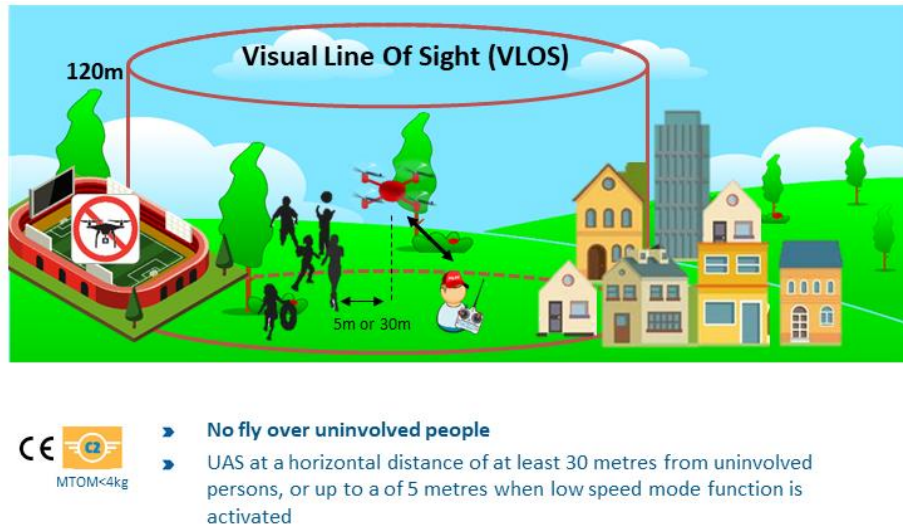


Figure 1-12 - Open (Subcategory - A2)

1.9.10.3 A3 – Fly Far from People

A3 subcategory drones cannot be operated in urban environments and should be at least 150m away from commercial, industrial, and residential areas (see Figure 1-8). These subcategory drones can weigh up to 25kg (*What are the requirements under the subcategories of the ‘open’ category?, no date b*).

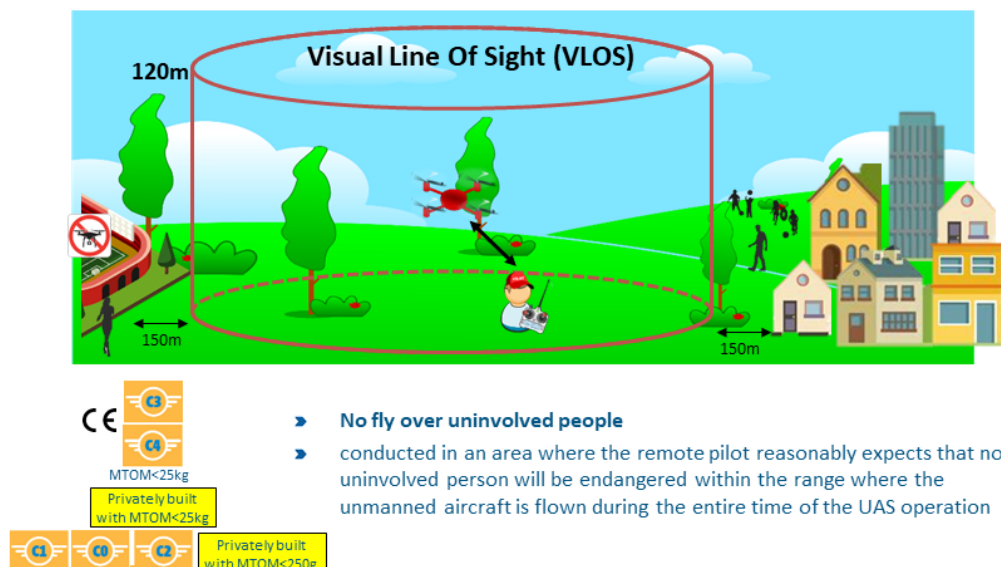


Figure 1-13 - Open (Subcategory - A3)

Table 1-1 Illustrates a matrix representation of classes and subcategories in effect from 1st January 2021.

Table 1-2 - Drone Regulations and Classifications

Class	Mass with payload	Subcategory	Area of Operation	Registration	Competency	Age requirement
C0						
	<250g		Fly over uninvolved people except for crowds	Only if the camera included		Minimum 12 with parental supervision, else 16
C1					User manual	
		A1 - Fly Over People			User manual	
	<900g		No intentional flight over uninvolved people	Yes	Online training and test	
C2						
	<2kg					
		A2 - Fly Near People	More than 50m from uninvolved people			
			Distance of 30m or more else 5m with low-speed mode	Yes		
	<4kg					
C3					User manual	
					Online training and test	16 or above
					A2 CofC	
C4	<25kg	A3 - Fly Far People	Minimum distance 150m	Yes		

1.10 Anatomy of the Quad-rotor Helicopter

A quadrotor helicopter is a type of rotary-wing aircraft that utilises four attached rotors. However, the simple design allows the aircraft to manoeuvre itself with six degrees of freedom (Naidoo, Stopforth and Bright, 2011).

Due to the multiple-rotor nature of the quadcopter, unless a sophisticated control module is incorporated, the craft can easily and quickly become unstable and very burdensome to control. The drone can manipulate its yaw and pitch by adjusting its rotors' rotation speed accordingly. This can be done via a radio frequency connection with a controller or by the drone itself utilising its own pre-programmed software.

The main elements of a typical quadcopter are shown in Figure 1-9 below.

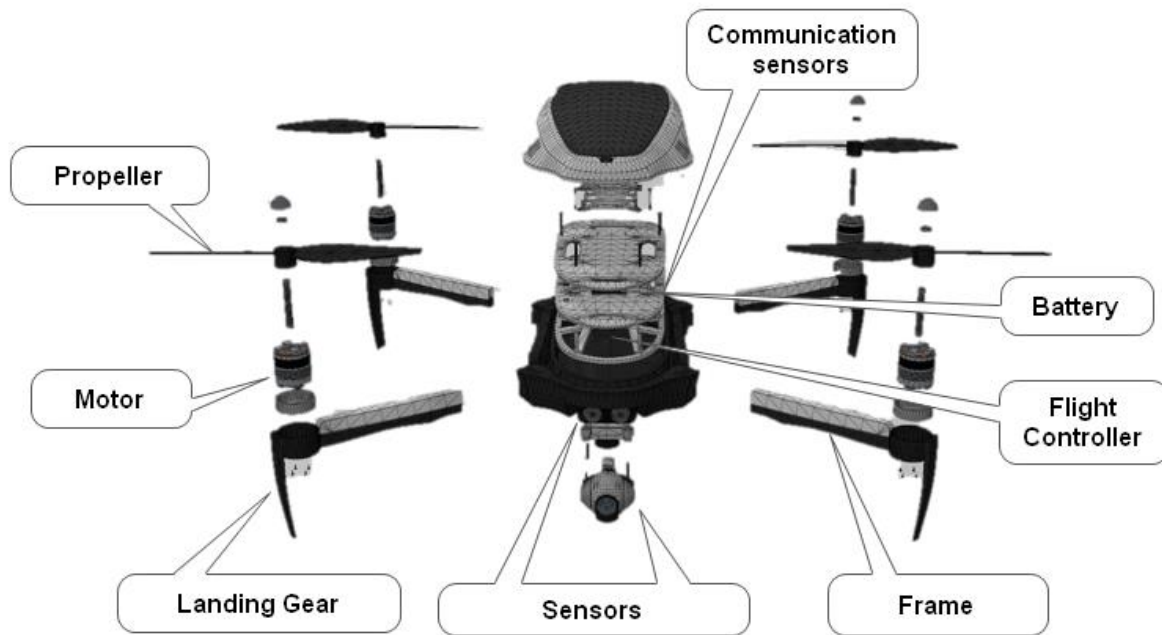


Figure 1-14 - Anatomy of a quadcopter (Ajay Joseph, 2017)

1.10.1 Frame

There are many different types of quadcopter frames. They vary in size and materials used, but the main thing they have in common is that they all provide a mounting platform for the components that make up the quadcopter. The frame is what holds everything together and gives it its shape. The frame will also determine how much payload you can carry, which is crucial if you want a gimbal or other accessories (Naidoo, Stopforth and Bright, 2011).

Most frames comprise carbon composite parts or cheaper materials such as aluminium which are both strong yet light whilst also being easy to repair or replace (Martinetti, Margaryan and Dongen, 2018). More sophisticated drones can be fitted with more complex components to suit the task required, such as an attachment of a camera.

Most quadcopter frames come in a cross-frame design with the propeller blade motors attached to each of the four arms. The main electrical components, such as the battery and flight control board, are located within a housing unit at the frame's centre. Figure Figure 1-10 below represents the most common configuration of quadcopters. X frames have an advantage over + configuration, especially if a camera is mounted on the quadcopter. A camera on top will usually impede its vision if a "+" configuration is adopted. The standard type of frame configuration is illustrated in Figure 1-10 below.

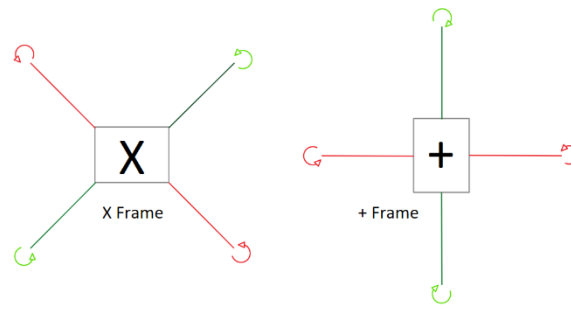


Figure 1-15 - Types of frame configuration (x and +)

Landing gears help support quadcopters during take-offs and landings. Shock absorption is essential when it comes to landing, and hence, landing gears can be flexible yet sturdy at the same time to avoid that. Choosing the landing gear needs to come from considering the gears' rigidity to withstand a forceful landing entry yet light enough that the overall weight does not strain the rotors so much that lift cannot be achieved. Most landing gears are static components and do not move ('Anatomy-of-a-Drone0415.pdf', no date).

1.10.2 Motors and Propellers

Quadcopters usually use brushless motors to rotate the propeller blades. The brushless rotors involved in a quadcopter are incorporated due to their efficiency in generating more torque, less noise and an increased lifetime (Javir *et al.*, 2015). The only downside to brushless dc motors is their high price cost. Each motor is controlled individually by an electronic speed controller that relays information from the flight control board to control the speed at which each blade should rotate, resulting in the desired movement of the quadcopter to occur. The blade's size is directly proportional to the amount of lift that it can achieve and how much power it draws to provide that lift (Luukkonen, no date).

1.10.3 Battery

Lithium Polymer (LiPo) batteries are most commonly used in the majority of quadrotor helicopters. These batteries are known to be lightweight with a high capacity and discharge whilst also being able to be recharged multiple times. This is useful as quadrotor copters are known for their need to draw large amounts of energy in a short period of time (Šustek and Úředníček, 2018).

1.10.4 Sensors

Quadcopters require sensors to establish a connection with the real-world physics it's in and to be able to interact with it accordingly. Most drones incorporate: accelerometers, gyroscopes,

cameras and GPS receivers. They all work in conjunction with one another to send feedback data to the drone's flight control board, which allows it to stay in equilibrium when in flight and allows it to correct its pitch, yaw and roll needed to avoid a failure resulting in a crash (Milionis, no date).

1.10.5 Flight Control Board

A flight control board is a crucial component that processes input data from the sensors and sends information to the appropriate output elements for the needs of the drone to keep itself airborne and stable. It is imperative the flight control board is always functional, as, without it, the rotor-wing craft will not be able to keep itself stable in the air as it requires accurate and precise control of its rotors to stay aloft. The flight control board is also the command unit that follows the user's input and relays the information to the rotors to control its direction, yaw, and pitch ('61806036.pdf', no date) (McBride, no date).

1.11 Chapter Summary

The technology to develop and produce UAVs was initially costly and not readily available to the general public. However, there has been an increasing demand for UAVs among military and civilians over the past few years. This has led to a reduced cost of UAVs in the market. This has happened because the technology to produce UAVs has become more widespread, and manufacturers have started to use cheaper materials to create UAVs. This has led to a reduction in the cost of UAVs. It is hard to imagine that UAVs can fly as high as the current commercial aircraft fly, which will give more flexibility than aircraft in the market today. The key areas where quadcopters will become beneficial are described in Chapter 1. The primary advantage is vertical take-off and landing (VTOL). The drone industry is expected to create hundreds of thousands of jobs in the next 10 years. As a result of technological advances, more and more public agencies and higher education institutions are considering implementing drone flight programs.

The Civil Aviation Authority has just released new UK drone regulations in January 2021. While it is not intended to be comprehensive, this chapter attempts to provide a useful overview of some critical points. The regulations' basic requirements are: A drone pilot must be 18 years or older and have a valid UK-issued photo ID; in addition to this, they must not have any criminal convictions for illegal drone use. A drone pilot cannot carry out any commercial drone operations unless they are a licensed commercial drone operator. Ultimately, this chapter described the basic applications and anatomy of a quadcopter.

This chapter explains the history of quadcopter helicopters. Eventually, advancements in technology made a significant impact on quadcopter research and development. The literature review describes how researchers have contributed to developing quadcopters. It also explains what techniques they have used and the result of their study, wherever possible. The content also consists of research results, justification and benefits of particular methodologies and outcomes. Some of the states of the art sensors found in the literature review are expensive and beyond the scope of this research. However, this research is primarily focused on rapid prototyping. Some researchers have used Atmega328p to prove the working principle and motivate others for studies. A similar approach is considered in this research paper where complicated terms and equations are simplified for the reader to follow.

Chapter 2 Basic Concept and Rigid Body Dynamics

This chapter describes the conventions of rigid body rotation and translation. It gives an overview of how quadcopters work, notations being implemented, and the conventional right-hand rule, which defines the direction of movement/rotation along an axis. The chapter highlights the phenomenon called singularity, where a rigid body's forces and moments do not affect the dynamics of the rigid body system's generalised coordinates. It describes the derivation of rotational matrices. It explains why the sequence of rotation is important and how it affects the transfer matrix.

2.1 Basic concepts of the quadrotor

A quadcopter consists of four motors, two on the left and two on the right. As with a conventional helicopter design, rotors are attached to a central hub connected via shafts to the motors at each corner. However, for a quadcopter, instead of having just one rotor spinning clockwise or counter-clockwise on an axis through its vertical centre point, as in most traditional helicopters, that power control is separated into each motor with no direct connection between them. Essentially each of the four shafts is like a standard rotary engine running at an independently controlled RPM rate.

In addition to having no direct connection between the motors, another added benefit of having independent motor control is that it can also control each rotor individually in multiples. This allows a quadcopter to obtain 6 degrees of freedom (6 DoF). 6 DoF is achieved by the ability to move left and right, up and down, and rotate clockwise and anticlockwise in a fixed location.

Left and right movement is achieved by increasing two of the motors at a speed (two left motors or two right motors) and proportionally decreasing the speed of the other two motors. A left turn is demonstrated in Figure 2-1.

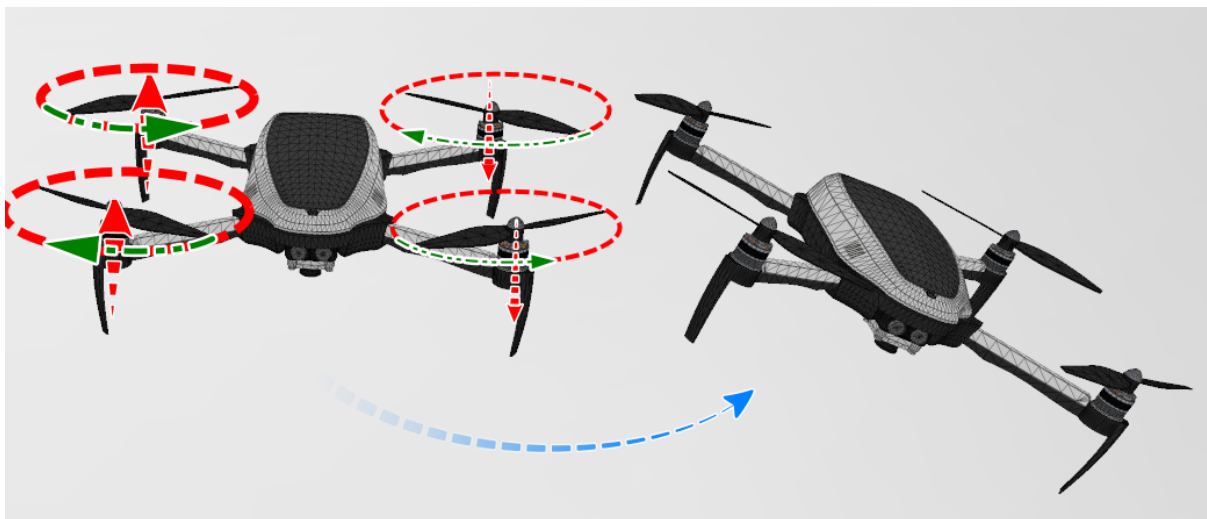


Figure 2-1 - Left turn

Similarly, forward and backwards movement is achieved by increasing two of the motor speeds (two front motors or two rear motors) and proportionally decreasing the speed of the other two motors (Figure 2-2)

.

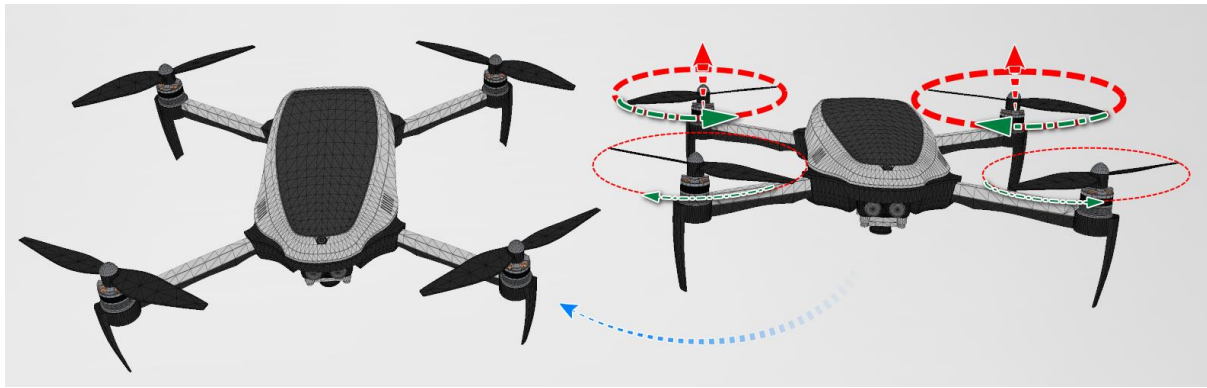


Figure 2-2 - Forward movement

To rotate the quadcopter a diagonal pair of motors is spun faster and the other diagonal pair is proportionally slower. An example is shown in Figure 2-3.

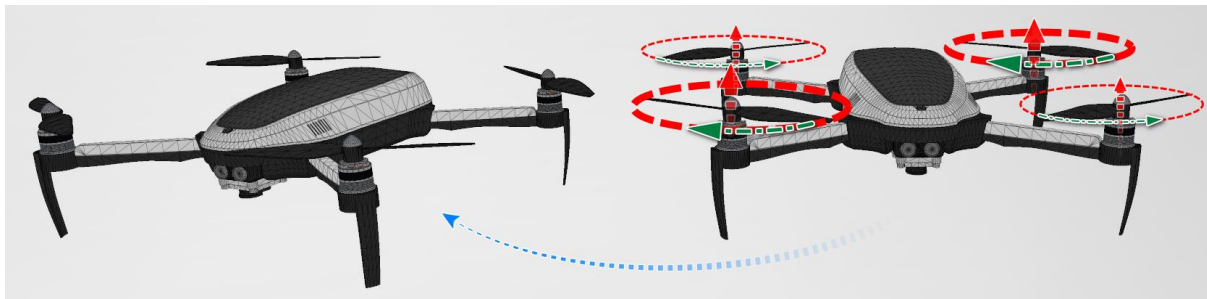


Figure 2-3 - Yaw rotation

The aircraft is primarily governed by the three major axes, namely, X, Y, and Z (Johnson, 2013). The quadcopter's movement is based on the rotational speed of each of the motors; the change in speed changes the position.

The quadrotor is modelled with its four rotors in a cross-configuration style (Figure 1-15). Two of the motors rotate counter-clockwise, while the other two rotate clockwise, given that the front-left and the rear-right motors rotate counter-clockwise. In contrast, the other two rotate clockwise, and gyroscopic effects and aerodynamic torques tend to cancel each other resulting in zero net force acting on the body of the quadcopter (*test1*, no date).

There are two controllers to manage displacement and angular rate. The X_r , Y_r , and Z_r inputs (see Figure 2-4) are different to the axes, these are input displacements (except Z_r which has input in angles) representing how far the quadcopter must travel/rotate and this information is fed into one of the controllers (the outer controller). Similarly, pitch, roll, and yaw take in data coming from the outer controller and converts it to the angular velocity. This will be further discussed in later chapters.

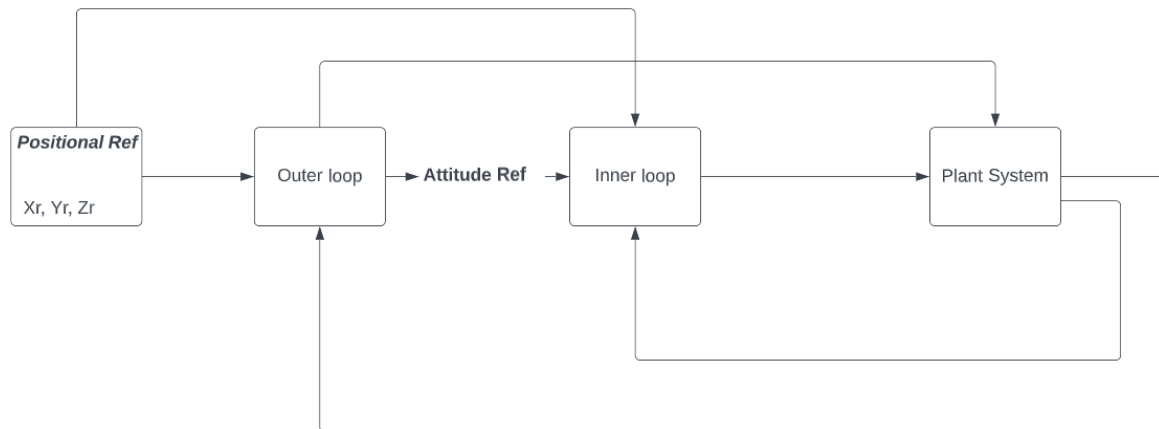


Figure 2-4 - Overall block diagram

2.1.1 Yaw

Yaw is the vertical axis that passes through the geometric centre of the quadcopter. The rotational force vector of all four motors acts at the centre and cancels out each other at the exact geometric centre; in conditions when it does not cancel, and the resultant vector has net positive or negative magnitude, the quadcopter rotates about this axis clockwise or anti-clockwise respectively.

Yaw control in quadcopters is done by varying the angular velocity of the diagonal motor. The two diagonally located motors of your quad will spin faster than the other two resulting in a change in the angular moment. Figure 3-1 below represents yaw control using the right-hand rule.

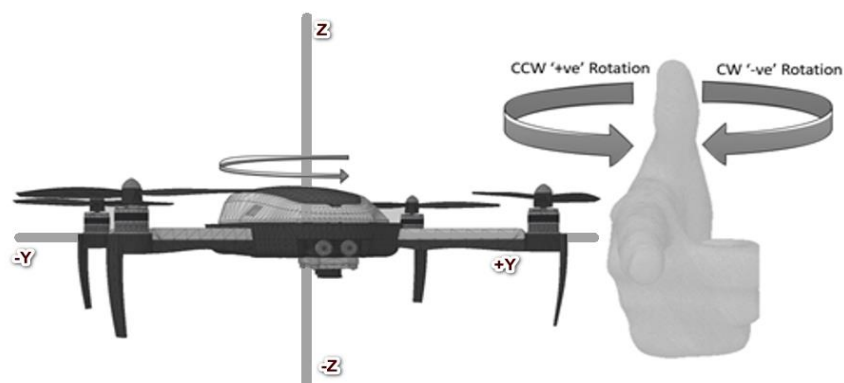


Figure 2-5 - Demonstration of Yaw manoeuvre along Z-axis

2.1.2 Pitch

It is the axis that passes horizontally parallel to the plane of the quadcopter extending towards the front and back end of the quadcopter. The rotational force vector of all four motors acts at the centre and cancels out each other at the exact geometric centre; when the resultant of the rotational force vector is not zero but either positive or negative, the quadcopter moves in the forward or backward direction, respectively.

Similar to the roll manoeuvre, the angular velocity of the front two motors is increased, and equally, the rear side two decreased to perform a manoeuvre. Figure 3-2 below represents a pitch manoeuvre.

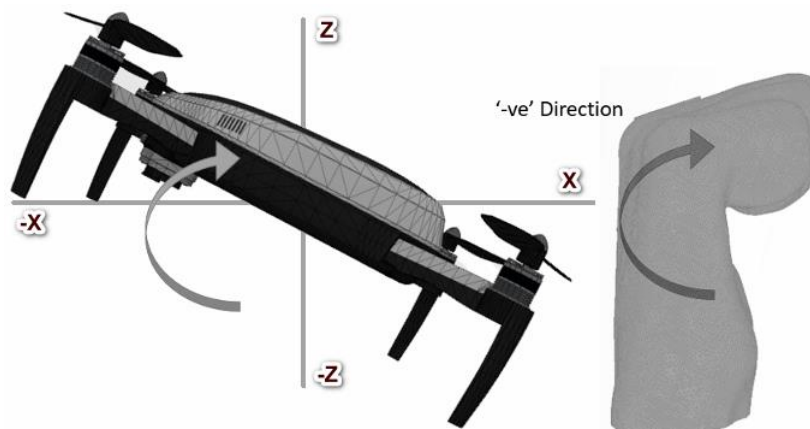


Figure 2-6 - Demonstration of Pitch manoeuvre along the Y-axis

2.1.3 Roll

It is the axis that passes horizontally parallel to the quadcopter plane extending from left to right. The rotational force vector of all four motors acts at the centre and cancels out each other at the exact geometric centre; when the resultant rotational force vector is not zero but either positive or negative, the quadcopter moves in the right or left direction, respectively.

In other words, the angular velocity of the left two motors increases, and equally, the right side two motors' angular speed decreases. This helps the quadcopter to maintain its altitude while performing the roll manoeuvre. Figure 3-3 below demonstrates the roll manoeuvre.

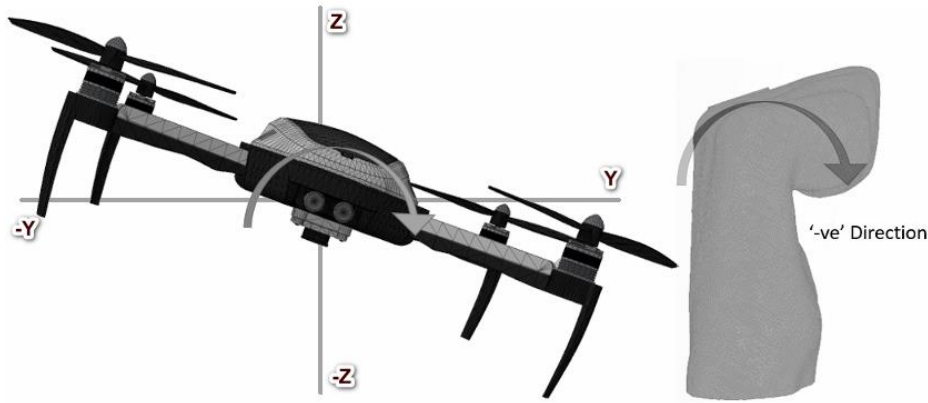


Figure 2-7 - Demonstration of Roll manoeuvre along the X-axis

2.1.4 Throttle – Vertical Motion

Throttle, also known as vertical take-off and landing (VTOL), is a term used to move the vehicle up or down in motion. Positive throttle makes the quadrotor fly higher and vice versa. A quadcopter is said to hover if it can maintain its height at a particular position without performing other manoeuvres. For a quadrotor to hover, all four propellers are made to maintain the motor's angular velocity. The hover state is often taken as an equilibrium point for linearisation because when two diagonal pairs of motors are rotating at the same angular velocity but in different directions (CCW and CW), the total angular momentum is zero.

2.2 Coordinate frames

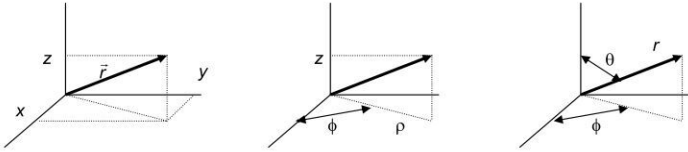
In general, a coordinate frame of reference enables measurements in one coordinate system to be converted to another coordinate system. One can then apply geometry to understand such relationships; for example: if the two systems have their axes aligned, then movement parallel to the axis will be translated into movement along that axis. A frame of reference fixes the relationship between certain spatial coordinates in the coordinate system. In effect, it determines the origin of a coordinate system, and can be viewed as determining how observers view the universe. In this respect, it is similar to a map projection—a choice of orientation or vantage point.

Rigid body movement in a coordinate frame is used to localise objects in a real-time simulation (DiSalle, 2020). This is used to calculate how the entities (quadcopter) move around in space and could use this information to render them on screen. Two rigid body world coordinate frames: one which is absolute (inertial/earth) and another which is relative to the object (body/vehicle). The absolute frame defines the world's origin (the centre point) and its direction. The relative frame defines the position of the entity of interest at any given time.

These are then calculated to determine how these entities move around in the world space if their relative position is known with respect to the known origin (world centre). This is achieved by transforming the two coordinate systems and multiplying their respective vectors together. This is referred to as rigid body matrix multiplication since matrices are used for the transformation.

The whole idea of coordinate systems is to describe a location in space. There are four types of coordinate systems: Cartesian, Cylindrical, and Spherical Figure 2-8.

Cartesian, Cylindrical Polar, and Spherical Polar Coordinates



	Cartesian	Cylindrical Polar	Spherical Polar
$x =$	x	$\rho \cos \phi$	$r \sin \theta \cos \phi$
$y =$	y	$\rho \sin \phi$	$r \sin \theta \sin \phi$
$z =$	z	z	$r \cos \theta$
Unit Vectors (orthonormal)	$\hat{e}_x = \hat{i}$ $\hat{e}_y = \hat{j}$ $\hat{e}_z = \hat{k}$	$\hat{e}_\rho = \cos \phi \hat{i} + \sin \phi \hat{j}$ $\hat{e}_\phi = -\sin \phi \hat{i} + \cos \phi \hat{j}$ $\hat{e}_z = \hat{k}$	$\hat{e}_r = \sin \theta \cos \phi \hat{i} + \sin \theta \sin \phi \hat{j} + \cos \theta \hat{k}$ $\hat{e}_\theta = \cos \theta \cos \phi \hat{i} + \cos \theta \sin \phi \hat{j} - \sin \theta \hat{k}$ $\hat{e}_\phi = -\sin \phi \hat{i} + \cos \phi \hat{j}$
Infinitesimal Displacement	$d\vec{r} = dx \hat{i} + dy \hat{j} + dz \hat{k}$	$d\vec{r} = d\rho \hat{e}_\rho + \rho d\phi \hat{e}_\phi + dz \hat{e}_z$	$d\vec{r} = dr \hat{e}_r + r d\theta \hat{e}_\theta + r \sin \theta d\phi \hat{e}_\phi$
$ d\vec{r} ^2 \equiv (ds)^2$	$(ds)^2 = (dx)^2 + (dy)^2 + (dz)^2$	$(ds)^2 = (d\rho)^2 + \rho^2 (d\phi)^2 + (dz)^2$	$(ds)^2 = (dr)^2 + r^2 (d\theta)^2 + r^2 \sin^2 \theta (d\phi)^2$
Volume Element	$dV = dx dy dz$	$dV = \rho d\rho d\phi dz$	$dV = r^2 \sin \theta dr d\theta d\phi$

Figure 2-8 - Coordinate systems (Yumpu.com, no date) (William W. Warren J)

In marine navigation, the heading is the direction of a ship's bow relative to the geodetic datum. Pitch is the up-down angle of the ship's nose. Bank is the left-right angle of a ship's hull. Yaw is the left-right angle of a ship's stern relative to its forward motion. A good description of these terms in use can be found in Bowditch (Bowditch, no date, p. 6).

A commonly used coordinate system in quadcopter design is Cartesian. In the Cartesian coordinate system, three numbers are used to specify a location in space (e.g., the X-, Y- and Z-coordinates of a point). One number is associated with each axis so that any point in space can be specified using an ordered triplet of numbers (X, Y, and Z). In coordinate frame systems, the conventional axis is two types: East, North, Up and North, East, Down frame.

As described above, a rigid body movement (in Cartesian) is represented in inertial and body frames. The inertial earth frame (E-frame) can be expressed as O_E, X_E, Y_E, Z_E . Where, the Z_E axis point upwards w.r.t the earth, Y_E towards the West, X_E towards the North and O_E is the axis of origin. Whereas the body-fixed frame (B-frame) is attached to the quadcopter's body and is expressed O_B, X_B, Y_B, Z_B (Figure 2-9) (Figure 5. Movement of quadcopter, body frame and inertial quadcopter..., no date).

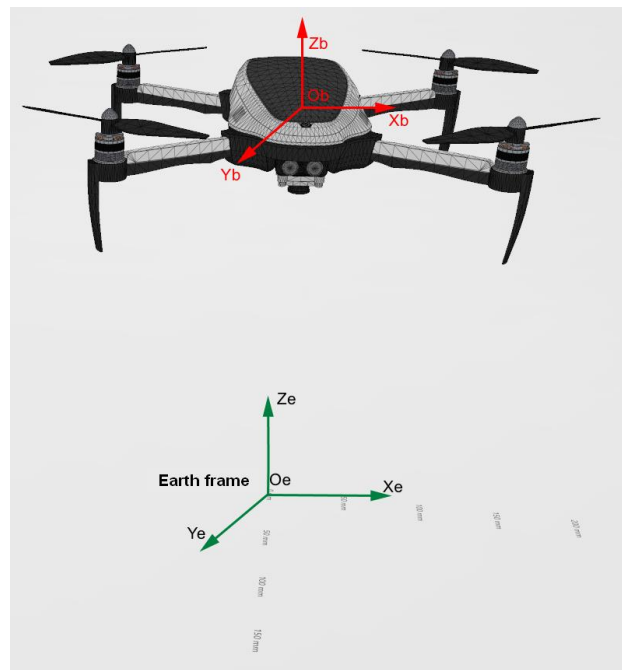


Figure 2-9 - Earth and body frame

2.3 Euler Angles

Euler angles are a series of three angles along three predefined axes used to parameterise rotations (Converting to Euler & Tait-Bryan, no date). Euler angle rotations are a sequential sequence of rotations (6: Euler Angle Definition - Rigid Body Kinematics I | Coursera, no date). There are 12 possible sets of Euler angles. These 12 sets can be divided into symmetrical and asymmetrical sets. The symmetrical sets of angles are also known as Proper Euler angles, and they have the first and third axis the same ($z-x-z, x-y-x, y-z-y, z-y-z, x-z-x, y-x-y$). In contrast, asymmetric Euler angles (also known as Tait-Bryan angles) are sequence rotations where the axes do not repeat themselves ($x-y-z, y-z-x, z-x-y, x-z-y, z-y-x, y-x-z$) ('Euler angles', 2021).

The matrix of rotation can be derived by separating the individual axis of rotation and by defining independent angles to each axis as follows: $R_x(\phi)$ for rotational around the x-axis, $R_y(\theta)$ for rotation around the y-axis, and $R_z(\psi)$ for rotation around the z-axis.

Table 2-1 - Rotational Matrix (R_x , R_y , R_z)

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

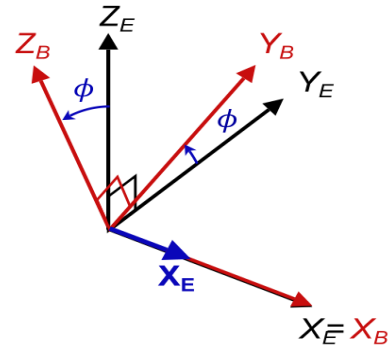


Figure 2-10 - X-axis rotation

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

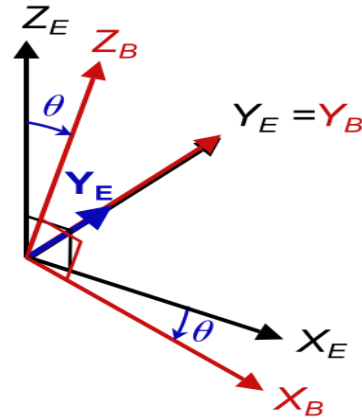


Figure 2-11 - Y-axis rotation

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

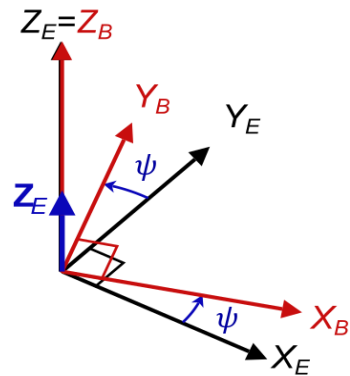


Figure 2-12 - Z-axis rotation

These axes of rotations are derived using the right-hand rule where the z-axis faces up. If the z-axis faces down using the right-hand rule, the "-ve" signs in each matrix will be displaced.

The position and orientation of an object are associated with an orthonormal reference frame. In other words, each axis of a chosen frame of reference must be perpendicular to the other.

Figure 3-7 and Figure 3-8 below represents inertial and body frame of reference, respectively. Generally, each of the axes denotes the direction vector or the norm as one. In both frames (O_E, O_B) have three linearly independent basis vectors $[X_E, Y_E, Z_E]^T$ and $[X_B, Y_B, Z_B]^T$ can be found, respectively.

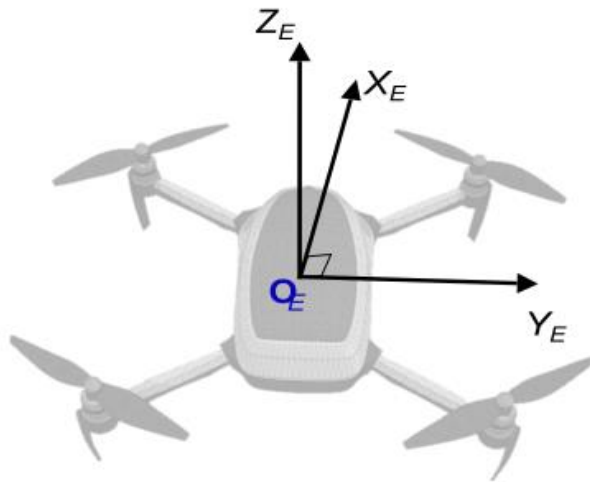


Figure 2-13 - Inertial frame of reference

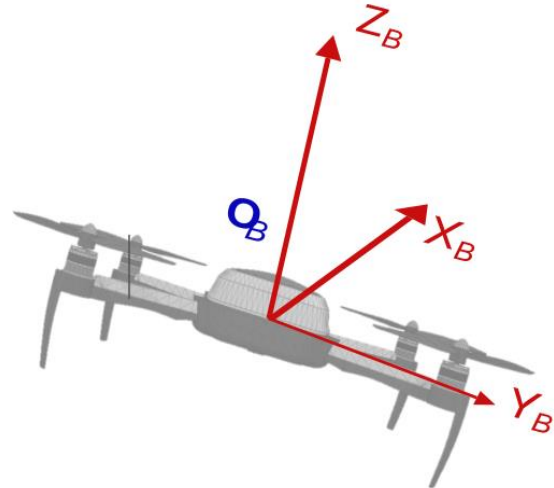


Figure 2-14 - Body frame of reference

2.4 Rotation about intermediate axes

As described above, the Euler rotation is a combination of a sequence of rotations. It multiplies the rotation matrix in a specified sequence. In other words, Euler angles comprise three successive sequences of rotations (Yaw, Pitch and Roll or 3-2-1). In MATLAB, the axis rotation sequence for Euler angles, by default, is in the order of z-axis, y-axis, and x-axis, as shown in Figure 3-9 below. It is important to adhere to the correct rotation sequence when Euler angles are provided.

Input Arguments

✓ **eul** — Euler rotation angles
 n -by-3 matrix

Euler rotation angles in radians, specified as an n -by-3 array of Euler rotation angles. Each row represents one Euler angle set.

Example: $[0 \ 0 \ 1.5708]$

> **sequence** — Axis rotation sequence
 "ZYX" (default) | "YZX" | "XYZ"

Figure 2-15 - MATLAB default Euler angle sequence (Convert Euler angles to rotation matrix
 - MATLAB eul2rotm - MathWorks United Kingdom, no date)

These three sequences can be divided into three different elemental rotations. Starting from the origin where both $O_E = R_E$ and $O_B = R_B$ are at the origin.

The first stage from its origin to the first rotation R_E to $R_1 = R_{R_1}^{R_E}(\psi) = \underset{\text{to}}{\overset{\text{from}}{R_{R_x}^{R_x}}}$ about the z-axis over the angle ψ as shown in Figure 3-10.

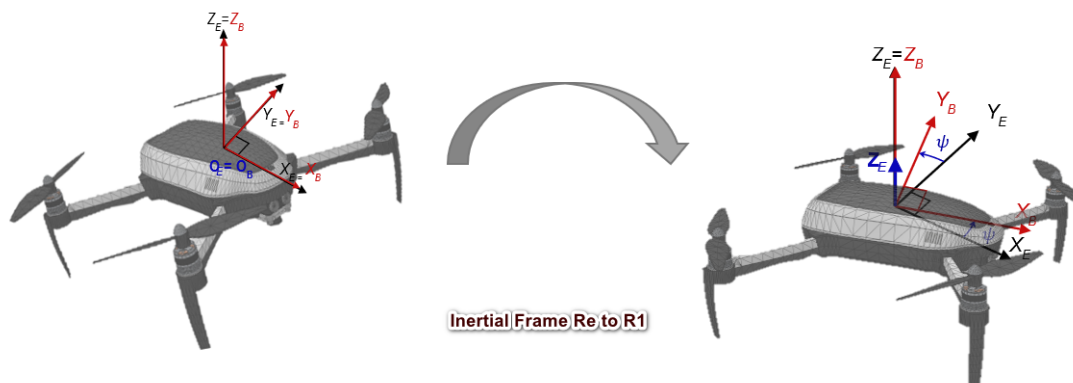


Figure 2-16 - Frame Re to R1

Second stage from R_1 to $R_2 = R_{R_2}^{R_1}(\theta)$ about the y-axis as shown in Figure 3-11

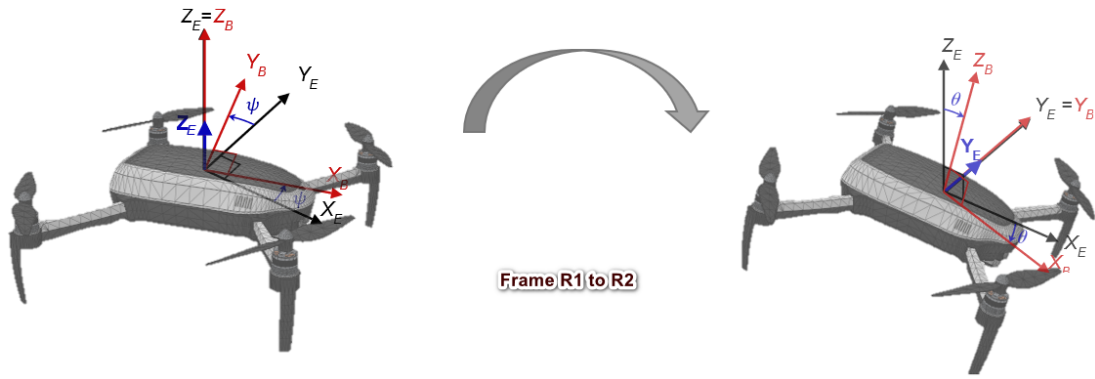


Figure 2-17 - Frame R1 to R2

Finally from R_2 to $R_B = R_{R_2}^{R_2}(\phi)$ about the x-axis gives the final orientation of the quadcopter as shown in Figure 3-12.

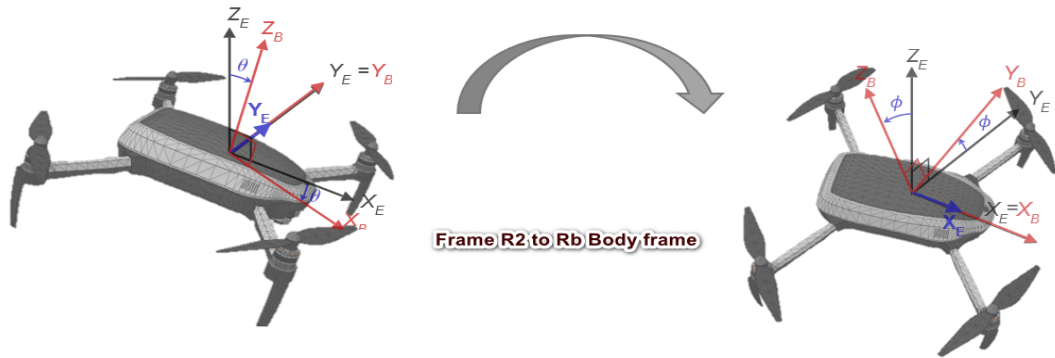


Figure 2-18 - Frame R2 to Rb

Therefore, the product of three basic rotation matrices gives a complete rotation matrix.

Where the total rotational matrix R_θ ;

$$R_\theta = R_{R_B}^{R_E} = R_{R_1}^{R_E} R_{R_2}^{R_1} R_{R_B}^{R_2} = R_x(\phi) R_y(\theta) R_z(\psi) \quad 2.1$$

$$R_\theta = \begin{bmatrix} \cos\theta\cos\psi & \cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi & \sin\phi\sin\psi + \cos\phi\cos\psi\sin\theta \\ \sin\psi\cos\theta & \cos\psi\cos\phi + \sin\psi\sin\theta\sin\phi & \sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\phi\cos\theta \end{bmatrix} \quad 2.2$$

2.5 Kinematics of a quadcopter

Kinematics is the branch of mechanics that describes the motion of objects. Kinematics is within the field of classical mechanics and deals with the motion of a robot without concern for its mass or inertia, only considering interaction forces (*What is Kinematics? Definition,*

Formula, Derivation, Sample Problems - GeeksforGeeks, no date). It analyses body movement without regard to forces or energy changes. In other words, kinematics studies how things move, not why they move in specific ways and/or at specific speeds. It is important to understand differential equations to describe kinematic variables such as velocity, acceleration, and time.

Ideally, a quadcopter consists of IMU, which returns acceleration (accelerometer – $\ddot{X}, \ddot{Y}, \ddot{Z}$) and angular velocity (gyroscope – p, q, r). The p, q , and r are the body axis angular rates and are not equal to Euler rates $\dot{\phi}, \dot{\theta}, \dot{\psi}$. In other words, the body axis angular rates (p, q, r) from the IMU cannot be used to determine Euler angles ϕ, θ, ψ see Figure 3-13 below.

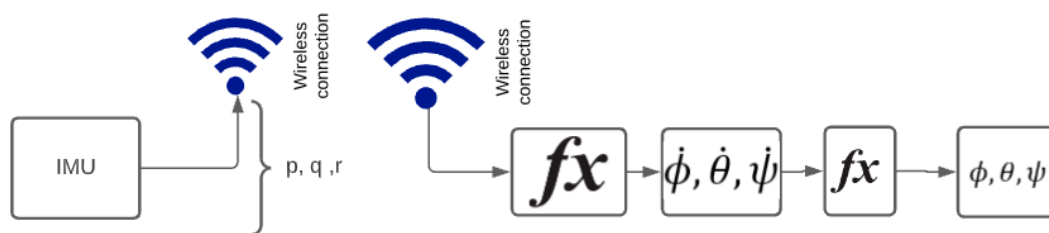


Figure 2-19 - Body axis angular rate to Euler rates

The angular velocity of an object will be constant if the object is rotating at a constant rate. However, the Euler rates run at a higher frequency because they rely on the instantaneous angles between the body frame and the inertial frame.

2.6 The inner and outer loop

As referred to in Figure 2-4, there are two controllers present to control the inner loop (attitude) and outer loop (position) of the system. The quadcopter should achieve a certain orientation to reach the desired position. The outer loop controller delivers U_1, θ_r, ϕ_r values to the inner loop controller as shown in Figure 2-20. Essentially the outer loop controller should eliminate e_ϕ and e_θ to zero.

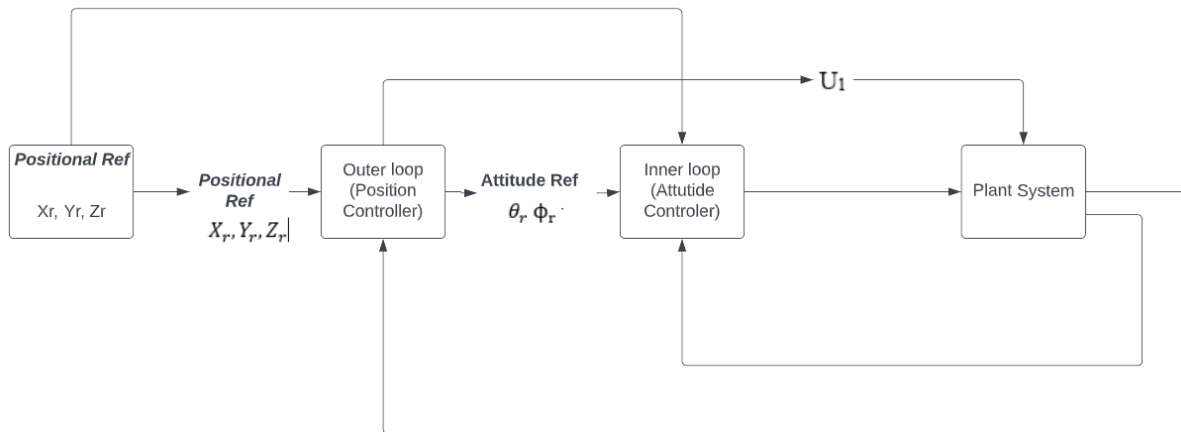


Figure 2-20 - Inner and outer loop controller block diagram

There is a crucial yaw angle (ψ_r) which is omitted from the block diagrams. However, this is not relevant to achieve the desired position. The reference yaw angle is delivered straight to the inner loop controller. Conventional yaw angle positive rotation is shown in Figure 2-21 (Figure 1.3: A common convention for the angles yaw, pitch and roll., no date).

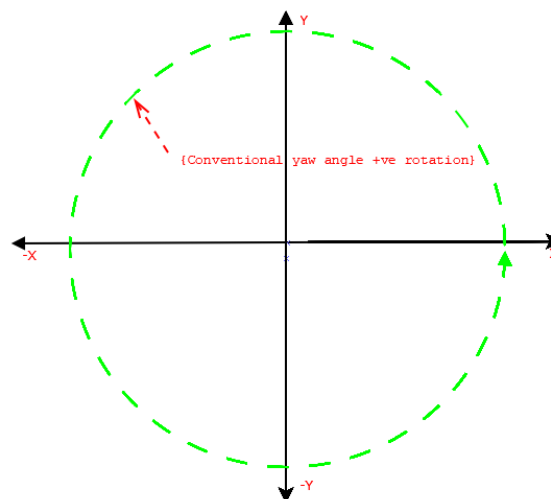


Figure 2-21 - Conventional yaw angle +ve rotation

There is an anomaly to consider when rotating the quadcopter more than 2π radians (360°). If the quadcopter is required to rotate, for example, 3 circles (6π) with an increment of 0.1 radians, the array should not reset after every 2π radians. The array should continue to increment up until its desired yaw command $[0, 0.1, 0.2 \dots 2\pi, \mathbf{2\pi} + 0.1, \dots 6\pi]$. Otherwise, the controller may not provide the required output.

If the inner and outer loops are running at the same time intervals, the inner loop won't be able to reach the setup (ϕ_r, θ_r) of the previous iteration. Before it reaches its setpoint the outer loop will recalculate and set a different set point for the inner loop controller. There

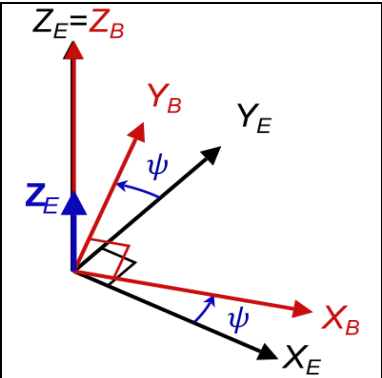
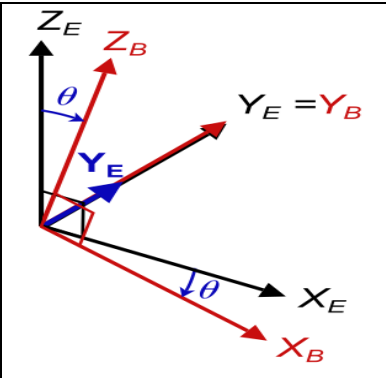
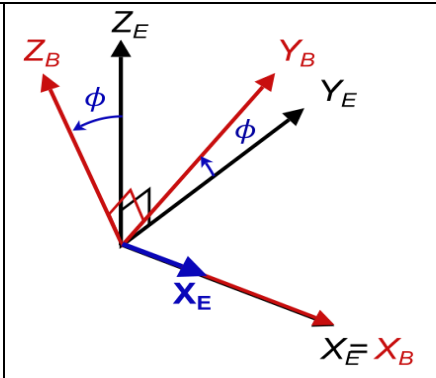
should be enough time for the inner loop to calculate and reach its setpoint and to accept the next iteration setpoint. A ratio of 5:1 is used in this research. This means every 1 interaction of the outer loop, the inner loop is calculated 5 times with every 100ms interval. Consequently, the inner loop is known as a higher dynamic loop or high-frequency loop and the outer loop with low frequency.

As described above the Euler angle sequence is made up of three successive rotations. Consider $\vec{\omega}$ as angular velocity vector. The magnitude of $\vec{\omega}$ represents the rate at which the earth frame is rotating with respect to the body frame. The following expression must be satisfied to establish the relationship between p, q, r and $\dot{\phi}, \dot{\theta}, \dot{\psi}$.

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = f(\vec{\omega}_{R_B}^{R_E}), \text{ where } \vec{\omega}_{R_B}^{R_E} = \vec{\omega}_B^E = \begin{bmatrix} p \\ q \\ r \end{bmatrix} \text{ (wrt body frame)} \quad 2.3$$

Table 3-2 below represents components of the angular velocity vector where $\vec{\omega}_1^E$, $\vec{\omega}_2^1$ and $\vec{\omega}_B^2$ describes the direction and magnitude of vectors along z, y, and x axes, respectively.

Table 2-2 - Intermediate angular velocity vector

		
$\vec{\omega}_1^E = \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix}$	$\vec{\omega}_2^1 = \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix}$	$\vec{\omega}_B^2 = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix}$

The angular velocity vector $\vec{\omega}_B^E$ is the summation of the following intermediate vectors

$$\vec{\omega}_B^E = \vec{\omega}_B^2 + \vec{\omega}_2^1 + \vec{\omega}_1^E \quad 2.4$$

To solve the equation above, the intermediate frames need to be converted into the body frame. These intermediate frames were derived above in equation 3-1 as $R_{R_1}^{R_E}, R_{R_2}^{R_1}, R_{R_B}^{R_2}$ with corresponding rotational angles.

$$\vec{\omega}_B^E = \vec{\omega}_B^2 + \overbrace{R_{R_B}^{R_2}(\phi) \times \vec{\omega}_2^1}^{\text{Converts to the body frame}} + \overbrace{R_{R_B}^{R_2}(\phi) \times R_{R_2}^{R_1}(\theta) \times \vec{\omega}_1^E}^{\text{Converts to the body frame}} \quad 2.5$$

$$\begin{aligned} \vec{\omega}_B^E &= \begin{bmatrix} \dot{\phi} \\ 0 \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \times \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} \\ &+ \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \times \begin{bmatrix} \cos\theta & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} \end{aligned} \quad 2.6$$

$$\vec{\omega}_B^E = \begin{bmatrix} \dot{\phi} + \dot{\psi} \sin(\theta) \\ \dot{\theta} \cos(\phi) - \dot{\psi} \cos(\theta) \sin(\phi) \\ \dot{\psi} \sin(\phi) + \dot{\psi} \cos(\phi) \cos(\theta) \end{bmatrix} \quad 2.7$$

$$\vec{\omega}_B^E = \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & \sin(\theta) \\ 0 & \cos(\phi) & -\cos(\theta) \sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \cos(\theta) \end{bmatrix} \times \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \quad 2.8$$

$$\overbrace{\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}}^{\Phi} = \overbrace{\begin{bmatrix} 1 & \sin\phi + \theta & \cos\phi + \theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sec\theta * \sin\phi & \sec\theta * \cos\phi \end{bmatrix}}^{T_\Theta} \times \begin{bmatrix} p \\ q \\ r \end{bmatrix} \quad 2.9$$

Where $\dot{\Phi} = \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}$ and $\Phi = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}$ denote three Euler angles. Note $\Phi \neq \phi$

$$T_\Theta = \begin{bmatrix} 1 & \sin\phi + \theta & \cos\phi + \theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sec\theta * \sin\phi & \sec\theta * \cos\phi \end{bmatrix} \quad 2.10$$

2.7 Summary

The chapter describes two fundamental tools required to manipulate rigid body movement. They are the rotation and translation motion of a rigid body. The rotational and transfer matrix allows users to manipulate data from sensors such as IMU.

The matrix R_Θ is also known as the direction cosine matrix (DCM). The advantage of DCM is it becomes an intermediate system to convert to and from Euler angles and other rotational techniques.

The disadvantage of this technique is a phenomenon known as gimbal lock, also known as a singularity (Hemingway and O'Reilly, 2018) (*Understanding Euler Angles* | CH Robotics, no date). The second Euler angle always enters singularity when it exceeds a known range of angular rotation. In asymmetric Euler angles, the singularity occurs at $\pm 90^\circ$ and in symmetrical Euler angles, the singularity can be found at 0° or 180° . This limitation makes it less attractive for large reorientations of the quadcopter (9: *Euler Angle Differential Kinematic Equations - Rigid Body Kinematics I*, no date). As an example, a 90° pitch will enter the quadcopter into a gimble lock in an asymmetric system.

The singularity phenomena can be avoided by limiting angular rotation to the quadcopter or by using a different technique called quaternion. Quaternions are widely used in the gaming and aviation industries, and it is proven to be robust. However, the mathematics involved is significantly complex, and it takes a considerable amount of computer resources.

The sequence adopted above (intrinsic) represents Euler angles defined relative to the intermediate frame rotate with respect to one another. In other words, to represent a rigid body rotation from $R_{RB}^{R_E}$ the intermediate frames change their orientation after each elemental rotation. However, the extrinsic element of rotation is defined as Euler angles relative to a reference frame that remains motionless and does not move. The sequence of order matters when the intrinsic element of rotation is being implemented to determine the kinematical equations.

Chapter 3 – Analytical and Dynamical Model of Quadcopter

This chapter covers in-depth the fundamental dynamics of the quadcopter. This includes focusing on deriving mathematical models using Newton's and Euler's equations. The chapter illustrates how to combine translation and rotation physics to determine various aspects of quadcopter dynamics. The models developed in this chapter become an integral part of the entire quadcopter. Having a stable plant model allows any controller to be used and tuned. In other words, the quadcopter is not limited to the controllers being used in this research. This chapter looks at the following changes and forces applied to the quadcopter.

- Translation from the earth frame and body frame to determine the quadcopter's position.
- Combining translational motion with the rotational motion to achieve 6 DoF.
- The mass movement of inertia to measure quadcopters' resistance to change in the rotational direction.
- Derivation of actuator dynamics.
- Forces applied to a quadcopter etc.

3.1 Quad-rotor Modelling Assumptions

Mathematical models are used as tools to investigate the underlying phenomena. In the 20th century, mathematical models of increasingly complex systems were developed in physics, chemistry, biology, and social science. A model may serve either a predictive or a purely conceptual purpose. Principles abstracted from the actual physical system are used in the model, and these principles may be quite different from the real world. For example, when constructing a physical model of an object, one usually makes certain assumptions about its works. These assumptions are based on knowledge of the real world and interpreted through known evidence. A model is not expected to behave exactly like the real world, but it should share those characteristics essential to the model's intended purpose. Mathematical models are often hypothetical in the sense that they are simplified representations of reality. They are hypothetical in the sense that they do not try to replicate reality exactly. Instead, they remain close enough to reality to be useful and practical, while remaining open to refinement or improvement. Mathematical models can usually be tested and refined to a very high degree of accuracy, so they are used for making predictions about future events. There are also models which are created to fit past data instead of being used to make predictions. A model may be considered a good model if it fits past data well or a poor one if its predictions are inaccurate and do not fit the data. Some of the models are from prior work by Dr Tim Poyi, especially 3.8 Actuator Dynamics (DC-motor).

Some of the assumptions made are as follows.

The quadcopter is symmetric in shape, ie one half of the quadcopter replicates the other half.

To reduce complex calculations, the origin of the body frame (O_B) is located at the centre of mass of the quadcopter.

The body frame used to do the calculations is aligned with the three principal axes.

An overview of the plant model can be seen in Figure 3-1.

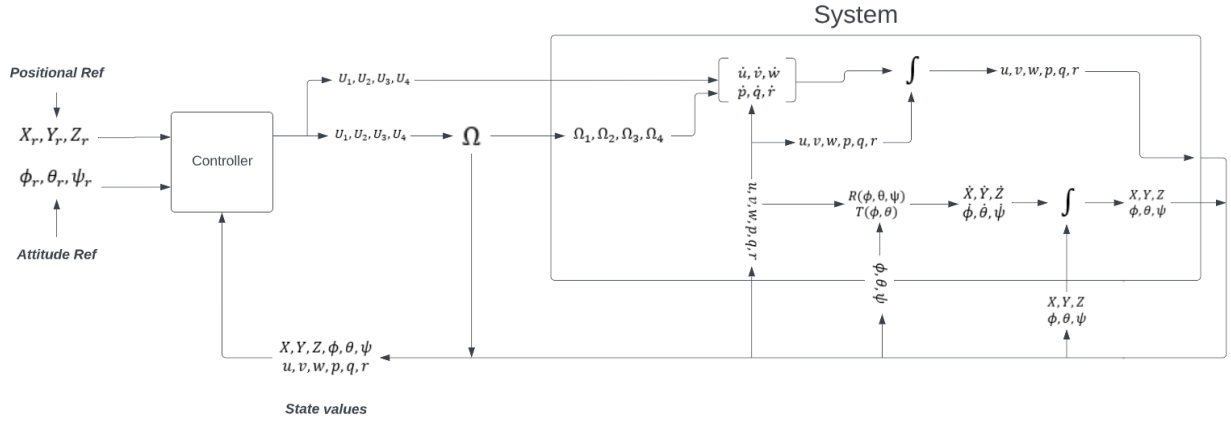


Figure 3-1 - Plant/System model block diagram

3.2 The dynamic equation for translation motion

In dynamics, which is the field of complex systems, the effects of an object's mass or inertia are considered within the laws of motion. Newton's second law of motion and Euler angles are used to describing the relationship between the moments and aerodynamic forces acting on the quadcopter('Modeling Vehicle Dynamics - Quadcopter Equations of Motion', 2017). A translational movement to the right or left is achieved by tilting the quadcopter with a rolling motion and increasing all the thrust produced compared to a stationary flight in order to maintain airborne along the Z axis. In addition to left and right translational motion, forward and reverse translational motion can also be achieved by tilting the drone with a pitching motion and increasing all thrust (Nirina, Auguste and Auguste, 2022).

This section considers developing equations for translation motion, in other words, these equations are used to track the movement of a quadcopter from one point to the next (not rotation). This also considers the different points of reference, i.e. conversion from earth frame and body frame.

$$F = ma \quad 3.1$$

Using Euler's rotation equation and Newton's law, the following equation can be derived (see Table 4-1)

$$F^E = m \ddot{\Gamma}^E \quad 3.2$$

The orientation of a body is disregarded in this section. Here $\overline{\Gamma}^E = [X \ Y \ Z]^T$ refers to the translation motion/position vector described in the inertial frame. According to Newton's second law:-

$$\vec{F} = ma = m \left(\frac{d\vec{v}}{dt} \right) = m\vec{v} \quad 3.3$$

The position vector can be written in a different form by considering the unit vectors $(\hat{i}, \hat{j}, \hat{k})$, where the unit vectors have a magnitude of 1.

$$\vec{r}^E = X\hat{i} + Y\hat{j} + Z\hat{k} \quad 3.4$$

Taking the derivative of the position vector and using the product rule gives,

$$\dot{\vec{r}}^E = \dot{X}\hat{i} + \dot{Y}\hat{j} + \dot{Z}\hat{k} + X\dot{\hat{i}} + Y\dot{\hat{j}} + Z\dot{\hat{k}} \quad 3.5$$

In this case, the axes of the inertial frame are fixed and $\dot{\hat{i}}, \dot{\hat{j}}, \dot{\hat{k}} = 0 \frac{\text{units}}{s}$. This means the unit vector does not change magnitude or direction, resulting in,

$$\dot{\vec{r}}^E = \dot{X}\hat{i} + \dot{Y}\hat{j} + \dot{Z}\hat{k} + \cancel{X\dot{\hat{i}}} + \cancel{Y\dot{\hat{j}}} + \cancel{Z\dot{\hat{k}}} = \dot{X}\hat{i} + \dot{Y}\hat{j} + \dot{Z}\hat{k} \quad 3.6$$

The derivative of eq 4.6 gives the acceleration vector

$$\ddot{\vec{r}}^E = \ddot{X}\hat{i} + \ddot{Y}\hat{j} + \ddot{Z}\hat{k} + \cancel{\dot{X}\dot{\hat{i}}} + \cancel{\dot{Y}\dot{\hat{j}}} + \cancel{\dot{Z}\dot{\hat{k}}} \quad 3.7$$

Therefore, the sum of the forces (net force) can be expressed as follows

$$\begin{aligned} \vec{F}^E &= m\ddot{\vec{r}}^E \text{ or } \sum \vec{F}^E = m\ddot{\vec{a}} \text{ or } \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = m \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} \text{ or } F_x\hat{i} + F_y\hat{j} + F_z\hat{k} \\ &= m(\ddot{X}\hat{i} + \ddot{Y}\hat{j} + \ddot{Z}\hat{k}) \end{aligned} \quad 3.8$$

The control torque and control forces are valid only on the body frame. The control forces are thrust generated by all four propellers U_1, U_2, U_3, U_4 , throttle, roll, yaw and pitch, respectively acting on the body frame. The equation for translation motion from the body frame must be expressed in the inertial frame.

$$\vec{v}^B = [u, v, w]^T \quad u \neq U_1, U_2, U_3, U_4 \quad 3.9$$

Here (u, v, w) represents translation motion in the x, y and z direction within the body frame (Figure 3-2).

Equation 4.9 can be expressed in unit vector on the body frame

$$\vec{v}^B = u\hat{i} + v\hat{j} + w\hat{k} \quad 3.10$$

Figure 4-1 below represents thrust forces acting on the body frame.

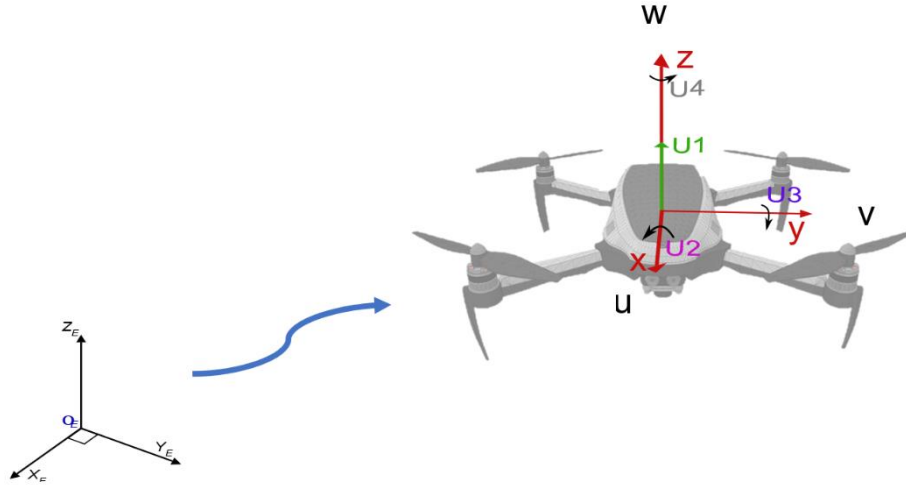


Figure 3-2 - Velocity expressed in the body frame

The derivative of 4.10 gives

$$\dot{\vec{v}} = \dot{u}\hat{i} + \dot{v}\hat{j} + \dot{w}\hat{k} + u\dot{\hat{i}} + v\dot{\hat{j}} + w\dot{\hat{k}} \quad 3.11$$

Resulting in a net force vector

$$\vec{F}_{net}^B = m \overbrace{\left(\dot{u}\hat{i} + \dot{v}\hat{j} + \dot{w}\hat{k} + u\dot{\hat{i}} + v\dot{\hat{j}} + w\dot{\hat{k}} \right)}^{\dot{\vec{v}}} \quad 3.12$$

Note the time derivative of unit vectors is not zero because the unit vector on the body frame changes its direction as it moves but not its magnitude. By using the right-hand rule or cross product the $\dot{\hat{i}}, \dot{\hat{j}}, \dot{\hat{k}}$ can be obtained.

Consider the angular velocity vector ($\vec{\omega}^B$) (or general rotation axis), which rotates all body frame axes with respect to the inertial frame, rotates at an angle α radian.

$$\vec{\omega}^B = [p \ q \ r]^T \frac{rad}{sec} \text{ where } \omega \neq w \quad 3.13$$

Note w is translation motion in (Hoffmann *et al.*, 2004; Barbary and ElAzeem, 2021, 2021; *New drone technology unveiled, your mind can now instruct a drone to fly* | WION, 2021; *Internet Growth Statistics 1995 to 2022 - the Global Village Online*, 2022b) the z -direction in the body frame (Figure 3-3).

The change in axes of unit vectors ($\hat{i}, \hat{j}, \hat{k}$) with respect to time can be represented as follows.

$$d\vec{\hat{i}} = \hat{i}(t + dt) - \hat{i}(t)$$

$$d\vec{\hat{j}} = \hat{j}(t + dt) - \hat{j}(t) \quad 3.14$$

$$d\vec{\hat{k}} = \hat{k}(t + dt) - \hat{k}(t)$$

By using the cross product, the change in the unit vector with respect to time as given in equation 4.14 can be expressed in the actual unit vector ($\hat{i}, \hat{j}, \hat{k}$) and the angle of rotation ($d\alpha$) radian.

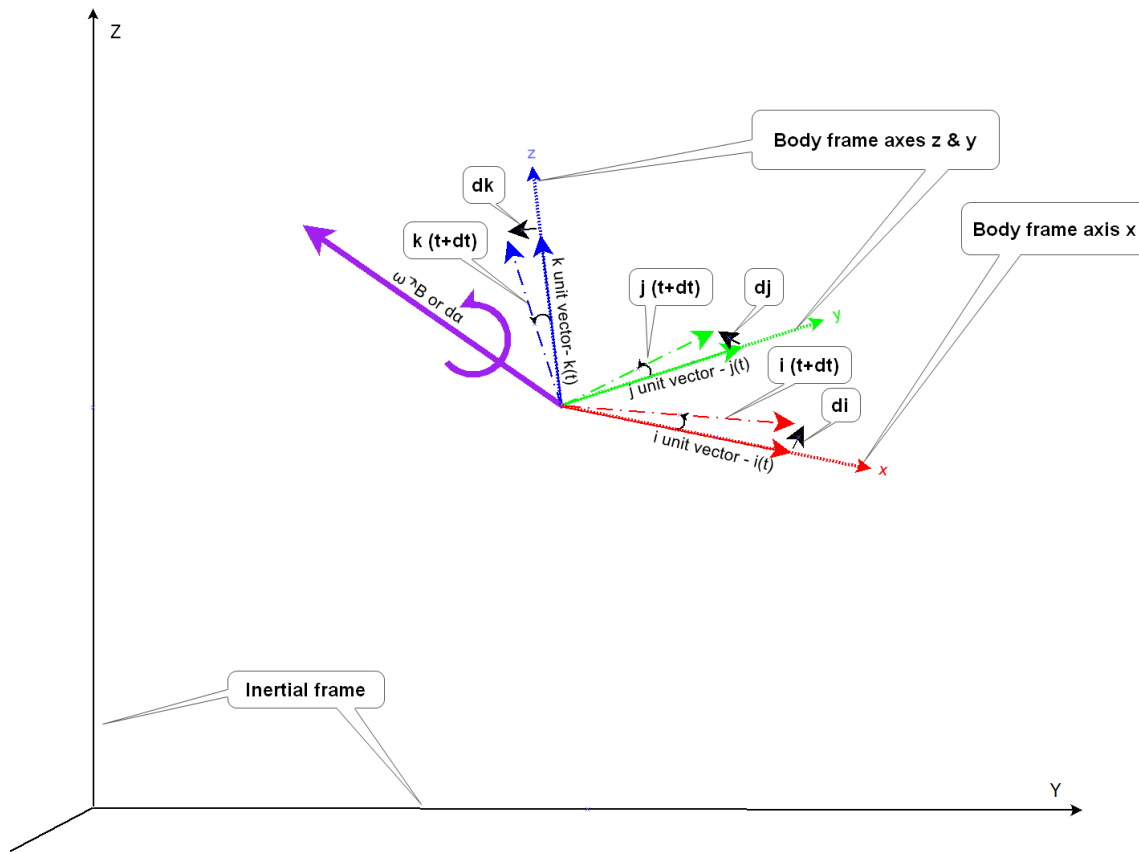


Figure 3-3 - Body frame components

Here $d\alpha$ represents the change in α radian with respect to time. In other words, the cross-product decomposes the $d\alpha$ rotation vector into components perpendicular to $\hat{i}, \hat{j}, \hat{k}$. By taking cross-product, we can relate the change in the unit vector with the actual unit vector and the change in $d\alpha$ radians (Figure 3-3).

$$d\hat{i} = d\alpha \times \hat{i} \quad d\hat{j} = d\alpha \times \hat{j} \quad d\hat{k} = d\alpha \times \hat{k} \quad 3.15$$

Therefore, the time derivate

$$\frac{\dot{\hat{i}}}{dt} = \frac{\overline{\omega}^B}{dt} \times \hat{i} \quad \frac{\dot{\hat{j}}}{dt} = \frac{\overline{\omega}^B}{dt} \times \hat{j} \quad \frac{\dot{\hat{k}}}{dt} = \frac{\overline{\omega}^B}{dt} \times \hat{k} \quad 3.16$$

Substitute the equations from 4.16 into equation 4.12 where $\dot{\hat{i}} = \overline{\omega}^B \times \hat{i}$, $\dot{\hat{j}} = \overline{\omega}^B \times \hat{j}$, $\dot{\hat{k}} = \overline{\omega}^B \times \hat{k}$

$$\vec{F}_{net}^B = m \left(\dot{u}\hat{i} + \dot{v}\hat{j} + \dot{w}\hat{k} + u(\overline{\omega}^B \times \hat{i}) + v(\overline{\omega}^B \times \hat{j}) + w(\overline{\omega}^B \times \hat{k}) \right) \quad 3.17$$

$$\vec{F}_{net}^B = m \left(\overbrace{\dot{u}\hat{i} + \dot{v}\hat{j} + \dot{w}\hat{k}}^{\dot{\vec{v}}_{xyz}^B = [\dot{u}, \dot{v}, \dot{w}]^T} \right) + \overbrace{\overline{\omega}^B \times u\hat{i} + \overline{\omega}^B \times v\hat{j} + \overline{\omega}^B \times w\hat{k}}^{\overline{\omega}^B \times \vec{V}^B} \quad 3.18$$

$$\vec{F}_{net}^B = m \left(\dot{\vec{v}}_{xyz}^B + \overline{\omega}^B \times \vec{V}^B \right) \quad 3.19$$

The dynamic equation for translation motion in the body frame can be expressed as follow,

$$\begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = m \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} + \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} u \\ v \\ w \end{bmatrix} \quad 3.20$$

Equation 3.20 shows how the quadcopter translates in space. However, the system will only consist of three degrees of freedom if it is only translated (X, Y and Z). The problem is the quadcopter doesn't only translate in space but also changes its attitude. Section 3.3 incorporates the orientation angles (ϕ, θ, ψ) to control the quadcopter's attitude.

3.3 The dynamic equation for rotational motion

The previous section 3.2 describes the translation motion of the quadcopter in space. If considering only the translation motion, then the system is limited to 3 degrees of freedom system where independent variables are X, Y and Z for the position. However, the quadcopter not only translates in space, it changes its attitude, which consists of independent angles ϕ, θ, ψ (for orientation), also known as Euler angles. The translational motions (x, y, and z) are accomplished by rotational motions (roll, pitch, and yaw). In order to stabilise the translational movements in quadrotor stabilisation control, the rotational movements must be stabilised (Abdulkareem *et al.*, 2022). By combining the translation and orientation of the quadcopter,

the system becomes a 6 degree of freedom of the system. In other words, the previous section relates to $F = ma$ in the body frame with respect to the inertial frame and this section focus on determining the moment of inertia with the mass moment of inertia and angular acceleration $M = I\ddot{\omega}$ in the body frame.

Let the angular momentum vector be,

$$\vec{H}^B = I_{xyz}^B \times \vec{\omega}^B \leftrightarrow \begin{bmatrix} H_x \\ H_y \\ H_z \end{bmatrix} = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \quad 3.21$$

Here the I^B is the mass moment inertial matrix.

3.3.1 Mass moment of inertia

This section determines the resistance to change in the rotational direction of a quadcopter. The moment of inertia has the same relationship to angular acceleration as mass has to linear acceleration (*Mass Moment of Inertia*, no date).

Since the quadcopter is considered to be symmetric, the off-diagonal elements are zero. Hence the mass moment of inertia I_{xx} , I_{yy} and I_{zz} can be represented as follows

$$I^B = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \quad 3.22$$

Figure 4-2 below represents the moment of inertia acting on various axes.

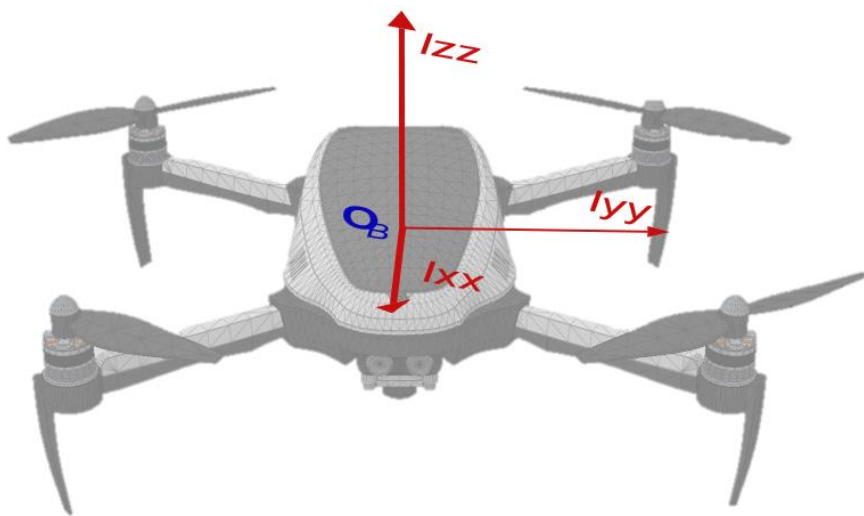


Figure 3-4 - Moment of Inertia in body frame

Rewriting the angular vector in terms of the unit vector gives

$$\vec{H}^B = H_x \hat{i} + H_y \hat{j} + H_z \hat{k} \quad 3.23$$

$$\vec{H}^B = (I_{xx} \times p \times \hat{i}) + (I_{yy} \times q \times \hat{j}) + (I_{zz} \times r \times \hat{k}) \quad 3.24$$

$$\vec{H}^B = \dot{I}_{xx}^B p \hat{i} + \dot{I}_{yy}^B q \hat{j} + \dot{I}_{zz}^B r \hat{k} + I_{xx}^B \dot{p} \hat{i} + I_{yy}^B \dot{q} \hat{j} + I_{zz}^B \dot{r} \hat{k} + I_{xx}^B p \dot{\hat{i}} + I_{yy}^B q \dot{\hat{j}} + I_{zz}^B r \dot{\hat{k}} \quad 3.25$$

The instantaneous time change of angular momentum $\dot{\vec{H}}^B$ is further simplified. Where equation 3.17 from the previous section 4.2 the $\dot{\hat{i}}, \dot{\hat{j}}$ and $\dot{\hat{k}} = \vec{\omega}^B \times (\hat{i} \text{ or } \hat{j} \text{ or } \hat{k})$

$$\dot{\vec{H}}^B = \overbrace{(I_{xx}^B \dot{p} \hat{i} + I_{yy}^B \dot{q} \hat{j} + I_{zz}^B \dot{r} \hat{k})}^{I^B \dot{\vec{\omega}}^B} + [I_{xx}^B p \overbrace{(\vec{\omega}^B \times \hat{i})}^{\dot{\hat{i}}} + I_{yy}^B q \overbrace{(\vec{\omega}^B \times \hat{j})}^{\dot{\hat{j}}} + I_{zz}^B r \overbrace{(\vec{\omega}^B \times \hat{k})}^{\dot{\hat{k}}}] \quad 3.26$$

The equation can be further simplified as

$$\dot{\vec{H}}^B = I^B \dot{\vec{\omega}}^B + (\vec{\omega}^B \times I^B \vec{\omega}^B) = \dot{\vec{H}}_{xyz} + \vec{\omega}^B \times \vec{H}^B \quad 3.27$$

The sum of the moments in time derivative is given as

$$\sum \vec{M}^B = \vec{M}_{net}^B = \begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix} = \frac{d\vec{H}^B}{dt} = \frac{d(I^B \times \dot{\vec{\omega}})}{dt} \quad 3.28$$

3.4 Newton-Euler equation

Newtonian mechanics is sometimes the most popular method for analysing mechanical systems, although it is not the only one. Another useful technique is lagrangian mechanics. The use of energies and generalised coordinates in Lagrangian mechanics rather than forces and constraints in Newtonian mechanics is, in essence, what distinguishes it from Newtonian mechanics. In comparison to Newtonian mechanics, lagrangian mechanics are also more adaptable to other physical theories. Lagrangian mechanics is more helpful in describing the laws of nature and applying them to other physical theories, such as general relativity and quantum field theory, although Newtonian mechanics is more applicable to many engineering sectors and practical usage (Hirvonen, no date).

The Newton-Euler equations combine forces and moments into a state-space (SS) form. These equations were derived in sections 3.2 and 3.3.

$$\vec{F}_{net}^B = m (\dot{\vec{v}}_{xyz}^B + \vec{\omega}^B \times \vec{v}^B) \quad 3.29$$

$$\Sigma \vec{M}^B = \vec{M}_{net}^B = \begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix} = I^B \dot{\vec{\omega}}^B + (\vec{\omega}^B \times I^B \vec{\omega}^B) \quad 3.30$$

Using matrix manipulation and mass moment of inertia matrix gives

$$\begin{bmatrix} \vec{F}^B \\ \vec{M}^B \end{bmatrix} = \begin{bmatrix} m I_{3 \times 3} & 0_{3 \times 3} \\ 0_{3 \times 3} & I^B \end{bmatrix} \begin{bmatrix} \dot{\vec{v}}^B \\ \dot{\vec{\omega}}^B \end{bmatrix} + \begin{bmatrix} \vec{\omega}^B \times (m \vec{v}^B) \\ \vec{\omega}^B \times (I^B \vec{\omega}^B) \end{bmatrix} \quad 3.31$$

Expanding equation 4.31, yields

$$\begin{bmatrix} F_x \\ F_y \\ F_z \\ M_x \\ M_y \\ M_z \end{bmatrix} = \begin{bmatrix} m & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 \\ 0 & 0 & m & 0 & 0 & 0 \\ 0 & 0 & 0 & I_{xx} & 0 & 0 \\ 0 & 0 & 0 & 0 & I_{yy} & 0 \\ 0 & 0 & 0 & 0 & 0 & I_{zz} \end{bmatrix} \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \\ \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} + \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} m u \\ m v \\ m w \end{bmatrix} + \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} I_{xx} p \\ I_{yy} q \\ I_{zz} r \end{bmatrix} \quad 3.32$$

Some literature ('Modelling, Identification and Control - Tommaso Bresciani.pdf', no date, p. 12) represents the Newton-Euler equation derived above 4.31 in a different format.

$$\vec{\Lambda}^B = M^B \dot{\vec{v}}^B + C^B(\vec{v}^B) \vec{v}^B \quad 3.33$$

$$C^B(\vec{v}^B) = \begin{bmatrix} 0_{3,3} & -mS(v^B) \\ 0_{3,3} & -S(I\omega^B) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & mw & -mv \\ 0 & 0 & 0 & -mw & 0 & mu \\ 0 & 0 & 0 & mv & -mu & 0 \\ 0 & 0 & 0 & 0 & I_{zz}r & -I_{yy}q \\ 0 & 0 & 0 & -I_{zz}r & 0 & I_{xx}p \\ 0 & 0 & 0 & I_{yy}q & -I_{xx}p & 0 \end{bmatrix} \quad 3.34$$

Note that $v^B \neq v$.

3.5 Identification of forces and moments of the quadcopter

Before Newton-Euler formulation is expressed in state-space (SS) form, it is important to identify the physical effects of quadcopter due to gyroscopic, gravitational and forces with torques from equation 3.33. $\vec{\Lambda}^B$ consists of the sum of the forces and moments.

$$\vec{\Lambda}^B = \begin{bmatrix} \vec{F}^B \\ \vec{M}^B \end{bmatrix} = \begin{bmatrix} \vec{F}_1 + \vec{F}_2 + \vec{F}_3 \dots \vec{F}_n \\ \vec{M}_1 + \vec{M}_2 + \vec{M}_3 \dots \vec{M}_n \end{bmatrix} \quad 3.35$$

3.5.1 Gravitational force

The equivalent force of the gravity vector (gravitational force) acts through the centre of mass of the quadcopter because the quadcopter is mass symmetric. Also, the centre of mass is aligned with the origin of the body frame. So, the origin of the body frame is also the centre of mass.

$$\vec{\Lambda}_G^B = \begin{bmatrix} \vec{F}_g^B \\ 0_{3 \times 1} \end{bmatrix} \quad 3.36$$

Here $\vec{\Lambda}_G^B$ is a 6x1 vector representing gravity vector acting on the body frame, and $\vec{F}_g^B$ consists of three elements F_x, F_y, F_z . Note the force of gravity acting on the drone has a negative sign, and it is represented in the inertial frame $F_g = -m \times g$. Therefore, the force of gravity in the inertial frame is given as

$$\vec{F}_g^E = R_{zyx} \times \vec{F}_g^B \quad 3.37$$

$$\vec{F}_g^B = (R_{zyx})^{-1} \times \vec{F}_g^E \quad 3.38$$

$$\therefore \vec{\Lambda}_G^B = \begin{bmatrix} \vec{F}_g^B \\ 0_{3 \times 1} \end{bmatrix} = \begin{bmatrix} (R_{zyx})^{-1} \times \vec{F}_g^E \\ 0_{3 \times 1} \end{bmatrix} = \begin{bmatrix} (R_{zyx})^{-1} \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} mg \cdot \sin\theta \\ -mg \cdot \cos\theta \cdot \sin\phi \\ -mg \cdot \cos\theta \cdot \cos\phi \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad 3.39$$

3.5.2 Gyroscopic effect on a quadcopter

The gyroscopic effect is described as the change in angular momentum's direction. The gyroscopic effect is a result of the angular momentum conservation principle. The quadrotor helicopter is capable of performing the following forces and moments: lifts caused by rotor rotation, pitching moment and rolling moment caused by the difference in lift between the four rotors, gravity, gyroscopic effect, and aerodynamic torque effect, i.e. yawing moment. Only the quadrotor helicopter with a lightweight construction displays the gyroscopic effect. Unbalanced rotational speeds of the four rotors produce a yawing moment. In essence, the gyroscopic effect and yawing moment are caused by the reaction torques. The reaction torques are the result of the rotor's rotation; their direction is the exact opposite of the rotor's rotation. When two rotors rotate in the opposite direction, the yaw moment is nullified. When the

quadrotor hovers, the gyroscopic effect and aerodynamic torque effect cancel out (He and Zhao, 2014).

When the quadcopter pitches forward, the rotation of the propellers will create a torque that would force the quadcopter to roll away from the pitch direction. If a negative pitch is executed, motors 1 and 4 will have to fight against this torque by rotating clockwise. The gyroscopic effect does not occur when the quadcopter makes a rotation about the Z-axis because the propellers attached to the body can only rotate about the same Z-axis.

Consider a propeller blade with a separate body frame axis rotating at angular velocity $\Omega \frac{\text{rad}}{\text{s}}$ as shown in Figure 3-5. The propeller has a mass moment of inertia along three axes. However, since the only axis of rotation is along the z-axis so other axes (x and y) can be disregarded. Let J_{TP} be the mass moment of inertia of the propeller blade (in $\text{kg} \cdot \text{m}^2$) along z-axis (only axis of rotation with respect to propeller rotation).

Assuming all four propeller blades are the same, by using the right-hand rule, the angular momentum $\vec{H}_M$ is determined as follows:

$$\vec{H}_1 = J_{TP}\Omega_1, \vec{H}_2 = -J_{TP}\Omega_2, \vec{H}_3 = J_{TP}\Omega_3, \vec{H}_4 = -J_{TP}\Omega_4 \quad 3.40$$

Note $\vec{H}_2$ and $\vec{H}_4$ have negative angular momentum. The use of the right-hand rule suggests a clockwise rotation resulting in a vector facing in a downward direction. In the meantime, Figure 4-4 represents the thrust forces produced by the motors. Even though the motors spin clockwise and anti-clockwise direction, the blades are aerodynamically designed to have thrust force pushing upward direction.

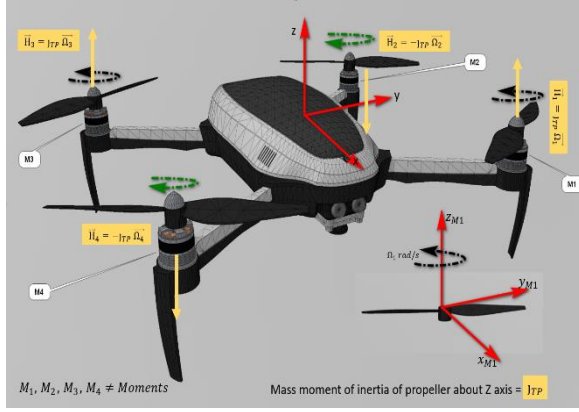


Figure 3-5 - Gyro effect - propeller properties(Ajay Joseph, 2017)

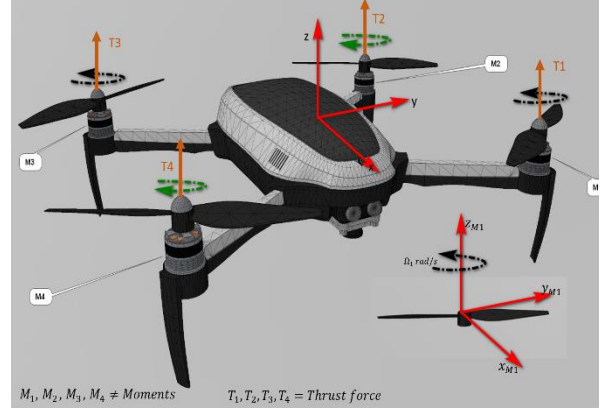


Figure 3-6 - Thrust(Ajay Joseph, 2017)

Consider the total angular velocity of the propellers (Ω_T rads/sec) to qualify the gyroscopic effects.

$$\vec{\Omega}_T = \vec{\Omega}_1 - \vec{\Omega}_2 + \vec{\Omega}_3 - \vec{\Omega}_4 \quad 3.41$$

The expression for the moment from the gyroscopic effect for each motor separately. The drone can rotate about its three-body frame, axis X, Y and Z, and the rotation about these axes produce angular velocity. This was described previously in section Kinematics of a quadcopter equation 3-3.

The total moment produced by the gyroscopic effect per motor can be expressed as:

$$M_{GRM1} = \vec{\omega}^B \times \vec{H}_{M1} = \vec{\omega}_B \times J_{TP} \times \vec{\Omega}_1 \quad 3.42$$

$$M_{GRM2} = \vec{\omega}^B \times \vec{H}_{M2} = \vec{\omega}_B \times -J_{TP} \times \vec{\Omega}_2 \quad 3.43$$

$$M_{GRM3} = \vec{\omega}^B \times \vec{H}_{M3} = \vec{\omega}_B \times J_{TP} \times \vec{\Omega}_3 \quad 3.44$$

$$M_{GRM4} = \vec{\omega}^B \times \vec{H}_{M4} = \vec{\omega}_B \times -J_{TP} \times \vec{\Omega}_4 \quad 3.45$$

Here M_{GRM1} represents the gyroscopic effect produced by motor 1 and $\vec{H}_{M1}$ is the angular momentum of motor 1. Expanding equation for M1 gives the following

$$\vec{M}_{GRM1} = \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ H_1 \end{bmatrix} = \begin{bmatrix} \vec{i} & \vec{j} & \vec{k} \\ p & q & r \\ 0 & 0 & H_1 \end{bmatrix} = \vec{i} \begin{bmatrix} q & r \\ 0 & H_1 \end{bmatrix} - \vec{j} \begin{bmatrix} p & r \\ 0 & H_1 \end{bmatrix} + \vec{k} \begin{bmatrix} p & q \\ 0 & 0 \end{bmatrix} \quad 3.46$$

$$\therefore \vec{M}_{GRM1} = \vec{i}(q \cdot H_1) - \vec{j}(p \cdot H_1) + \vec{k} \cdot 0 = \begin{bmatrix} qH_1 \\ -pH_1 \\ 0 \end{bmatrix} = \begin{bmatrix} q J_{TP}(-\Omega_1) \\ -p J_{TP}(\Omega_1) \\ 0 \end{bmatrix} \quad 3.47$$

$$\vec{M}_{GRM2} = \begin{bmatrix} -q J_{TP}(\Omega_2) \\ p J_{TP}(\Omega_2) \\ 0 \end{bmatrix}; \vec{M}_{GRM3} = \begin{bmatrix} q J_{TP}(\Omega_3) \\ -p J_{TP}(\Omega_3) \\ 0 \end{bmatrix}; \vec{M}_{GRM4} = \begin{bmatrix} -q J_{TP}(\Omega_4) \\ p J_{TP}(\Omega_4) \\ 0 \end{bmatrix} \quad 3.48$$

The total moment is given as

$$\vec{M}_{GR} = \vec{M}_{GRM1} + \vec{M}_{GRM2} + \vec{M}_{GRM3} + \vec{M}_{GRM4} \quad 3.49$$

$$\vec{M}_{GR} = \begin{bmatrix} q J_{TP}(-\Omega_1) \\ -p J_{TP}(\Omega_1) \\ 0 \end{bmatrix} + \begin{bmatrix} -q J_{TP}(\Omega_2) \\ p J_{TP}(\Omega_2) \\ 0 \end{bmatrix} + \begin{bmatrix} q J_{TP}(\Omega_3) \\ -p J_{TP}(\Omega_3) \\ 0 \end{bmatrix} + \begin{bmatrix} -q J_{TP}(\Omega_4) \\ p J_{TP}(\Omega_4) \\ 0 \end{bmatrix} \quad 3.50$$

$$\vec{M}_{GR} = \begin{bmatrix} q J_{TP}(\Omega) \\ -p J_{TP}(\Omega) \\ 0 \end{bmatrix} N.m \quad 3.51$$

Here $\Omega = \Omega_T$ as shown by equation 4.41.

Substitute the gyroscopic moment to force and moment vector $\vec{\Lambda}_{GR}^B$.

$$\vec{\Lambda}_{GR}^B = \begin{bmatrix} 0 \\ 0 \\ 0 \\ q J_{TP}(\Omega) \\ -p J_{TP}(\Omega) \\ 0 \end{bmatrix} \quad 3.52$$

3.5.3 Control moment/torque

Finally, the quadcopter is exposed to the four control inputs U_1, U_2, U_3 , and U_4 as shown in Figure 4-5.

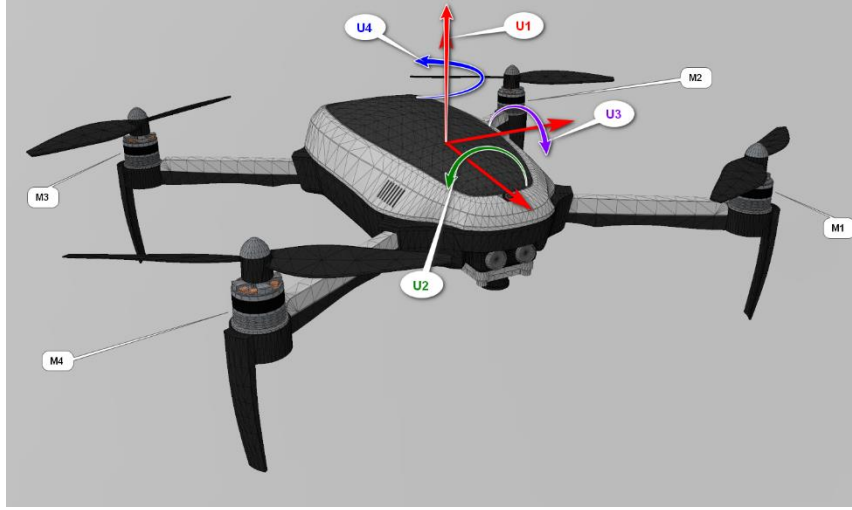


Figure 3-7 - Control inputs/moment/torque (Ajay Joseph, 2017)

Let $\vec{\Lambda}_{IN}^B$ be control input vector. The difference between the force of gravity and control inputs forces/torques is that control inputs are represented in the body frame. Since it is expressed in the body frame, the rotational matrix and transfer matrix are not required.

$$\vec{\Lambda}_{IN}^B = \begin{bmatrix} 0 \\ 0 \\ U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} \quad 3.53$$

The thrust acts in the body frame direction, hence U_1 is in the third element and the U_2, U_3, U_4 are control moments for roll pitch and yaw, respectively.

The total net force and moment vector as given in section 3.5 by eq. 4.35 can be expressed in terms of three different force moment vectors. This is given by equation 4.54.

$$\vec{\Lambda}^B = \vec{\Lambda}_{GR}^B + \vec{\Lambda}_G^B + \vec{\Lambda}_{IN}^B \quad 3.54$$

3.6 Quadcopter State Variable definition

The state variable is a function that describes how the world changes with time. Aircraft designers use this relationship between velocity and position to compute forces acting on the aeroplane as it moves through space, allowing them to predict how an airframe will perform in flight. In this way, derived mathematical models of the aircraft design help predict the results more accurately using real experimental data instead of relying on assumptions or testing in wind tunnels.

Rearranging the Newton-Euler form from equation 4.33 and substituting equation 4.54 allows the quadcopter model to be represented in state-space (SS) form.

$$\vec{\Lambda}^B = M^B \dot{\vec{v}}^B + C^B \vec{v}^B \quad 3.55$$

$$\dot{\vec{v}}^B = (M^B)^{-1} [-C^B \vec{v}^B + \vec{\Lambda}^B] \quad 3.56$$

$$\dot{\vec{v}}^B = (M^B)^{-1} [-C^B \vec{v}^B + \vec{\Lambda}_{GR}^B + \vec{\Lambda}_G^B + \vec{\Lambda}_{IN}^B] \quad 3.57$$

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \\ \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \frac{1}{m} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{m} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{m} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{I_{xx}} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{I_{yy}} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{I_{zz}} \end{bmatrix} \begin{bmatrix} -mwq + mvr + mg \sin\theta + 0 + 0 \\ mwp - mur - mp \cos\theta \sin\phi + 0 + 0 \\ -mvp + muq - mg \cos\theta \cos\phi + 0 + U_1 \\ -I_{zz} r q + I_{yy} . t q + 0 + q J_{TP} \Omega + U_2 \\ I_{xx} r p - I_{xx} p r + 0 - p J_{TP} \Omega + U_3 \\ -I_{yy} q p + I_{xx} p q + 0 + 0 + U_4 \end{bmatrix} \quad 3.58$$

$$\dot{u} = vr - wq + g . \sin\theta$$

$$\begin{aligned} \dot{v} &= wp - wr - g . \cos\theta . \sin\phi \\ \dot{w} &= uq - up - g . \cos\theta . \cos\phi + \frac{U_1}{m} \\ \dot{p} &= \frac{I_{yy} - I_{zz}}{I_{xx}} q r + \frac{J_{TP}}{I_{xx}} q \Omega + \frac{U_2}{I_{xx}} \\ \dot{q} &= \frac{(I_{zz} - I_{xx})}{I_{yy}} p r - \frac{J_{TP}}{I_{yy}} p \Omega + \frac{U_3}{I_{yy}} \\ \dot{r} &= \frac{I_{xx} - I_{yy}}{I_{zz}} p q + \frac{U_4}{I_{zz}} \end{aligned} \quad 3.59$$

The above equations, 3.59 will be simplified and used in the control algorithms later. The kinematics equation can also be used to apply the angular equations to angular fixed references.

The mathematical notations used to analyse the quadcopter's dynamics and its state variables are expressed, Table 3-1 below.

Table 3-1 - State Variables

Vector	Sub-Vector	State	Description
ξ	Γ_E Linear Position	X	The position of the centre of gravity in the inertial frame
		Y	The position of the centre of gravity in the inertial frame
		Z	The position of the centre of gravity in the inertial frame
	Θ_E Angular Position	φ	Roll angle
		θ	Pitch angle
		ψ	Yaw angle
ν	ν_B linear velocity	u	X linear velocity of the centre of gravity in the body-fixed frame with respect to the inertial frame
		v	Y linear velocity of the centre of gravity in the body-fixed frame with respect to the inertial frame
		w	Z linear velocity of the centre of gravity in the body-fixed frame with respect to the inertial frame
	ω_B angular velocity	p	Roll angular velocity in the body-fixed frame with respect to the inertial frame
		q	Pitch angular velocity in the body-fixed frame with respect to the inertial frame
		r	Yaw angular velocity in the body-fixed frame

3.7 Aerodynamic properties of the propeller blade

Airfoil theory is a branch of aerodynamics that deals with the study of the forces that are produced by the interaction of a moving airfoil (the wing of an aeroplane or a sailboat) and the air.

A relatively simple method of predicting the performance of a propeller (as well as fans or windmills) is the use of Blade Element Theory. The methodology is based upon the assumption that the blade is composed of several small, simple airfoils and that the resultant forces and moments can be calculated by considering the way in which the fluid/air interacts with the blade. In practice, the method is extremely difficult to implement as a complete model, and as such, the analysis is often simplified to a single airfoil section, Figure 3-8.

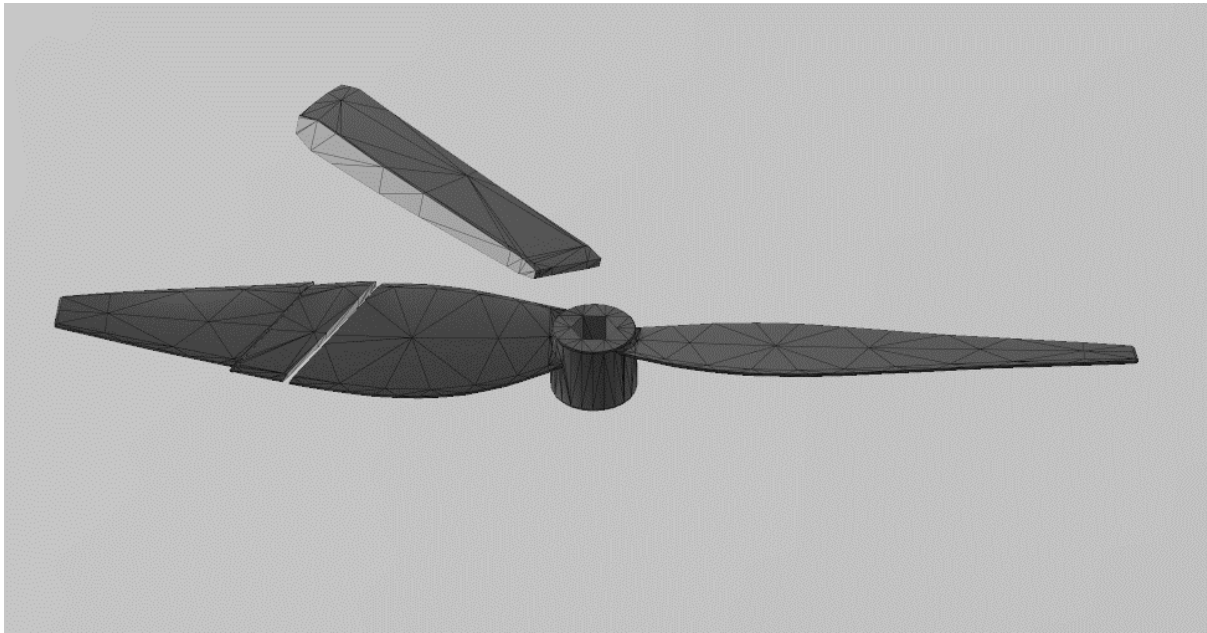


Figure 3-8 – Airfoil (Ajay Joseph, 2017)

As an example, a propeller blade may be assumed to have a uniform cross-section and an elliptical (or circular) shape. The airfoil section at the blade tip is defined as a circular arc, and the cross-sectional area is assumed to be constant. The chord length, B in Figure 3-9 is the distance from the blade root to a given point on the blade chord and is the characteristic distance over which the flow is assumed to be uniform. The blade angle, α , is the angle between the blade chord at a given point and the blade span and is the angle between the blade chord and the plane of rotation.

The blade element theory assumes that the blade can be decomposed into a number of small airfoils, as shown in Figure 3-9-1. The airfoil can be assumed to be an infinitesimally small, thin, flat plate, and the forces acting on this cross-section are similar to those of a full-scale airfoil. The forces acting on the airfoil lift, drag, and moment, Figure 3-9, can be calculated using standard formulas for the respective forces, and the total force on the blade can be found by summing the forces on each airfoil section.

The moment is due to the torque applied to the blade by the power source. The torque must be reacted by the blade, and the moment is calculated by multiplying the torque by the distance from the axis of rotation to the airfoil. The reaction torque is calculated by summing the moments on each airfoil section.

The drag force is due to the fluid acting on the surface of the airfoil and is calculated based on the airfoil shape and the velocity of the fluid at the point. The force is calculated by summing the drag forces on each airfoil section.

The lift force is due to the pressure acting above the airfoil and the velocity acting below the airfoil. The pressure above the airfoil is the static pressure plus the dynamic pressure, and the total pressure above the airfoil is found by summing the pressures on each airfoil section. The dynamic pressure is based on the velocity of the airfoil. The static pressure is found by summing the pressures on each airfoil section. The lift is found by calculating the angle between the lift vector and the chord line.

The friction drag is due to the friction between the blade and the surrounding fluid. This value is found by summing the drag forces on each airfoil section. The interference drag is due to the blade passing through the air at a different speed than the air passing over the blade. The interference drag is found by summing the drag forces on the airfoil sections on the back and front of the blade.

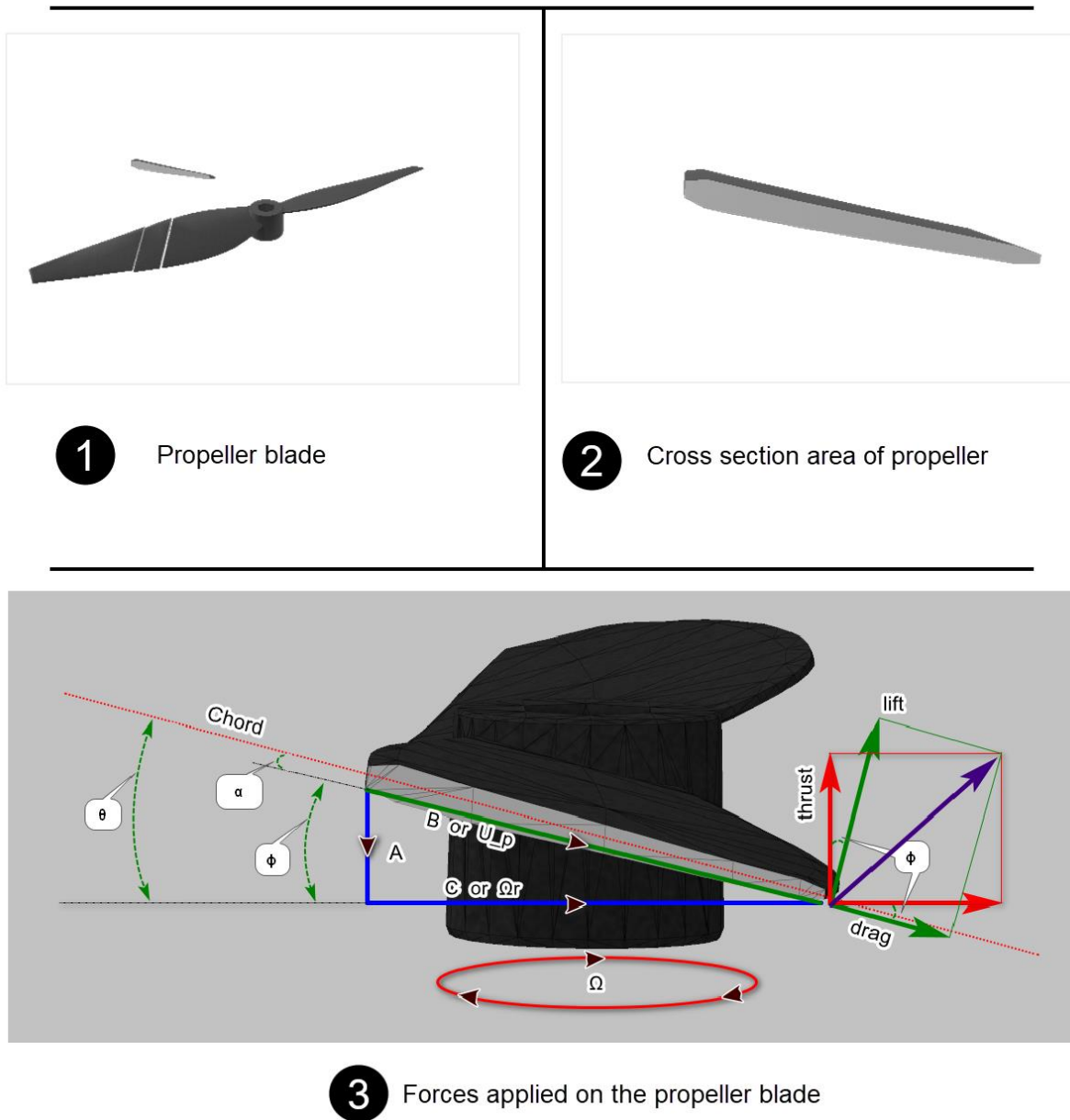


Figure 3-9 - Propeller blade (Ajay Joseph, 2017)

In Figure 3-9-3, the A, B and C components and angles ($\theta, \phi, \alpha \neq \text{euler angles}$) represent

- A or V_p – Rotor-induced air velocity due to stream tube effect in $\frac{m}{s}$
- C – Air velocity Ωr in m/s
- B or U_p – Resultant vector of A and C in m/s
- θ – Angle between air velocity and the chord line of a propeller blade (pitch angle of a prop)
- α – Angle of attack formed between resultant vector and chord line.
- ϕ – Angle between the air velocity vector and resultant vector.

The magnitude of the resultant vector B or U_p can be calculated using the Pythagoras theorem.

$$B = \sqrt{A^2 + C^2} \text{ or } U_p = \sqrt{V_p^2 + \Omega r^2} \quad 3.60$$

Due to the resultant air velocity, the airfoil will exert an aerodynamic force dF as shown in Figure 3-10. The total aerodynamic force is the sum of the pressure on the element and the force due to the circulation. The pressure on the element is directed away from the axis of rotation and is equal in magnitude to the force due to the circulation. The total aerodynamic force is, therefore, the sum of the pressure on the element and the force due to the circulation.

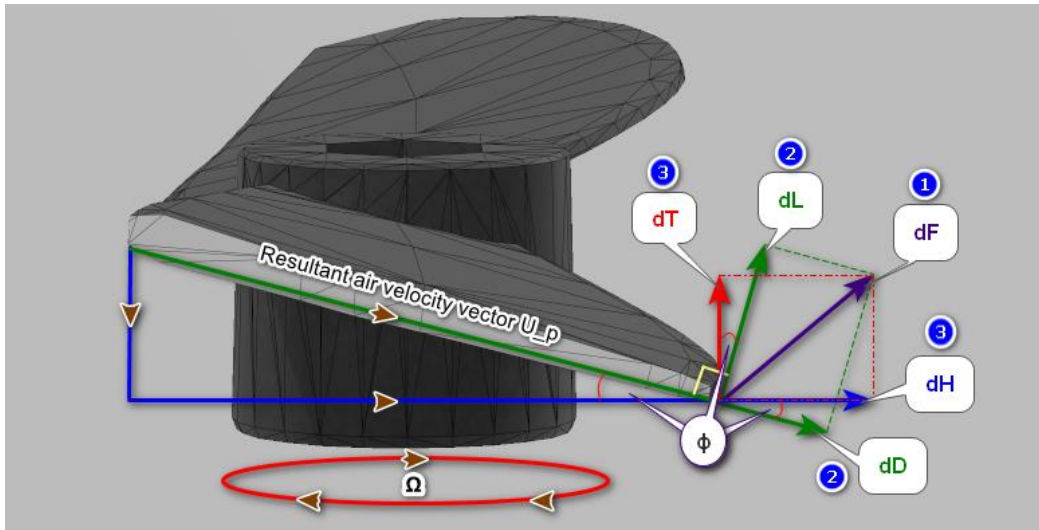


Figure 3-10 – Determine differential thrust (Ajay Joseph, 2017)

Here

- dL – Differential lift force
- dD – Differential drag
- dT – Differential thrust
- dF – Resultant force due to air velocity

The objective is to determine the thrust generated by each propeller. The differential thrust is a function of differential lift force and differential drag force.

$$dT = dL \times \cos(\phi) - dD \times \sin(\phi) \quad 3.61$$

$$dL = C_l \times \frac{1}{2} \times \rho \times U_p^2 \times c_p \times dr \quad 3.62$$

$$dD = C_d \times \frac{1}{2} \times \rho \times U_p^2 \times c_p \times dr \quad 3.63$$

Here C_l is a dimensionless quantity, and it is known as the lift coefficient. This quantity is dependent on three parameters: α, M, Re which represents the angle of attack, Mach number and Reynolds number, respectively. The angle of attack is defined as the angle between the chord line of the wing and the relative wind direction, Figure 3-9-3. It is measured by the angle between the chord line of the wing and the velocity vector of the flow. It is not the same as the angle of incidence. Typically, an angle of attack in the range of 0° to 10° is used. The Mach number is the ratio between the speed of the air and the speed of sound. The Reynolds number is the ratio between the density of air ρ , velocity of air V and viscous forces μ of the fluid; $Re = \frac{\rho V \times L}{\mu}$. It depends on the size and shape of the airfoil and its speed. The Reynolds number is typically used to describe the boundary layer of the air. The boundary layer is the layer of the flow where viscous and turbulent effects are important. It describes whether the blade will provide a smooth laminar flow or turbulent flow, as shown in Figure 3-11.

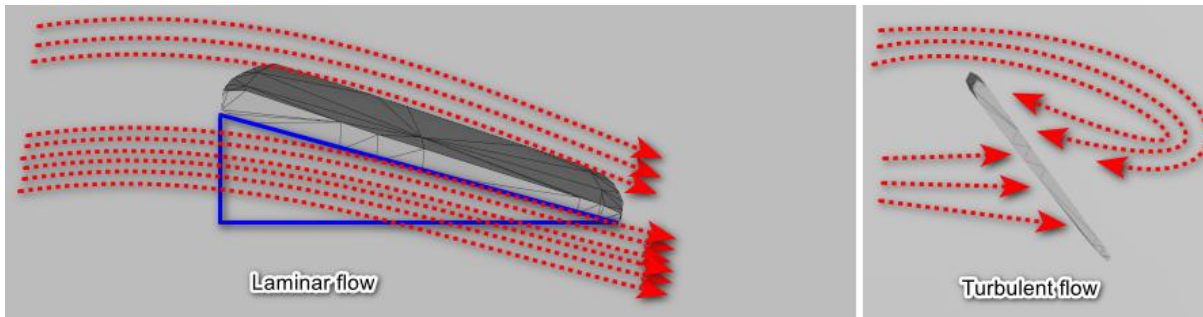


Figure 3-11 - Laminar and turbulent flow (Ajay Joseph, 2017)

So, from the equation 3.62 dL , ρ is the air density in $\frac{kg}{m^3}$, U_p^2 is the resultant force vector, c_p is the chord length and dr is the differential width of the airfoil.

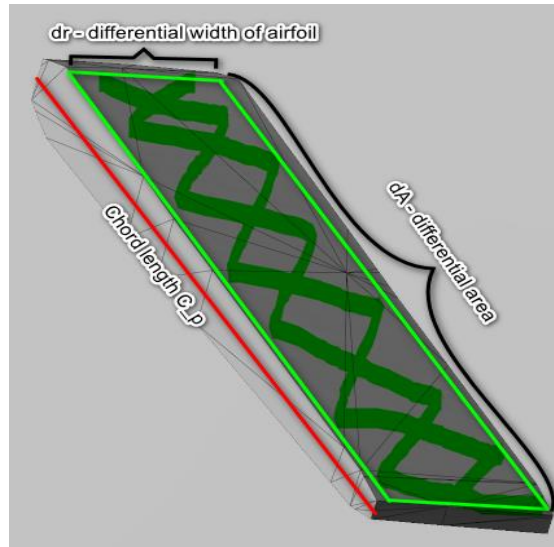


Figure 3-12 - Airfoil differential width (Ajay Joseph, 2017)

Finally, the drag coefficient C_d mentioned in equation 3.63 is another dimensionless quantity describing the contribution of drag force on an airfoil. The higher the drag coefficient, the higher the differential drag force is larger and vice-versa. Different shapes have different drag coefficients, and some of them are shown in Figure 3-13. A lower drag coefficient is desirable in aerospace design, and the lowest drag coefficient is achieved by having a streamlined body. Note that the lift coefficient and drag coefficients describe an object's shape properties, not its size.

Shape		Drag Coefficient
Sphere		0.47
Half-sphere		0.42
Cube		1.05
Streamlined Body		0.04
Streamlined Half-body		0.09

Measured Drag Coefficients

Figure 3-13 - Drag coefficients of various shapes (Gustafsson, 2009)

As mentioned above in equation 3.60, the resultant force U_p can be determined by.

$$U_p = \sqrt{A^2 + C^2} = \sqrt{(\Omega r)^2 + V_p^2} \quad 3.64$$

Therefore, the differential drag force;

$$dL = C_l \times \frac{1}{2} \times \rho \times [(\Omega r)^2 + V_p^2] \times c_p \times dr \quad 3.65$$

$$dD = C_d \times \frac{1}{2} \times \rho \times [(\Omega r)^2 + V_p^2] \times c_p \times dr \quad 3.66$$

Assuming the induced air velocity on the airfoil is close to zero or $V_p \approx \Omega r$, i.e. by taking a small-angle approximation of ϕ , Figure 3-9-3), the equation can be re-written as follows;

$$dL = C_l \times \frac{1}{2} \times \rho \times \Omega r^2 \times c_p \times dr \quad 3.67$$

$$dD = C_d \times \frac{1}{2} \times \rho \times \Omega r^2 \times c_p \times dr \quad 3.68$$

This results in differential thrust force dT to be, using equation 3.60;

$$dT = dL \times \cos(\phi) - dD \times \sin(\phi) = dL - dD \times \phi \quad 3.69$$

Substitute differential lift and drag coefficients to differential thrust coefficient equation

$$dT = (dL - dD \times \phi) = (C_l \times \frac{1}{2} \times \rho \times \Omega^2 r^2 \times c_p \times dr - C_d \times \frac{1}{2} \times \rho \times \Omega^2 r^2 \times c_p \times dr) \times \phi \quad 3.70$$

Rearranging the equation will result in thrust force with respect to r , $\frac{dT}{dr}$

$$\frac{dT}{dr} = \Omega^2 \times \frac{1}{2} \times \rho \times c_p [C_l - C_d \times \phi] r^2 \quad 3.71$$

Now integrate to determine the thrust produced by the propeller T_{m1}

$$\int_0^{T_{m1}} dT = 2 \times \int_0^R \Omega^2 \times \frac{1}{2} \rho c_p [C_l - C_d \times \phi] r^2 dr \quad 3.72$$

$$T_{m1} = \Omega^2 \times \rho \overbrace{\int_0^R c_p [C_l - C_d \times \phi] r^2 dr}^{C_T} \quad 3.73$$

$$T_{m1}(\Omega) = C_T(\Omega^2) \quad 3.74$$

These results can be expressed in input control vectors where U_1 is total thrust acting on the body frame direction and the U_2, U_3, U_4 are control moments for roll pitch and yaw, respectively.

$$U_1 = T_{m_1} + T_{m_2} + T_{m_3} + T_{m_4} \quad 3.75$$

$$U_1[N] = C_T \Omega_1^2 + C_T \Omega_2^2 + C_T \Omega_3^2 + C_T \Omega_4^2 \quad 3.76$$

$$U_2[Nm] = M_{M_2} - M_{M_4} = C_T \times \Omega_2^2 \times l - C_T \times \Omega_4^2 \times l = C_T \times l(\Omega_2^2 - \Omega_4^2) \quad 3.77$$

$$U_3[Nm] = M_{M_3} - M_{M_1} = C_T \times \Omega_3^2 \times l - C_T \times \Omega_1^2 \times l = C_T \times l(\Omega_3^2 - \Omega_1^2) \quad 3.78$$

Here l represents the length from the axis of rotation to the propeller. To determine U_4 the horizontal component dH must be calculated, Figure 3-10. This is established by taking dL and dD vectors as follows.

$$dH [N] = dL \times \sin(\phi) + dD \times \cos(\phi) \quad 3.79$$

As mentioned above, the ϕ is considered a small angle, so the equation can be written as follows.

$$dH = dL \times \phi + dD \quad 3.80$$

Using the above equation, the differential air resistance moment dQ shown in Figure 3-14 can be determined.

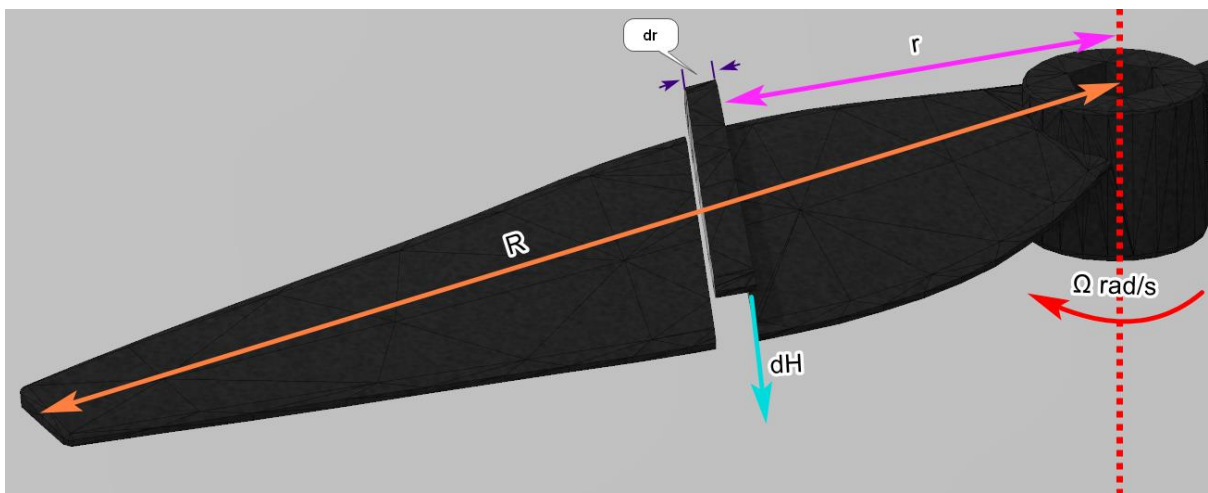


Figure 3-14 - Differential air resistance moment (Ajay Joseph, 2017)

$$dQ[Nm] = dH \times r = (dL \times \phi + dD) \times r \quad 3.81$$

As per equations 3.62 and 3.63, the differential air resistance can be written as

$$dQ = [C_l \times \frac{1}{2} \times \rho \times U_p^2 \times c_p \times dr + C_d \times \frac{1}{2} \times \rho \times U_p^2 \times c_p \times dr] \times r \quad 3.82$$

$$dQ = \Omega^2 \times \frac{1}{2} \times \rho \times C_p \times r^3 [C_l \times \phi + C_d] \times dr \quad 3.83$$

The total air resistance moment can be calculated by integrating from the centre to the end of the blade.

$$\int_0^Q dQ = \int_0^R \Omega^2 \times \frac{1}{2} \times \rho \times C_p \times r^3 [C_l \times \phi + C_d] \times dr \quad 3.84$$

Since there are two blades per motor, the equation is then multiplied by two. This results in a simplified equation per motor (e.g., motor 1).

$$\int_0^{Q_{M1}} dQ = 2 \times \int_0^R \Omega_1^2 \times \frac{1}{2} \times \rho \times C_p \times r^3 [C_l \times \phi + C_d] \times dr \quad 3.85$$

$$Q_{M1} = \Omega_1^2 \times \rho \int_0^R C_p \times r^3 [C_l \times \phi + C_d] \times dr \quad 3.86$$

The moment applied by the quadcopter per motor $M_{D_{M1}}$ is given as

$$M_{D_{M1}} = -Q_{M1} \quad 3.87$$

Therefore,

$$M_{D_{M1}} = \Omega^2 \times (-1) \times \rho \int_0^R C_p \times r^3 [C_l \times \phi + C_d] \times dr \quad 3.88$$

The torque factor C_Q is the simplified version of equation 3.87.

$$M_{D_{M1}} = C_Q \times \Omega_1^2 \quad 3.89$$

Note $C_Q \neq C_T$ where C_T is the thrust factor (see equation 3.74) in $N.s^2$ and C_Q is known as torque factor in $N.m.s^2$.

The total moment applied can be expressed as M_{DM_T}

$$M_{DM_T} = M_{DM_1} + M_{DM_2} + M_{DM_3} + M_{DM_4} \quad 3.90$$

$$M_{DM_T} = C_Q\Omega_1^2 - C_Q\Omega_2^2 + C_Q\Omega_3^2 - C_Q\Omega_4^2 \quad 3.91$$

Using the right-hand rule Ω_2 and Ω_4 are rotating clockwise, which makes them negative. In other words, $M_{DM_2} = -C_Q\Omega_2^2$ and $M_{DM_4} = -C_Q\Omega_4^2$

So the net moment applied by the quadcopter to the motors is,

$$M_{DM_T} = C_Q(\Omega_1^2 - \Omega_2^2 + \Omega_3^2 - \Omega_4^2) \quad 3.92$$

If the net moment of rotors is in the positive direction, then the quadcopter will experiment moment in the opposite direction U_4 .

$$U_4[Nm] = -M_{DM_T} = -C_Q(\Omega_1^2 - \Omega_2^2 + \Omega_3^2 - \Omega_4^2) \quad 3.93$$

$$U_4[Nm] = C_Q(-\Omega_1^2 + \Omega_2^2 - \Omega_3^2 + \Omega_4^2) \quad 3.94$$

Therefore, the system of equation U_1, U_2, U_3 , and U_4 are

$$U_1[N] = C_T\Omega_1^2 + C_T\Omega_2^2 + C_T\Omega_3^2 + C_T\Omega_4^2 \quad 3.95$$

$$U_2[Nm]l = C_Tl(\Omega_2^2 - \Omega_4^2)$$

$$U_3[Nm] = C_Tl(\Omega_3^2 - \Omega_1^2)$$

$$U_4[Nm] = C_Q(-\Omega_1^2 + \Omega_2^2 - \Omega_3^2 + \Omega_4^2)$$

Rearrange the system of equations in matrix form,

$$\begin{bmatrix} \overrightarrow{U} \\ \frac{U_1}{C_T} \\ \frac{U_2}{C_T \times l} \\ \frac{U_3}{C_T \times l} \\ \frac{U_4}{C_Q} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & -1 \\ -1 & 0 & 1 & 0 \\ -1 & 1 & -1 & 1 \end{bmatrix} \times \begin{bmatrix} \overrightarrow{\Omega_T^2} \\ \Omega_1^2 \\ \Omega_2^2 \\ \Omega_3^2 \\ \Omega_4^2 \end{bmatrix} \quad 3.96$$

By inverting the system of equations to make $\Omega_{1 \rightarrow 4}$ as the subject in matrix form gives.

$$\begin{aligned} S \overrightarrow{\Omega_T^2} &= \overrightarrow{U} \\ S^{-1} \times S \times \overrightarrow{\Omega_T^2} &= \overrightarrow{U} \\ \therefore \overrightarrow{\Omega_T^2} &= S^{-1} \overrightarrow{U} \end{aligned} \quad 3.97$$

Here, the inverse matrix S^{-1}

$$S^{-1} = \begin{bmatrix} \frac{1}{4} & 0 & -\frac{1}{2} & -\frac{1}{4} \\ \frac{1}{4} & \frac{1}{2} & 0 & \frac{1}{4} \\ \frac{1}{4} & 0 & \frac{1}{2} & -\frac{1}{4} \\ \frac{1}{4} & -\frac{1}{2} & 0 & \frac{1}{4} \end{bmatrix} \quad 3.98$$

$$\begin{bmatrix} \Omega_1^2 \\ \Omega_2^2 \\ \Omega_3^2 \\ \Omega_4^2 \end{bmatrix} = \begin{bmatrix} \frac{1}{4} & 0 & -\frac{1}{2} & -\frac{1}{4} \\ \frac{1}{4} & \frac{1}{2} & 0 & \frac{1}{4} \\ \frac{1}{4} & 0 & \frac{1}{2} & -\frac{1}{4} \\ \frac{1}{4} & -\frac{1}{2} & 0 & \frac{1}{4} \end{bmatrix} \times \begin{bmatrix} \frac{U_1}{C_T} \\ \frac{U_2}{C_T \times l} \\ \frac{U_3}{C_T \times l} \\ \frac{U_4}{C_Q} \end{bmatrix} \quad 3.99$$

3.8 Actuator Dynamics (DC-motor)

The brushless dc motor is made of permanent magnets and the electromagnets are placed on its shafts. When a current is passed through the coils, they generate an electromagnetic field with North and South poles that repel each other, thus causing rotation around both shafts' common axis. The direction of movement can be reversed by reversing the polarity of the current in a DC circuit. This technology provides many advantages over brushed motors - there is no rotor or brush wear because there are no commutators and brushes. Smaller, lighter, and more efficient, however complex to operate than brushed motors.

To model the system, three steps have been adopted, firstly model the electrical motor; secondly, find a relation between the angular velocity of the shaft and the voltage applied; and finally, establish a relation between the thrust force, angular speed, and voltage which all these parameters are considered as variable linked with constant.

3.8.1 Voltage and angular velocity

The electrical representation of the DC motor is shown in Figure 4-6, a model which binds electrical and mechanical quantities. The model consists of an inductor $L(H)$, resistor $Z(\Omega_m)$ and a voltage supply $e(V)$ (‘Heba_ElKholly_Thesis_S2014.pdf’, no date).

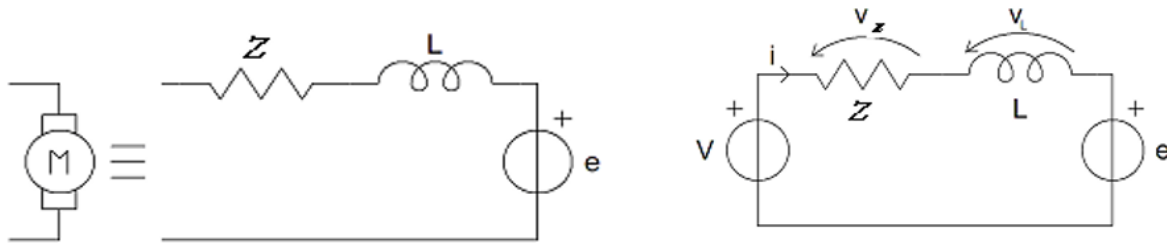


Figure 3-15 - DC motor model (Poyi, 2014)

The following equation is determined by using Kirchhoff's voltage law

$$v = v_z + v_L + e \quad 3.100$$

Where voltage across the resistor R and inductor L is expressed as $V_R [V]$ and $V_L [V]$ respectively.

$$v = Zi + \frac{L\delta i}{\delta t} + K_e\Omega \quad 3.101$$

Here $i[A]$ is motor current, K_e is motor constant in $\left[\frac{Vs}{rad}\right]$ and Ω_m is motor angular speed in $\frac{rad}{s}$.

However, to simplify equation 4.61, there are a few assumptions that must be made. As very known in the modern small DC-Motor, the inductance effect could be neglected due to its small value.

$$v = Zi + K_e \Omega \quad 3.102$$

Now the second part of the rotor dynamics considered is the relationship between the motor torque, the propeller torque and the angular speed, which is expressed in the following equation,

$$J\dot{\Omega} = \tau_m - \tau_d \quad 3.103$$

Here J is the motor moment of inertia in Nms^2 , $\dot{\Omega}_m$ represents the motor angular acceleration in $\frac{rad}{s^2}$, τ_m and τ_d are the motor torque and the load torque in Nm .

To find a relationship between the voltage and the angular velocity of the propeller, the following equations below have been used.

$$\tau_m = K_q i \quad 3.104$$

Where the motor torque τ_m is proportional to the electrical current i through K_q [Nm/A]. From the electrical power ($P_E[W]$) and mechanical power ($P_M[Nm/s]$) assumes the two constants K_e and K_q have the same size even if the physical unit changes('Heba_ElKholy_Thesis_S2014.pdf', no date).

$$P_E = P_M \quad \left| \begin{array}{l} P_E = ei = K_e i \Omega \\ P_M = \tau_m \Omega = K_q i \Omega \end{array} \right. \Rightarrow K_e = K_q \quad 3.105$$

The Drag torque is proportional to the square of the propeller's speed by the drag coefficient and is expressed in the following equation

$$\tau_d = D\Omega^2 \quad 3.106$$

The relationship between the angular velocity and voltage can be expressed by substituting equations 4.64, 3.105 and 4.66

$$V = \frac{ZD\Omega^2}{K_q} + \frac{ZJ\dot{\Omega}}{K_q} + K_e\Omega \quad 3.107$$

3.8.2 Voltage and thrust of the motor

The final step of the rotor dynamic is to determine a relation between the voltage [V], the angular velocity [Ω] and the thrust force [F], which is produced by each motor torque is the input of the quadcopter system('Heba_ElKholy_Thesis_S2014.pdf', no date).

From the given equation 4.64, the applied motor current can be represented as follows

$$\frac{\tau_m}{K_q} = i \quad 3.108$$

By using Joule's law, the electrical power can be determined.

$$P = VI = \frac{\tau_m}{K_q} V \quad 3.109$$

The mechanical output power of η motor's efficiency

$$P_m = \eta P = \eta \frac{\tau_m}{K_q} V \quad 3.110$$

The ratio of induced power P_h to the mechanical power put into the propeller P_m is represented as a figure of merit f .

$$f = \frac{P_h}{P_m} \quad 3.111$$

Rearrange equation 4.71 and substitute it with equation 4.70 gives

$$P_h = \eta f \frac{\tau_m}{K_q} V \quad 3.112$$

Ideally the power P_h applied at hover equals the thrust force times the speed is represented as

$$P_h = Fv_h \quad 3.113$$

The following equation is obtained by using the momentum theory

$$v_h = \sqrt{\frac{F}{2\rho A}} \quad 3.114$$

Now substitute the equations 4.73 to 4.74

$$\eta f \frac{bF}{Kq} V = F \sqrt{\left(\frac{F}{2\rho A}\right)} \quad 3.115$$

Finally, the quadcopter thrust output and voltage input expression is represented by rearranging equation 4.75.

$$F = 2\rho A \left[\frac{f\eta b}{Kq} \right]^2 V^2 \quad 3.116$$

Control system overview

To control any kind of prototype, some useful techniques could be deployed. This section is an introduction to the two technics adopted in this thesis to control the system which are the Kalman filter and the PID controller.

3.8.3 Kalman filter

The Kalman filter is a predictor-corrector for the linear-quadratic problem, which is the key issue for estimating an instantaneous linear dynamic system perturbed by Gaussian white noise in order to reduce the calculated error covariance (*Dr. Kostas Alexis - Dr. Kostas Alexis Personal Website*, no date).

The invention of statistical method estimators has allowed electronic systems to monitor the most complex dynamic systems, such as ongoing manufacturing processes, aircraft, ships, and spacecraft. To manipulate any dynamic system, however, a good understanding of the system to its closest precision must be created using state-space (SS) representation or a model of the dynamic process using differential equations. The Kalman filter will use the dynamic model to provide the mean for the missing information, forecast the future of the dynamic system, and monitor the mechanism that is difficult to control. From a practical standpoint, these are the

Kalman filter mathematical tool (it does not solve the problem, but it improves the efficiency of the system), a computer programme, statistical characterization of a problem estimation, and a learning method in a limited context (Dorf and Bishop, 2010).

The objective of KF's mathematical equations is to estimate the state space variable x of a discrete time-controlled mechanism governed by the linear stochastic difference equation (Dorf and Bishop, 2010).

$$X_k = A_{x_{k-1}} + B u_k + W_{k-1} \quad 3.117$$

$$Y_k = C x_k + v_k \quad 3.118$$

Here, the matrix $A_{n \times n}$ elements relate to the previous time step state vector x_{k-1} to the current state vector x_k . The matrix $B_{n \times 1}$ elements relate the control input U to the state vector x . The matrix $C_{m \times n}$ elements which relate the measurement x state to the measurement y . W and V represent a random variable of the process and the measurement noise, respectively. They are assumed to be white and independent from each other, with normal probability distributions:

$$\begin{aligned} P(w) &= N(0, Q) \\ P(V) &= N(0, R) \end{aligned} \quad 3.119$$

The KF process the measurements in two stages: In the next step state of the system is predicted based on the previous measurement, and the current state of the system is calculated based on the measurement at the current time step (Jansen, no date).

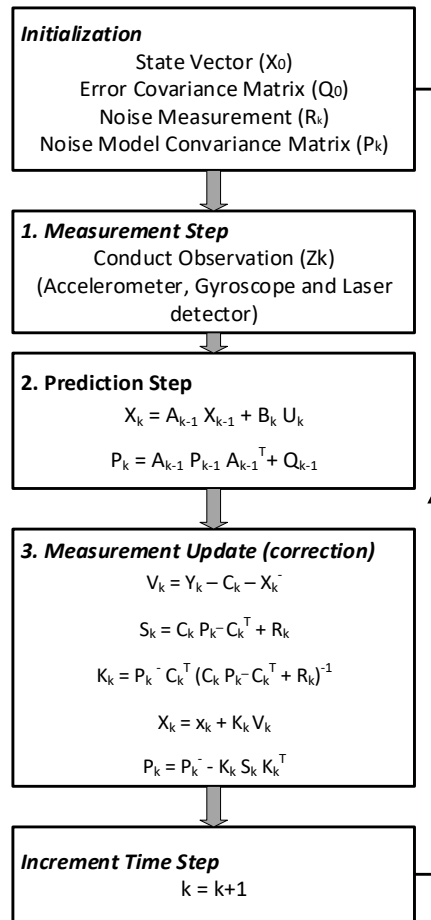


Figure 3-16 - Flow diagram of KF (Stevens and Lewis, 2003)

Where the notation used is:

- X_k and P_k are the predicted mean and covariance of the state, respectively, on the time step k before seeing the measurement.
- $\hat{X}_k$ and $\hat{P}_k$ are the estimated mean and covariance of the state, respectively, on time step k after seeing the measurement.
- Y_k is the mean of the measurement of time step k .
- V_k is the innovation or the measurement residual on time step k .
- S_k is the measurement prediction covariance on the time step k .
- K_k is the filter gain, which tells how much the predictions should be corrected on time step k .

3.8.4 PID controller

The Proportional Integrator Derivative (PID) controller commonly known for its simplicity, is used for process control in different industries sectors. The PID controller is based on three actions proportional (P), integral (I), and derivative (D) (Figure 4-8), each one of these actions

consists of different roles in order to minimize the error between the desired value or the command and the measured value from the process, the proportional action reduces the difference between the commanded and measured values (the error), the integral action integrates the error, and the derivative action is the rate. There are four compositions of the controller depending on the nature of the variable controlled and the process dynamic, the composition of the controller could be proportional only (P), proportional-derivative (PD), proportional-integral (PI), and proportional-integral-derivative (PID). The action is enabled or adjusted by finding the right gain K_p , K_i , and K_d associated with the appropriate actions so the system response can be tuned refer to the equation below(Dorf and Bishop, 2010).

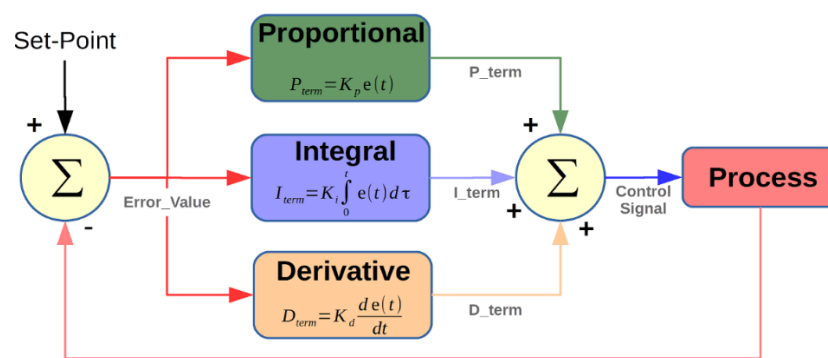


Figure 3-17 - Block diagram of PID controller (Srinivasan K, 2016)

In order to tune any controller, there are four major specifications to tune a system of closed-loop step response such as the steady-state, which is the value where the system settles to, Rise time when the output reaches 90% of the desired value; overshoot is deviation above the steady-state; time response is the time to converge to the steady-state; and the steady-state error which is the difference between steady-state and the desired value. To reach the specification required, the actions gains effect are described in Table 4-2 below, so tuning a system is about balancing the effects(Carrillo *et al.*, 2010).

Table 3-2 -PID controller response characteristics

	<i>Rise time</i>	<i>Overshoot</i>	<i>Settling time</i>	<i>Steady-state error</i>
<i>K_p</i>	Decrease	Increase	Small Change	Decrease
<i>K_d</i>	Small Change	Decrease	Decrease	Small Change
<i>K_i</i>	Decrease	Increase	Increase	Eliminate

To tune the PID loop, the proportional gain K_p is the first to start with, so it brings the system to oscillate around the desired value. The next action is to activate the derivative gain K_d to reduce the oscillation and settling time and increase the rise time. The last step action is to tune K_i to eliminate the offset between the desired value and the process value.

3.9 Chapter Summary

This chapter presents intensive research and understanding of the quadrotor dynamic model concept and introduced the control algorithms that could be applied. Several points have been considered the first is the description of the mechanical concept of the quadcopter and the mathematical equations that enable to extract the different relationships between the propulsion electrical system, the thrust produced and the manoeuvrability of the system in linear approximation. It helps to determine the necessary hardware that will be used for the real test.

The dynamical modelling includes the derivation of translation and rotational motion of the quadcopter. Combining forces and moments to formulate Newton-Euler equations and determining the state variables. It explains the aerodynamic properties of the propeller blade as well as actuator dynamics. It also gives an overview of the controller and filter technique being implemented. Namely, PID controller and Kalman filter respectively.

Chapter 4 – Controller Design Methodology and Experimental Extraction of the Physical Parameters

This chapter starts with linearising the quadcopter model. The linearised equations are used to develop the control system and implement the Kalman Filter(KF). KF helps to eliminate plant and measurement noises. The chapter also focuses on extracting hardware parameters through approximating and empirical testing. Deriving the physical parameters enables to development of a more accurate model of the quadcopter.

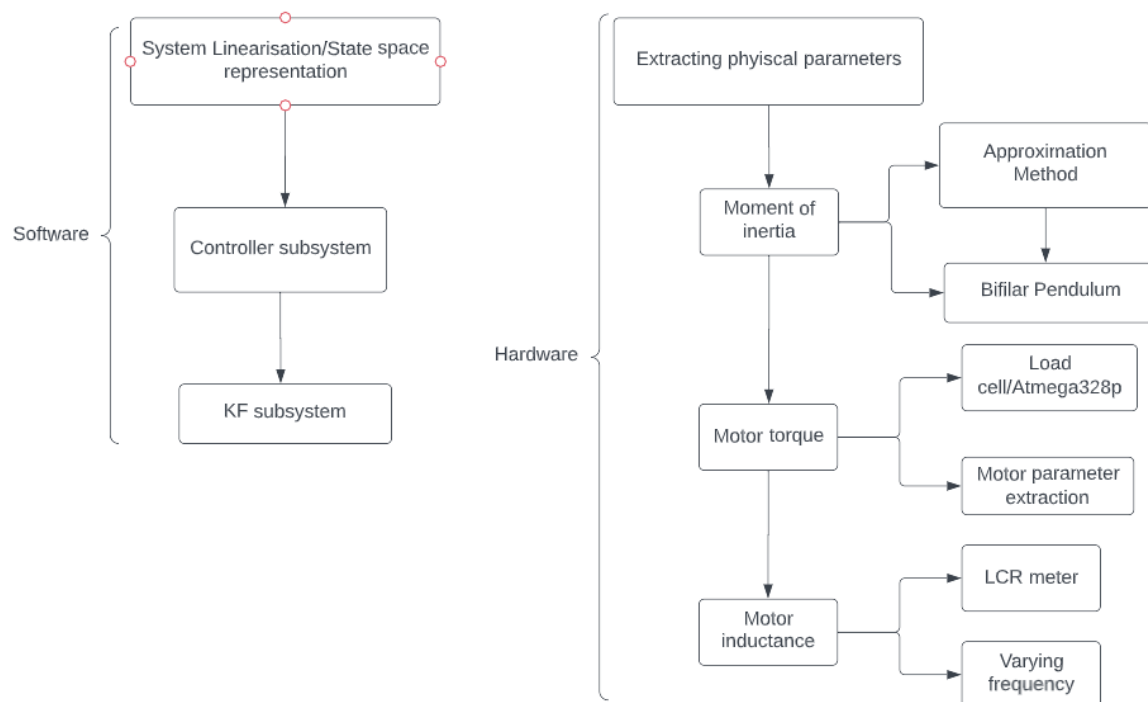


Figure 4-1 – Chapter 4 Overview

4.1 Linearisation and State-space (SS) Representation for Quadcopter

In many practical problems, the motion of a system is governed by a set of nonlinear equations of motion. The equations of motion can be linearized about some arbitrary state of initial conditions. The reason for linearising the model is to make the output predictable. The linearised equations can be directly associated with state-space equations derived in previous chapters (3.6).

An alternative to the state-space equation is the transfer function (TF). One way to represent a transfer function is as the ratio of the Laplace transform of the output to the Laplace transform of the input, with the assumption that all initial conditions are equal to zero. In order to calculate the transfer function of a control system, it is necessary to assume that all of the initial conditions of the differential equations are equal to zero. A given system will always have the same transfer function because it is a static quantity. It is possible to arrive at an accurate estimate of the system's output by performing a straightforward calculation known as the inverse Laplace of the product of the transfer function and the information that is provided ('Transfer function Definition - Advantages & Disadvantages | Electricalvoice', 2017). The output is given as follows

$$Effect = \frac{Output}{Input} \equiv T(s) = \frac{Y(s)}{X(s)} \quad 4.1$$

The transfer function model cannot be used to analyse non-linear systems because of this limitation. In transfer function models, the initial conditions are not taken into account, hence the consequences that are caused by those conditions are also ignored.

Linearization is a critical feature that allows the use of the linearized equations of motion to predict the consequences of external forces and boundary conditions on the motion of the system. The concept of linearization is also essential for reasons of computational efficiency. It helps with tuning the control system a lot easier. However, the disadvantage is if there are external disturbances, it will significantly affect its performance.

The equations for translation motion are derived in Chapter 3, section 3.2 and the state-space equations are given in equation

3.59. The state-space equations are represented in the body frame. Converting body frame equations to an inertial frame equation make it easier to develop a robust feedback system. Therefore, equations of motion in the inertial frame for position control need to be determined in the form of $\ddot{X}$, $\ddot{Y}$, and $\ddot{Z}$.

The state-space equation

3.59 were derived from Newton's second law, which describes the net force (eq 3.12) in the body frame.

$$\overrightarrow{F_{net}} = \frac{d(m\vec{v})}{dt} = \frac{dm}{dt}\vec{v} + m\frac{d\vec{v}}{dt} \quad 4.2$$

Here, m is the mass of the quadcopter and $\vec{v}$ represents velocity vector. Since the mass is constant, the equation becomes.

$$\overrightarrow{F_{net}} = m\frac{d\vec{v}}{dt} \quad 4.3$$

The net force vector equation had to be rewritten in terms of individual forces (equation 3.57). Here, $\vec{\lambda}_{GR}^B$, $\vec{\lambda}_G^B$, $\vec{\lambda}_{IN}^B$ are gyroscopic force moment vector, gravity force moment vector and input force moment vectors, respectively. This equation also includes three dimensions for the moment, but here we only need the first three elements from the force moment vectors.

$$\dot{\vec{v}}^B = (M^B)^{-1} [-C^B \vec{v}^B + \vec{\lambda}_{GR}^B + \vec{\lambda}_G^B + \vec{\lambda}_{IN}^B] \quad 4.4$$

To derive the equation in the inertial frame, the forces experienced by the quadcopter need to be considered and convert the equations (eq 3.4 - 3.8) using Newton's second law of motion (to the inertial frame $\overrightarrow{\Gamma^E} = [X \ Y \ Z]^T$). From equation 4.3, the second derivative of position vector yields acceleration vectors, where $\ddot{\overrightarrow{\Gamma^E}} = \frac{d\vec{v}}{dt}$.

From the equation derived in Chapter 3, equation 3.4, the unit vectors $\overrightarrow{\Gamma^E} = [X \ Y \ Z]^T$ can be rewritten in terms of unit vectors $\hat{i}, \hat{j}, \hat{k}$.

$$\overrightarrow{\Gamma^E} = X \hat{i} + Y \hat{j} + Z \hat{k} \quad 4.5$$

The first derivative will result in the following equation.

$$\dot{\vec{r}}^E = (\dot{X}\hat{i} + X\dot{\hat{i}}) + (\dot{Y}\hat{j} + Y\dot{\hat{j}}) + (\dot{Z}\hat{k} + Z\dot{\hat{k}}) = \dot{X}\hat{i} + \dot{Y}\hat{j} + \dot{Z}\hat{k} \quad 4.6$$

Here, $\hat{i} = \hat{j} = \hat{k} = 0$

Finally, the second derivative results in.

$$\ddot{\vec{r}}^E = \ddot{X}\hat{i} + \ddot{Y}\hat{j} + \ddot{Z}\hat{k} \quad 4.7$$

Hence the force in the inertial frame can be expressed as follows.

$$\begin{bmatrix} F_X \\ F_Y \\ F_Z \end{bmatrix}_{net}^E = m \begin{bmatrix} \ddot{X} \\ \ddot{Y} \\ \ddot{Z} \end{bmatrix} \quad 4.8$$

Now expressing the net force vector in terms of its individual forces experienced by the quadcopter will include $\vec{\Lambda}_{GR}^B, \vec{\Lambda}_G^B, \vec{\Lambda}_{IN}^B$. The derivation of these equations can be found in 3.36,

3.52 and

3.53.

$$\vec{\Lambda}_G^B = \begin{bmatrix} \vec{F}_g^B \\ 0_{3 \times 1} \end{bmatrix}, \quad \vec{\Lambda}_{GR}^B = \begin{bmatrix} 0 \\ 0 \\ 0 \\ q J_{TP}(\Omega) \\ -p J_{TP}(\Omega) \\ 0 \end{bmatrix}, \quad \vec{\Lambda}_{IN}^B = \begin{bmatrix} 0 \\ 0 \\ U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} \quad 4.9$$

By neglecting all moments and considering only the forces acting on the body frame yields,

$$\vec{\Lambda}_G^B = [\vec{F}_g^B], \quad \vec{\Lambda}_{GR}^B = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \quad \vec{\Lambda}_{IN}^B = \begin{bmatrix} 0 \\ 0 \\ U_1 \end{bmatrix} \quad 4.10$$

Therefore, the force vectors in the inertial frame are determined by multiplying the rotation matrix equation 2.2 with force input vectors, Equations 4.10 results in the following expression.

$$\vec{F}_G^E = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix}, \quad \vec{F}_{GR}^E = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \quad \vec{F}_{IN}^E = R \times \begin{bmatrix} 0 \\ 0 \\ U_1 \end{bmatrix} \quad 4.11$$

$$\vec{F}_{IN}^E = \begin{bmatrix} (\sin\phi\sin\psi + \cos\phi\cos\psi\sin\theta) \times U_1 \\ (\sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi) \times U_1 \\ (\cos\phi\cos\theta) \times U_1 \end{bmatrix} \quad 4.12$$

Hence,

$$m\ddot{\vec{r}}^E = m \begin{bmatrix} \ddot{X} \\ \ddot{Y} \\ \ddot{Z} \end{bmatrix} = \vec{F}_G^E + \vec{F}_{GR}^E + \vec{F}_{IN}^E = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + R \times \begin{bmatrix} 0 \\ 0 \\ U_1 \end{bmatrix} \quad 4.13$$

Rearrange equation, 4.13 to get

$$\begin{bmatrix} \ddot{X} \\ \ddot{Y} \\ \ddot{Z} \end{bmatrix} = \begin{bmatrix} (\sin\phi\sin\psi + \cos\phi\cos\psi\sin\theta) \times \frac{U_1}{m} \\ (\sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi) \times \frac{U_1}{m} \\ (\cos\phi\cos\theta) \times \frac{U_1}{m} \end{bmatrix} \quad 4.14$$

The main target of the quadrotor stability control is to provide the values of the motors' voltage which maintains the helicopter in a certain desired position in different manoeuvres heights, roll, pitch, and yaw. The process of control must pass by inverse kinematics and inverse dynamics. Whereas, the inverses operations are not always possible and not straightforward like the direct ones so, for these reasons further simplification has been taken into consideration to simplify the inverse model and could be implemented in the control algorithm(Bouabdallah, 2007).

Thus, the equations can be further simplified and rearranged according to the following considerations:

The gyroscopic effects and Coriolis-centripetal form (described in 3.5.2) have been neglected so, the motion of the quadrotor is assumed to be close to the hovering condition, which means a small angular variation of roll and pitch occurs.

The attitude in the earth frame is determined by the acceleration of the Euler angles, which determines the angular accelerations. They are not equal to the angles of the quadrotor measured in its fixed frame. The transfer matrix between the Earth frame and the Body frame defines the relationship between the angular velocities in both frames. Hence, in the hovering condition, it is close to the identity matrix so, the acceleration

equations have been stated directly to the Euler angle accelerations(‘Modelling, Identification and Control - Tommaso Bresciani.pdf’, no date).

The purpose of the control is to develop an algorithm that is used to give the right signals to the propellers to get to the right X and Y position; for that reason, a control architecture has been designed according to the attitude and altitude requirement, it is worthy to notice that the Euler angles and their time derivatives do not depend on translation components (X and Y). However, the translations depend on the angles.

Considering 4.14 above, the equation can be expressed in matrix form as

$$\begin{aligned}\ddot{X} &= \frac{U_1}{m} = -g\theta \\ \ddot{Y} &= \frac{U_1}{m} = g\phi \\ \ddot{Z} &= \frac{U_1}{m} - g \\ \dot{p} &= \frac{U_2}{I_{xx}} \\ \dot{q} &= \frac{U_3}{I_{yy}} \\ \dot{r} &= \frac{U_4}{I_{zz}}\end{aligned}\tag{4.15}$$

Assuming that the rates p, q, and r are equal to the Euler angle ϕ , θ and ψ , respectively.

The system progresses in time is described using the classical state-space representation equations

$$\dot{x}(K) = A(K)x(K) + B(K) \times u(K)\tag{4.16}$$

$$y(K) = C(K)x(K) + D(K) \times u(K)\tag{4.17}$$

In general, these equations describe the evolution of a linear time-varying system, where $x(t)$ is the vector of states, $y(t)$ is the output vector, and $u(t)$ is the input vector. However, the conventional state-space representation of a quadcopter considers the following matrices A, B, C and D linear time-invariant realisation of the system, assuming no change over time and it will be:

$$\dot{x} = Ax + Bu \quad 4.18$$

$$y = Cx + Du \quad 4.19$$

Now, to express both the state-transition model according to these equations, a set of state-space vectors $X = (\dot{z}, z, \dot{\phi}, \phi, \dot{y}, y, \dot{\theta}, \theta, \dot{x}, x, \dot{\psi}, \psi)^T$ and input vectors $U = (U_1, U_2, U_3, U_4)$ have been implemented. Equations, 4.18 and 4.19 show that each system (height, roll, pitch and yaw) acts independently from the other on the quadcopter's body. For this reason, the state-space representation has been divided into four manoeuvres vertical, latitudinal, longitudinal, and directional independent from each other so, the quadcopter's dynamics are decoupled and can be studied as sub-systems state vector $X = (\text{Vertical}, \text{Directional}, \text{Latitude}, \text{longitudinal})$

4.1.1 Z-axis movement - Altitude

The system describes the dynamics of the VTOL movements of the quadcopter, according to this state-space equation.

$$\begin{bmatrix} \ddot{Z} \\ \dot{Z} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} * \begin{bmatrix} \dot{Z} \\ Z \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} * \frac{1}{m} * U_1 \quad 4.20$$

4.1.2 Yaw movement - Directional

The directional movement is related to the yaw angle, which is the movement around the Z-axis, and the state-space matrix equation, which describes the system, is as follows below.

$$\begin{bmatrix} \ddot{\psi} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} * \begin{bmatrix} \dot{\psi} \\ \psi \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} * \frac{1}{I_{zz}} * U_4 \quad 4.21$$

4.1.3 Rotation along Y axis - Latitudinal

The latitudinal dynamic that manages the pitch movement is described by the state-space equation cited below.

$$\begin{bmatrix} \ddot{\phi} \\ \dot{\phi} \\ \ddot{y} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & -g & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} * \begin{bmatrix} \dot{\phi} \\ \phi \\ \dot{y} \\ y \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} * \frac{1}{I_{xx}} * U_3 \quad 4.22$$

4.1.4 Rotation along X axis - Longitudinal

The longitudinal dynamic rules the roll movement and is represented by the following state-space representation.

$$\begin{bmatrix} \ddot{\Theta} \\ \dot{\Theta} \\ \ddot{x} \\ \dot{x} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & g & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} * \begin{bmatrix} \dot{\Theta} \\ \Theta \\ \dot{y} \\ y \end{bmatrix} + \begin{bmatrix} \frac{1}{I_{zz}} \\ 0 \\ 0 \\ 0 \end{bmatrix} * U2 \quad 4.23$$

4.1.5 Controllability and Observability of the systems

Before proceeding to build controllers for each subsystem, controllability and observability have been tested, controllability is crucial to determine the state variable stability of the system feedback, or in other meaning, Controllable systems are those possible to be moved from an initial state at a time initial time, to another stable state within an infinite span of time, and it is verified using the rank of the matrix expressed in equation (Dorf and Bishop, 2010).

$$Co = [B \ AB \ A^2B \ \dots \ A^{n-1}B] \quad 4.24$$

Where A is an n-by-n matrix, B is an n-by-m matrix, and C has n rows and nm columns. If Co rank has full rank, the system is controllable.

The second propriety checked is the observability of the system using the state space representation of the quadrotors shown in equations 4.34 to 4.23. The observability matrix O_b which will determine either full rank or not to determine the observability of the system.

$$Ob = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix} \quad 4.25$$

4.2 Flight control Algorithm

The Flight control algorithm acquires the data from the measurement estimator (in this case Kalman Filter) as inputs and compares it to the desired value received from the trajectory generator. The desired trajectory will pass through different computations during which a lot of constants and variables are involved; all the controller blocks are shown in Figure 4-2, which describes the inverse dynamics of the quadrotor. The flight controller is based on two cascading PID controllers, one for the position as an inner loop and the other for altitude and attitude, which is the inner loop.

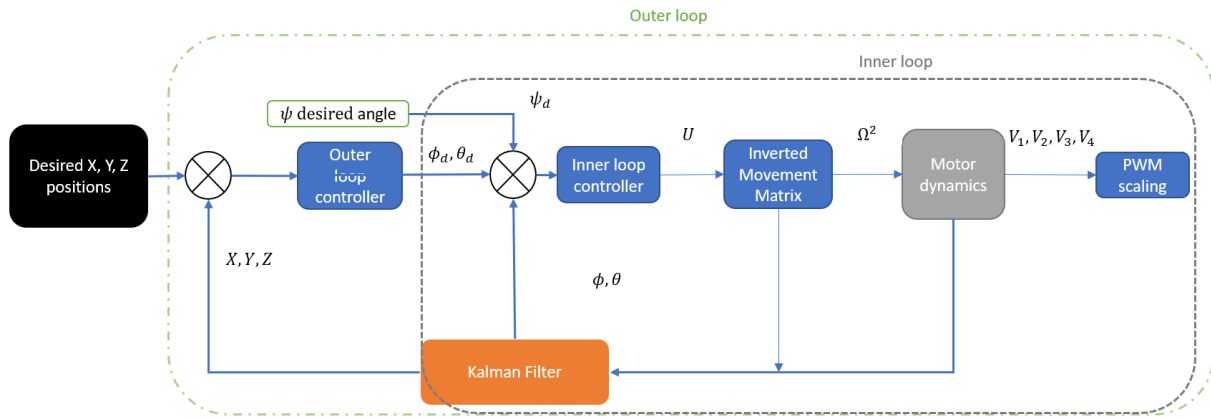


Figure 4-2 - Flight controller block diagram (Ajay Joseph, 2017)

4.2.1 Control Technique

As mentioned earlier, the control technique adopted is composed of the traditional PID structure associated with the Quadcopter Mixer, which relates the PID controller error to the desired input voltage for the system to reach the set point, the PID and the quadcopter mixer are enhanced by saturation block which determines the linear zone of the error and the actuator to avoid any undesirable effects such as derivate called integral wind-up.

4.2.2 Outer loop

This Loop runs at a low frequency than the inner loop and provides autonomous control of the copter by introducing the X [m] and Y [m] coordinate and then converting it into the attitude command in terms of Roll and Pitch, respectively. The PID controllers developed for X and Y are shown in Figure 4-3.

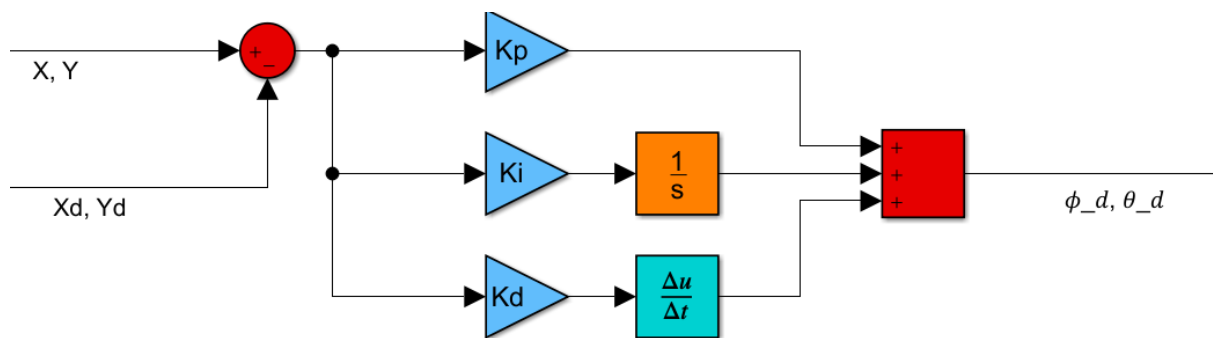


Figure 4-3 - X and Y controller design

Where X_d [m] and Y_d [m] represents the desired coordinate, X [m] and Y [m] are the coordinates estimated from KF. The error provided by the PID is divided by the g [$m.2$] to result in the desired ϕ_d, θ_d .

4.2.3 Inner loop

The inner loop represents the core control for the Quadcopter as its process directly the balance,

4.15 is used in this block to transfer an angular acceleration to translational movement by creating torque around the x and y-axis. The control technique used in this loop is PID, and it is applied for Roll ϕ , Pitch θ and Yaw Ψ (see section 2.6).

4.2.4 Throttle controller

This PID will control the height of the vehicle, and it has been improved by a saturator at the output. The values max and min of the saturation block have been extracted from the calibration of the model.

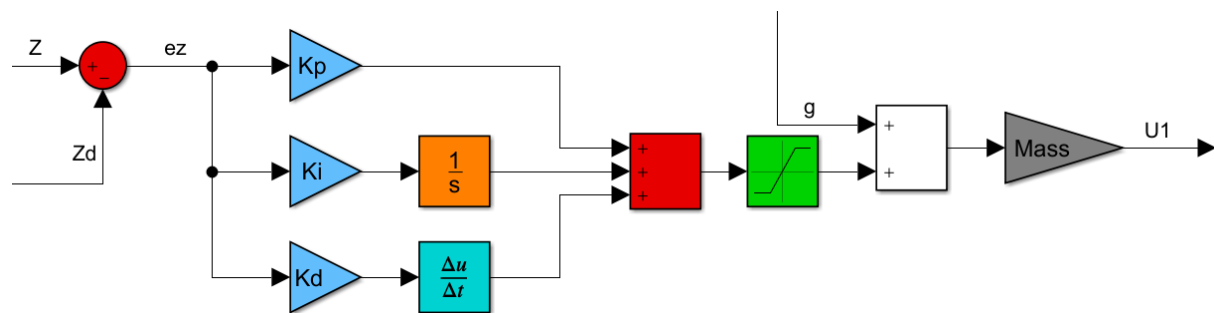


Figure 4-4 - Throttle controller block

Z_d : represents the desired throttle [m], Z : is the throttle estimated from the KF [m], e_z : is the throttle error [m], g : gravity acceleration [m.s²], m : mass of the vehicle [kg], and U_1 : required thrust [N].

4.2.5 Roll and Pitch controller

The two PID controllers build for the Roll and Pitch are similar even in their saturation values. The only difference, as mentioned before, is the Roll acts around the x-axis, and the Pitch acts around the y-axis Figure 4-5.

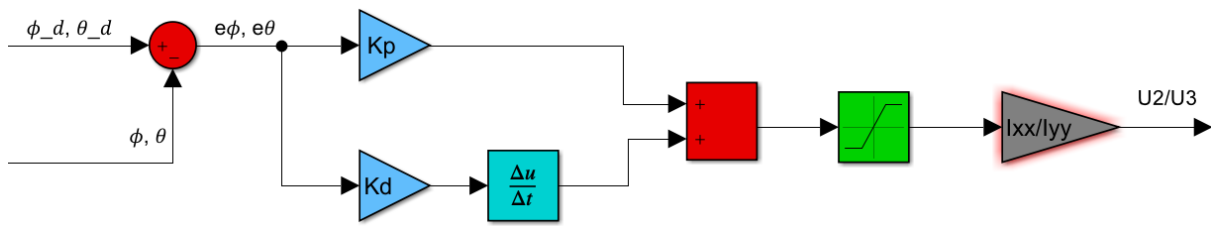


Figure 4-5 - Roll and Pitch controller block

ϕ_d [rad], θ_d [rad] represents the desired Roll and Pitch angles (introduces manually or provided by the outer loop), ϕ [rad] and θ [rad] are the measured angles, $e\phi$ [rad] and $e\theta$ [rad]: Roll and Pitch errors, I_{xx} [N.m] and I_{yy} [N.m]: the inertia body moment around the x-axis and y-axis, and the last $U2$ [N.m] and $U3$ [N.m]: the required Roll and Pitch torques, are. This design comes from equation

4.15, which relates the Roll and Pitch control to $U2$ and $U3$.

4.2.6 Yaw controller

Finally, the yaw controller block is represented in Figure 4-6 below.

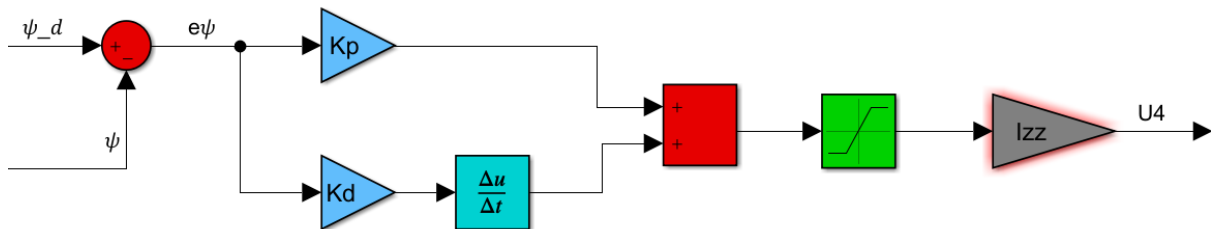


Figure 4-6 - Yaw controller block

ψ_d [rad s⁻¹]: signifies the desired yaw angle, ψ [rad]: signifies the yaw angle estimated from KF, $e\psi$ [rad]: the yaw error, I_{zz} [N.m]: the inertia body moment around the z-axis, and $U4$ [N.m] the required yaw torque.

4.3 Kalman Filter Algorithm

To build Kalman Filter, the linear state-space representation refer to equation

4.15 and the KF basis has been used. The actual approach KF algorithm is created to estimate the state variable vector for each manoeuvre separately following the state-space representation, assuming that the quadcopter is working in 6 degrees of freedom using an Inertial measurement unit consisting of a three-axis gyroscope, and accelerometer.

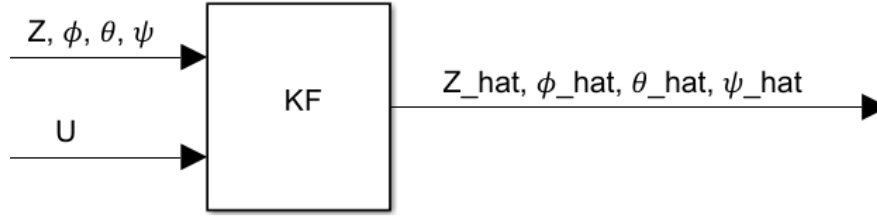


Figure 4-7 - KF block diagram

The variances of the measurement noise v , the R matrix and the process noise w , the Q matrix are considered time-invariant in this study and used to correct the drift and bias due to sensor measurement noise and the process, respectively. The KF for each manoeuvre is represented as follows:

4.3.1 Altitude

The state-space representation in equation 4.34 in the section for the vertical position is used to estimate the $\hat{x}$ vector,

$$\dot{\hat{x}}_K = \begin{bmatrix} \ddot{\hat{Z}}_K \\ \dot{\hat{Z}}_K \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} \dot{\hat{Z}}_k \\ \hat{Z}_K \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \times U_1 + w_K \quad 4.26$$

$$\begin{bmatrix} \dot{\hat{Z}}_k \\ \hat{Z}_K \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} \dot{\hat{Z}}_k \\ \hat{Z}_K \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \times U_1 + v_k \quad 4.27$$

Where,

$\hat{Z} [k]$ altitude estimation [m].

$\dot{\hat{Z}}[k]$ Vertical velocity estimation [m/s].

U_1 the input thrust [N].

W : is the process noise, and v is the measurement noise with known variances Q and R .

Q is a diagonal matrix with 2x2 and R diagonal matrix with 1x1 dimension.

4.3.2 Directional

To build the KF for directional movement, equation 4.35 has been used, and the estimated $\hat{x}$ vector represent the new measured value.

$$\hat{x}_k = \begin{bmatrix} \ddot{\hat{\phi}}_k \\ \dot{\hat{\phi}}_k \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} \hat{\phi}_k \\ \dot{\hat{\phi}}_k \end{bmatrix} + \begin{bmatrix} 1 \\ \frac{1}{I_{zz}} \\ 0 \end{bmatrix} \times U_4 + w_k \quad 4.28$$

$$\hat{y}_k = \begin{bmatrix} \dot{\hat{\phi}}_k \\ \hat{\phi}_k \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} \dot{\hat{\phi}}_k \\ \hat{\phi}_k \end{bmatrix} + \begin{bmatrix} 1 \\ \frac{1}{I_{zz}} \\ 0 \end{bmatrix} \times U_4 + v_k \quad 4.29$$

4.3.3 Latitudinal

To build the KF for latitudinal movement equation 4.22 has been used, and the estimated $\hat{x}$ vector represent the new measured value.

$$\hat{x}_k = \begin{bmatrix} \ddot{\hat{\phi}}_k \\ \dot{\hat{\phi}}_k \\ \ddot{\hat{y}}_k \\ \dot{\hat{y}}_k \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & -g & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} \hat{\phi}_k \\ \dot{\hat{\phi}}_k \\ \hat{y}_k \\ \dot{\hat{y}}_k \end{bmatrix} + \begin{bmatrix} 1 \\ \frac{1}{I_{xx}} \\ 0 \\ 0 \\ 0 \end{bmatrix} \times U_3 + w_k \quad 4.30$$

$$\hat{y}_k = \begin{bmatrix} \dot{\hat{\phi}}_k \\ \hat{\phi}_k \\ \dot{\hat{y}}_k \\ \hat{y}_k \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \dot{\hat{\phi}}_k \\ \hat{\phi}_k \\ \dot{\hat{y}}_k \\ \hat{y}_k \end{bmatrix} \times U_3 + v_k \quad 4.31$$

Where,

$\hat{\phi}_k$ is angle estimation around X-axis [deg];

$\dot{\hat{\phi}}_k$ is angular velocity estimation around x axis [deg/s];

$\ddot{\hat{\phi}}_k$ is angular acceleration estimation around x axis [deg/s];

$\ddot{\hat{y}}_k$ is linear acceleration estimation in the y-axis [deg/s];

$\dot{\hat{y}}_k$ linear velocity estimation in the y-axis [deg/s];

$\hat{y}_k$ position estimation in y-axis [deg/s];

U_3 : the input thrust [N.m];

W: is the process noise, and v: is the measurement noise with known variances Q and

R;

Q is a diagonal matrix with 4x4 and R diagonal matrix with 2x2 dimension;

4.3.4 Longitudinal

To build the KF for latitudinal movement the equation (IV.19) has been used, and the estimated $\hat{x}$ vector represents the new measured value.

$$\hat{x}_k = \begin{bmatrix} \ddot{\hat{\theta}}_k \\ \dot{\hat{\theta}}_k \\ \ddot{\hat{x}}_k \\ \dot{\hat{x}}_k \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & -g & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} \hat{\theta}_k \\ \dot{\hat{\theta}}_k \\ \hat{x}_k \\ \dot{\hat{x}}_k \end{bmatrix} + \begin{bmatrix} \frac{1}{I_{yy}} \\ 0 \\ 0 \\ 0 \end{bmatrix} \times U_3 + w_k \quad 4.32$$

$$\hat{y}_k = \begin{bmatrix} \dot{\hat{\theta}}_k \\ \hat{\theta}_k \\ \dot{\hat{x}}_k \\ \hat{x}_k \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \dot{\hat{\theta}}_k \\ \hat{\theta}_k \\ \dot{\hat{x}}_k \\ \hat{x}_k \end{bmatrix} \times U_3 + v_k \quad 4.33$$

Where,

$\hat{\theta}_k$ is angle estimation around X-axis [deg];

$\dot{\hat{\theta}}_k$ is angular velocity estimation around the y-axis [deg/s];

$\ddot{\theta}_k$ is angular acceleration estimation around the y-axis [deg/s²];

$\ddot{\hat{x}}_k$ is linear acceleration estimation in the y-axis [deg/s²];

$\dot{\hat{x}}_k$ linear velocity estimation in the x-axis [deg/s];

$\hat{y}_k$ position estimation in x-axis [deg/s];

U_3 : the input thrust [N.m];

W: is the process noise, and v: is the measurement noise with known variances Q and R;

Q is a diagonal matrix with 4x4 and R diagonal matrix with 2x2 dimension;

4.4 Identification of the physical parameters

Extraction of physical parameters is required to be added to the developed subsystem. Some of them are Ixx, Iyy, mass, motor characteristics etc.

4.4.1 Moment of Inertia

Determining the moment of inertia parameters is essential for the rotational dynamics of the quadcopter. Some of the methods to calculate the moment of inertia are CAD modelling, approximation, and the Bifilar pendulum experiment.

In this section, the moment of inertia is found using the approximation method and Bifilar pendulum methods. Then the results are compared against Parrot Mambo's parameters. These parameters are provided by Parrot Mambo Simulink Package Figure 4-8.

```
>> Vehicle.Airframe.inertia

ans =

    1.0e-04 *

    0.5829         0         0
         0    0.7169         0
         0         0    1.0000

fx >>
```

Figure 4-8 - Simulink Package Parrot Mambo

4.4.1.1 Moment of Inertia Approximation

This section is used to extract the hardware parameter of the quadcopter by approximation. The moment of inertia of a rod (Figure 4-9) of length L can be calculated as follows (*Model, Simulate and Control a Drone in MATLAB & SIMULINK*, 2020):

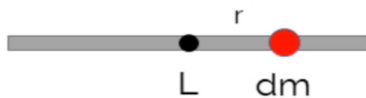


Figure 4-9 - Moment of inertia of a rod

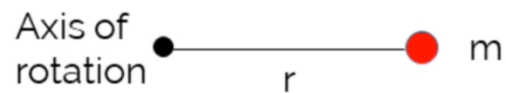


Figure 4-10 - Moment of inertia of point mass

$$I_{(xx,yy,zz)} = \int_0^M r^2 dm \quad 4.34$$

R is the radius or distance from the centre to the end of the frame, and d_m represents the rod's mass. Equation 4.4 can be simplified by considering the quadcopter as four rods of equal mass symmetrically placed about the quadcopter's centre of mass, as shown in Figure 4-11 and Figure 4-12.

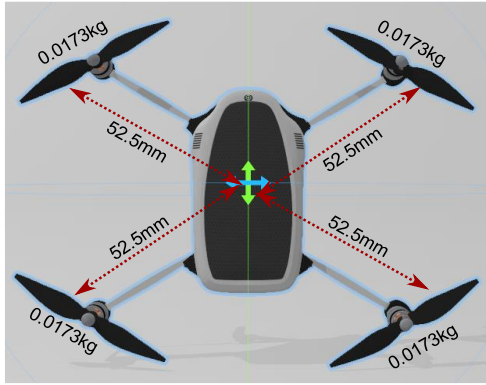


Figure 4-11 - Quad moment of inertia

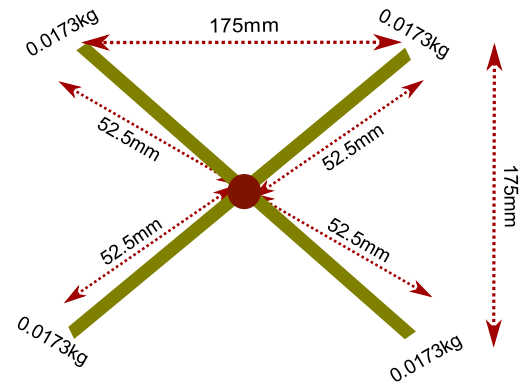


Figure 4-12 - Moment of inertia of quad equivalent (I_{xx} , I_{yy} , I_{zz})

$$I_{(xx,yy,zz)} = \int_0^M \iota A_c dx \quad 4.35$$

$$I_{(xx,yy,zz)} = \iota A_c \frac{1}{3} x^3 \Big|_0^{L_r} \quad 4.36$$

$$I_{(xx,yy,zz)} = \iota A_c \frac{1}{3} L_r^3 \quad 4.37$$

Where ι , A_c and L_r are the density of the material, cross-sectional area, and distance from the axis of rotation, respectively. The density of the material and cross-sectional areas is neglected in this calculation. The reason for neglecting the density and cross-sectional area is because this experiment is used for an unevenly shaped object with no information required about the properties of the object. This experiment solemnly depends on the frequency at which the object oscillates.

Therefore, the equation to determine the moment of inertial along the x, y and z-axis of the quadcopter can be further simplified as:

$$I_{(xx,yy,zz)} = \frac{1}{3} \times \frac{m}{4} \times \left(\frac{L_r}{2} \right)^2 \times 4 \quad 4.38$$

4.4.1.2 Bifilar pendulum

In this section, a well-known technique is used to determine the moment of inertia of the quad. This technique is used by NASA.

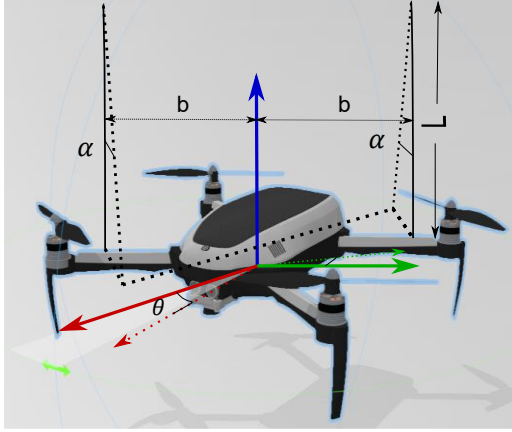


Figure 4-13 - Bifilar Pendulum

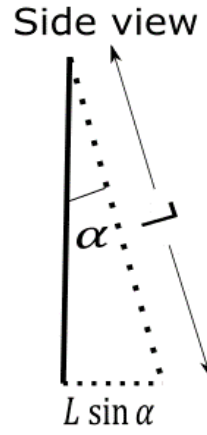


Figure 4-14 - Derivation of angles

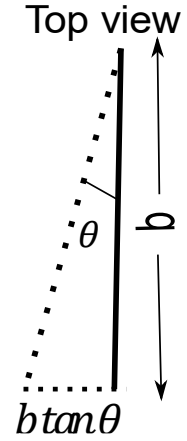


Figure 4-15 - Derivation of displacement

Figure 4-13 represents the setup of the bifilar pendulum experiment on the quadcopter. The expression for the moment of inertia can be determined by using a small-angle approximation. The small-angle approximation helps avoid non-linear trigonometric terms (sin, cos, tan etc.). This technique will only work if the angle at which the bifilar pendulum is swung is kept minimal.

Assuming the angles in Figure 4-13 are oscillating at a small degree, it becomes safe to assume $\sin \alpha \approx \alpha$ and $\tan \theta \approx \theta$. So, the equation can be simplified as follows.

$$b \tan \theta = L \sin \alpha \quad 4.39$$

$$\alpha = \frac{b\theta}{L} \quad 4.40$$

The forces acting is illustrated in their horizontal and vertical component level in Figure 4-16.

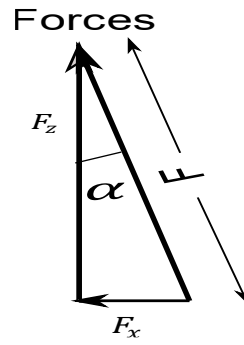


Figure 4-16 - Bifilar pendulum forces

$$F = F_z + F_x \quad 4.41$$

Where,

$$F_z = mg \text{ and } F_x = F_z \tan \alpha \quad 4.42$$

Once again, if the angle α is small enough, $\tan$ can be eliminated from the equation resulting.

$$F_x \approx \frac{mg}{2} \times \alpha \quad 4.43$$

From equation 4.40;

$$F_x = \frac{mg}{2} \times \frac{b\theta}{L} \quad 4.44$$

Since the moment is force multiplied by the distance, the "-ve" sign represents the force acting in the opposite direction to movement, resulting in

$$M = -b \times F_x \quad 4.45$$

$$M = -\frac{mgb^2\theta}{L} \quad 4.46$$

Sum of moments, where θ represents the angle tilted by the quadcopter.

$$\epsilon M = I\alpha = I \ddot{\theta} \quad 4.47$$

Note the denominator "2" in equation 4.44 is removed as the quadcopter is suspended on both sides using string. The differential equation derived in 4.47 can be solved by assuming θ as a sinusoidal motion and by representing θ in complex exponential term.

$$\theta = Ae^{i\omega t} \quad 4.48$$

$$\dot{\theta} = i\omega Ae^{i\omega t} \quad 4.49$$

$$\ddot{\theta} = -\omega^2 Ae^{i\omega t} \quad 4.50$$

Substituting the equations above gives;

$$-\frac{mgb^2}{L} \times Ae^{i\omega t} = I(-\omega^2) \times Ae^{i\omega t} \quad 4.51$$

$$\text{Since } \omega = 2\pi f = \frac{2\pi}{T}$$

$$\omega^2 = 4\pi^2 f^2 = \frac{4\pi^2}{T^2} \quad 4.52$$

Substitute

$$\frac{mgb^2}{L} = I \left(\frac{4\pi^2}{T^2} \right) \quad 4.53$$

$$\therefore I = \frac{mgb^2 T^2}{4\pi^2 L} \quad 4.54$$

If the distance between the two strings is D then the equation becomes:

$$I = \frac{mgT^2 D^2}{16\pi^2 L} \quad 4.55$$

4.4.1.3 Results

The approximation method and the experimental method discussed above are compared and evaluated in this section. Figure 4-17 and Figure 4-18 represent the setup used to determine the moment of inertia using the bifilar pendulum method.



Figure 4-17 - Bifilar pendulum test Parrot Mambo



Figure 4-18 - Bifilar Pendulum custom quad

Table 4-1 - Moment of Inertia results Parrot Mambo

$kg.m^2$	SIMULINK PACKAGE	APPROXIMATED	BIFILAR PENDULUM	BIFILAR CUSTOM QUAD
I_{xx}	0.00005829	0.000176097	0.0000796965	0.0075
I_{yy}	0.00007169	0.000176097	0.000075126	0.0075
I_{zz}	0.0001	0.000253575	0.00015902	0.0013

4.4.2 Motor Torque – Load cell

This section evaluation the thrust force exerted by the propeller at various speeds. An HX711 amplifier module along with a 5Kg 4-wire load cell was used to take measurements (see Figure 4-19). The reason for choosing these components is because the HX711 amplifier has a 24-bit resolution and the 4-wire load cell has a full Wheatstone bridge embedded into it.

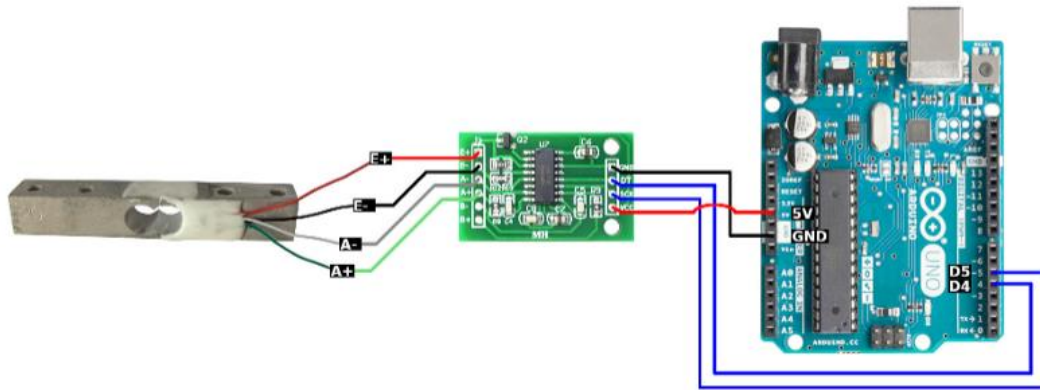


Figure 4-19 - Load cell circuit diagram (4-Wire Load Cell (1/5/10/20/200kg) with HX711 and Arduino - Circuit Journal, no date)

The following steps were taken to calibrate the scale.

Run the `set_scale()` function, with empty parameters and open the serial monitor.

Place a known weight on the scale and record the reading. In this case, it was a 35g object with returned value of -14563.00.

Divide the number by 35g, resulting in -415.534. This tares the scale.

Substitute the above value in `set_scale()` as a parameter.

The code was modified to suit the experiment.

Measured Weight	631.6
Measured Weight	630.7
Measured Weight	633.9
Measured Weight	630.9
Measured Weight	629.7
Measured Weight	633.0
Measured Weight	637.3
Measured Weight	630.6
Measured Weight	631.2
Measured Weight	634.1
Measured Weight	636.7
Measured Weight	635.1
Measured Weight	635.0
Measured Weight	634.2
Measured Weight	637.2
Measured Weight	636.8
Measured Weight	639.8
Measured Weight	637.5
Measured Weight	636.7
Measured Weight	635.5
Measured Weight	638.1
Measured Weight	640.3
Measured Weight	633.4
Measured Weight	636.6
Measured Weight	629.5
Measured Weight	630.2

Figure 4-20 - Load cell measurement

4.4.2.1 Test rig for motor parameter extraction

To determine BLDC motor parameters, a makeshift experimental setup was devised as shown in Figure 6-14. The experiment was carried out on one of four BLDC motors rated at 850KV (850Kv Motor AC2830-358 Build Your Own Drone, no date). The jDrones ESC Model AC20-1.0 (ArduCopter ESC 20 Amp for QuadCopter, HexaCopter and OctoCopter use. Build Your Own Drone, no date) was used to regulate the motor. A PSU with a current limited to 7A and a voltage of 11V was used to supply ESC, while under test. The ESC was regulated with a 10K Ω potentiometer from the microcontroller. The experiment was carried out in three different PWM frequencies 200, 300 and 500Hz. This gives an appropriate range of V_{rms} associated with the PWM duty cycle. The microcontroller generates an 8-bit PWM (0-255). A handheld tachometer was also used to measure the RPM of the propeller.

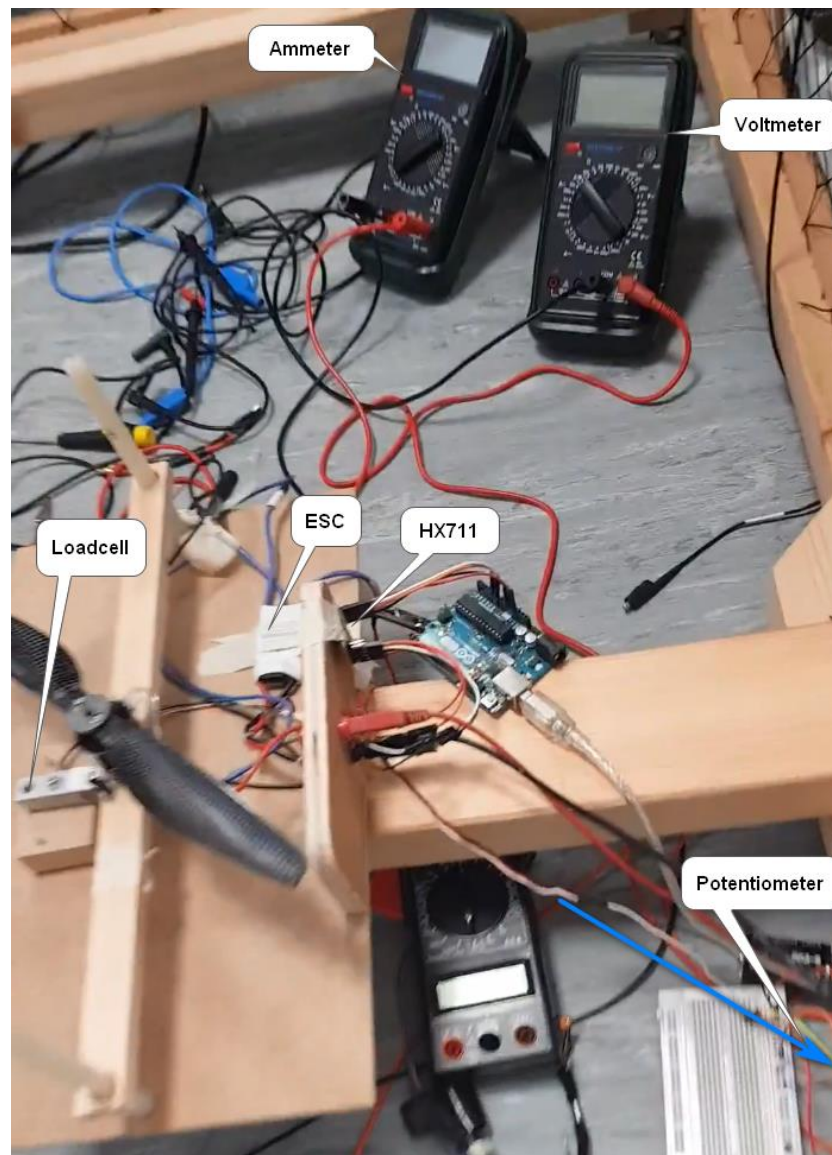


Figure 4-21 - Thrust measurement platform setup

The motor parameters as per the datasheet are given in Table 4-2 below.

Table 4-2 - Thrust measurements from the datasheet

Scale	25%	50%	70%	100%
Current (Amp)	1.26	3.72	7.69	13.3
Power (Wattage)	18.7	55.2	109.2	189.3
Thrust (gram)	151	406	702	970

4.4.3 Parameter extraction procedure

The characteristics of the BLDC motor including angular velocity and thrust are determined using empirical measurements recorded in Table 4-3 below. The full table of content can be

found in APPENDIX I - Load cell which calculates essential parameters and linear approximation of the drive train (motor and propeller combined).

Table 4-3 - BLDC initial measurements

Current (A)	Vrms (V)	PWM (V)	Weight (g)	Angular Speed [RPM]
0.3	4.3	4.183	-67.9	1050
0.6	5.04	4.252	-36.6	1404
0.9	5.68	4.301	-4.7	1816
1	5.68	4.325	-0.9	1799
1.1	5.83	4.355	22	2027
1.4	6.26	4.385	50.2	2311
1.6	6.46	4.426	64.8	2435
1.8	6.72	4.449	94.8	2676
2.1	6.94	4.482	118.8	2875
2.3	7.14	4.516	142.3	3028
2.5	7.32	4.555	160.8	3173
2.8	7.62	4.586	205.8	3501
3	7.72	4.622	222.6	3560
3.4	7.96	4.65	256.4	3151
3.8	8.31	4.675	313.3	4023
3.9	8.4	4.701	321.6	4104
4.3	8.73	4.739	356.5	4350
4.7	9.01	4.77	393.9	5416
4.9	9.13	4.808	415.2	4621
5.4	9.42	4.819	474.1	4901
5.7	9.53	4.855	492	4981
6	9.52	4.884	535.2	5132
6.2	9.51	4.92	549.4	5238

The following equations convert RPM to radians per second to determine the angular speed and thrust in Newton. In ('Static Thrust Calculation', 2011) paper the author takes a different approach to determine thrust by using propeller properties such as diameter, the velocity of air, the density of air etc.

$$\omega_m \left[\frac{rad}{sec} \right] = \omega_m [RPM] \frac{2\pi}{60} \quad 4.56$$

$$T[N] = T[g] \frac{1}{1000} \left[\frac{kg}{g} \right] g \left[\frac{m}{s^2} \right] \quad 4.57$$

The velocity constant K_v of the motor can be cross-verified by using the following equation and by obtaining the linear equation from the chart in Figure 4-22. The K_v constant is given in motor's specification. In this case, the motor has a K_v of 850 as per the datasheet. The discrepancy in the result is due to the imperfections in measurements (handheld tachometer).

$$K_v = \frac{RPM}{V}$$

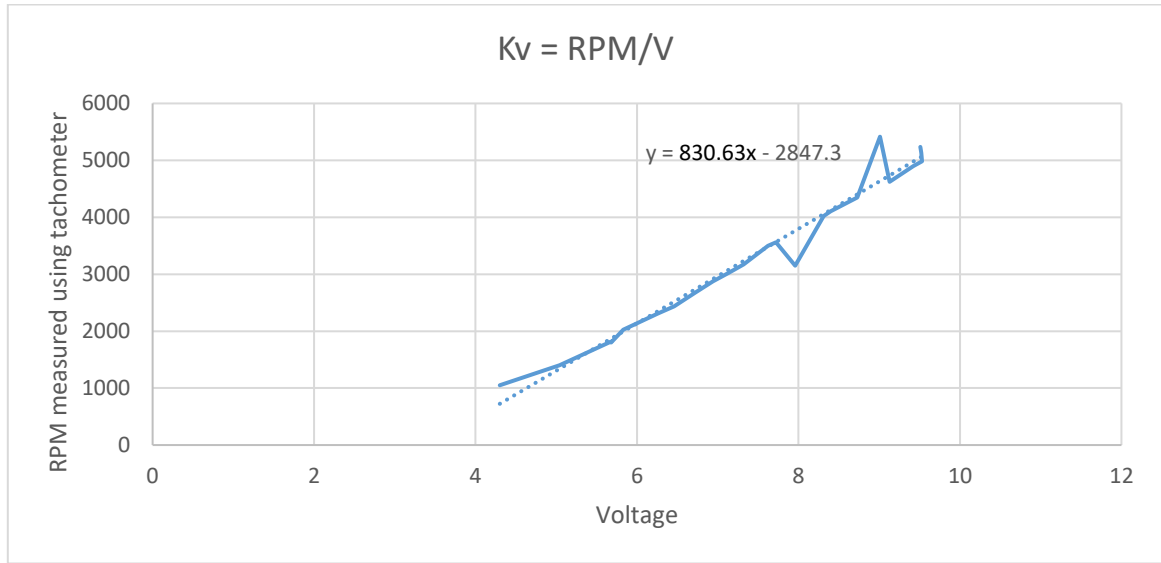


Figure 4-22- K_v constant

As described in Chapter 4, section 4.7.1, equation 4.62 the inductance value is neglected. From this equation, we can find a linear relationship between the motor angular velocity and over current $\frac{\omega_m}{i}$ and motor voltage over current $\frac{v}{i}$. The following approach is found in Tommaso Bresciani's paper ('download.pdf', no date, p. 88) equation 6.66.

$$\frac{v}{i} = Z + k_e \left(\frac{\omega_m}{i} \right) \quad 4.58$$

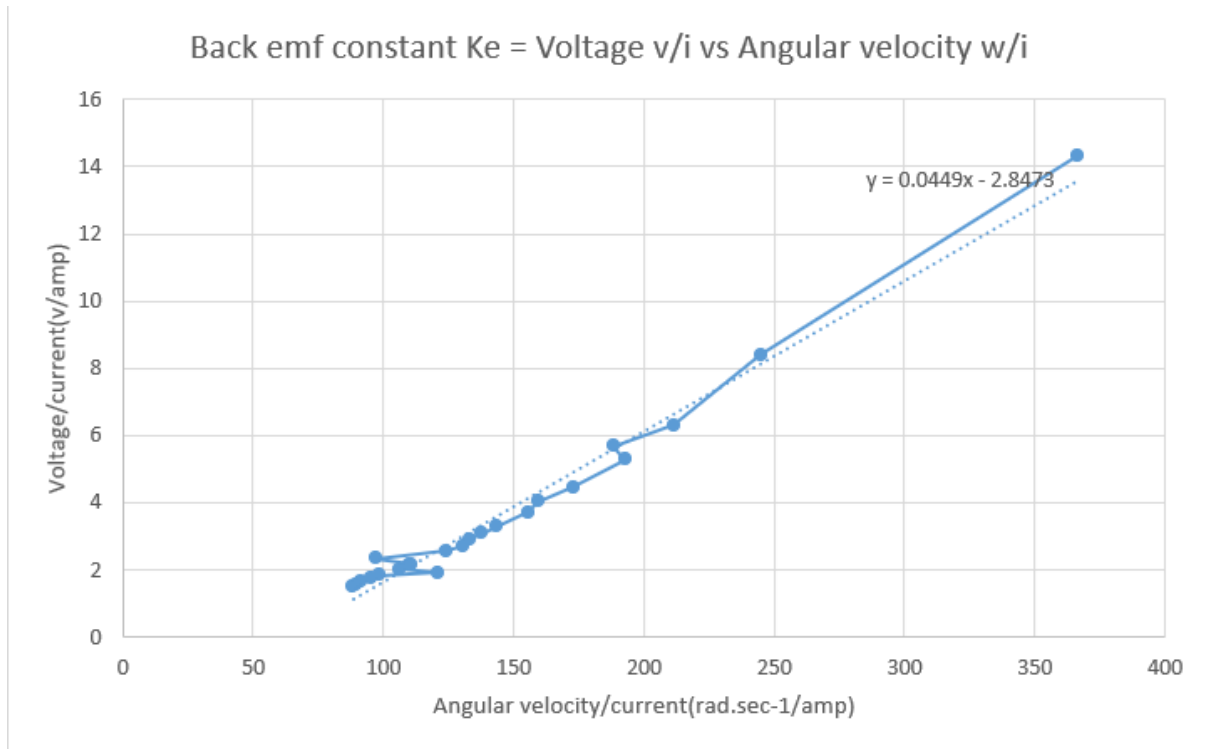


Figure 4-23 - Back emf constant K_e

From Figure 4-23 above the back emf constant $k_e = 0.0449$ V.s/rads

Similarly, the equation for force can be determined using the equation derived in Chapter 5, equation 4.76 or 6.26. To estimate the coefficient between voltage square (V^2) and thrust (F) can be obtained from the following Figure 4-24.

$$F = 2\rho A \left[\frac{f\eta b}{Kq} \right]^2 V^2 \quad 4.59$$

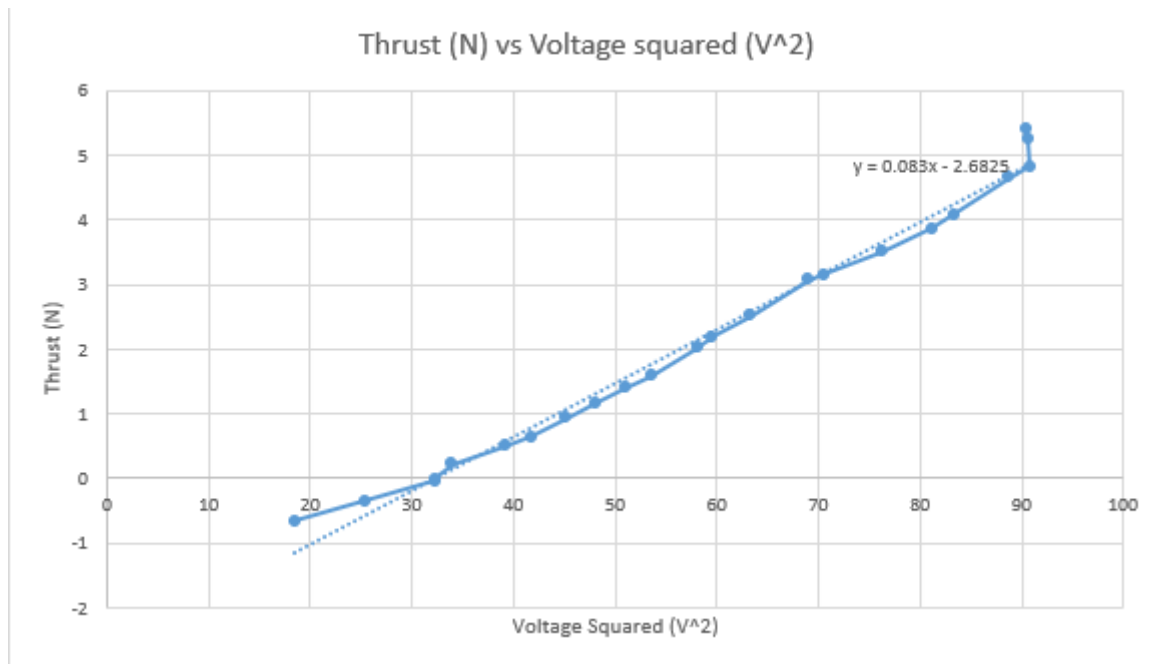


Figure 4-24 - Thrust and Voltage squared relationship

By taking the relative relationship between angular velocity (ω_m) and the thrust, the thrust coefficient (b) can be identified. In this journal (Al, Dewi and Hidayat, 2018) the thrust coefficient is derived using a different equation, and the thrust coefficient is known as C_T .

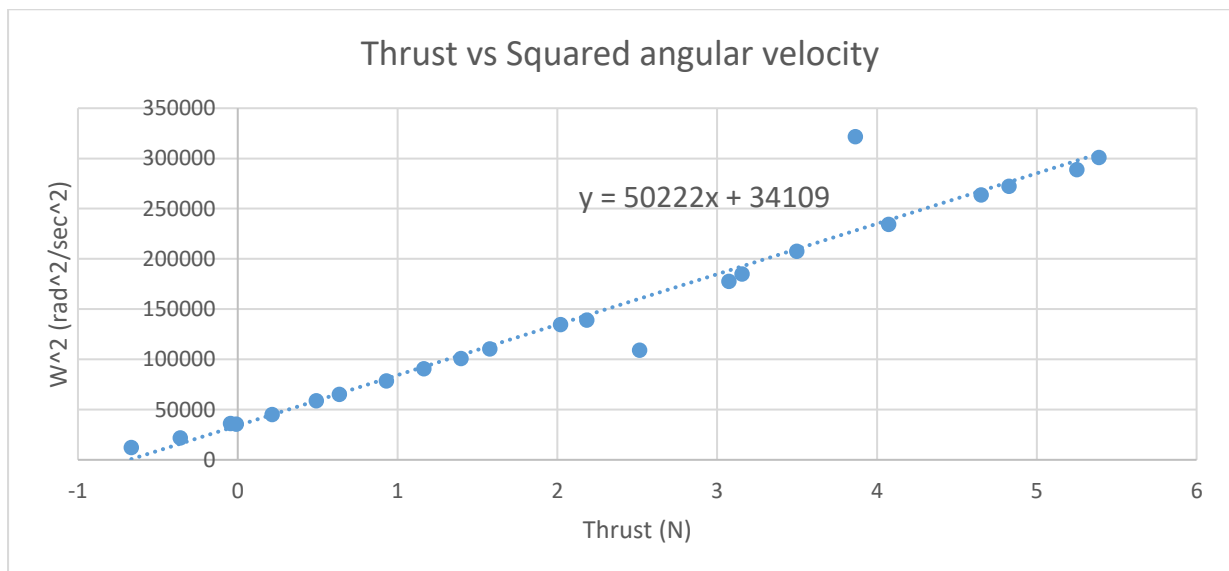


Figure 4-25 - Relationship between thrust and squared angular velocity

According to the equation

3.99 and the length of the frame used, the thrust factor will be;

$$b = 6.3 \times 10^{-5}$$

Figure 4-26 is used to determine the linear regression between the voltage (V) and the propeller velocity squared (Ω^2), to build the inverse relationship between the voltage and the angular velocity the coefficient of the linear regression has been used.

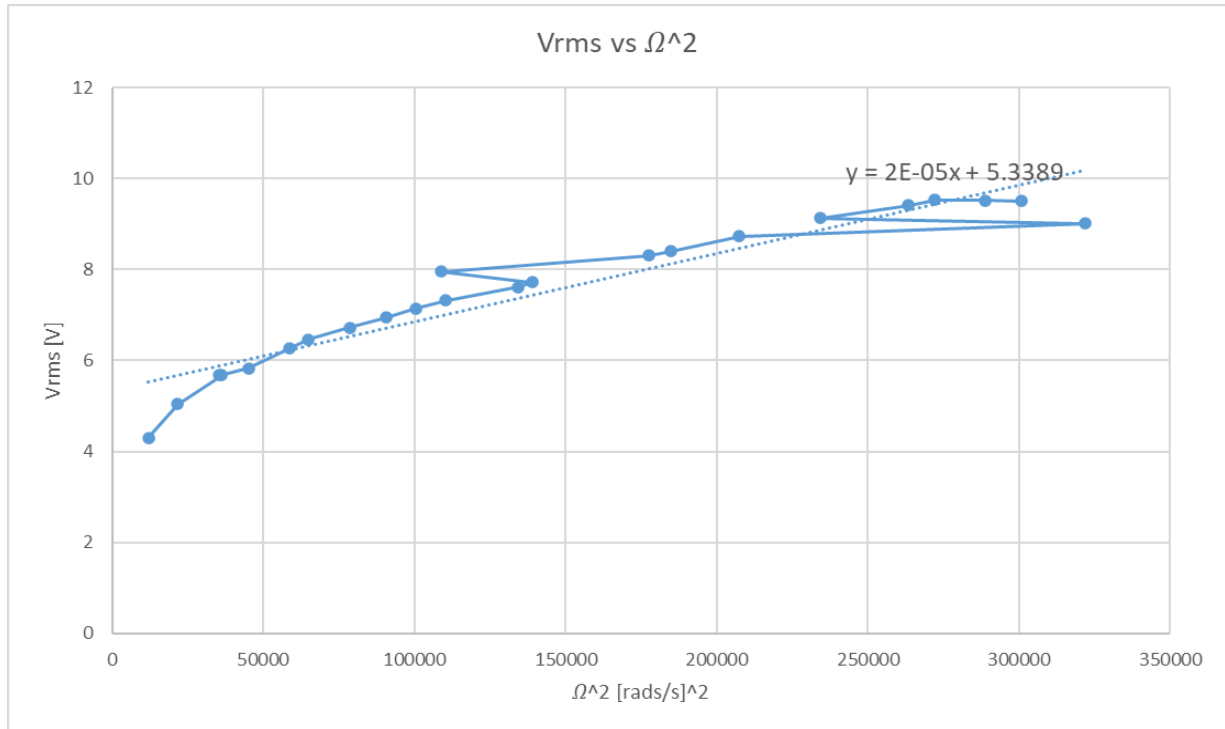


Figure 4-26 - Voltage vs Angular velocity squared

Figure 4-27 allows scaling the PWM ESC regulator and the voltage (V) which will help later to build the controller voltage command, and show the linear graph.

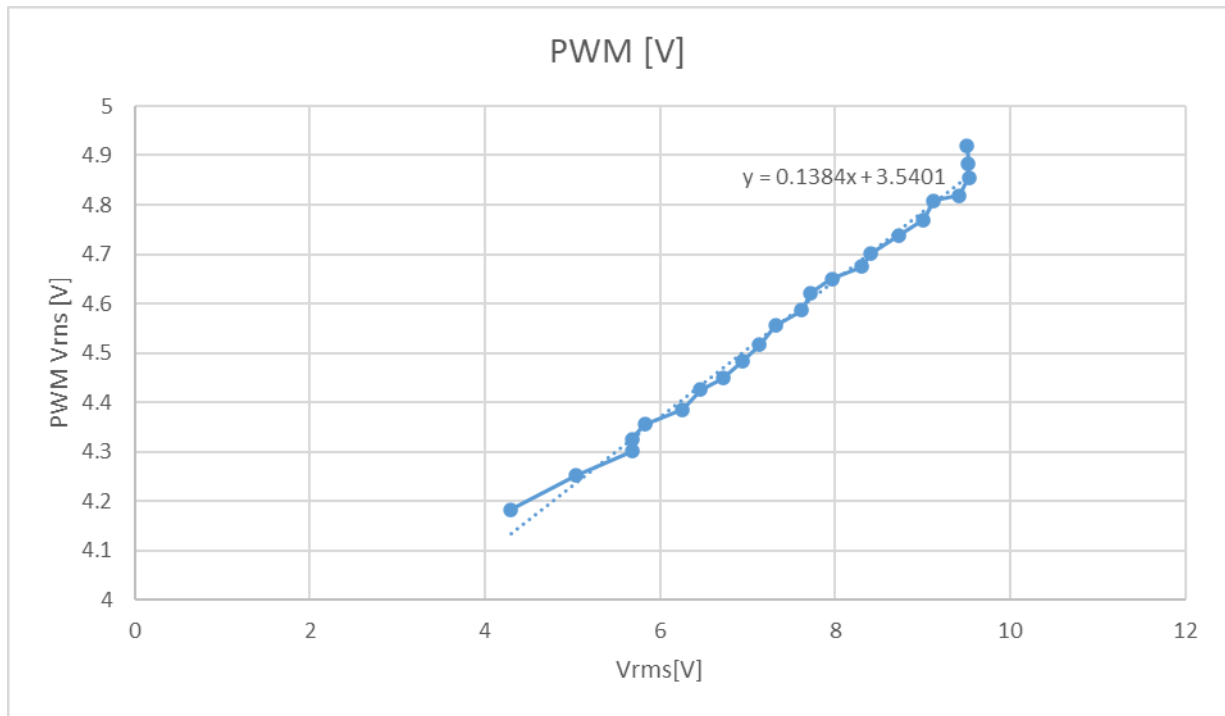


Figure 4-27 - PWM vs Voltage input

4.4.4 Empirical measurement of motor inductance

The importance of having an accurate inductance value cannot be understated, especially when developing an equivalent model for a robust control system. The inductance value is relatively small; therefore, they are neglected in the majority of research papers. Also, the equipment required to obtain accurate readings is uncommon and expensive, such as an LCR meter. To overcome this issue, a few methods were attempted to measure the approximate inductance value of each winding of the BLCD motor. The methods are compared against LCR meter readings shown in Figure 4-28 below. These readings were taken while the rotor housing was intact on the brushless dc motor; hence the rare earth magnets (permanent magnets) may cause slightly inaccurate readings.

	Jdrone Inductance L (uH)			
Freq (kHz)	Red-Black	Red-Blue	Black-Blue	
25	26.49	28.58	25.74	
10	27.47	30	26.8	
1	28.8	31.7	31.3	
0.12	30	31	36	
0.1	31	52	34	
Average uH	28.752	34.656	30.768	31.39

Figure 4-28 - LCR meter readings

4.4.4.1 Vary frequency until half of supply voltage peak-to-peak

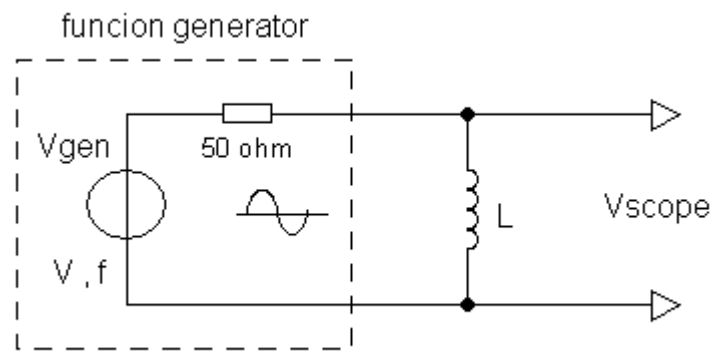


Figure 4-29 - Method 1 - Equivalent circuit diagram(A Simple Method to Measure Unknown Inductors, no date)

Measure the output impedance of the signal generator

- a) Measure V_{p-p} using signal generator probes with no load (see Figure 4-30).



Figure 4-30 – Signal generator impedance - i/p

- b) Load the function generator with a load resistor, measure V_{p-p} using a resistor in series with probes as shown in Figure 4-31.



Figure 4-31 - Signal generator impedance - with load

c) Measure the resistor value (see Figure 4-32).



Figure 4-32 - Signal generator impedance

Use the following equation to determine the output impedance of the signal generator

$$Z_{out} = R_L \times \left(\frac{V_{in}}{V_L} - 1 \right) = 0.987 \times \left(\frac{4.04}{3.84} - 1 \right) = 51.406 \, \Omega \quad 4.60$$

The result confirms the impedance is close enough as stated on the signal generator 50Ω.

2. Determining frequency

Set signal generator V_{p-p} as required ($4 V_{p-p}$ in this example).

Connect the inductor/windings in series

Vary the frequency to obtain half the set input V_{p-p} ($2V_{p-p}$ in this case)

Make note of the frequency when the V_{p-p} reaches step c.

3. Calculate the inductance

$$L = \sqrt{\left(\frac{1}{3}\right)} \times \frac{Z_{out}}{2\pi f} \quad 4.61$$

	Freq (KHz)	L(uH)
Red-Black	280	16.8700
Red-Blue	270	17.4948
Blue-Black	298	15.8510

Figure 4-33 - Inductance at higher frequencies - Method 1 (Half input V)

4.4.4.2 Further analysis

There were two more attempts made to determine the inductance of the BLDC motor. One was using simply increasing/decreasing the frequency until the voltage drop across a series resistor and coil windings are equal. Once the frequency is obtained, rearrange the equation $L = \frac{X_L}{2\pi f}$.

Here the X_L is the resistor in the circuit which should be equivalent to the output impedance of the signal generator (51.406Ω as calculated above) (*Coil Inductance, Measurement and Calculation.wmv*, no date).

The second method follows the document provided by Tektronix inductance measurement (*Capacitance and Inductance Measurements Using an Oscilloscope and a Function Generator*, no date) and KTE direct(*KDE Direct - Measuring Motor Inductance*, no date). It has a similar setup and an oscilloscope with 2 channels is used to measure the frequency, phase difference and amplitudes of channel 1 and channel 2 (see Figure 4-34). According to the document the frequency is set at 10 kHz. However, at 10 kHz the signal is too low at channel 2. So it was increased to 150 kHz as per KTE direct.

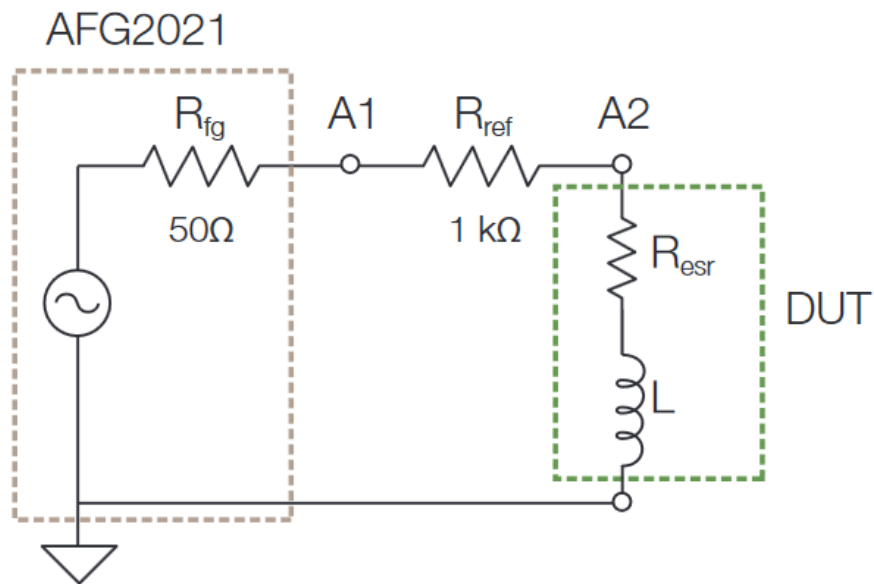


Figure 4-34 - Inductance measurement - Tektronix circuit diagram(Capacitance and Inductance Measurements Using an Oscilloscope and a Function Generator, no date)

The parameters are set as follows;

Function generator amplitude = 2.5V

Reference resistor = 1 kΩ

Function generator frequency = 150kHz

Function generator waveform = Sinewave

The phase difference and the amplitude values measured at 150 kHz frequency can be seen in Figure 4-35.

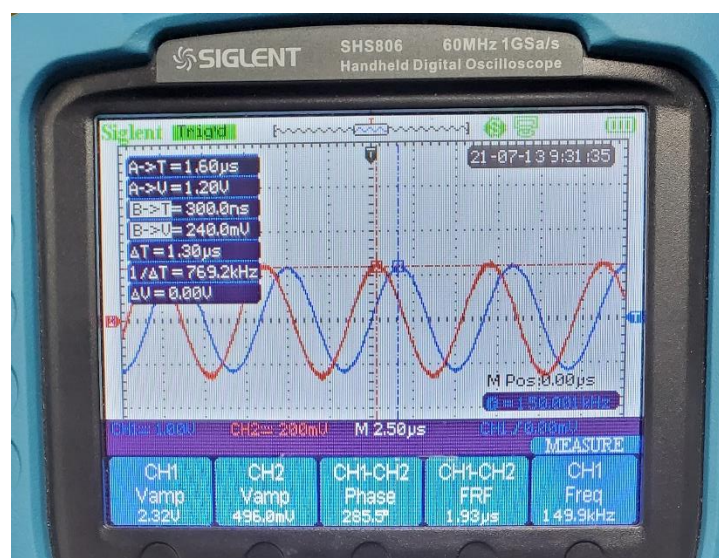


Figure 4-35 - Tektronix method result

The following equations are used to calculate impedance Z , coil resistance R_{ESR} and inductance L .

$$\theta = 360 \times freq_{in} \times phase\ difference\ in\ s$$

$$\theta = \frac{360}{period} \times \Delta = \frac{360}{6.667us} \times 1.3us$$

$$\theta = 70.2^\circ$$

$$Z = \frac{V_{ch2}}{\sqrt{V_{ch1}^2 - 2V_{ch1}V_{ch2}\cos(\theta) + V_{ch2}^2}}$$

$$Z = \frac{496mV}{\sqrt{2.32V^2 - 2 \times 2.32 \times 496mV \times \cos(70.2) + 496mV^2}}$$

$$Z = 0.2376\ \Omega$$

$$\alpha = \theta - \tan^{-1} \frac{-V_{ch2} \times \sin(\theta)}{V_{ch1} - V_{ch2} \times \cos(\theta)}$$

$$\alpha = 70.2^\circ - \tan^{-1} \frac{-496mV \times \sin(70.2^\circ)}{2.32 - 496mV \times \cos(70.2^\circ)} = 82.826^\circ$$

$$R_{ESR} = Z \times \cos(\alpha)$$

$$R_{ESR} = 0.2376 \times \cos(82.826) = 19.67\Omega$$

$$L = \frac{Z \times \sin(\alpha)}{2\pi f}$$

$$L = \frac{0.2376 \times \sin(82.826)}{2\pi \times 150kHz} = 0.00000025\ H$$

4.4.5 Parameters implemented in the simulation

After carrying out the experimental test of the motors with propellers, using the test quadcopter build, the table below shows the parameters extracted, and the extraction procedures used in various ways to determine them. First from the previous study and then exploiting the datasheet, and finally, the experiment test is done.

Table 4-4 - Table of physical parameters

Parameters	Symbol	Value	Unit	Sources
Rotational inertia along the x-axis, y-axis	I_{xx}, I_{yy}	0.0075	Kg.m ²	Bifilar pendulum
Rotational inertia along the z-axis	I_{zz}	0.0013	Kg.m ²	Bifilar pendulum
Total Mass of the Quadcopter	M	0.65	Kg	Datasheet
Drag Constant	d	7.50E-7	N.s ²	Previous study
Propeller Radius	R	0.15	m	Datasheet
Propeller Disc Area	A	0.0490873	m ²	Datasheet
Rotor Inertia	J	6.04E-5	Kg.m ²	Previous study
Arm Length	L	0.23	m	Datasheet
Motor Resistant	Z	0.06	Ohms	Experiment & Datasheet
Motor Torque Constant	K_q	0.00039	Nm/A	Experiment & Datasheet
Motor Speed Constant	K_e	0.00039	V.s/rad	Experiment
Motor Efficiency	η	80	%	Datasheet
Air Density	ρ	1.2	Kg.m ³	Previous study
Figure of merit	f	0.5	NIL	Previous study
Thrust / Torque Constant	b	0.0008145	m	Experiment

4.5 Summary

This chapter describes the linearisation of system equations and why SS equational was chosen over the Transfer function. Also, how to extract the essential physical parameters of a quadcopter. Using the Laplace transform, the complicated time-domain equations can be reduced to simple algebraic form. It gives the mathematical model of the entire system as well as each component of the system. For a known transfer function, it is simple to determine the output response for any reference input. It helps determine essential system properties such as poles, zeros, etc. Using the transfer function, it is straightforward to analyse the system's stability. It is useful to correlate output and input. However, the Transfer function cannot be used in a non-linear system. Also, the system derived from Transfer function equations considers initial conditions as zero.

The chapter explains the use of Parrot Mambo as a reference and compares data available in Simulink. Different approaches are taken to determine the values, especially to obtain the moment of inertia.

There are two methods to determine the thrust-to-weight ratio of a motor, empirical data from manufacture or bench test. Since the manufacturer's data was unavailable, the latter option had to be taken. Another reason is the propeller properties are also unavailable.

The primary reason for taking empirical measurements is because lack of information in the datasheet, especially BLDC parameters.

Chapter 5 - Simulation of Quadcopter Model in MATLAB/Simulink

This chapter provides the development of a dynamic model of a quadcopter. It is focused on modelling the quadrotor in MATLAB/Simulink based on derived theoretical equations and parameters extracted through various experiments in previous chapters. This includes modelling subsystems for throttle, roll, pitch, yaw etc. These subsystem models combine the heart of the system. From here onwards, various controllers can be tested, and new parameters can be entered to observe system response.

5.1 MATLAB/Simulink Software

MATLAB is a programming platform designed specifically for engineers and scientists to analyse and design systems and products that transform our world. MATLAB is widely used in academia, for teaching and research. It is also used by thousands of professionals in the aerospace, automotive, communications, electronics, financial services, healthcare, manufacturing, pharmaceutical, and software industries. It is easy to learn yet has the power to solve tough technical computing challenges. MATLAB is a high-level language for technical computing that integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. It is used for a wide range of applications, including signal and image processing, communications, control design, test and measurement, financial modelling, and computational biology.

The power of MATLAB comes from its ability to solve technical computing challenges using existing code and libraries, its familiar mathematical notation, and its interactive environment. It has a simple syntax, familiar mathematical notation, and an interactive execution environment. MATLAB required very little or no programming experience. The MATLAB toolbox contains standard functions for control and estimation, including filters, Kalman filtering, particle filters, state-space models, and numerous other functions that support these areas. You can extend the capabilities of MATLAB by writing programs or by using optional toolboxes such as Signal Processing Toolbox™ and Simulink®.

Simulink is a MATLAB-based graphical programming environment for modelling, simulating, and analysing multi-domain dynamical systems. Its primary interface is a graphical block diagramming tool and a customizable set of block libraries. The Simulink language has three types of elements: blocks, which represent physical or mathematical entities; wires, which connect blocks; and functions (or "blocks" in the Simulink Blocks language), which perform computations on signals.

The equations and models' subsystems are developed using the "+" quadcopter configuration.

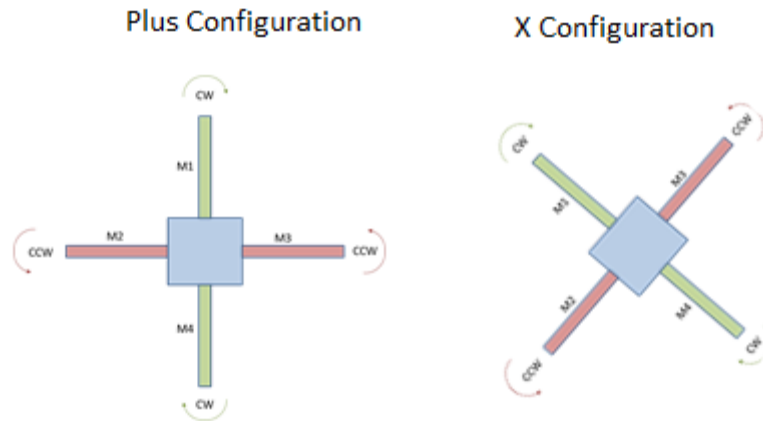


Figure 5-1 - Quadcopter configurations

5.2 Model Implementation in MATLAB/Simulink

This section covers integrating and developing subsystems on the Simulink model from equations derived in the above chapter. Figure 5-2 below gives a summary of the entire system in a block diagram. Where the highlighted block diagram represents the subsystem to be focused on in this section. This will be used as a template throughout this section to focus readers' attention.

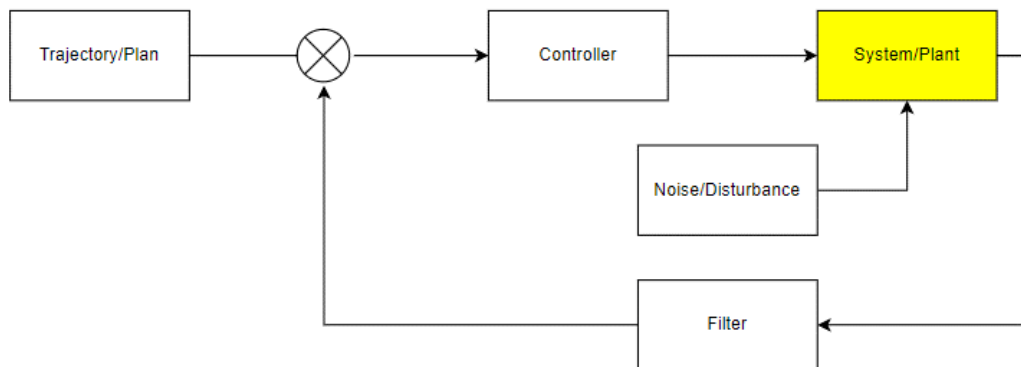


Figure 5-2 - Block diagram representing the architecture of the system - Plant subsystem

5.2.1 Actuator Subsystem

The electric motor takes in voltage and converts it to motor speed or angular velocity (rad/sec). The derivation of the equation can be found in 3.8 above. The section focuses on building a single-motor model. Three more replicates are made of motor model and connected to input voltages, V1, V2, V3, and V4 respectively.

$$\Omega = \frac{K_q}{J_R} \times \int V - K_e \Omega - \frac{RD}{K_q} \times \Omega^2 \quad 5.1$$

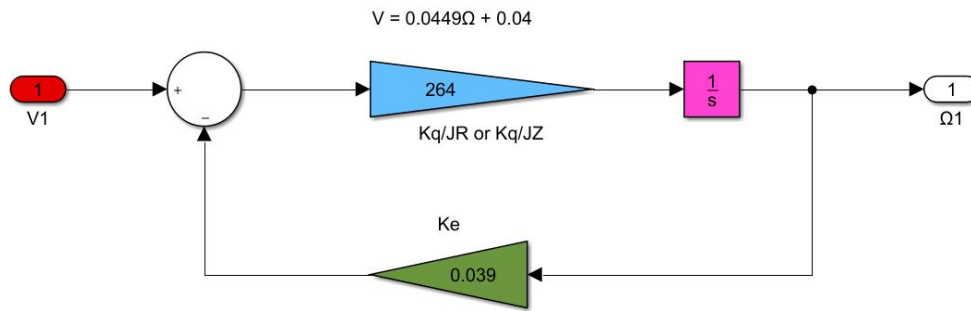


Figure 5-3 - Actuator subsystem

5.2.2 Roll Subsystem

The roll subsystem consists of constants derived from Table 4-4. There are two voltages V_2 and V_4 representing right and left motors of the quadrotor, in which one voltage is subtracted in one of these motors and by the same amount added to other motors to perform roll manoeuvre.

$$\ddot{\phi} = \frac{2\rho A}{m} \left[\frac{f\eta k_t}{k_q} \right]^2 (V_2^2 - V_4^2) \quad 5.2$$

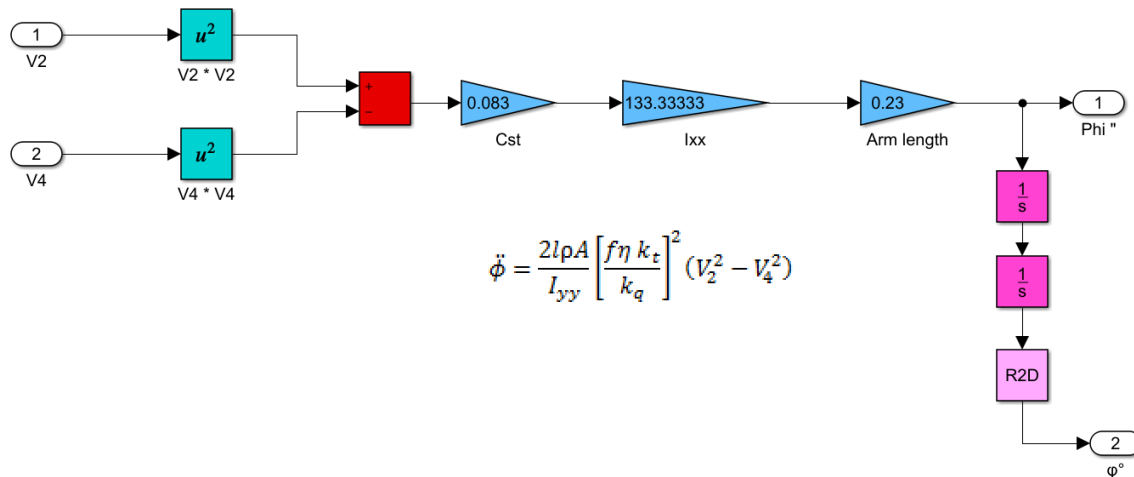


Figure 5-4 - Roll subsystem

5.2.3 Pitch Subsystem

$$\ddot{\theta} = \frac{2\rho A}{m} \left[\frac{f\eta k_t}{k_q} \right]^2 (V_3^2 - V_1^2) \quad 5.3$$

The pitch subsystem is similar to the roll subsystem. There are two voltages V_1 and V_3 representing front and back motors of the quadrotor, in which one voltage is subtracted in one of these motors and by the same amount added to the other motor to perform pitch manoeuvre.

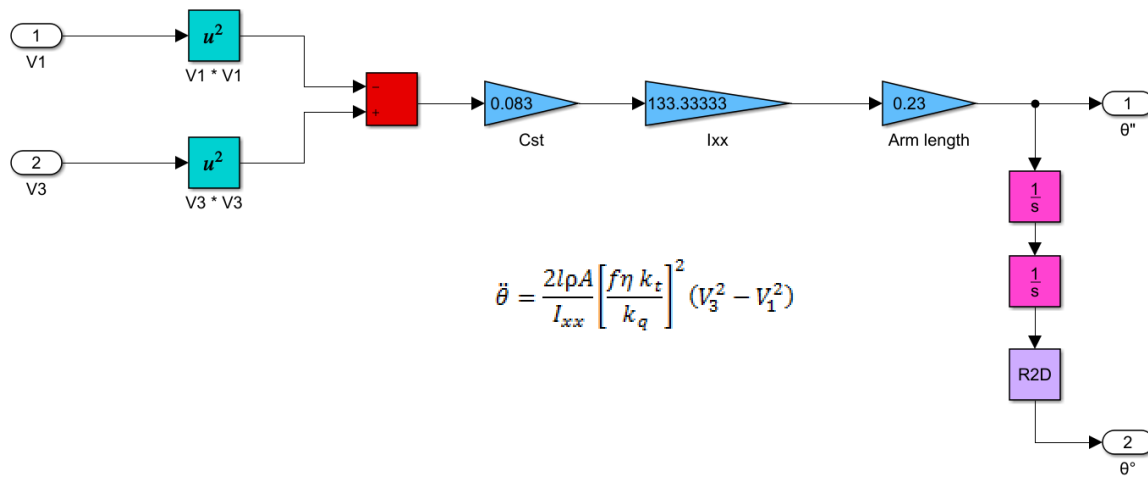


Figure 5-5 - Pitch subsystem

5.2.4 Yaw Subsystem

In the yaw subsystem, there are four angular velocities and four angular displacements multiplied with constant gain from Table 4-4.

$$\ddot{\psi} = \frac{J}{I_{zz}} (\dot{\Omega}_1 + \dot{\Omega}_3 - \dot{\Omega}_2 - \dot{\Omega}_4) + \frac{D}{I_{zz}} (\Omega_1^2 + \Omega_3^2 - \Omega_2^2 - \Omega_4^2) \quad 5.4$$

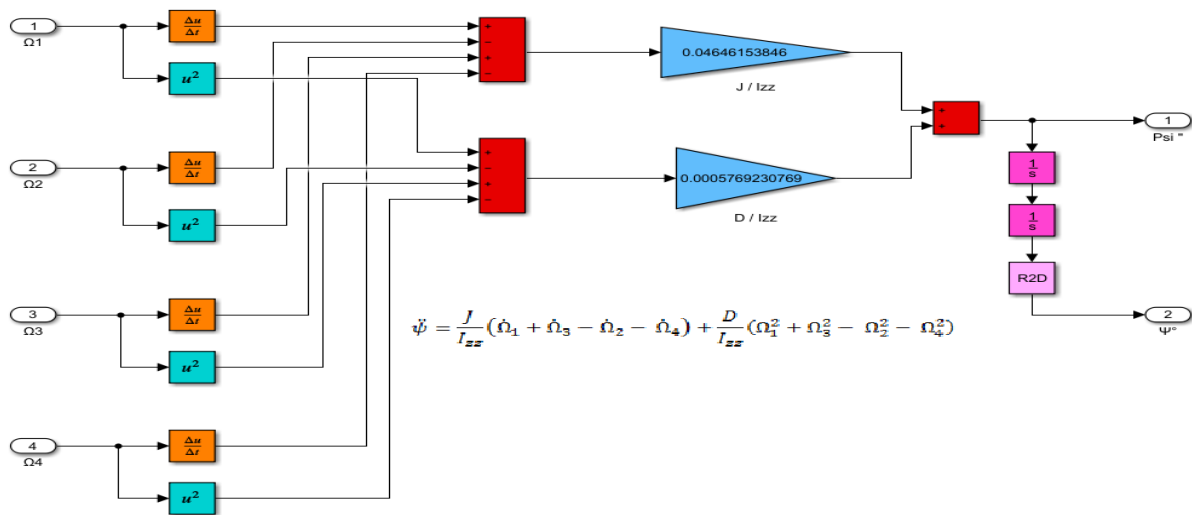


Figure 5-6 - Yaw subsystem

5.2.5 X-Motion Subsystem

$$\ddot{x} = \frac{2\rho A}{m} \left[\frac{f\eta k_t}{k_q} \right]^2 (V_1^2 + V_2^2 + V_3^2 + V_4^2) (\cos\phi \sin\theta \cos\psi + \sin\phi \sin\psi) \quad 5.5$$

Equation 5.5 is based on differential equations for motion along the x-axis. The result is given in acceleration along the x-axis. Integrating the output once gives linear velocity and twice gives the position of the quadrotor along the x-axis (see Figure 5-7 below).

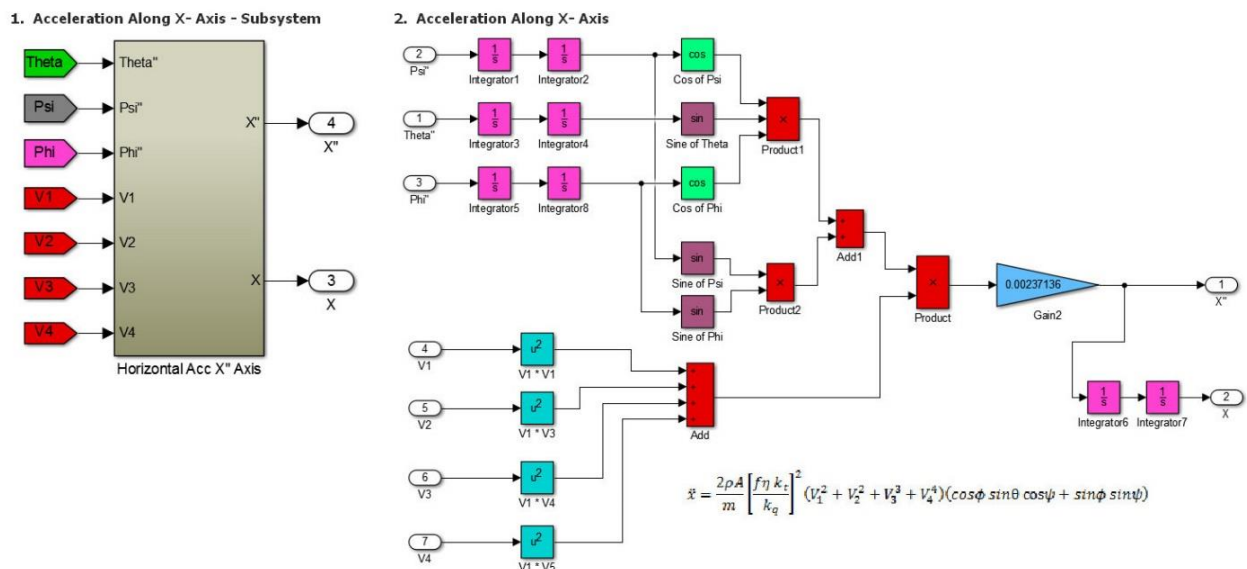


Figure 5-7 - Linear acceleration along the x-axis

5.2.6 Y-Motion Subsystem

$$\ddot{y} = \frac{2\rho A}{m} \left[\frac{f\eta k_t}{k_q} \right]^2 (V_1^2 + V_2^2 + V_3^2 + V_4^2)(\sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi) \quad 5.6$$

Equation 5.6 is based on differential equations for motion along the y-axis. The result is given in acceleration along the y-axis. Integrating the output once gives linear velocity and twice gives the position of the quadrotor along the y-axis (see Figure 5-8 below).

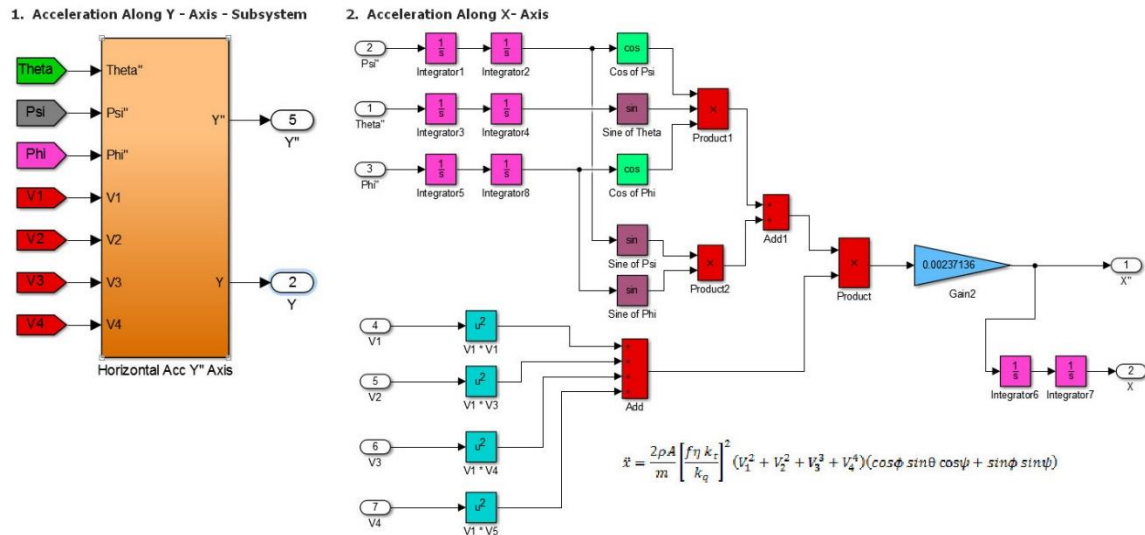


Figure 5-8 - Linear acceleration along the y-axis

5.2.7 Z-Motion Subsystem

$$\ddot{z} = \frac{2\rho A}{m} \left[\frac{f\eta k_t}{k_q} \right]^2 (V_1^2 + V_2^2 + V_3^2 + V_4^2)(\cos\theta\cos\phi) - g \quad 5.7$$

The equation is based on differential equations for motion along the z-axis. The variable inputs required to determine z-acceleration are pitch and roll's angular displacement as depicted in image 2 of Figure 5-9. There is also an additional constant, force of gravity, 9.81. The result is given in acceleration along the z-axis. Integrating the output once gives linear velocity and twice gives the position of the quadrotor along the z-axis

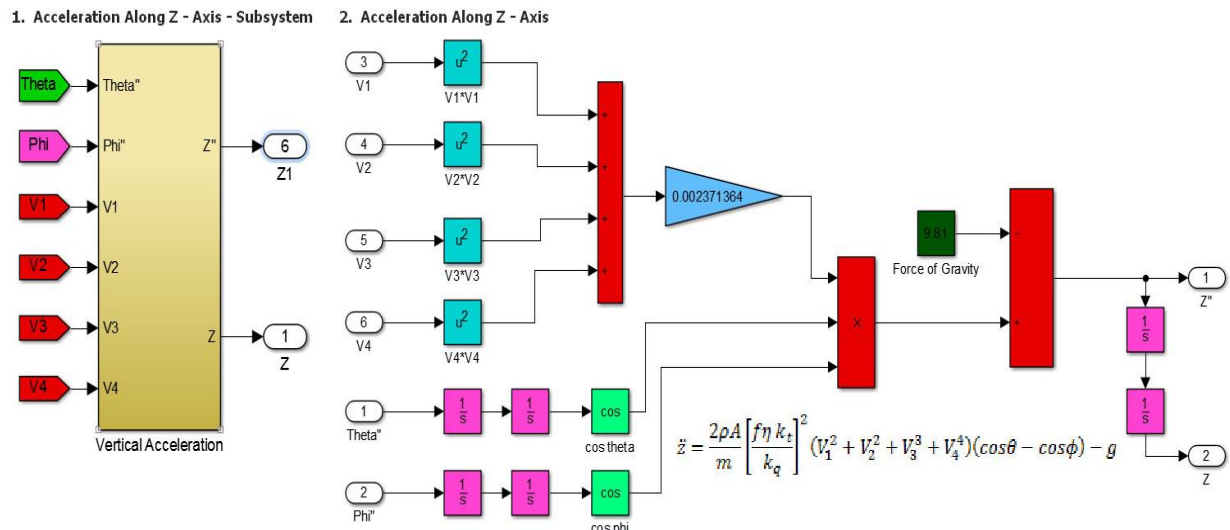


Figure 5-9 - Vertical acceleration along the z-axis

5.3 Mathematical Subsystem Model

Figure 5-10 shows the Simulink subsystem model comprised of mathematical equations.

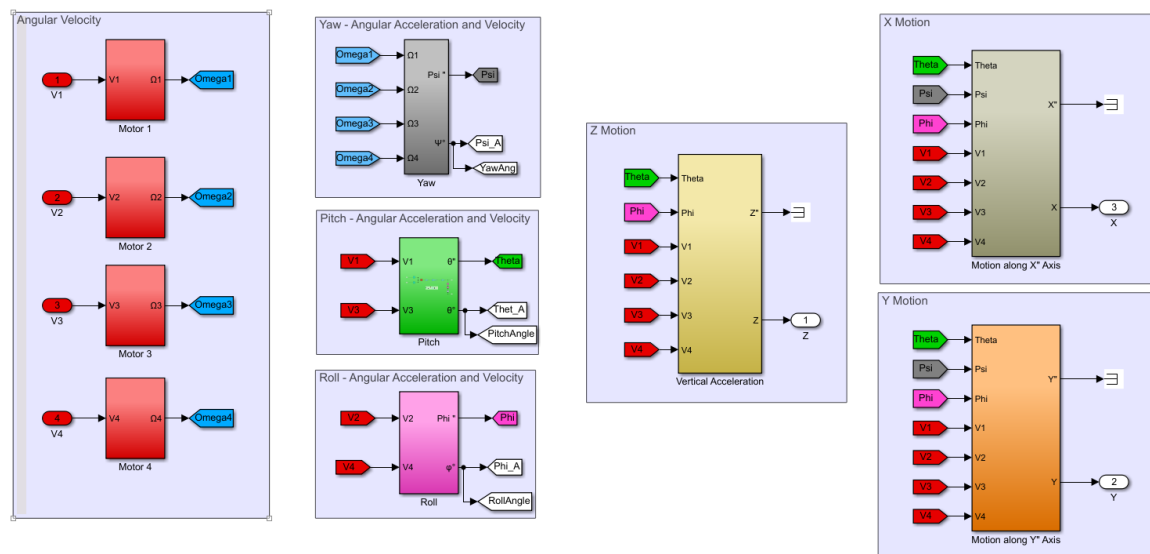


Figure 5-10 - Mathematical model subsystem

5.4 Motor Mixer

The following block set in Figure 5-11 consists of a motor mixer that diverts signals to appropriate motors. The block set takes in torque/moment parameters coming from the controller and converts them to the voltage output. The output voltages are channelled to appropriate quadcopter dynamic subsystems developed above for further processing.

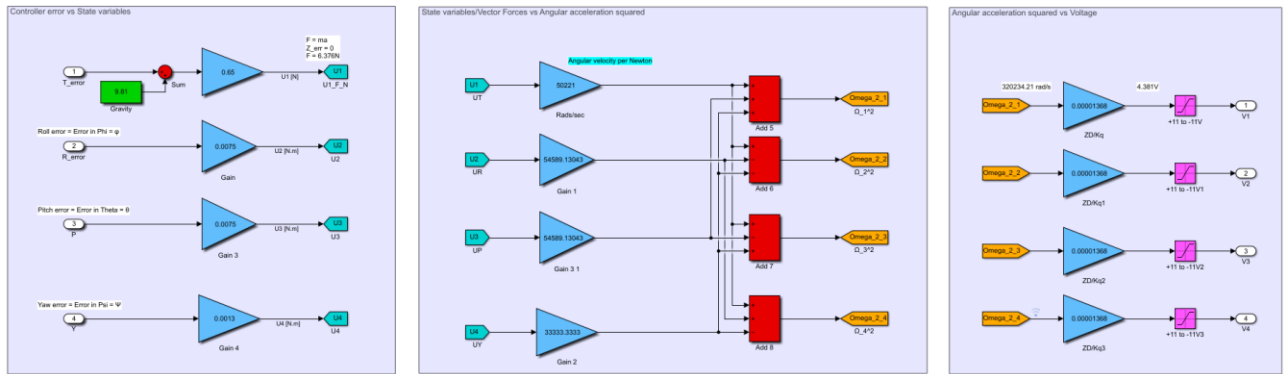


Figure 5-11 - Motor Mixer

5.5 The Open Loop Model

A control system is a collection of mechanical or electronic components that, through the use of control loops, are used to regulate the operation of other devices or systems. Control systems are almost always run by computers these days. Control systems are an essential component of both the manufacturing sector and the automation sector. There are primarily two categories of control loops: open loops, which function with input from humans and closed loops, which run completely independently of humans. Some loops provide the ability to toggle between the open and closed states. When the switch is in the closed position, the tunable loop is fully automated, but when it is open, it is managed manually.

This subsection deals with combining both controller and plant subsystems as shown in Figure 5-12.

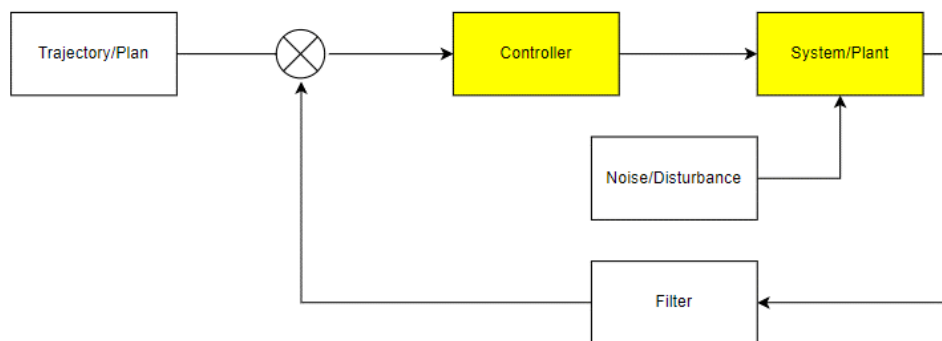


Figure 5-12 - Block diagram representing the architecture of the system - Controller and Plant

Figure 5-13 below represents an open-loop model. Control designs, integration of sensors and assigning and planning set points begin from this model.

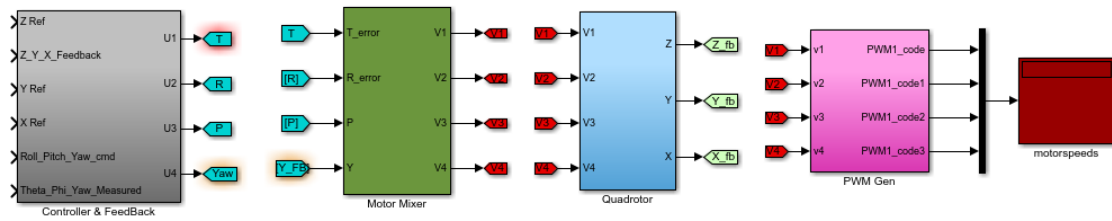


Figure 5-13 - Open-loop model

Before a controller is made an open loop test is carried out to ensure all connections are made. Although the output from an open-loop test may not be comprehensible, especially when the simulation is run for a longer period, it enables a system designer to visualise random outputs before a controller is incorporated. The system designer can fine-tune the controller and fix errors designed so far by running Simulink for a few seconds. The open-loop test is conducted using a joystick by a human with random inputs, therefore an input reference/setpoint is unavailable.

5.6 Controller

A control system is a collection of mechanical or electronic components that, through the use of control loops, are used to regulate the operation of other devices or systems. Control systems are almost always run by computers these days. Control systems are an essential component of both the manufacturing and the automation sector.

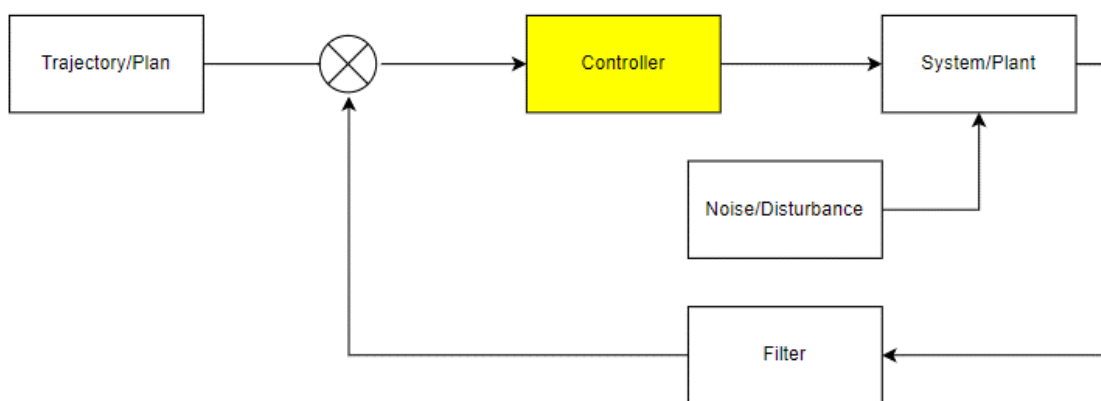


Figure 5-14 - Block diagram representing the architecture of the system - Controller subsystem

The controller being used is a PID controller. PID controller controls both the inner and outer loop of the system Figure 5-15. The parameters were tuned using the trial and error method.

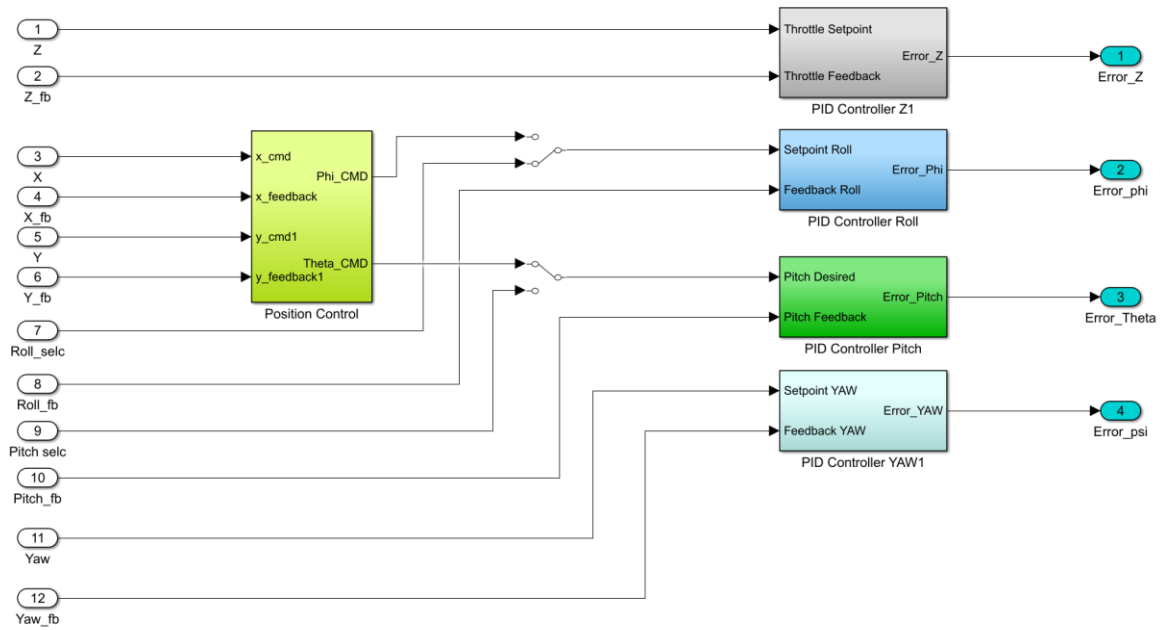


Figure 5-15 - Inner and outer loop

5.7 Kalman filter modelling

The KF in this Simulink model is intended to estimate the actual output of the quadcopter and filter out any noise generated during processing. One of the key steps in this work is the implementation of KF in Simulink. Filtering is performed using KF blocks and block input parameters in the Simulink Control System Toolbox library.

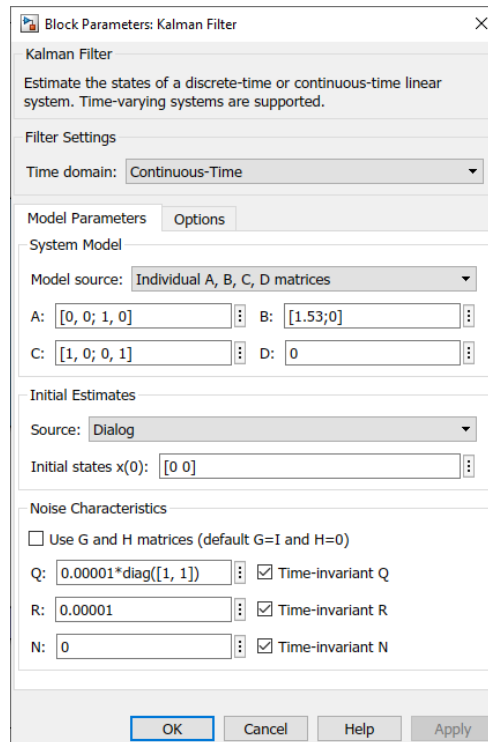


Figure 5-16 - Kalman filter block parameters

Time Domain: Continues or Discrete chosen depending on the application;

Model state-space representation A, B, C, D matrices;

y[n]: Current measurement as input for the Block;

u: which is the process input;

X [0]: initial state for variable;

Q matrix covariance process noise $w[n]$, is assumed to be time-invariant is less performance than the time-variant, but has a lower computational cost and is easier to design.

R matrix covariance of the measurement noise $v[n]$ which assumes the sensor bias.

N matrix assumes that there is no correlation between measurement and process noises.

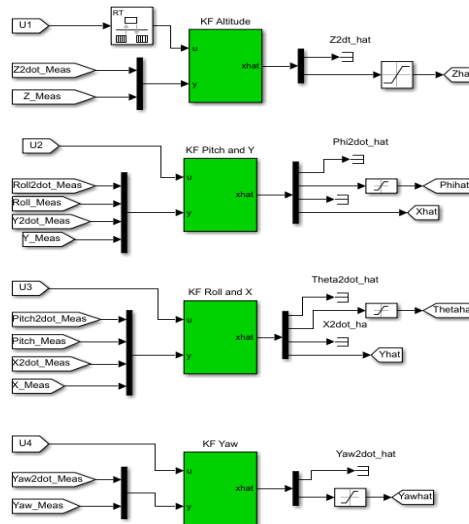


Figure 5-17-Pitch, Roll, Yaw, Altitude - KF Simulink block diagram

The main parameters which are influenced by the KF filter are the state variables and the covariance matrices of Q and R. As can be seen in Figure 5-17, independent KF filters are implemented for each manoeuvre and all physical parameters are considered within each block for best response.

5.8 Running the Simulation

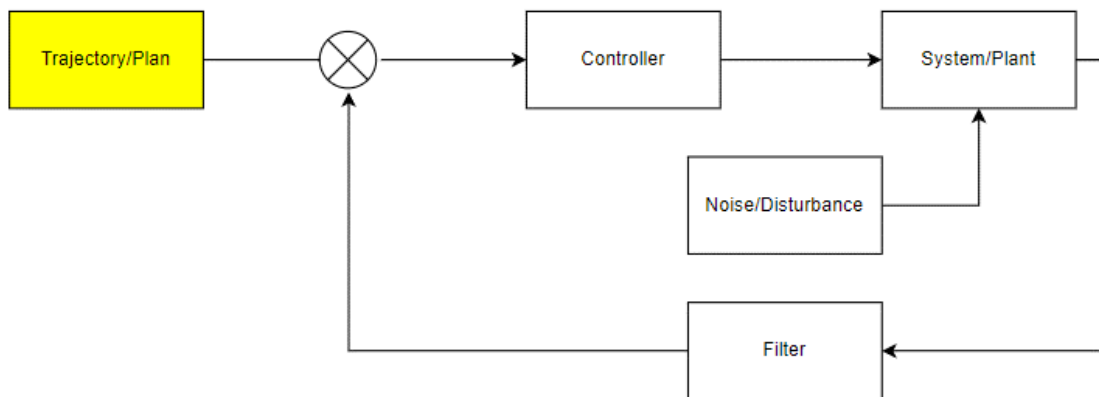


Figure 5-18 - Block diagram representing the architecture of the system - Trajectory subsystem

The trajectory planner (Figure 5-18) is implemented in this section. Figure 5-19 below is used as a reference to test the closed-loop system. This reference will vary through this research to determine the system tolerance and limitations of the controller.

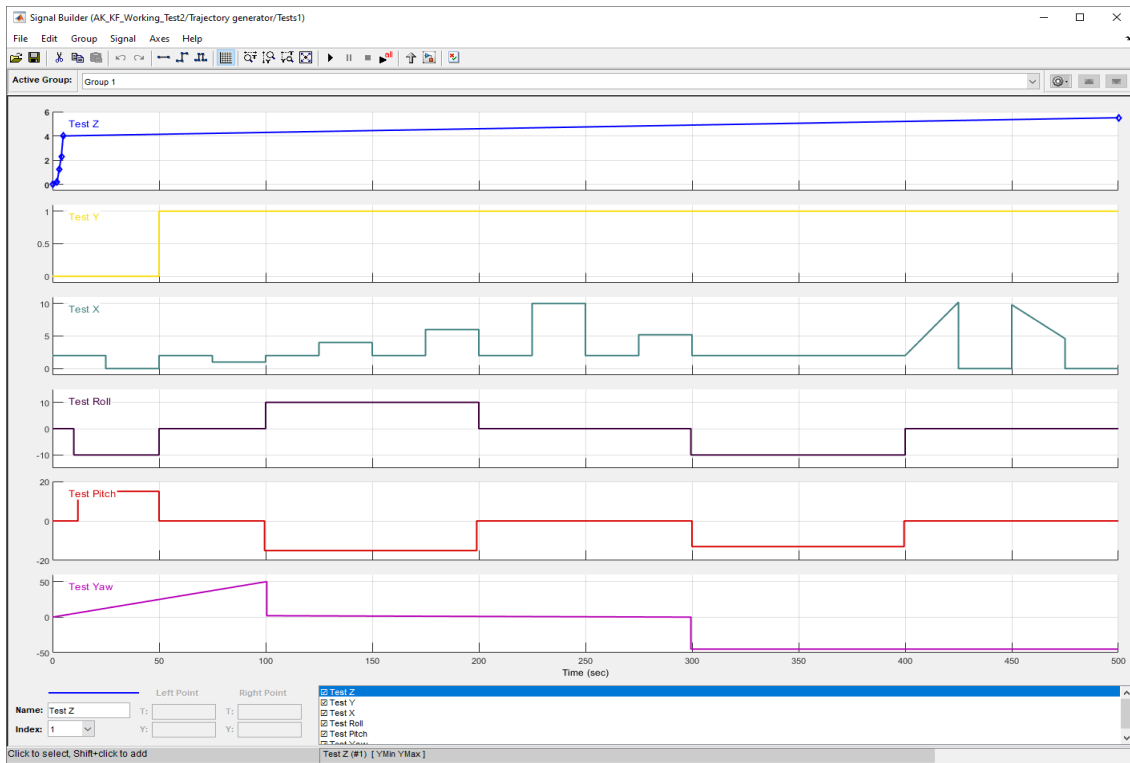


Figure 5-19 - Open loop trajectory planner

The simulation results were executed using a signal generator to mimic the trajectory (Figure 5-20) of the quadcopter and a Cyborg 3D Rumble Force joystick (Figure 5-21) was also used as an input to the quadcopter model. Initially, the signal generator was used to automate trajectory in order to tune controllers and for debugging.

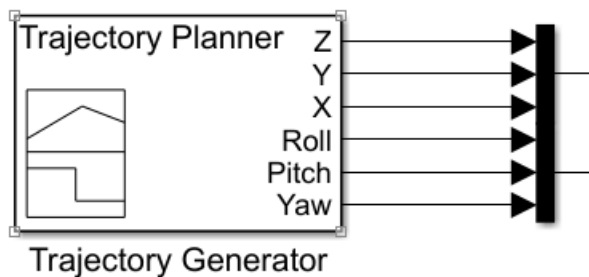


Figure 5-20 - Signal generator



Figure 5-21 - Joystick

5.8.1 Noise

The noise block sets shown in Figure 5-22 (process noise) consist of random Gaussian white noise for plant for thrust vector input U_T (U_1, U_2, U_3, U_4) and state vectors z, ϕ, θ, ψ . The

process noise is added into motor mixer subsystem. Also, the measurement noise is added which replicates noise generated in unfiltered sensors. The noise frequency and amplitude of the plant (process noise) and sensor noise are determined by random selection and from previous studies. The noise of both plant and sensor of the real quadcopter is calculated by The process noise that goes through the dynamics of the quadcopter and the measurement noise is directed to the KF subsystem for estimation and prediction. Figure 5-23 represents the block set where noise is generated and Figure 5-24 shows measurement noise entering the KF filter block.

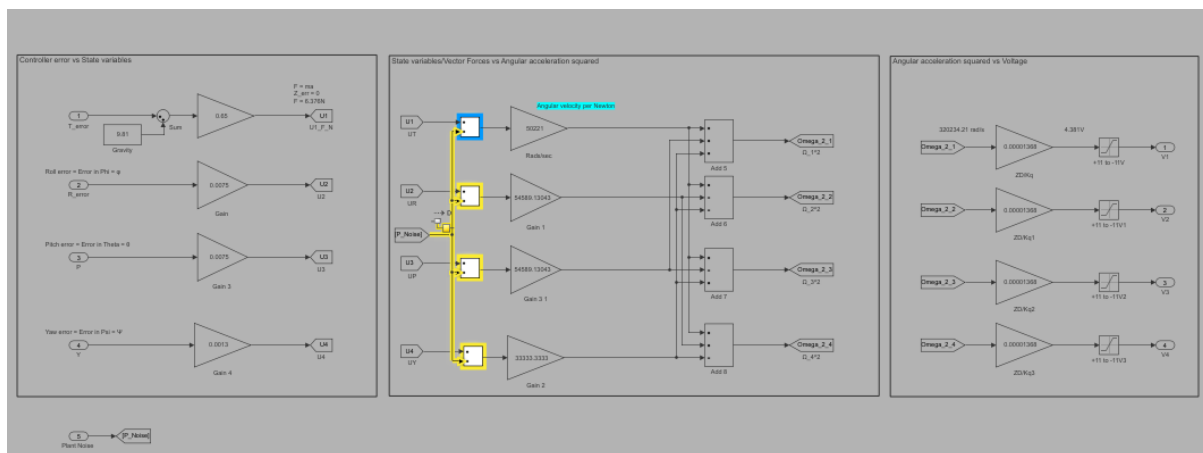


Figure 5-22 - Noise block set to plant model

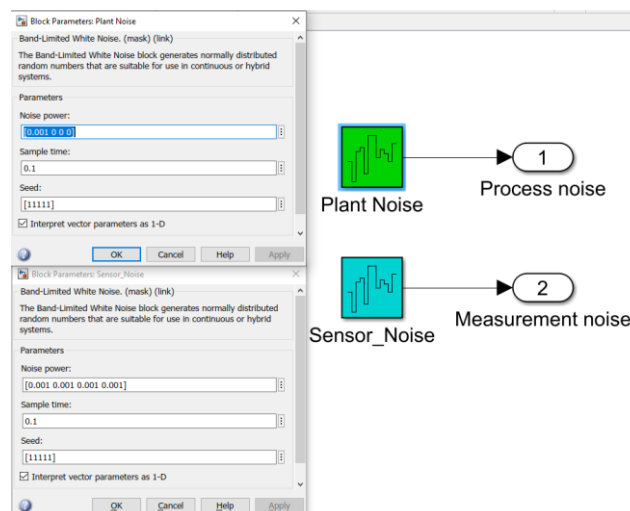


Figure 5-23 - Gaussian white noise generator block

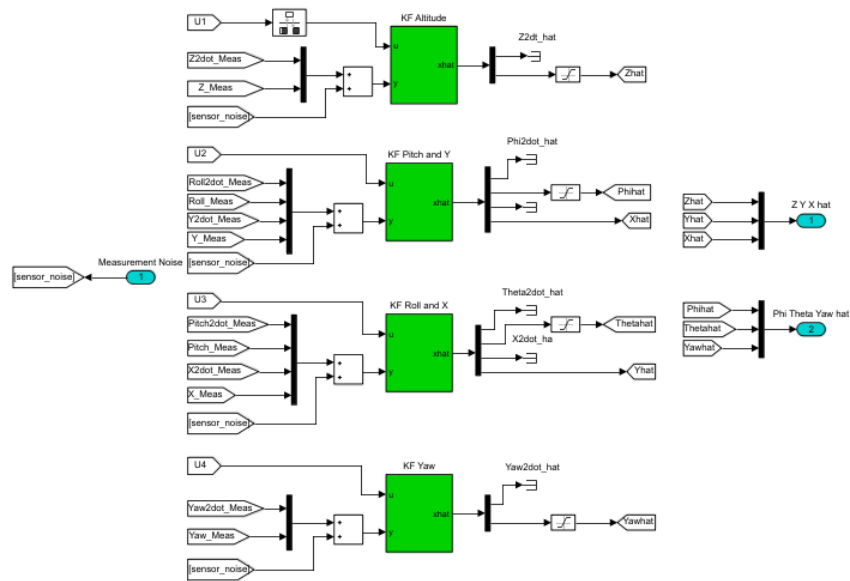


Figure 5-24 - Measurement noise in KF

5.8.2 PWM output

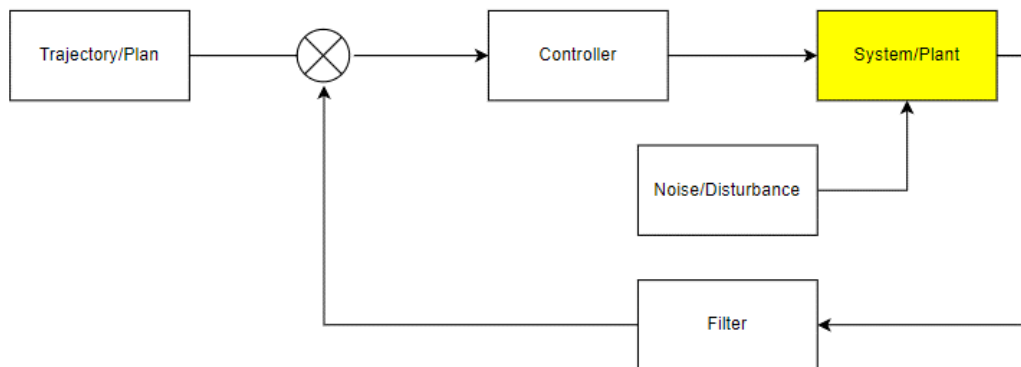


Figure 5-25 - Block diagram representing the architecture of the system - PWM O/P

Figure 5-25 above shows where the PWM block is generated within the system. The block set below generates PWM values as an unsigned 16-bit integer by scaling the calculated voltage from the model and sending it remotely across to the microcontroller or running onboard. The scaling is achieved from the linear equation derived using empirical measurement in Chapter 4 - Motor Torque – Load cell can be seen in Figure 4-27.

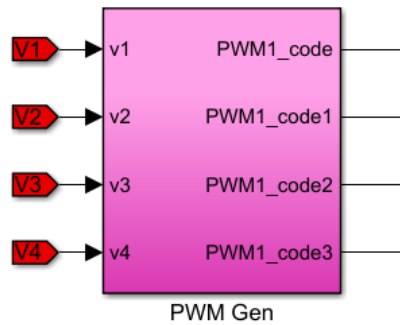


Figure 5-26 - PWM block set

5.8.3 Throttle (Vertical Motion)

Figure 5-27 represents the altitude response of the system using the PID controller and KF filter. The throttle setpoint was set to 5 meters and the response is measured up to 600 seconds. As it can be seen, there is an overshoot of 0.3 meters which can be future reduced by tuning the PID controller. Reducing the transient response will also help with tuning a robust model.

The KF matrix is considered a time-variant system with an initial value of the state variable estimated to be zero. The result of KF along with the raw Gaussian noise can be seen in Figure 5-27 below.

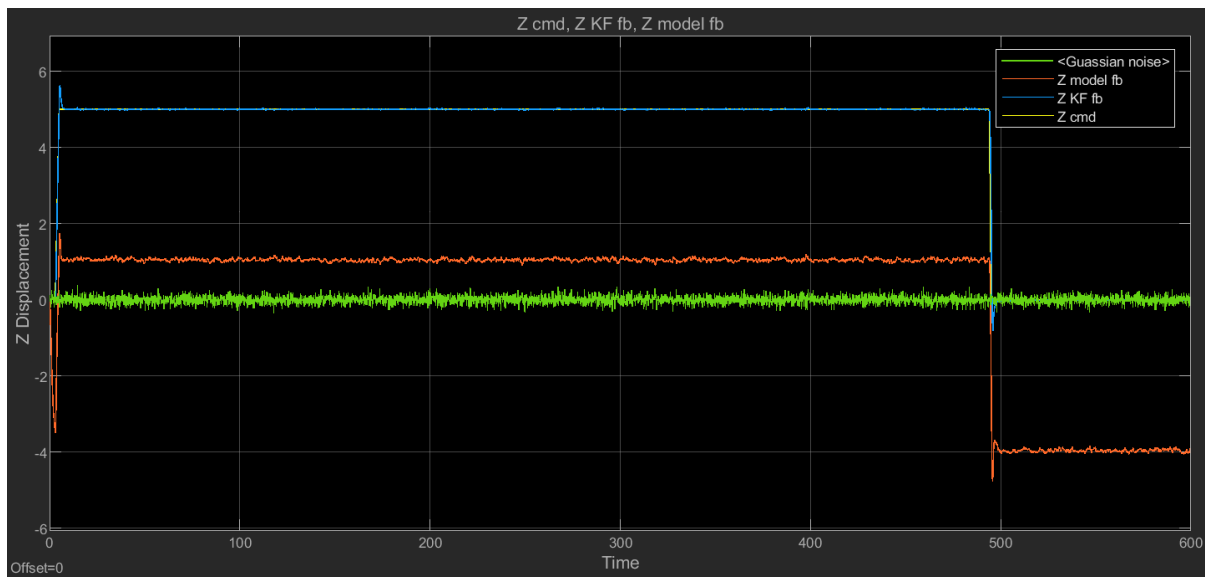


Figure 5-27 - Z displacement with noise

Table 5-1 represents the overall response of the PID controller, along with KF with and without its implementation.

Table 5-1 - Throttle - PID and KF response

Controller	Steady-state (s)	Offset (m)
PID with KF	3	0.5
PID without KF	5	-4

The by using Q and R matrices (Figure 5-28) the KF reduces the noise error and estimates the attitude correctly and proving its ability to operate as an optimal observer.

Noise Characteristics

☐ Use G and H matrices (default G=I and H=0)

Q: ☒ Time-invariant Q

R: ☒ Time-invariant R

N: ☒ Time-invariant N

OK Cancel Help Apply

Figure 5-28 - Attitude - KF Q and R matrix

5.8.4 Roll and Pitch

The commands given to test roll and pitch are shown in Figure 5-29, which simulates the hover scenario at 5m whilst roll and pitch commands are executed in sequence. The attitude (roll and pitch) stability is performed using the PID controller.

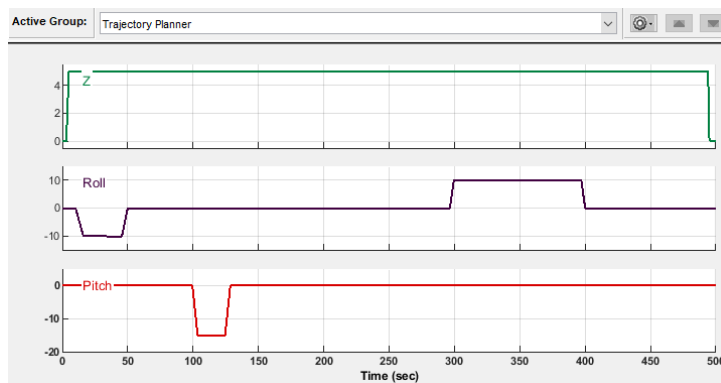


Figure 5-29 - Roll and pitch command

The results of attitude control can be seen in Figure 5-30. The roll was set to 10 and -10 degrees at various time intervals and the pitch angle was set to -15 when the roll is back to zero, mimicking a real-life system command. As it can be noticed in Figure 5-30 the simulation follows roll and pitch commands as expected with a slight overshoot.

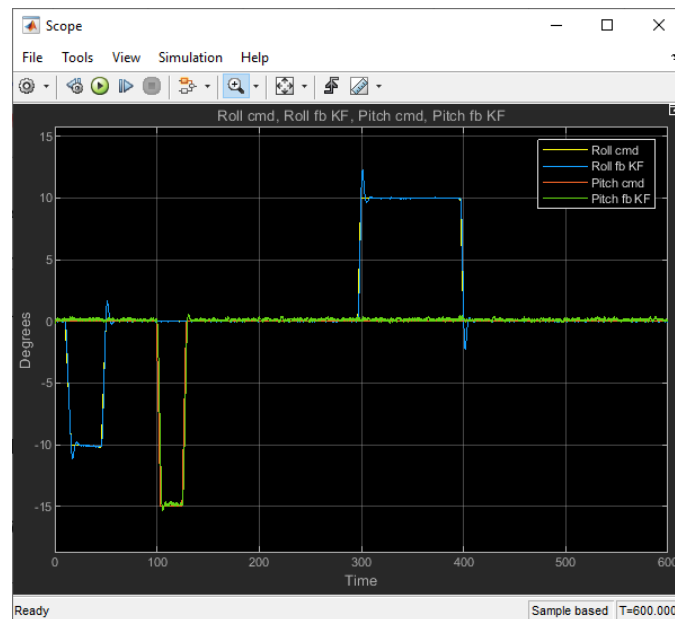


Figure 5-30 - Roll and Pitch KF

The optimal Q and R matrices for pitch and roll are shown in Figure 5-31 and the performance with and without the KF is represented in Table 5-2.

Table 5-2 - Pitch and Roll PID response

Controller	Manoeuvre	Steady-state (s)	Offset (deg)
PID with KF	Roll	3	2
	Pitch	4	1
PID without KF	Roll	4	3
	Pitch	6	3

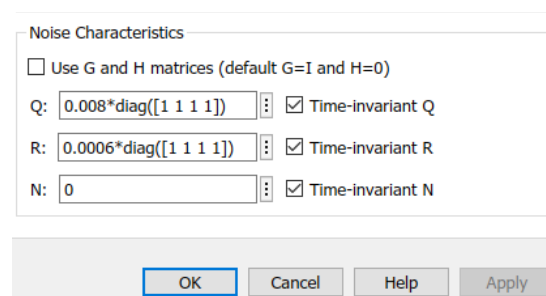


Figure 5-31 - Roll and Pitch - Q and R matrices

5.8.5 Yaw

In this scenario, the quadcopter hovers at 5m and the heading of the quadcopter is rotated along Z-axis. The controller used for the yaw manoeuvre is a simple PD controller.

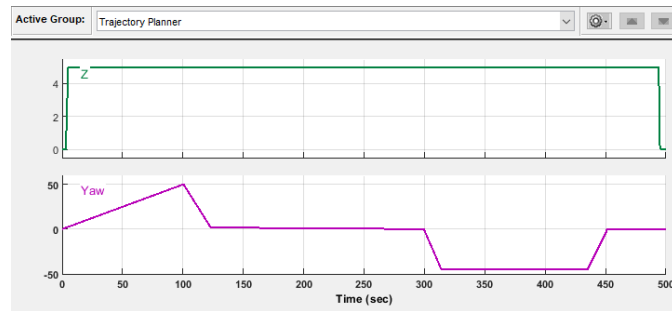


Figure 5-32 - Yaw command

Figure 5-33 shows the responses at different commands, where the induced noise is assumed identical to the pitch and roll scenarios mentioned above.

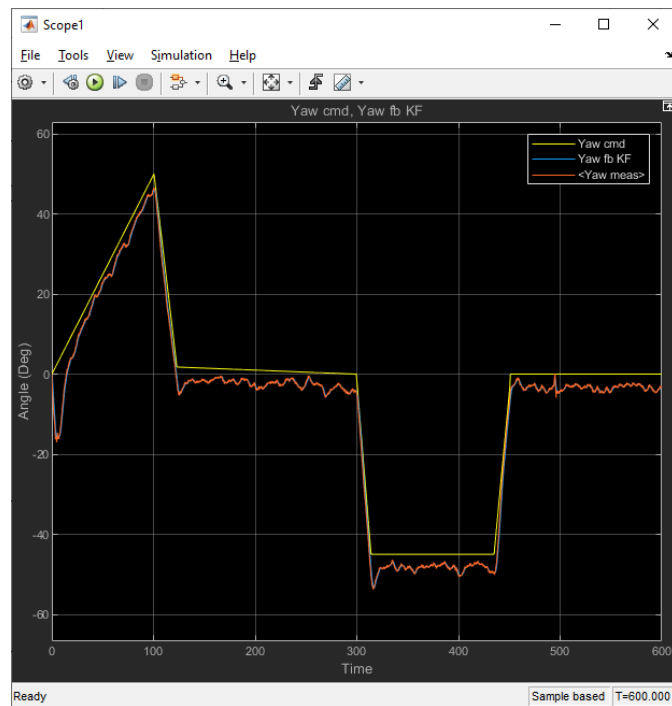


Figure 5-33 - Yaw KF response

Table 5-3 below shows responses based on Q and R matrices of KF.

Table 5-3 - Yaw PID KF response

Controller	Steady-state (s)	Offset (m)
PID with KF	10	0.3
PID without KF	14	0.9

The effectiveness of the KF filter and PID controller can be seen clearly in the above examples. There is an offset bias in the yaw manoeuvre, however, it is insignificant. It is very evident that KF is working as an optimal observer, estimating the new altitude value correctly by decreasing the noise error using Q and R diagonal matrices with the following tuned coefficient (Figure 5-34).

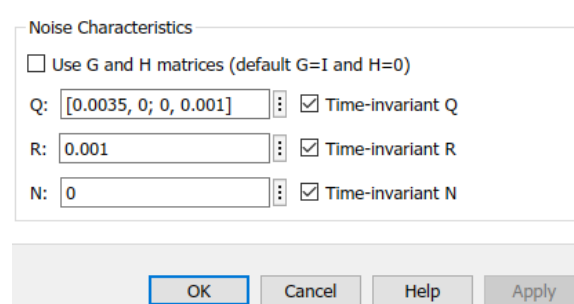


Figure 5-34 - Yaw Q and R matrices

Figure 5-35 represents a complete simulation model with KF.

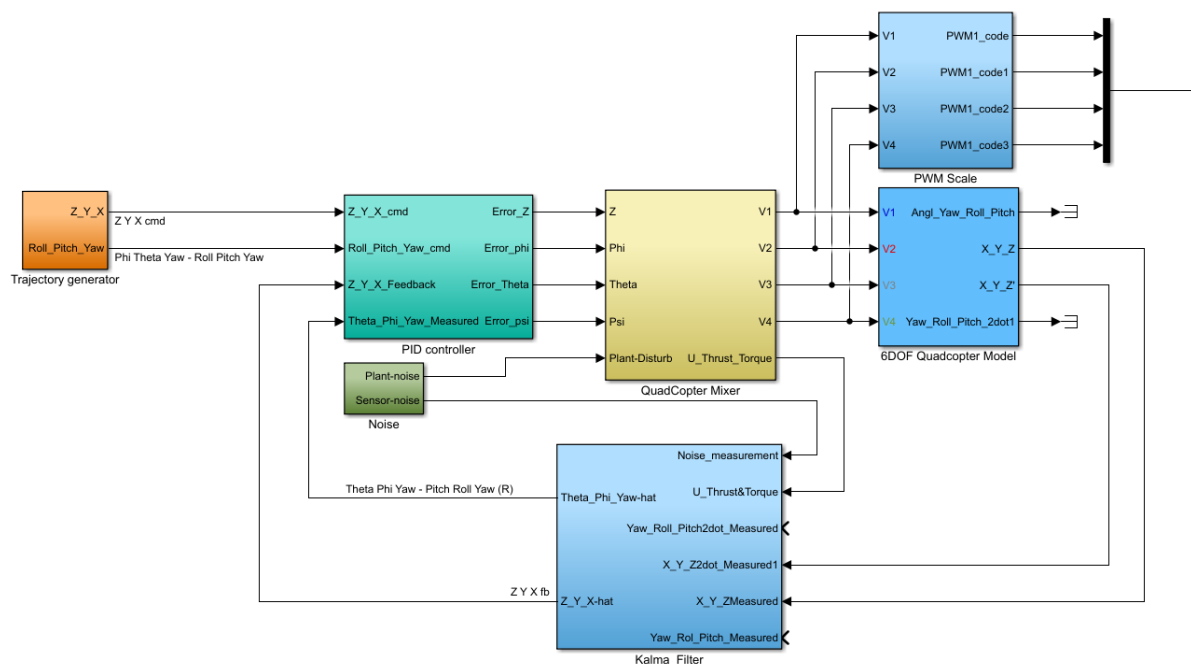


Figure 5-35 - Complete simulation model

5.9 Summary

The chapter describes the mathematical models of the quadcopter. These mathematical equations were derived from previous chapters. Once the quadcopter's mathematical model, the motor mixer subsystem, and controllers were combined, a few basic tests were carried out

to ensure the system behaves as expected. The chapter also highlights results with and without having a Kalman filter in the model.

The controller being used is a PID controller. Kalman filter act as a noise filter, Gaussian noise is injected into the system to fine-tune the Kalman filter. As it can be seen from the results without KF the steady-state reaches 14 seconds with an offset of 0.9m.

Chapter 6 – 3D trajectory generation and Computer-Aided Design Modelling of Quadcopter

This chapter describes the 3D trajectory generation procedure for a quadrotor. It accompanies the MATLAB code for generating the 3D trajectory of a quadrotor given its Simulink dynamical model. The 3D trajectory tracking will enable to visualisation of the performance of a quadrotor using any controllers. First, the quadrotor dynamics and controller are briefly described. Then the process of setting up the Simulink file containing the quadrotor open/closed-loop controller is detailed. After that, the MATLAB code to create the quadrotor trajectory is described in detail. The chapter also focuses on implementing various types of CAD models in MATLAB to visualise the trajectory of the quadcopter. The open-source software Windows 3D builder was used to develop and test CAD models in MATLAB. It shows how to manipulate a replica of the Parrot Mini drone's model from Window's 3D builder and several approaches taken to achieve the results. Any CAD model of formats given in this chapter can be manipulated in MATLAB for visualisation.

6.1 Quadrotor Dynamics

The quadrotor dynamics are described by a 6-degree of freedom model (three translations and three rotational) as mentioned in ('1-0-1-k-j-astrom-pid-controllers-theory-design-and-tuning-2ed.pdf', no date). Rotor speeds are the only four independent inputs on a quadrotor helicopter.

A quadrotor consists of four motors, which generate lift and propulsion. In the current study, DC motors constitute the rotors, which convert the electrical energy (from batteries) into rotational movement. The DC motor speed at each instant and the voltage required are calculated by the controller.

6.2 Quadrotor Controller

The quadrotor controller takes as input the reference trajectory at each instant, and the measured output (the altitude and Euler angles). The primary objective of the controller is reference tracking, i.e. the manoeuvre specified as the reference must be exactly followed by the quadrotor. At this stage, external disturbances and noises are not accounted for. The reference input, the vertical, pitch, yaw and roll are decoupled, so separate PID controller loops are included along each of these channels.

6.3 Simulink Model

To access and manipulate the outputs of the Simulink model to create a trajectory, the outputs are to be exported to the MATLAB workspace. This is done via multiple 'To Workspace' blocks as shown in Figure 6-1 below. Once the model is run in Simulink, these blocks will place the following values in the MATLAB workspace: displacement, angles, motor speeds and controls. Of these, the displacement (X, Y, and Z) and angles (Phi, Theta, and Psi) are required for plotting a 3D trajectory.

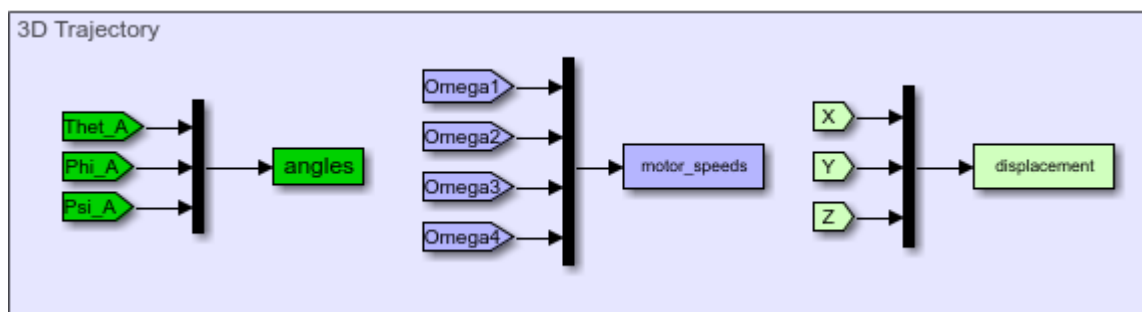


Figure 6-1 - Workspace for 3D visualisation

While including these blocks, the format in which the constituent data is presented may be selected. The default format is a 'time series'. This format is changed to 'array' so that the data

is available in a format suitable for manipulation. This is done by double-clicking the relevant *To Workspace* block and choosing *Array* in the drop-down menu for ‘*Save format*’.

After running the Simulink model, the variables appear in the Workspace in the form of matrices. In this case, the angles and displacement have 493 rows representing each step in the simulation. The 3 columns in the displacement matrix represent the X, Y, and Z positions respectively. While in the angles matrix they represent pitch, roll and yaw respectively.

6.4 Main MATLAB Script

Having set up the Simulink file to return data to the workspace, trajectory plotting can now be implemented. After clearing the workspace, the Simulink model is simulated using the MatLab command *sim ()* (see Figure 6-2 below). This evaluates the Simulink model with extension ‘.slx’ with the model parameters included in the Simulink model, and then places the displacement and angle data in the MatLab workspace.

```
8 -      clc; clear all; close all;
9
10      %% Simulating closed loop system
11 -      sim('Quad_closedloop');
```

Figure 6-2 - *Sim()* command

The following lines of code gathered relevant data from the workspace variables and assign them to X, Y, Z, and pitch, roll, yaw. These are required for plotting the quadrotor trajectory. The ‘*scale_factor*’ setting is for choosing how much larger or smaller the quadrotor 3D model appears on the plot. Currently, this value is at 1:1. For the model to appear 10 times smaller on the plot, the ‘*scale_factor*’ should be 10. Similarly, for the quadrotor model to appear 10 times larger, choose ‘*scale_factor*’ = 0.1.

The ‘*Step*’ represents the step size considered for the quadrotor to traverse on the trajectory. At a step size of 2, the quadrotor appears on the trajectory at every other time step. The ‘*viewang*’ is the viewing angle, taken as the input to MATLAB’s *view ()* command.

Finally, these values are input into the function *AnimationQuad.m*’ to plot the 3D trajectory as in Figure 6-3 below.

```

30 % Plot quadrotor trajectory from displacement and angle values
31 - M = AnimationQuad(x, y, z, pitch, roll, yaw, scale_factor, step, viewang);
32 % t;

```

Figure 6-3 - AnimationQuad function

6.5 The AnimationQuad function

This function shown in Figure 6-4 generates the quadrotor trajectory accounting for the displacement and orientation at the time step specified. The first few lines initialize the function by checking for the correctness of the number of input values, and including default values if some of the inputs are not specified.

```

1  function [M]=AnimationQuad(x,y,z,pitch,roll,yaw,scale_factor,step,varargin)
2
3  %% Function pre-processing
4  - if step<1
5  -     step=1;
6  - end
7
8  - if nargin==9
9  -     theView=cell2mat(varargin(1));
10 - end
11
12 - if nargin<9
13 -     theView=[82.50 2];
14 - end
15
16
17 - mov=nargout;
18
19 - cur_dir=pwd;
20
21 - tic;

```

Figure 6-4: I/P parameter to AnimationQuad function – displacement, angles & variable arg

6.6 Quadcopter 3D modelling

Computer-aided design, CAD, is a digital representation of an object or collection of things. The aerospace industry and all the other types of engineers will benefit from 3-dimensional models as they assist in designing essential components. 3D drawings enable engineers to visualise it easily and hence, improve the quality of design. Although 2D drawings can provide basic information that requires minimal effort if amendments are needed, these drawings will struggle to capture deeper design complexities, including assembly and fit. Additionally, alongside 2D drawings, there is often a need for physical prototypes to depict the functionality of a product design. 3D drawings will decrease the time taken to analyse and build a design concept. The manufacturing process of a product requires a lot of clarity, and 3D drawings help

engineers to precisely analyse components of a design in more depth (Shaqura and Shamma, 2017).

The demand for developers to prospect new ideas that allow for better communication with consumers can be achieved through the incorporation of 3D models and visualisations. 3D models in engineering allow for the presentation of results, new advances and insights on potential trends within the design and engineering fields (*The Importance of 3D Visualisation In The Aerospace Industry*, 2017).

Visually, a quadrotor may be plotted in MATLAB using simple lines, and manipulating its directions at each point in time using its dynamics('Quadcopter Dynamics, Simulation, and Control.pdf', no date). A better option, however, is to have a 3D model of the quadrotor, because it enables the viewer to perceive its orientation during flight.

The 3D models are from Windows 3D builder's library and also a simple mono-colour model representing the custom build quadcopter model (*3D Builder user's manual*, no date) was created using Windows 3D Builder.

Two file formats are chosen due to their popularity and availability on CAD software. They are STL and Wavefront OBJ formats.

6.7 Standard Tessellation Language File

The Standard Tessellation Language, STL, file consists of a few backronyms such as Standard Tessellation Language, Standard Triangle Language, and Stereolithography is a file format describing surfaces with logically placed triangular meshes, aka facets. It is a series of tessellated triangles representing a 3-dimensional surface geometry. This format is utilised in cross-platform CAD development software for rapid prototyping, 3d printing etc. Figure 7-8 represents the STL format of Parrot Mambo, and Figure 7-9 custom quadcopter used in this report on Windows 3D builder.

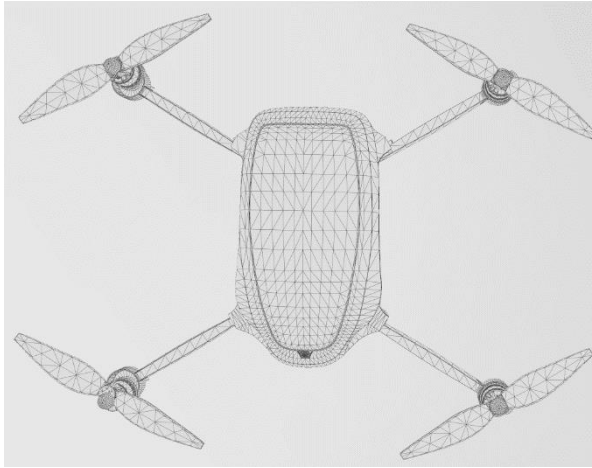


Figure 6-5 - Parrot Mambo STL

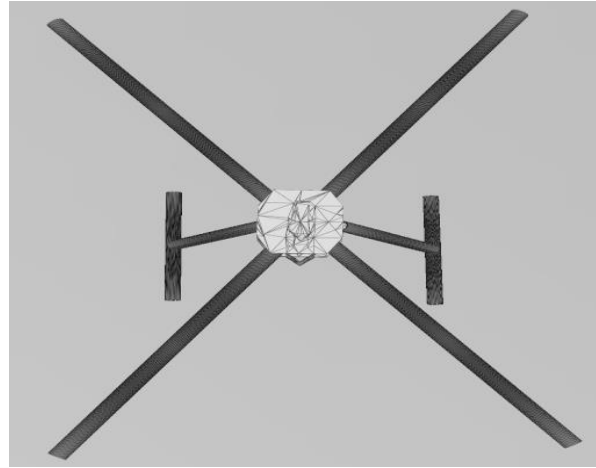


Figure 6-6 - Custom Quad STL

6.7.1 ASCII or Text format

ASCII file format has very minimal use due to its large size. This format is mainly used for testing new CAD interfaces (*STL (STereoLithography) File Format, ASCII, 2019*).

Each facet consists of a unit normal and three vertices of Cartesian coordinates (X, Y and Z). A unit normal is a unit vector with a line perpendicular to the triangle (cross product of two matrices), with a length of 1. The unit normal and each vertex are described by three coordinates, making 12 values for each facet (*The StL Format | fabbers.com, no date*).

Converting a surface into tessellated triangles is elegant in theory but tricky in practice, especially when the geometry contains complex curvatures. There is a set of rules, conventions, and algorithms to follow to manipulate the STL file. It is essential to be consistent with the way facets are oriented to define which face represents the inside and outside of the surface. There are two ways to describe the orientation of the facets to represent which way the facets are facing, i.e. inside and outside of the surface. First, the normal vector should face an outward direction and secondly, when looking at the object from outside, the vertices must be listed in an anti-clockwise manner, as illustrated in Figure 6-7.

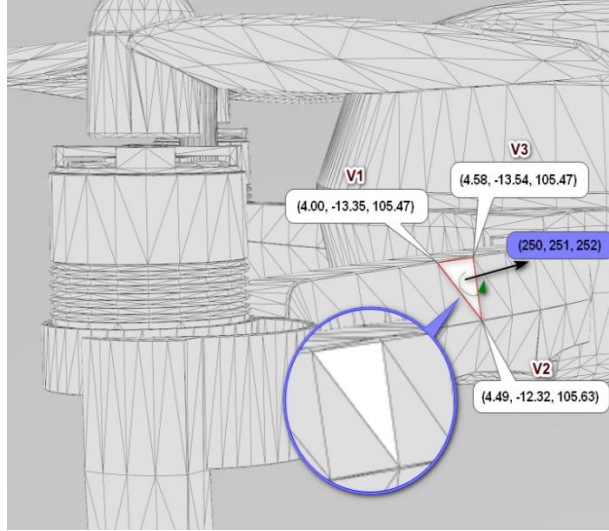


Figure 6-7 - STL ASCII representation

$$\hat{n} = \frac{(\vec{v}_2 - \vec{v}_1) \times (\vec{v}_3 - \vec{v}_1)}{|(\vec{v}_2 - \vec{v}_1) \times (\vec{v}_3 - \vec{v}_1)|} \quad 6.1$$

Equation

6.2 shows the syntax for the ASCII file format, where the name is optional. However, a space must be retained after "solid"; otherwise, the software can misinterpret it as a binary STL file.

$$\begin{array}{l} \text{Solid nme} \\ \left(\begin{array}{l} \text{facet normal } n_x n_y n_z \\ \text{outer loop} \\ \text{vertex } v_{x1} v_{y2} v_{y3} \\ \text{vertex } v_{x2} v_{y2} v_{z3} \\ \text{vertex } v_{x3} v_{y3} v_{z3} \\ \text{end loop} \\ \text{end facet} \end{array} \right) \\ \text{endSolid} \end{array} \quad 6.2$$

6.7.2 Binary Format

Compared to the ASCII format, the binary format is smaller in size; therefore, uses less computational power, making it ideal for most computer applications. The binary file format contains 80-bytes of character header followed by an unsigned integer of 32-bit (4-byte) little-endian. The 80-byte character header is generally ignored while reading the file. The 32-bit data indicates the number of triangular facets in the file i.e. $2^{32} = 429, 496, 796$ number of triangular facets can be accommodated. Following that, data consists of triangle vertices representing normal vectors (which way triangles are facing) and coordinates of each vertex.

Each triangle contains 50 bytes of data and carries 12 bytes of 32-bit IEEE floating-point numbers.

In MATLAB terms, these data are single-precision variables stored as 4-byte, except "Attribute byte count". The first three 4-bytes (12-byte) are for the normal vector of a triangle. The next eight 4 bytes (32 bytes) are for vector coordinates X, Y and Z. Finally, 2 bytes are for "Attribute Byte Count", and have unsigned integers (Burns, 1993). An overview is demonstrated in Figure 6-8 below using the block diagram.

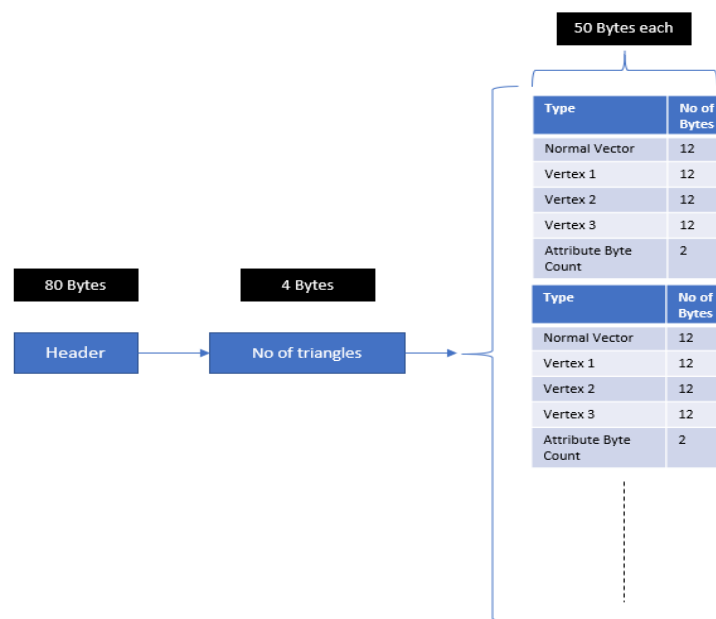


Figure 6-8 - STL Binary format

6.7.3 STL file results

The *stlread()* is provided by Eric Johnson and was last updated on 20th July 2011 (*STL File Reader*, no date). The error encountered with this function was, it returns "Points" and "ConnectivityList" rather than "Vertices", "Faces" and "Normal". Figure 6-9 shows the time taken to call the original function and the output results.

```
>> tic; ParrotSTLMambo = stlread('ParrotMambo.stl'); toc
Elapsed time is 0.150284 seconds.
>> ParrotSTLMambo

ParrotSTLMambo =

    triangulation with properties:

        Points: [49933x3 double]
    ConnectivityList: [99980x3 double]
```

Figure 6-9 - *stlread()* corrupt file

The example code snippet provided by Eric Johnson below shows the function should return data as expected (F , V and N).

```
% STLREAD imports geometry from an STL file into MATLAB.
% FV = STLREAD(FILENAME) imports triangular faces from the ASCII or binary
% STL file indicated by FILENAME, and returns the patch struct FV, with fields
% 'faces' and 'vertices'.
%
% [F,V] = STLREAD(FILENAME) returns the faces F and vertices V separately.
%
% [F,V,N] = STLREAD(FILENAME) also returns the face normal vectors.
%
% The faces and vertices are arranged in the format used by the PATCH plot
% object.
```

The *stlread()* file was renamed to overcome this error.

6.8 Wavefront OBJ file

Wavefront Technologies developed the OBJ file format for their Advanced Visualizer 3D graphics software package. Unlike the STL file, the OBJ file extension is a text file-based format, meaning these files can be opened or edited using a simple text file.

Similar to STL file format, Wavefront OBJ files define the geometry of an object and OBJ files may be stored in binary (*.MOD* extension) or ASCII (*.obj* extension). The commonly used format is ASCII since the binary file format is undocumented and proprietary (*Alias/Wavefront OBJ File Format*, no date).

An ideal obj file format consists of a minimum of three data: vertex, texture and normal coordinates. Firstly, vertex positions "v" (see Figure 7-13) which represent the 3D coordinates of the model. Secondly, the list of items beginning with "vt" are texture coordinates. The "vt" list consists of information on how to map the texture with floating-point values between 0 and 1 (see Figure 6-11). "Vt" is also known as the UV position of texture coordinate. Finally, "f" represents face vertices. The "vn" list is an optional data normal vector missing in the Windows 3D builder-generated obj file. The normal vector carries the same purpose as mentioned in the STL section above.

The most important data is faces with the keyword "f", which combines vertex position coordinates and texture coordinates (and vector normal if available). For each vertex, "vt" and "vn" may be associated, which indicates how to map the texture and normal at a point respectively. An example of a face having **v**, **vt**, and **vn** is "f 1/1/1 2/2/2 3/3/3", where each element consists of respective data.

In this scenario, each face is defined by two numbers: vertex position and texture coordinate index, as shown in Figure 6-13. Since Windows 3D builder does not provide normal vertices for OBJ extension, the order of texture coordinates determines the way polygons face. Note that the face position starts from 1 and not 0 as in most programming languages.

```
1 # Exported from 3D Builder
2 o Object.1
3 v -19.457867 8.555571 105.474213
4 v -19.706694 8.041900 105.474213
5 v -19.346790 8.383405 105.626572
6 v -19.595608 7.869697 105.626572
7 v -19.457867 8.555571 104.789581
8 v -19.706694 8.041900 104.789581
```

Figure 6-10 - Geometric vertices

```
1893
1894 vt 0.234655 0.356029
1895 vt 0.235749 0.357386
1896 vt 0.234443 0.356605
1897 vt 0.235582 0.357990
1898 vt 0.991786 0.738558
1899 vt 0.994245 0.738558
```

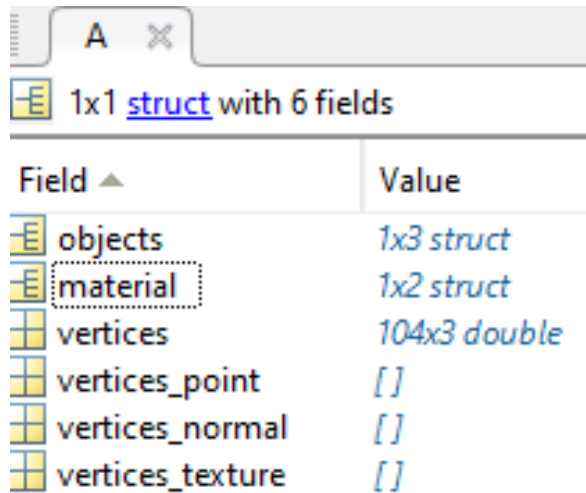
Figure 6-11- Texture vertices

```
3665
3666 f 1/1 2/2 3/3
3667 f 133/175 1/1 3/3
3668 f 3/3 548/593 133/175
3669 f 549/594 133/175 548/593
3670 f 548/593 507/552 549/594
3671 f 518/563 549/594 507/552
```

Figure 6-12 - Face with vertex position & texture coordinate

6.8.1 Wavefront OBJ implementation

The model is exported into MATLAB in the OBJ format to represent the structure, and an MLT file to represent colour, texture, and material. MATLAB does not natively support OBJ or MTL files, so an external toolbox is employed for this purpose. The 3D files in OBJ and MTL format are read into MATLAB by using the **Wavefront OBJ** toolbox (*Wavefront OBJ toolbox - File Exchange - MATLAB Central*, no date). The imported file is saved in ".mat" format as QModel_simp.mat, and contains a MATLAB structure 'A' (see Figure 6-13 below) containing all the vertices, faces, and colours needed to create the quadrotor model.



Field ▲	Value
objects	1x3 struct
material	1x2 struct
vertices	104x3 double
vertices_point	[]
vertices_normal	[]
vertices_texture	[]

Figure 6-13: Object file

This structure is loaded into the MATLAB workspace. The vertices are fetched, with $V(:,1)$, $V(:,2)$, and $V(:,3)$ representing the quadrotors X, Y, and Z points respectively. $V(:,n)$ means select all rows and the n^{th} column only. The mean is subtracted from each vertex, to create a normalized matrix as shown in Figure 6-14 below.

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Load the quadcopter model and manipulate vertex positions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

load QModel_simp.mat;
V = A.vertices;

% Normalise vertex values
V(:,1)=V(:,1)-round(sum(V(:,1))/size(V,1));
V(:,2)=V(:,2)-round(sum(V(:,2))/size(V,1));
V(:,3)=V(:,3)-round(sum(V(:,3))/size(V,1));

```

Figure 6-14: Importing and refining vertices

Then, the quadrotor is scaled according to the required specification (see Figure 6-15 below).

```

33 - correction= max(abs(V(:,1)));
34 - V=V./(scale_factor*correction);

```

Figure 6-15: Correction and Scaling

Now, the displacement and angles are to be considered for plotting. A for loop is used (see Figure 0-4) to iterate through every data point (in this case there are 273 data points). Since

step = 2, every other point is considered for plotting. At the initial point, the platform is plotted. It is considered to be 3m wide, extending 1.5m along X-axis on either side of the origin.

At every iteration, the quadrotor model must be placed at the appropriate position (given by x, y, z) and at the correct orientation (given by the Euler angles – pitch, roll and yaw). This is done by first calculating these angles to radians from degrees and then calculating the rotational matrix ‘R’ as shown in Figure 6-16 below. The quadrotor vertices are rotated by multiplying by R, as well as translated to the correct position.

```

theta=deg2rad(pitch(i));
phi=deg2rad(roll(i));
psi=deg2rad(yaw(i));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Rotation matrix
R=[cos(psi)*cos(theta), -sin(psi)*cos(theta), sin(theta);
   cos(psi)*sin(theta)*sin(phi)+sin(psi)*cos(phi) ...
   -sin(psi)*sin(theta)*sin(phi)+cos(psi)*cos(phi) ...
   -cos(theta)*sin(phi);
   -cos(psi)*sin(theta)*cos(phi)+sin(psi)*sin(phi) ...
   sin(psi)*sin(theta)*cos(phi)+cos(psi)*sin(phi) ...
   cos(theta)*cos(phi)];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

Figure 6-16: Rotational matrix and degrees to radian conversion

Next, the patch () function is used to create the quadrotor shape onto the figure window (see Figure 6-17 below). The vertices are given by Vnew, i.e. the correctly scaled, oriented and translated vertex matrix. The faces represent the interconnection between vertices and are imported from the A structure mentioned above.

```

Vnew=V*R;

rif=[x(i) y(i) z(i)]; %displacement

X0= repmat(rif,size(Vnew,1),1); %bring it to correct size

Vnew=Vnew+X0;

% Patch the new quadrotor vertices on the axes
p = patch('faces',A.objects(3).data.vertices,'vertices',Vnew); hold on;
set(p,'facec',[0.24,0.65,0.24]); hold on;
set(p,'EdgeColor','none');
xlabel('X [m]');
ylabel('Y [m]');
zlabel('Z [m]');

```

Figure 6-17: Patch function

The continuously evolving quadrotor motion is captured in a movie M using MatLab's *getframe* function as shown in Figure 6-18 below.

```

%Capture video
if mov
    lighting gouraud;
    frame = frame + 1;
    M(frame) = getframe;
end

```

Figure 6-18: Movie in M file and Getframe function

At the end of the *for* loop, after the trajectory is traversed by the quadrotor, the displacement is plotted as shown in Figure 6-19 below.

```

%Plot the displacement trajectory
plot3(x,y,z,'--','LineWidth',2);
lighting gouraud;
grid on;
view(theView);

```

Figure 6-19: End - Displacement Trajectory

The function returns the movie M, which can be replayed using the movie (M) command. This concludes the MATLAB codes for the 3D trajectory platform.

6.8.2 Wavefront OBJ results

An issue was encountered when the patch function was executed. The results did not match the example provided by the developer in MATLAB. The Wavefront OBJ toolbox developed by Dirk-Jan Kroon `read_wobj()` is available in MATLAB add-ons. Although the example provided by MATLAB shows the function should return vertices and faces to be manipulated, it was returning corrupt data (see code snippet). The example shows by calling the 3rd structure from `A.Objects`; the variable should return 'Object.1', 'Colour' and 1x1 struct of "faces". In this case, it does not return as expected (as shown - in APPENDIX B).

Field	Value	Fields	data	type
objects	1x3 struct	1	'Object.1'	'o'
material	1x2 struct	2	'Green'	'usmtl'
vertices	104x3 double	3	1x1 struct	'f'
vertices_point	[]	4		
vertices_normal	[]	5		
vertices_texture	[]	6		

Field	Value	Fields	data	type
objects	1x168 struct	1	'Object.1'	'o'
material	[]	2	1x1 struct	'f'
vertices	49933x3 double	3	'Object.2'	'o'
vertices_point	[]	4	1x1 struct	'f'
vertices_normal	[]	5	'Object.3'	'o'
vertices_texture	60405x2 double	6	1x1 struct	'f'
		7	'Object.4'	'o'
		8	'Object.5'	'o'
		9	'Object.6'	'o'
		10	1x1 struct	'f'
		11	'Object.6'	'o'
		12	1x1 struct	'f'

Figure 6-20 - wavefront obj error

```
% Example,
% OBJ=read_wobj('examples\example10.obj');
% FV.vertices=OBJ.vertices;
% FV.faces=OBJ.objects(3).data.vertices;
% figure, patch(FV,'facecolor',[1 0 0]); camlight
```

An alternative to `read_wobj()` is `readobj` by Bernard Abayowa (*readObj - File Exchange - MATLAB Central*, no date). `Readobj()` file is significantly slower than `read_wobj()` (see Figure 7-26). However, to overcome this issue, the CAD file is loaded in a ".mat" format to MATLAB's current folder. In this case, it is saved as " ParrotOBJMambo.mat".

```
Command Window
>> tic;ParrotOBJMambo = readObj('Mambo.obj');toc
Elapsed time is 103.459799 seconds.
>> tic;ParrotOBJMambo = read_wobj('Mambo.obj');toc
Reading Object file : Mambo.obj
    % Exported from 3D Builder
Finished Reading Object file
Elapsed time is 11.876765 seconds.
```

Figure 6-21 - Time taken readObj() and read_wobj functions

After saving as ParrotOBJMambo.mat use the command shown below to access the CAD file's vertices and faces.

```
load (ParrotOBJMambo.mat)
V = ParrotOBJMambo.v;
F = ParrotOBJMambo.f.v;
```

The obtained results are expected, and it is shown in Figure 6-22.

```
>> ParrotOBJMambo

ParrotOBJMambo =

    struct with fields:

        v: [49933×3 double]
        vt: [60405×2 double]
        vn: []
        f: [1×1 struct]
```

Figure 6-22 - Wavefront OBJ result

6.9 3D animations

6.9.1 3D Quadcopter model

The 3D model from the Windows 3D builder used in this research can be seen in Figure 6-23. Similarly, the STL view and OBJ format used for visualisation is in Figure 6-24 and Figure 6-25 below respectively.

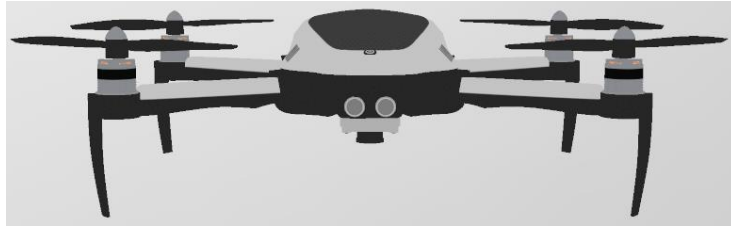


Figure 6-23 - Windows 3D builder model

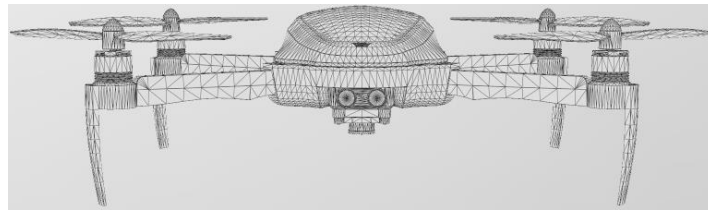


Figure 6-24 - STL view

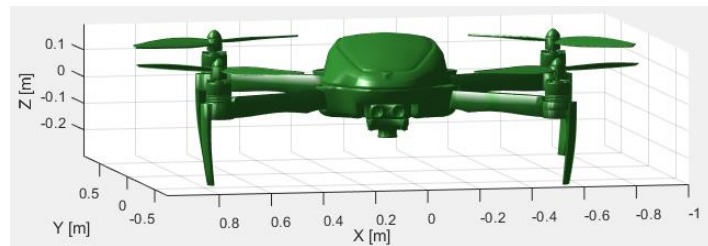


Figure 6-25 - OBJ format

6.9.2 Animation Results

Preliminary tests established the controller stability of the model, and it was observed in the graphical representation in the previous chapter Chapter 5. To visually see the performance of the quadcopter, the following commands were executed. The first command demonstrates throttle/altitude, where the quadcopter raises to 5 meters. The result representing the animation of the CAD file can be seen in Figure 6-26.

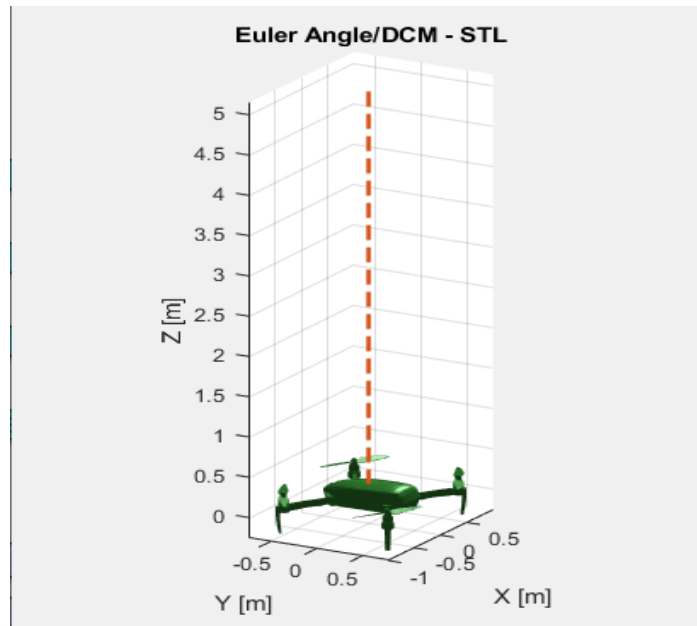


Figure 6-26 - Altitude command

The X and Y movements are demonstrated in Figure 6-27 and Figure 6-28 respectively. The objective is to move in X and Y directions by maintaining an altitude of 4 meters.

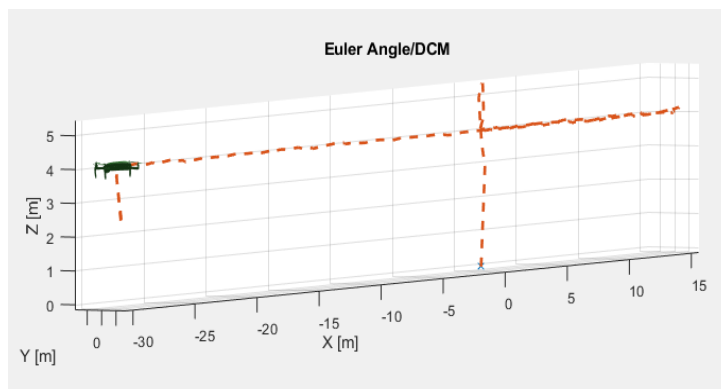


Figure 6-27 - Quad X directional movement

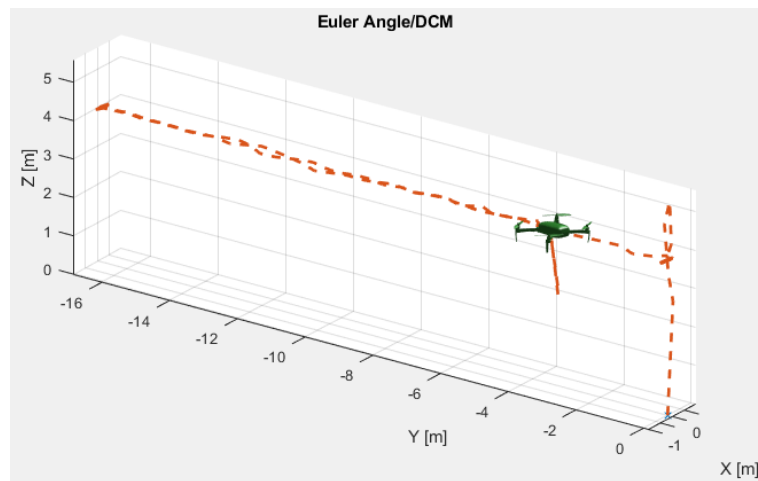


Figure 6-28 - Quad Y directional movement

As it can be seen there is a slight overshoot. In Figure 6-29, shown in graphical representation, the overshoot may not look obvious.

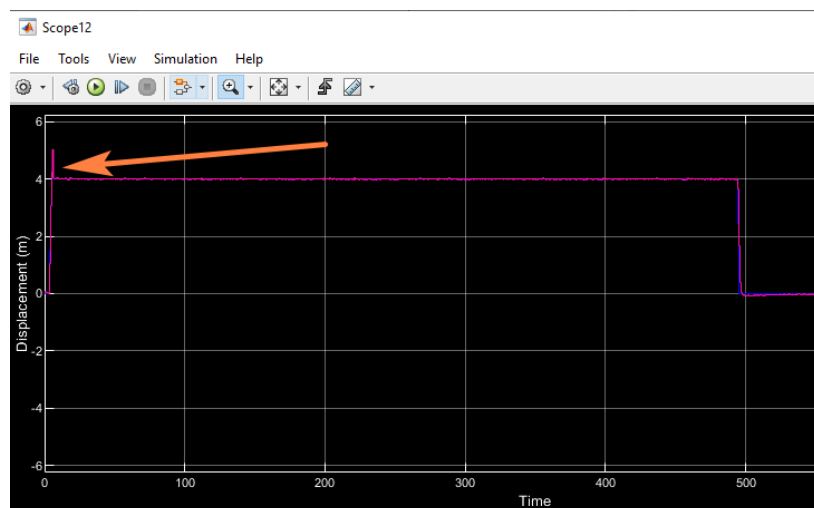


Figure 6-29 - Graphical representation of overshoot

Finally, all the test commands were executed to see the performance of the quadcopter. The signal builder sends an altitude command of 4 meters and performs X, Y and Yaw manoeuvres within 500 seconds (see Figure 6-30).

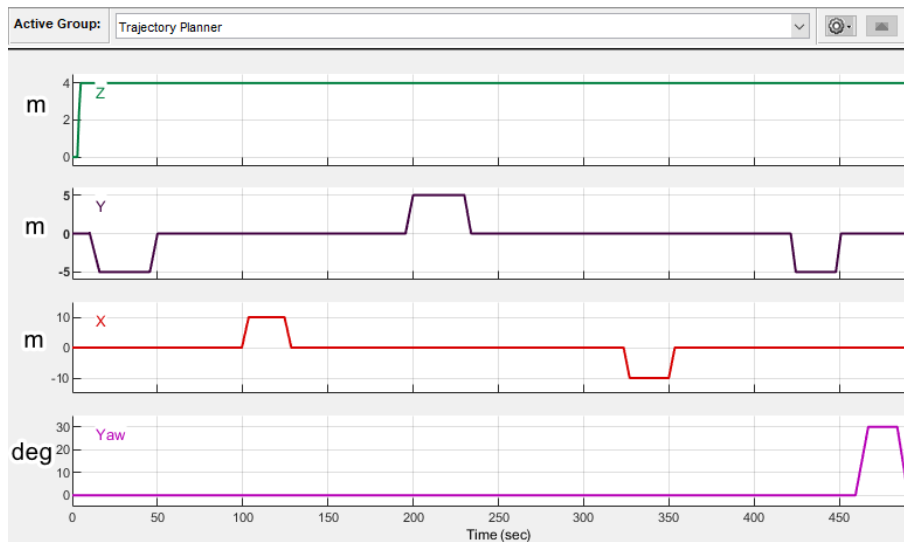


Figure 6-30 - Signal builder commands

The result in Figure 6-31 shows the performance of the quadcopter's trajectory.

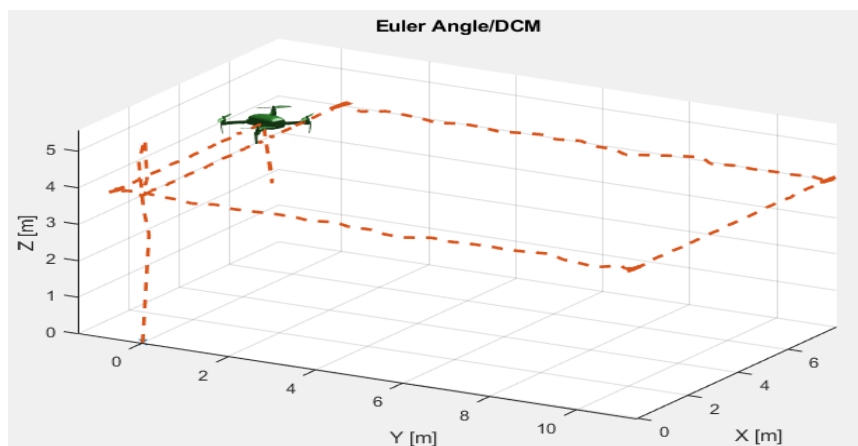


Figure 6-31 - Quad rectangular trajectory using all commands

6.10 Summary

This chapter describes how to manipulate open-source 3D models from Windows 3D builder in MATLAB. This chapter looks at how to troubleshoot some of the add-ons provided by MATLAB and overcome drawbacks. The probable cause for add-ons to fail could be due to a lack of support/update in up-versions of MATLAB.

The chapter compares two popular file formats used in 3D modelling and compares the size; time taken to simulate it on MATLAB. Unfortunately, Simulink doesn't support 3D visualisation using these file formats, so the data are sent to MATLAB workspace and displayed. It demonstrates following a planned trajectory.

Chapter 7 - Sensor Fusion and Wireless Communication Systems

This chapter involves developing a communication system using two devices – Atmega328p and Raspberry Pi 3 B+. The Atmega328p is used to send and receive data to the ground station (MATLAB/Simulink). RPi act as an onboard controller which takes command from the ground station and obeys the command in real-time during flight. The chapter explains the different types of sensors being used, how they are filtered, and data are combined to make accurate judgements. It also goes through some of the difficulties encountered while gathering data and how they are rectified. Figure 7-1 below is the overall architecture of this chapter.

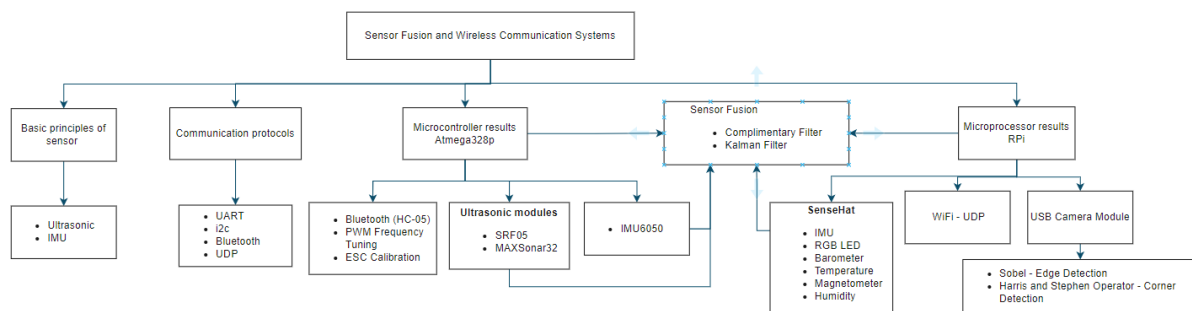


Figure 7-1 - Sensor Fusion and Wireless Communication - Overall architecture

7.1 Sensors and communication systems

7.1.1 The basic principle of sensors

7.1.1.1 *Ultrasonic*

The ultrasonic sensor makes use of the principle of ultrasound echoes. Simply put, the ultrasonic sensor sends out a pulse and waits for the time it takes for the echo to return. It then times the delay, multiplies the time by the speed of sound and calculates the distance from the sensor to the object. The main parts of an ultrasonic sensor are the transmitter and receiver. The transmitter emits ultrasonic sound waves, and the receiver detects the reflected sound waves. The transmitter normally consists of a piezoelectric ceramic material that converts electrical energy into mechanical vibration. The receiver converts the received sound waves into electrical energy. The distance to the object is calculated by measuring the time between the emission of the ultrasonic sound wave and the reception of the reflected signal, as shown in Figure 8-1 below. In an ultrasonic sensor, the timing of the emitted ultrasonic sound wave is controlled by a microcontroller.

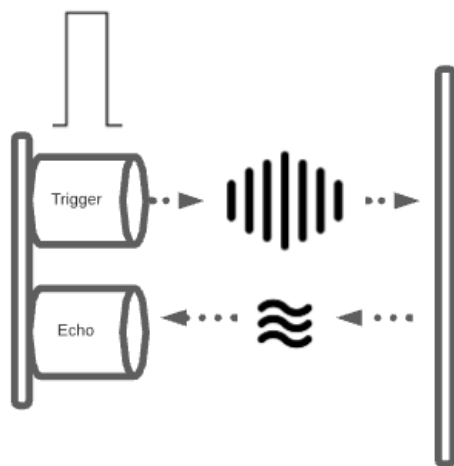


Figure 7-2 - Ultrasonic sensor SRF05

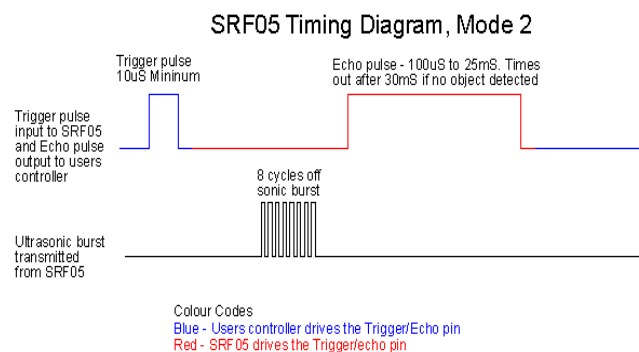


Figure 7-3 - Ultrasonic sensor SRF05 timing diagram

7.1.1.2 *IMU*

The IMU is typically used in navigation and guidance applications and is sometimes included in the heading for an aircraft navigation system, or flight control system. An IMU consists of an inertial measurement unit (IMU) and a processing unit. The IMU consists of three orthogonal rate sensors that measure linear and angular velocities (accelerometers), and three orthogonal rate sensors that measure linear acceleration along the three spatial axes (gyroscopes). The processing unit is usually a microprocessor that processes the raw signals

from the IMU and computes the orientation of the body together with the linear acceleration relative to the body. Accelerometers and gyroscopes are active devices that require a power source. The output of an accelerometer at any given instant represents the force applied at the centre of mass of the sensing element divided by the element's total mass. The output of a gyroscope at any given instant represents the angular velocity of the sensing element about the axis of the sensing element. The output of a gyroscope is typically referred to as the rate of rotation. The IMU is frequently used to estimate the orientation of the body based on accelerometers and gyroscopes. The orientation (attitude) of the body is represented by the three Euler angles (roll, pitch, and yaw).

7.1.2 Communication Systems

7.1.2.1 *UART*

UART is an acronym for Universal Asynchronous Receiver-Transmitter. It is a communication protocol that provides the capability to transmit and receive data via a serial connection. UART is a very basic protocol that is used for the lowest level of communication between devices. The UART protocol is a simple protocol that is used to send and receive data between two devices. The protocol is asynchronous, meaning that it does not require a clock signal to synchronize the sending and receiving of data. The protocol is simplex, half-duplex or full duplex. The UART protocol is also a serial protocol, meaning that it sends data one bit at a time.

7.1.2.2 *I2C*

The inter-integrated circuit (*i²c*) the protocol was developed by Philips Semiconductors in 1982 and was first published in December 1982. It is a multi-master, multi-slave, single-ended, serial, synchronous, half-duplex, bit-oriented protocol.

The I2C protocol specifies a communication model in which multiple devices may be connected to a single bus. Each device may be considered a "master" or a "slave". A master device may initiate a data transfer to a slave device by issuing a "master-send" command. A slave device may respond to a master-send command by issuing a "slave-send" command. A master device may also issue a "master-receive" command, which a slave device may respond to with a "slave-receive" command.

The I2C protocol specifies that a device may act as a master or a slave, but not at the same time. A device that acts as a master device may not act as a slave device, and a device that acts as a slave device may not act as a master device. The I2C protocol specifies that a device may

act as a "transmitter" or a "receiver" but not both at the same time i.e. read/write. A device that acts as a transmitter may not act as a receiver, and a device that acts as a receiver may not act as a transmitter.

7.1.2.3 *Bluetooth (HC-05)*

HC-05 module is an easy-to-use Bluetooth module. It supports both the Bluetooth HID and GATT profiles. It uses the UART to communicate with the host controller. The HC-05 does not have any onboard memory. The HC-05 has two modes of operation, Data mode and AT mode. The AT command mode is to change settings and the Data mode is for sending and receiving data. The module also has a built-in power management circuit that will turn itself off after 10 seconds of inactivity. This module can be powered by 3.3V or 5V. It also has an onboard 3.3V regulator.

7.1.2.4 *UDP*

UDP is a connectionless protocol, which means that it does not establish a dedicated end-to-end connection with the recipient of a message, but rather transports the message in the form of individual packets. UDP has a very simple header, which is short enough to be written on an application. The protocol does not provide any kind of error correction and relies on the application to detect any errors that might occur and request the retransmission of lost packets. As a result, UDP is typically used with (or over) TCP for critical applications.

UDP is a transport layer protocol, a transport protocol that provides a service at an intermediate level between an application and the network interface layer, containing the internet protocol. UDP is used extensively in multicast applications, such as video streaming. When sent over the internet protocol, UDP packets are sent to an IP address, but they do not guarantee reliable delivery. UDP is not a connection-oriented protocol. It does not establish a logical end-to-end connection with the transport layer at the other end of the transmission (called the "receiving application"), but instead sends a stream of individual "datagrams", or packets, to the receiving application.

A datagram is not delivered if it is not received at the destination. The UDP sender does not need to know if every packet reached the destination. The application that sends the UDP packet does not need to know if the packet was lost. The receiver will request the retransmission of lost data to the sender.

One of the main advantages of UDP is that it is stateless. The sender and receiver do not establish a connection. The receiver sends no acknowledgement, or an acknowledgement

indicating that it has successfully received the data. Therefore, there is no reason to create a state at the sender.

7.2 Atmega328p setup procedure

The objective of the Atmega328P microcontroller was to initially act as a device to send and receive data such as PWM, Ultrasonic measurements and IMU data. MATLAB/Simulink work as a ground station and runs model analysis in real time while taking away computationally demanding tasks from the microcontroller. Initially, sensors were tested on the microcontroller and verified, then migrated to MATLAB/Simulink wherever applicable.

7.2.1 Bluetooth (HC-05)

To pair two Bluetooth modules, the user must enter in AT command mode. A USB to FTDI adapter can be used to enter AT command mode by following the simple connection mentioned in Table 8-1.

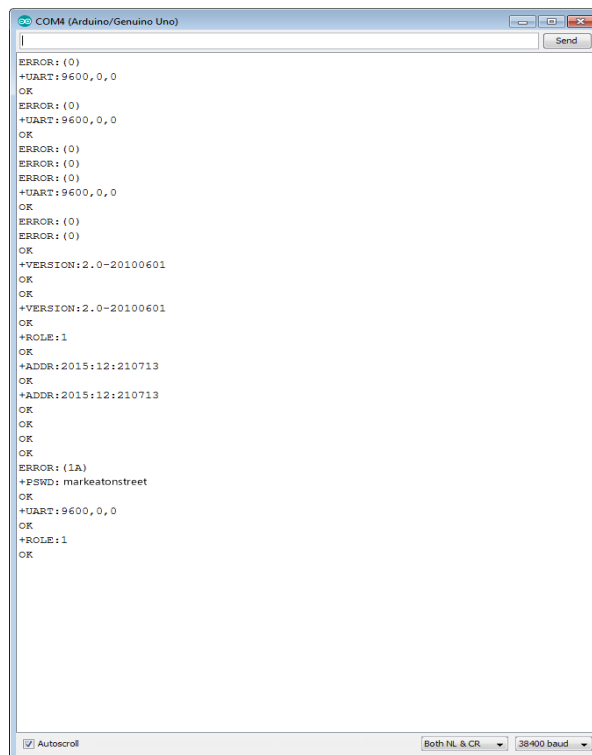
Table 7-1 - FTDI - HC05 Connection

EN/Wakeup/Key	3.3V (Press reset button if available)
Vcc	5v
Gnd	Gnd
Tx	Rx
Rx	Tx

The onboard LED light on HC-05 will flash every two seconds indicating the user has entered AT command mode successfully. The default baud rate for AT command is 9600, however, to enter AT command mode, the serial terminal window must be set at 38400 baud (sandeep jagdevFollow, no date). Some of the AT commands used to pair the modules are shown in Table 8-2.

Table 7-2 - AT commands

Commands	Parameter	Description
AT+ORGL		Resets to the original setting
AT+RMAAD		Resets to factory setting
AT+BAUD	115200	Sets baud rate
AT+ADDR?		Return address of the module
AT+ROLE	1	1 - Master, 0 – Slave
AT+CMODE	0	0 – Binds only with linked 'ADDR'
AT+LINK	ADDR	Connects the address to be paired
AT+BIND	ADDR	Set/Check fixed address



```

COM4 (Arduino/Genuino Uno)
|
ERROR: (0)
+UART:9600,0,0
OK
ERROR: (0)
+UART:9600,0,0
OK
ERROR: (0)
ERROR: (0)
ERROR: (0)
+UART:9600,0,0
OK
ERROR: (0)
ERROR: (0)
OK
+VERSION:2.0-20100601
OK
+VERSION:2.0-20100601
OK
+ROLE:1
OK
+ADDR:2015:12:210713
OK
+ADDR:2015:12:210713
OK
OK
OK
ERROR: (1A)
+PSWD: markeatonstreet
OK
+UART:9600,0,0
OK
+ROLE:1
OK
  
```

Figure 7-4 – Atmega328P BT connection result

7.2.2 ESC PWM frequency control

While carrying out an initial motor test in the test rig, it was noticed that ESCs were not entering the calibration mode, and the motors were not running in unison. After diagnosing the issue was realised that a fixed frequency had to be set on each PWM pin to enter ESC calibration mode. The default PWM frequencies are observed as shown in Figure 8-4.

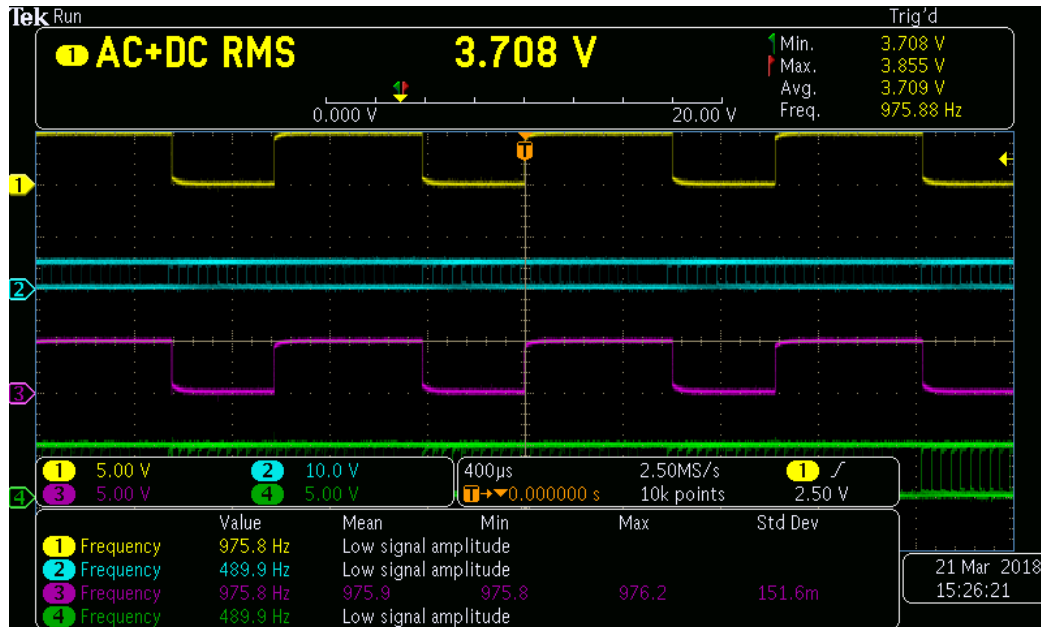


Figure 7-5 - Difference in PWM frequencies

There are two methods to generate PWM. Hardware and Software PWM methods. The hardware uses the dedicated pins provided by the microcontroller. Whereas the software PWM requires programming individual pins to convert those pins to generate PWM. In ATMEGA328P, 3 Timers generate Hardware PWM. Timer0, Timer1, and Timer2 correspond to output pins 5, 6, 9, 10, 3 & 11 respectively on the Arduino Uno board (refer to Table 8-3).

Table 7-3 - Timer Default Settings

Default	Timers - Counters		
	Timer 0	Timer 1	Timer 2
Timer (bits)	8	16	8
Frequency (Hz)	980	490	490
Resolution (Max)	255	65355	255

Default	Timers - Counters		
	Timer 0	Timer 1	Timer 2
Pin Numbers	5 & 6	9 & 10	3 & 11

The reason for not manipulating the frequency in Timer 0 is that any alteration to the pre-scaler value will disrupt the normal operation of functions such as delay(), millis(), micro() etc.

A. Mode of Operations

The ATMEGA328P can generate frequencies for PWM functionality up to 8 MHz. In order to customise the frequency, it is required to modify timers that contain registers. There are several modes of operation to choose from, and some of them are - Normal Mode, Fast PWM, Clear Timer on Compare Match (CTC) Mode, Phase Correct PWM and Frequency and Phase Correct PWM.

1. Normal Mode: Normal mode runs with default frequencies listed in Table 8-3.
2. Fast PWM Mode: This mode counts from Zero to Max (based on the resolution mentioned in TABLE II); after that, the timer count overflows to Zero. This mode produces a sawtooth waveform. The advantage of Fast PWM Mode is it can generate higher frequencies. The equation for calculating possible frequencies is given in 8.1. Where N is the prescale value.

$$f = \frac{f_{clk}}{N \times 256} \quad 7.1$$

3. CTC Mode: CTC is a widely used mode when implementing Hardware PWM. In CTC mode, the counter counts up in a sawtooth waveform. The counter can be cleared to Zero when the counter value (TCNTx) matches OCRxA (where x represents the timer number). The OCRxA defines the top value of the counter, hence also its resolution ('Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf', no date, p. 74). The waveform frequency equation is defined as shown in the equation.

$$f_{ocnx} = \frac{f_{clk}}{2 \times N \times (1 \times OCRnx)} \quad 7.2$$

4. Phase Correct PWM: In this mode, the counter counts in a triangle waveform. The timer counts from Zero to Max and does not overflow, like in the sawtooth waveform, and counts back from Max to Zero.

$$f = \frac{f_{clk}}{(N \times 510)} \quad 7.3$$

B. Configuring Timer/Counter Register

To configure Timer registers, the desired I/O Registers must be set to their associated timer. There are three I/O registers. Data Direction Register (DDRx), Pin Register (PINx) and Port Register (PORTx). Where x represents GPIO (General I/O) port A, B, C etc.

Figure 8-5 shows the bit structures of DDRx, PINx and PORTx. They are 8-bit registers.



Figure 7-6 - I/O Registers

DDRx pin saves configuration information for the pins of PORTx. Writing '0' make the corresponding pin an INPUT, and writing '1' makes the corresponding pin an OUTPUT. PORTx stores the logic value, which is being outputted on the physical pins of PORTx. PINx stores the current state, and the logic value of the physical pins of PORTx (Mayank, 2011).

Now that port operation registers and their I/O are initialised, it is required to manipulate Timer/Counter Control registers to obtain desired frequency. Mainly these registers are TCCRxA, TCCRxB, OCRxA and OCRxB.

1. TCCRxA: Timer/Counter Control Register A. The WGM11 (Waveform Generation Mode) and WMG10 bits (shown in Figure 8-6) are two of the four bits (only in Timer1) required to select the mode of operation. COM1A [1:0] and COM1B [1:0] must be set when using CTC mode; else, these bits have to be cleared.

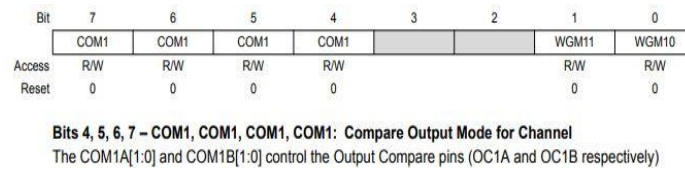


Figure 7-7 - TC1 Control Register A

2. TCCRxB: Timer/Counter Control Register B. In this register, the other two bits of WGM13 and WGM12 (as shown in Figure 8-7) must be set appropriately to choose the correct mode of operation. The three clocks select bits are chosen to select the clock source and pre-scaler values (CS [2:0]).

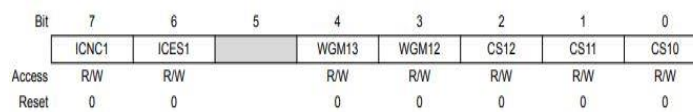


Figure 7-8 - WGM and Pre-scaler

3. OCRxA and OCRxB: These registers adjust the duty cycle to the desired PWM pins, especially in Phase Correct PWM mode.

C. Phase Correct PWM

This paper shows the setup procedures of Phase Correct PWM based on the guidance above. The reason for choosing Phase Correct PWM is because this mode has an additional pre-scaler value (32) in Timer 2 as shown in Table 8-4. Also, Timer 1 has a 16-bit timer which gives more resolution resulting in precise frequency selection.

Table 7-4 - Pre-Scaler and Timers

Pre-scaler values(N)	Timer 0	Timer 1	Timer 2
No Clock Source			
Clk _{I/O} /1			
Clk _{I/O} /8			
Clk _{I/O} /32	x	x	
Clk _{I/O} /64			
Clk _{I/O} /256			
Clk _{I/O} /1024			

It was established to set the operating frequency at 500 Hz ~ after conducting a series of thrust measurement tests. The thrust measurement was carried out on three different ESCs (Electronic Speed Controllers) and various frequencies (50 – 1 kHz). Out of all the different frequencies tested, the most linear result obtained was at 490.83 Hz. The list of experimental measurements taken is shown in Figure 8-8 below, which determines aerodynamic characteristics and motor parameters.

	A	B	C	D	E	F	G	H	I	J	K	L
1	A	Vrms	PWM	B	RPM	Omega Rad/s	N	Omega^2	I/Omg	V/Omg	Omg/I	V/I
3	0.3	4.3	4.183	-67.9	1050	109.9557429	-0.666099	12090.26539	0.00273	0.039106643	366.5191429	14.33333333
4	0.6	5.04	4.252	-36.6	1404	147.0265362	-0.359046	21616.80234	0.00408	0.034279526	245.044227	8.4
5	0.9	5.68	4.301	-4.7	1816	190.1710753	-0.046107	36165.03788	0.00473	0.029867844	211.3011948	6.311111111
6	1	5.68	4.325	-0.9	1799	188.3908395	-0.008829	35491.10839	0.00531	0.030150086	188.3908395	5.68
7	1.1	5.83	4.355	22	2027	212.2669436	0.21582	45057.25536	0.00518	0.027465416	192.9699488	5.3
8	1.4	6.26	4.385	50.2	2311	242.0073541	0.492462	58567.55943	0.00578	0.025866983	172.8623958	4.471428571
9	1.6	6.46	4.426	64.8	2435	254.9926037	0.635688	65021.22795	0.00627	0.025334068	159.3703773	4.0375
10	1.8	6.72	4.449	94.8	2676	280.2300647	0.929988	78528.88916	0.00642	0.023980296	155.6833693	3.733333333
11	2.1	6.94	4.482	118.8	2875	301.069296	1.165428	90642.72098	0.00698	0.023051172	143.3663314	3.304761905
12	2.3	7.14	4.516	142.3	3028	317.0914185	1.395963	100546.9677	0.00725	0.022517166	137.8658341	3.104347826
13	2.5	7.32	4.555	160.8	3173	332.275783	1.577448	110407.196	0.00752	0.022029893	132.9103132	2.928
14	2.8	7.62	4.586	205.8	3501	366.6238627	2.018898	134413.0567	0.00764	0.020784244	130.9370938	2.721428571
15	3	7.72	4.622	222.6	3560	372.8023282	2.183706	138981.5759	0.00805	0.020708025	124.2674427	2.573333333
16	3.4	7.96	4.65	256.4	3151	329.9719484	2.515284	108881.4867	0.0103	0.024123263	97.05057305	2.341176471
17	3.8	8.31	4.675	313.3	4023	421.2875748	3.073473	177483.2207	0.00902	0.019725244	110.8651513	2.186842105
18	3.9	8.4	4.701	321.6	4104	429.769875	3.154896	184702.1455	0.00907	0.019545344	110.1974038	2.153846154
19	4.3	8.73	4.739	356.5	4350	455.5309348	3.497265	207508.4325	0.00944	0.01916445	105.9374267	2.030232558
20	4.7	9.01	4.77	393.9	5416	567.1621937	3.864159	321672.954	0.00829	0.015886108	120.6728072	1.917021277
21	4.9	9.13	4.808	415.2	4621	483.9099884	4.073112	234168.8769	0.01013	0.018867145	98.75714049	1.863265306
22	5.4	9.42	4.819	474.1	4901	513.2315198	4.650921	263406.593	0.01052	0.01835429	95.04287404	1.744444444
23	5.7	9.53	4.855	492	4981	521.6091003	4.82652	272076.0535	0.01093	0.018270387	91.51036847	1.671929825
24	6	9.52	4.884	535.2	5132	537.4217833	5.250312	288822.1731	0.01116	0.017714206	89.57029721	1.586666667
25	6.2	9.51	4.92	549.4	5238	548.5220773	5.389614	300876.4693	0.0113	0.017337497	88.47130279	1.533870968

Figure 7-9 - Thrust Measurement

The Phase Correct PWM set-up procedure explained below is on Timer 1 only. These procedures can be implemented on other Timers with slight changes.

In ATMEGA328P 8-bit register called PORTB correspond to digital pins from 8 to 13, in which 9 and 10 are PWM compatible pins as mentioned in TABLE II. The following steps were carried out to set pins 9 and 10 to the desired frequency using Phase Correct PWM mode.

Port Operation Registers

PORTB = 0x00;

PINB = 0x00;

DDRB = (1<<1)|(1<<2);

Timer/Counter Registers

TCCR1A = 0b10100001;

TCCR1B = 0b00000011;

Duty Cycle

OCR1A = (duty cycle as required);

OCR1B = (duty cycle as required);

7.2.3 ESC calibration

The jDrones ESC Model AC20-1.0 being used needs throttle calibration (armed) during power-up. There is a special sequence to be followed which mimics joystick throttle control settings, where 100% throttle must be selected during power-up and gradually brought back to 0%. The throttle calibration can be materialised by following the audio feedback from the ESC. The joystick throttle control setting was replicated in the microcontroller as follows.

- Bring 8bit PWM values in descending order from 255 to 0 (100% - 0% duty cycle) in 5ms intervals.
- Wait 2 seconds
- Gradually increase the PWM every 50ms up to 12% duty cycle
- Finally, the 12% duty cycle back to 0%.

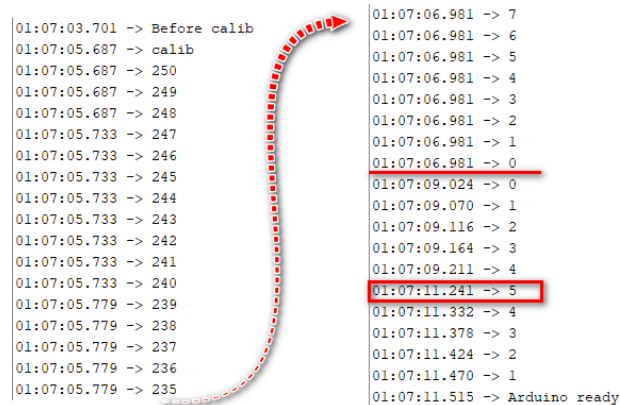
```
1 void calibration()
2 {
3   int writeValue = 250;
4   delay(2000);
5   Serial.println("calib");
6   for (writeValue = 250; writeValue >=0; writeValue--)
7   {
8
9     Serial.println(writeValue);
10    analogWrite (3, writeValue);
11    analogWrite (11, writeValue);
12    analogWrite (9, writeValue);
13    analogWrite (10, writeValue);
14
15    delay(5);
16  }
17  delay(2000);
18  for(writeValue = 0; writeValue < 5; writeValue++)
19  {
20    Serial.println(writeValue);
21    analogWrite (3, writeValue);
22    analogWrite (11, writeValue);
23    analogWrite (9, writeValue);
24    analogWrite (10, writeValue);
25    delay(50);
26  }
27  delay(2000);
28  for(writeValue = 5; writeValue > 0; writeValue--)
29  {
30    Serial.println(writeValue);
31    analogWrite (3, writeValue);
32    analogWrite (11, writeValue);
```

```

33   analogWrite (9, writeValue);
34   analogWrite (10, writeValue);
35   delay(50);
36 }
37 }
38

```

The arming/calibrating process is a one-time setup during powerup, so the function is only executed once in the “Setup” function.



```

01:07:03.701 -> Before calib
01:07:05.687 -> calib
01:07:05.687 -> 250
01:07:05.687 -> 249
01:07:05.687 -> 248
01:07:05.733 -> 247
01:07:05.733 -> 246
01:07:05.733 -> 245
01:07:05.733 -> 244
01:07:05.733 -> 243
01:07:05.733 -> 242
01:07:05.733 -> 241
01:07:05.733 -> 240
01:07:05.779 -> 239
01:07:05.779 -> 238
01:07:05.779 -> 237
01:07:05.779 -> 236
01:07:05.779 -> 235

01:07:06.981 -> 7
01:07:06.981 -> 6
01:07:06.981 -> 5
01:07:06.981 -> 4
01:07:06.981 -> 3
01:07:06.981 -> 2
01:07:06.981 -> 1
01:07:06.981 -> 0
01:07:09.024 -> 0
01:07:09.070 -> 1
01:07:09.116 -> 2
01:07:09.164 -> 3
01:07:09.211 -> 4
01:07:11.241 -> 5
01:07:11.332 -> 4
01:07:11.378 -> 3
01:07:11.424 -> 2
01:07:11.470 -> 1
01:07:11.515 -> Arduino ready

```

Figure 7-10 - ESC calibration sequence

7.2.4 Ultrasonic modules

7.2.4.1 SRF05

The SRF05 ultrasonic module was initially implemented in Atmega328p using the following code. It reads the sensor value using the function called `ultrasonic()` and this function returns the distance measured in data type float (32-bit – 4byte) (*float - Arduino Reference*, no date). The code snippet is given below.

```

1. float ultrasonic()
2. {
3.     digitalWrite(trig, LOW);
4.     delayMicroseconds(2);
5.
6.     digitalWrite(trig, HIGH);
7.     delayMicroseconds(10);
8.     digitalWrite(trig, LOW);
9.

```

```

10. dur = pulseIn(echo, HIGH);
11. delay(50);
12.
13. //Serial.println(dur);
14.
15. dis = ((dur * 0.034)/2 )/100;
16.
17. // Serial.println(dis);
18.
19. return dis;
20. }

```

Figure 8-10 shows a simple implementation of the KF filter on the SRF05 ultrasonic module. This was an attempt to filter noise produced by the ultrasonic sensor while implemented in an Atmega328p microcontroller. This ultrasonic sensor uses UART communication protocol.

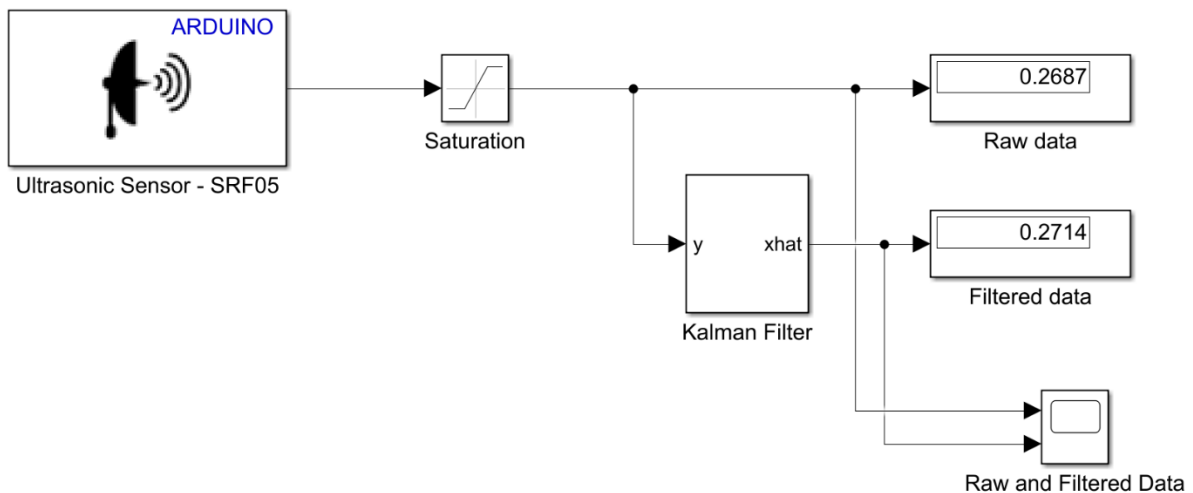


Figure 7-11 - KF implementation on SRF05

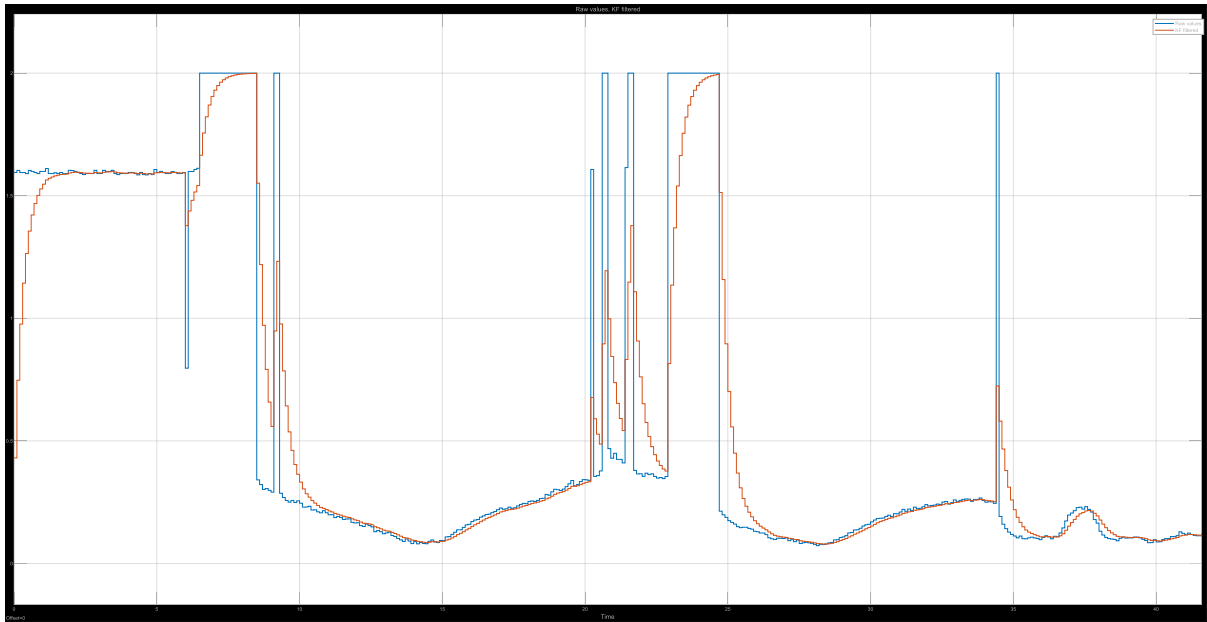


Figure 7-12 - Ultrasonic SRF50 results with KF

The disadvantage of this sensor is it requires a dedicated GPIO pin to read the measurement. In other words, the microcontroller will require 4 pins per ultrasonic unit. Therefore, an alternative is I2C ultrasonic sensor.

7.2.4.2 MAXSonar

The MAXSonar ultrasonic sensor adopts the I2C communication protocol. There can be up to 127 slave I2C devices on the I2C line. To communicate with an I2C device and its internal features, there are two crucial things necessary to know before communicating with any slave devices.

Slave devices address

Whether to perform a **READ** or **WRITE** command

The datasheet of an I2C device will contain this information. The essential factor to understand the purpose of the slave device is whether to execute **READ** or **WRITE** sequence. In this scenario, the slave device **MB1242** ultrasonic sensor is required to have a **WRITE** at the end of the slave address ('**0x70**'), followed by the pointer register requesting rang reading. Note the binary representation of the slave address contains '**0**' at the least significant bit which represents a **WRITE** command. In contrast, '**1**' represents a **READ** command. The pointer register is not available on the datasheet for this particular device. However, the register which is required to obtain the range reading is given as '**0x51**' as shown below.

Command	Hexadecimal Value	Decimal Value	Explanation
Take Range Reading	51	81	Sensor takes a new range reading

The initial sequence of operation carried out, as per the datasheet, using the i2c protocol for the ultrasonic module is illustrated in Figure 8-12 below.

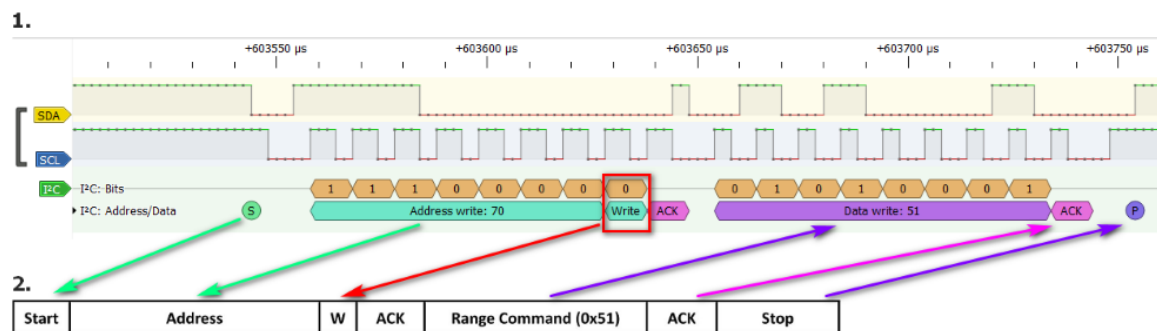


Figure 7-13 - I2c for MAXSonar Ultrasonic

The timing diagram shown above (see Figure 7-13) was obtained by a logic analyser and by running the Atmega328P code shown below.

```
#include "Wire.h"

void setup()
{
  Serial.begin(9600);    // Enabled for printing and debuggin
  Wire.begin();          // Essential for I2C communication
}

void loop()
{
  Wire.beginTransmission(0x70); // Slave address
  Wire.write(0x51);              // Register to receive range
  Wire.endTransmission();        // Ends I2C Transmission

  delay(100);
}
```

```

Wire.requestFrom(0x70, byte(2)); //Sends a READ request of 2 bytes

if(Wire.available() >= 2)
{
    byte FirstByte = Wire.read(); // First byte of data
    byte SecondByte = Wire.read();// Second byte of data

    word rangeReading = word(FirstByte, SecondByte); // Combine two bytes

    Serial.println(rangeReading); // Prints the reading
}
}

```

The working Arduino model was used as a reference to replicate the command in Simulink and then on MATLAB. There were several hindrances while executing the model in Simulink and MATLAB. The Simulink diagnostic viewer did not return any error; it stopped working after a few seconds, which made it difficult to debug. The second option was to use the MATLAB command window to replicate Arduino's working model.

```

1.  clc                                % Clears command window
2.
3.  fprintf('Cleared workspace\n');      % Displays message in command window
4.
5.  a = arduino('com4','uno','libraries','I2C')% Creates an arduino object 'a'
6.  scanI2CBus(a);
7.  mb1242 = device(a, 'I2CAddress', '0x70'); % Creates slave object 'mb1242'
8.
9.  fprintf('Initialisation Complete \n'); % Initial setup complete message
10.
11.

```

```

12. for i = i:50                                % Creates a loop to read no: of readings
13.   writeRegister(mb1242, 1, hex2dec('51'), 'uint8'); % WRITE request to register '0x51'
14.   pause(0.1);                                % 100mS delay - Important
15.   data = read(mb1242, 2, 'uint8');             % Sends a READ request the SLAVE 0x70 for
        2 Bytes of data
16.   bitconcat(fi(data(1),0,8,0), fi(data(2), 0, 8, 0))% Received 2 Bytes are combined
17. end
18.
19. fprintf('End of program\n');                  % Displays message in command window
20. % clear all
21. clear a                                       % Clears Arduino object 'a' and associated data
22. clear data
23. clear i
24. clear mb1242

```

7.2.5 Error checking and UART communication

The communication block comprises three blocks; Serial Receive, Serial Send and Serial Configuration blocks. Serial receive expects header **0x3C** and **0x3E** following that, 3 bytes of data for roll, pitch, and altitude respectively (yaw is ignored for simplicity see Figure 7-14).

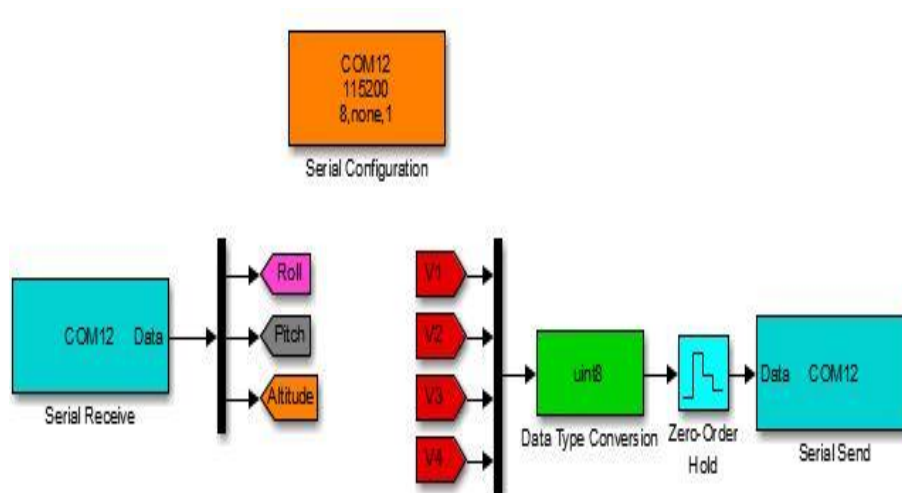


Figure 7-14 - Serial Configuration Simulink

The other objective was to send PWM motor signals from Bluetooth devices to Atmega328P. However, the received data was corrupted. The setup of having headers helped receive correct

PWM values. In loose terms, this setup can be called checksum. Figure 8-14 shows corrupted data captured while data was sent between devices.

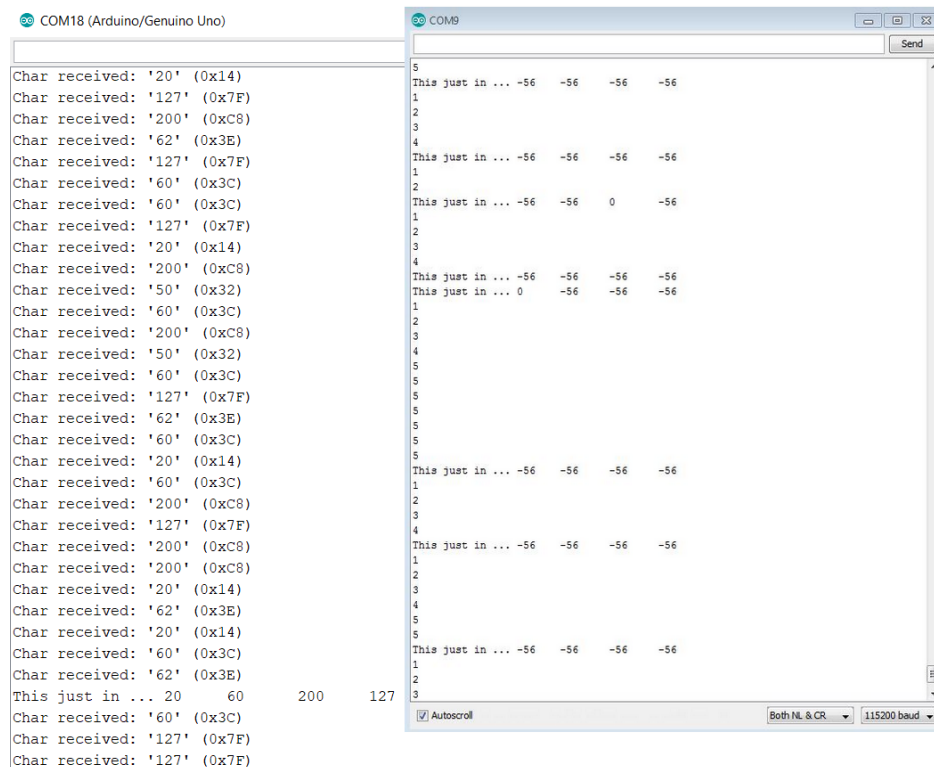


Figure 7-15 - Corrupt data

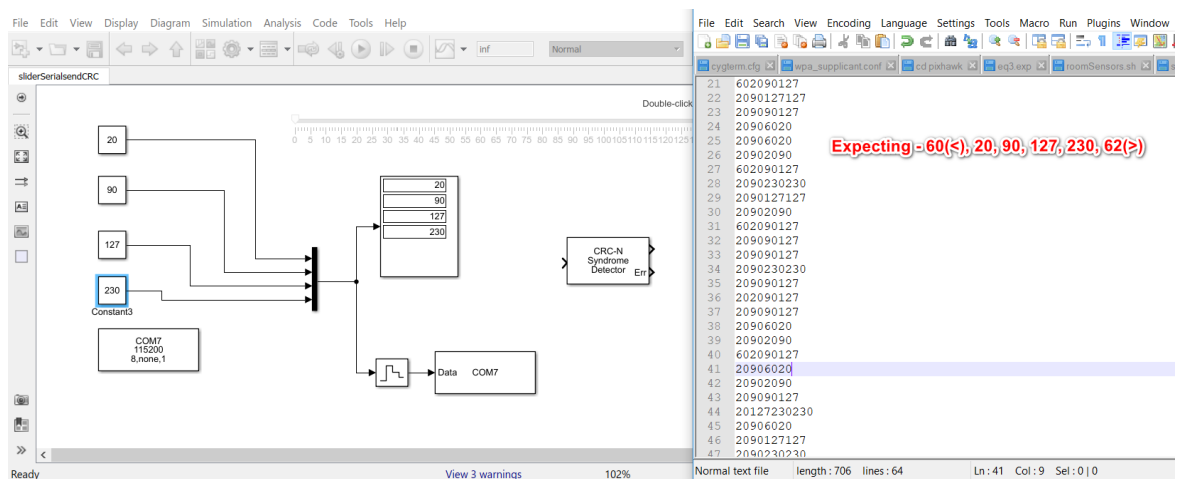


Figure 7-16 - Expected incoming data test

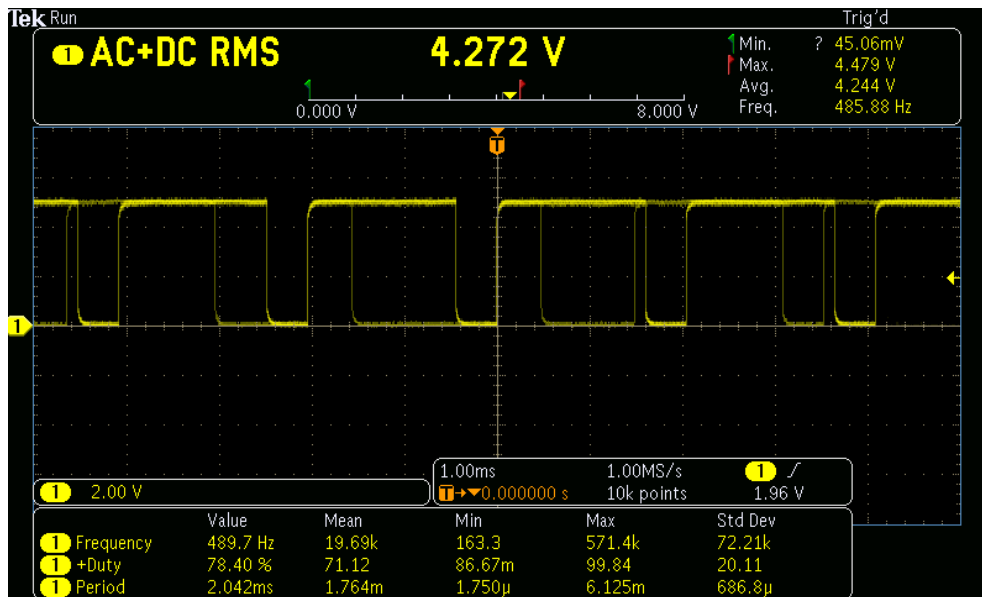


Figure 7-17 - Noise captured using the oscilloscope

To eliminate this error, the following code is implemented in Atmega328p with the function name `recvWithStartEndMarkers()`. It expects data arriving from MATLAB/Simulink, i.e. PWM motor signals. Initially, the function waits and reads data arrived in line numbers **8 – 12**. Then checks whether arrived data contains a starter maker **0x3C** or '**<**' in ASCII (set within the **Serial Send** block). This can be seen in lines **30-31**. Once the starter marker is received, the function looks for the end maker **0x3E** or '**>**'.

```

1 void recvBytesWithStartEndMarkers()
2 {
3     static boolean recvInProgress = false;
4     static byte ndx = 0;
5     byte startMarker = 0x3C;
6     byte endMarker = 0x3E;
7     byte rb;
8
9
10    while (Serial.available() > 0 && newData == false) {
11        rb = Serial.read();
12        Serial.println(rb);
13
14        if (recvInProgress == true) {

```

```

15     if (rb != endMarker) {
16         receivedBytes[ndx] = rb;
17         ndx++;
18         if (ndx >= numBytes) {
19             ndx = numBytes - 1;
20         }
21     }
22     else {
23         receivedBytes[ndx] = '\0'; // terminate the string
24         recvInProgress = false;
25         numReceived = ndx; // save the number for use when printing
26         ndx = 0;
27         newData = true;
28     }
29 }
30
31 else if (rb == startMarker) {
32     recvInProgress = true;
33 }
34 }
35 }

```

Following that, another function captures 4 bytes of data after ensuring the data has been captured between the start and end markers. Any other data received outside of this window will be discarded. This function separates the received 4 bytes into 1-byte data (line numbers **10-14**). Then they are sent to the appropriate ESCs.

```

1 void showNewData()
2 {
3     if (newData == true) {
4
5         for (byte n = 0; n < numReceived; n++) {
6             Serial.println(receivedBytes[n]);

```

```

7
8     }
9
10 //   Received data from Simulink
11
12     Serial.println(receivedBytes[0]);                               Serial.println(receivedBytes[1]);
13 Serial.println(receivedBytes[2]); Serial.println(receivedBytes[3]);
14
15     M1 = receivedBytes[0]; M2 = receivedBytes[1] ; M3 = receivedBytes[2]; M4 =
16 receivedBytes[3];
17
18 //     esc5.write(M1); //     esc6.write(M2); //     esc9.write(M3); //     esc10.write(M4); //
19 Serial.println();
20
21     analogWrite(3, M1); analogWrite(9, M2); analogWrite(10, M3); analogWrite(11, M4);
22
23     newData = false;
24 }
25 else
26 {
27     M1 = 0, M2 = 0, M3 = 0, M4 = 0;
28     //     esc5.write(M1);
29     //     esc6.write(M2);
30     //     esc9.write(M3);
31     //     esc10.write(M4);
32
33     analogWrite(3, M1); analogWrite(9, M2); analogWrite(10, M3); analogWrite(11, M4);
34 }
35 }

```

Although this error-checking setup is not robust, it was acceptable for the application.

7.2.1 IMU

The IMU used in this section is the MPU6050 module which works on the I2C protocol. Figure 7-18 gives the block diagram representation of a how-to setup of the internal registers and addresses to receive raw data to the microcontroller. Also, Figure 7-18 represents the wiring diagram.

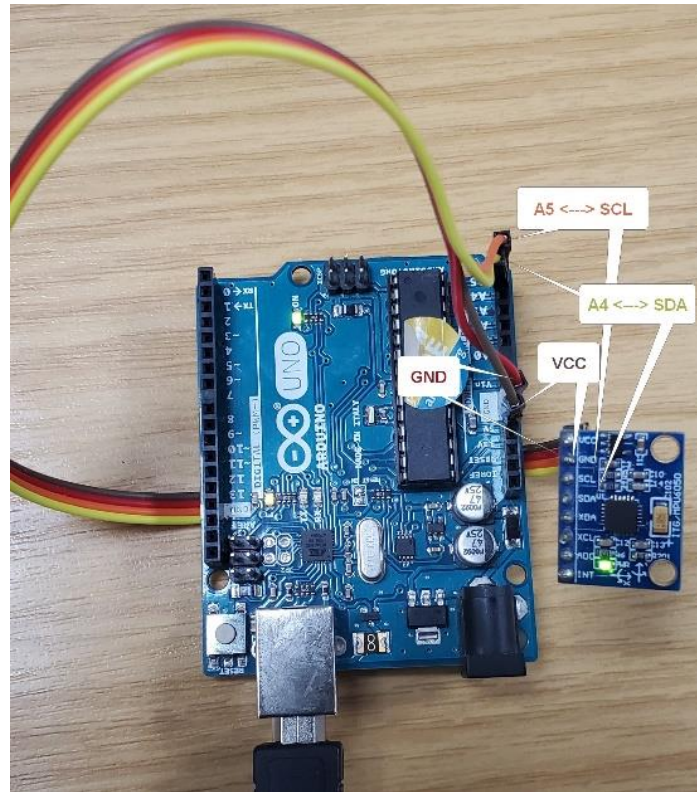


Figure 7-18 - IMU wiring diagram

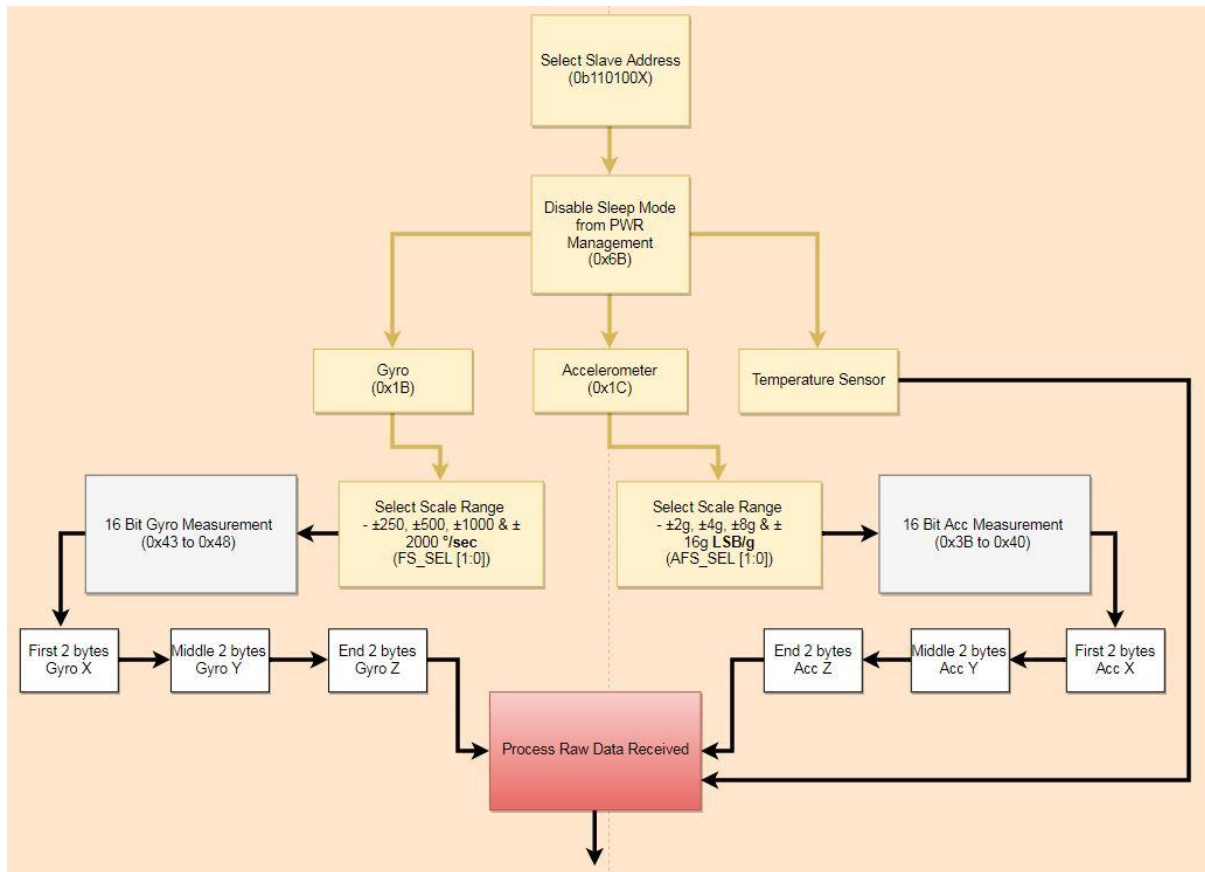


Figure 7-19 - IMU MPU6050 Block diagram

The block diagram Figure 7-19 can be divided into three stages – Initial setup, select range/sensitivity scale, and combining data to a data type using bit-shifting. The initial design can be seen in yellow blocks, and this is done one time during IMU power-up (*Create a 6dof IMU with a gyro and accelerometer for (Arduino) multicopters.*, no date).

```

1  void onetime_IMU_registerSetup(){
2    Wire.beginTransmission(0x68);
3    Wire.write(0x6B);
4    Wire.write(0x00);
5    Wire.endTransmission();
6    Wire.beginTransmission(0x68);
7    Wire.write(0x1C);
8    Wire.write(0x10);
9    Wire.endTransmission();
10   Wire.beginTransmission(0x68);
11   Wire.write(0x1B);

```

```
12    Wire.write(0x08);
13    Wire.endTransmission();
14 }
```

The grey and white boxes are implemented by calling ***conti_IMU_data()*** which reads data continuously. The read data are sent to their corresponding variables for further processing and filtering.

```
1 void conti_IMU_data(){
2   Wire.beginTransaction(0x68);
3   Wire.write(0x3B);
4   Wire.endTransmission();
5   Wire.requestFrom(0x68,14);
6   while(Wire.available() < 14);
7   accelX = Wire.read()<<8|Wire.read();
8   accelY = Wire.read()<<8|Wire.read();
9   accelZ = Wire.read()<<8|Wire.read();
10  temp = Wire.read()<<8|Wire.read();
11  gyroX = Wire.read()<<8|Wire.read();
12  gyroY = Wire.read()<<8|Wire.read();
13  gyroZ = Wire.read()<<8|Wire.read();
14 }
```

7.2.2 Sensor fusion

An IMU is a device that measures the linear and angular velocities and positions of an object. It consists of a triad of sensors: a tri-axial gyroscope which measures the rate of rotation about each axis of the body frame, a tri-axial accelerometer which measures the rate of acceleration about each axis of the body frame, and a tri-axial magnetometer that measures the magnetic field strength at the triad centre of gravity. The triad has three degrees of freedom (DOF) (one in the body frame and two in the Earth frame). In a typical application, IMU data is used to estimate the state of a vehicle, such as the attitude and angular velocity of its airframe.

An IMU is an inherently noisy system. Sensor noise and bias are the most common sources of errors. The sensor noise results from intrinsic sensor imperfections as well as noise from the

environment. The bias is the systematic error present in the measurement. Since an IMU is a three-degree of freedom system, there are six bias parameters to be estimated. Some of the common issues are;

Gyroscopes do not measure angular position but the angular rate (rads/s)

Accelerometers measure more than just linear acceleration (m/s^2), for instance, gravitational acceleration and Coriolis terms.

Converting the frame of reference from the body to the inertial frame.

Noise in measurements and offset.

Traditionally, a Kalman filter has been the preferred method for removing platform noise and bias in the sensor measurements. However, a Kalman filter requires the motion state to be accurately estimated, and this can become difficult in an Unmanned Air Vehicle. This is because the UAV is at the mercy of the wind, and as such, the motion state is tough to estimate. An alternative filter is a complimentary filter. This is simple to implement however has some drawbacks.

The problem of tracking an Unmanned Air Vehicle in a windy environment has been extensively studied. However, as long as the motion estimation is not too poor, the Kalman filter will give reasonable estimates of the state.

7.2.2.1 *Calibration*

Inertial measurement unit calibration is essential to keep a quadcopter straight. This means the flight controller receives accurate information about the direction of the quadcopter so it can make better decisions. In simple terms, calibration is a process of determining the angular offsets between the IMU and the main flight control board in your drone. These offsets are used to rotate any heading information from the IMU into absolute position information so that quadcopter knows precisely where its front and back ends are at all times. Usually, sensor calibration is required before each flight.

7.2.2.2 *Accelerometer*

An IMU accelerometer measures the acceleration of the device in each of the three axes. The acceleration is measured by detecting the force exerted on the MEMS element by the movement of a mass. It is possible to estimate roll and pitch using basic trigonometry using raw data obtained at every three axes. The yaw value cannot be estimated using an accelerometer reading because the gravity is the same regardless of its direction.

$$\phi_{acc} = \arctan\left(\frac{ccelY}{\sqrt{AccelY^2 + AccelZ^2}}\right) \times \frac{180}{\pi} \left[\frac{rads}{s}\right] \quad 7.4$$

$$\theta_{acc} = \arctan\left(\frac{AccelX}{\sqrt{AccelY^2 + AccelZ^2}}\right) \times \frac{180}{\pi} \left[\frac{rads}{s}\right] \quad 7.5$$

Since there will be high-frequency noise on the signal a low pass filter needs to be used. A moving average is implemented for a low pass filter in this example. As it can be seen in Figure 7-20 at higher frequencies motion the low pass filter attenuates the amplitude of the signal.

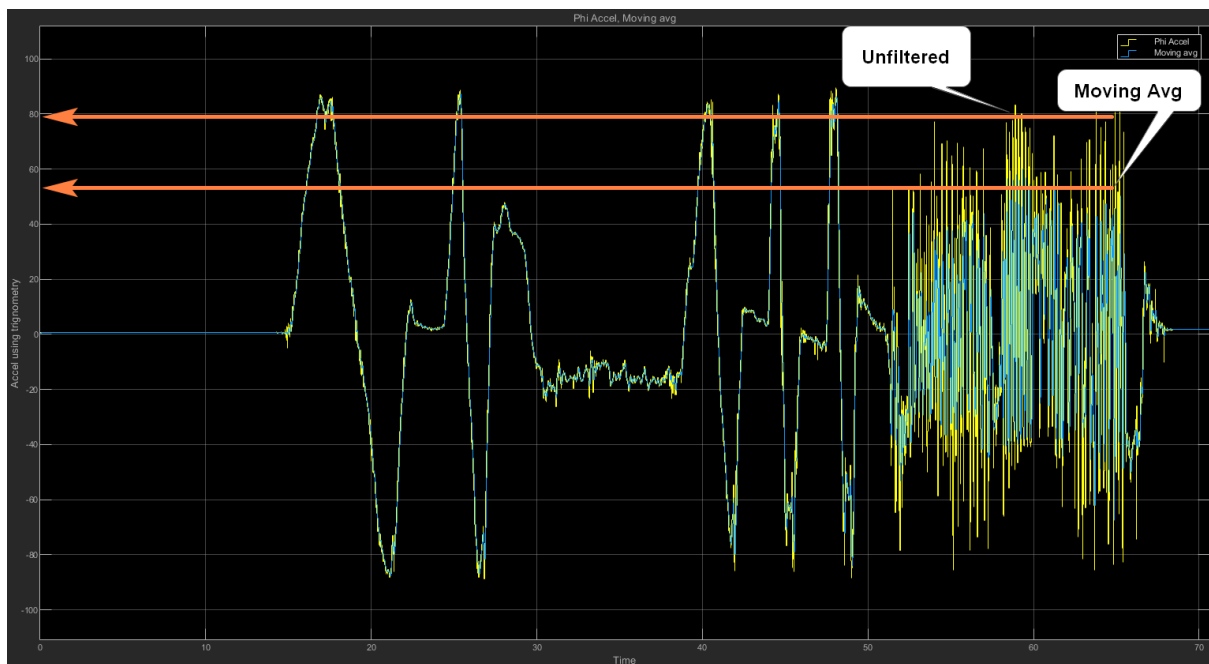


Figure 7-20 - Accelerometer moving average

The implementation of the accelerometer error in Simulink was as shown in Figure 7-21 below.

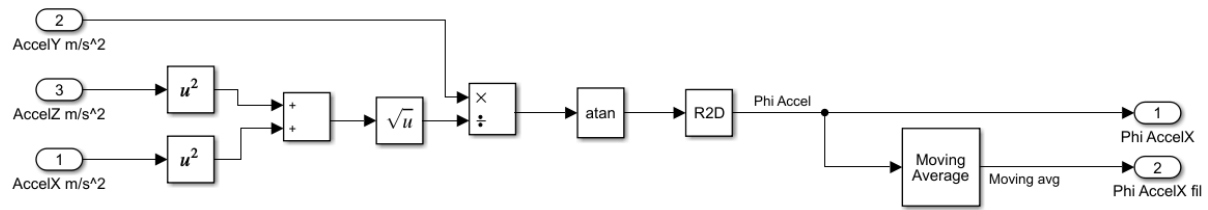


Figure 7-21 - Accelerometer block diagram demo

7.2.2.3 Gyroscope

The gyroscope in an IMU provides information about the rate of change of the angular orientation of the device. This information is obtained from the angular velocity measured by the gyroscope. The angular velocity of a device is the rate of change of the angle of the device measured in radians per second. This rate of change occurs around each axis in this body frame since the IMU is placed on the body of the quadcopter.

It is easy to assume that integrating the rate of change will obtain an angular position relative to the body frame. However, inherently these sensors have noise present in them. The effect known as gyroscopic drift occurs when these small noises are added and accumulated over time.

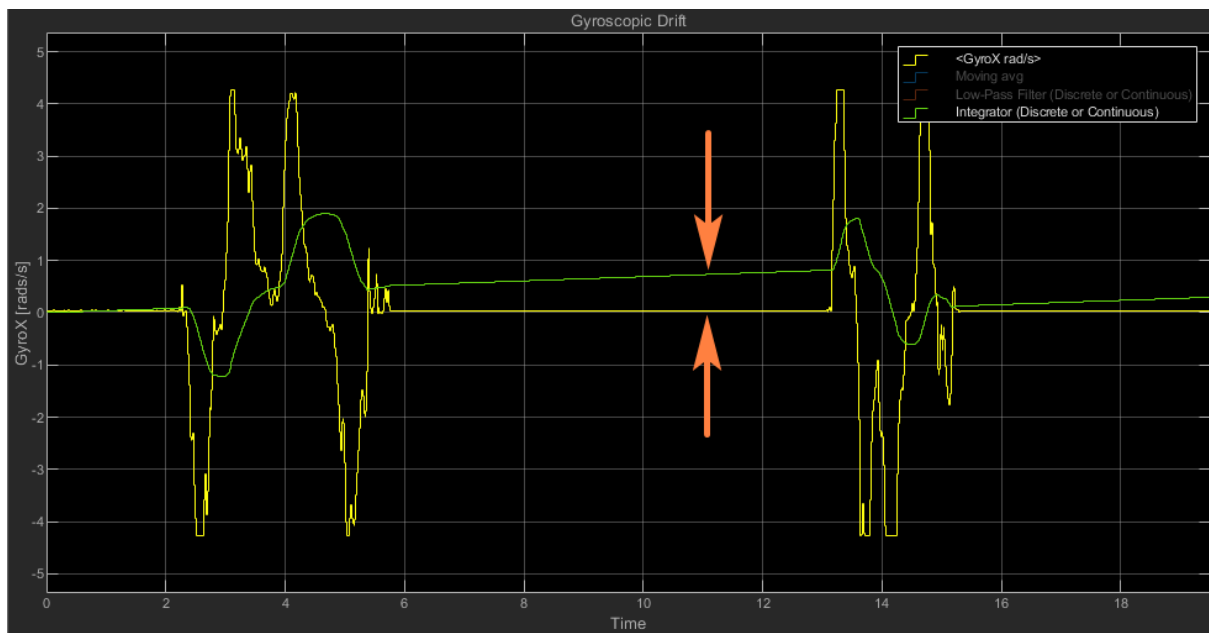


Figure 7-22 - Gyroscopic drift

By integrating angular rates of change in the body, the frame does not provide accurate roll, pitch, and yaw angles in the inertial frame because the rate of change of Euler angles relies on more than one body frame angular velocity. The transfer matrix derived in equation 2.9 is used to obtain rates of change in the inertial frame.

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi \tan\theta & \cos\phi \tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi \sec\theta & \cos\phi \sec\theta \end{bmatrix} \times \begin{bmatrix} p \\ q \\ r \end{bmatrix} \quad 7.6$$

7.2.2.4 Complimentary filter

From the previous section, it can be noticed both sensors have drawbacks. The accelerometers are only good when the system is motionless and the gyroscopes are for a short duration. As the name implies, the complimentary filter combines the desirable high-frequency characteristics of the gyroscopes and low-frequency characteristics of accelerometers.

The main advantage of a complimentary filter is it has only one adjustable/tuneable parameter, i.e., the gain of alpha α . However, increasing the alpha gain will result in heavy dependence on accelerometer data and therefore result in noisy data. It can be seen in Figure 7-23 that a gain of 0.1 depends more on gyroscope data. This makes high-frequency motion less attenuated and filters noise components reasonably well.

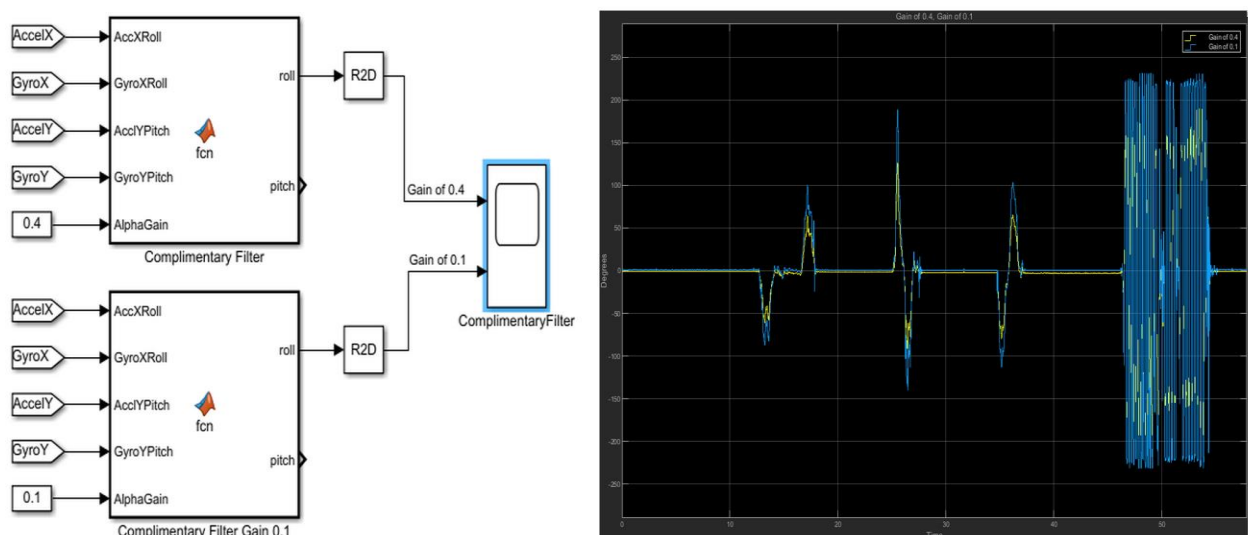


Figure 7-23 - Complimentary filter gain 0.4 and 0.1

The equation to implement a complimentary filter is given in equation 7.7 below. The live script of the complimentary filter can be found in APPENDIX G.

$$\dot{\phi} = (1 - \alpha) \times \phi_{gyro} + \alpha \times \phi_{acc} \quad 7.7$$

$$\dot{\theta} = (1 - \alpha) \times \theta_{gyro} + \alpha \times \theta_{acc}$$

To get the real angle at which the quadcopter has tilted, the transformation matrix must be incorporated before the compliment filter. The result shown in Figure 7-24 demonstrates roll and pitch movement and holding the desired angles.

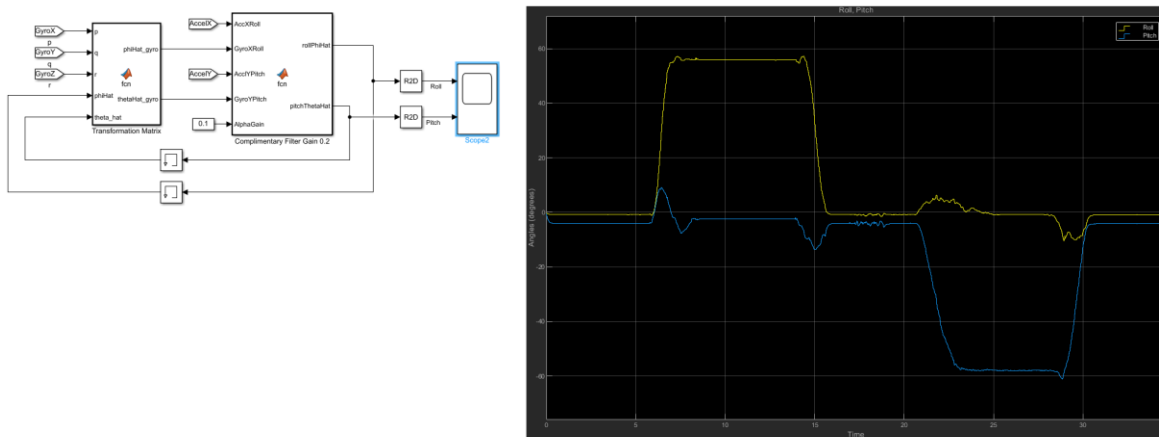


Figure 7-24 - Complimentary filter true angle

7.2.2.5 Kalman filter

The complimentary filter is easy to implement if considering the IMU has high accuracy, sensor noise is minimal, and the quadcopter does not encounter rigorous motion/manoeuvre. If these conditions are met, the alpha gain can be easily tunable. Kalman filter allows the system to choose gain coefficients mathematically through iterations. In other words, the Kalman filter not only takes sensor data into account but also the system dynamics. Since the Kalman filter was explained above, this section will focus on formulating generalised system dynamics and implementing the Kalman filter algorithm to be used on various systems ('IMU Attitude Estimation – Philip Salmony', no date).

The following considerations were made to develop a generalised system matrix.

- An accelerometer is used to take an estimate of angular positions.
- Remove the changing gyroscopic bias, which results in a gyroscopic drift

- To obtain the angular position, subtract the gyroscopic bias measured from the transformed angular rate of change.

Generalised measurement vectors, input vectors and state vectors are defined as follows

$$\vec{x}_t = \begin{bmatrix} \hat{\phi} \\ \alpha_{\hat{\phi}} \\ \hat{\theta} \\ \alpha_{\hat{\theta}} \end{bmatrix} \quad 7.8$$

$$\vec{u}_t = \begin{bmatrix} \dot{\phi}_G \\ \dot{\theta}_G \end{bmatrix} \quad 7.9$$

$$\vec{z}_t = \begin{bmatrix} \hat{\phi}_{acc} \\ \hat{\theta}_{acc} \end{bmatrix} \quad 7.10$$

Here the $\alpha_{\hat{\phi}}$ represents gyroscopic bias related to estimate $\hat{\phi}$. So, the generalised state-space equation can be written as.

$$\vec{x} = \begin{bmatrix} 1 & -t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -t \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \vec{x} + \begin{bmatrix} t & 0 \\ 0 & 0 \\ 0 & t \\ 0 & 0 \end{bmatrix} \vec{u} + \vec{w} \quad 7.11$$

$$\vec{y} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \vec{x} + \vec{v} \quad 7.12$$

Here $\vec{w}$ and $\vec{v}$ represents process noise and measurement noise covariance. The result of implementing KF on RPi can be seen in Figure 7-25 and Figure 7-26

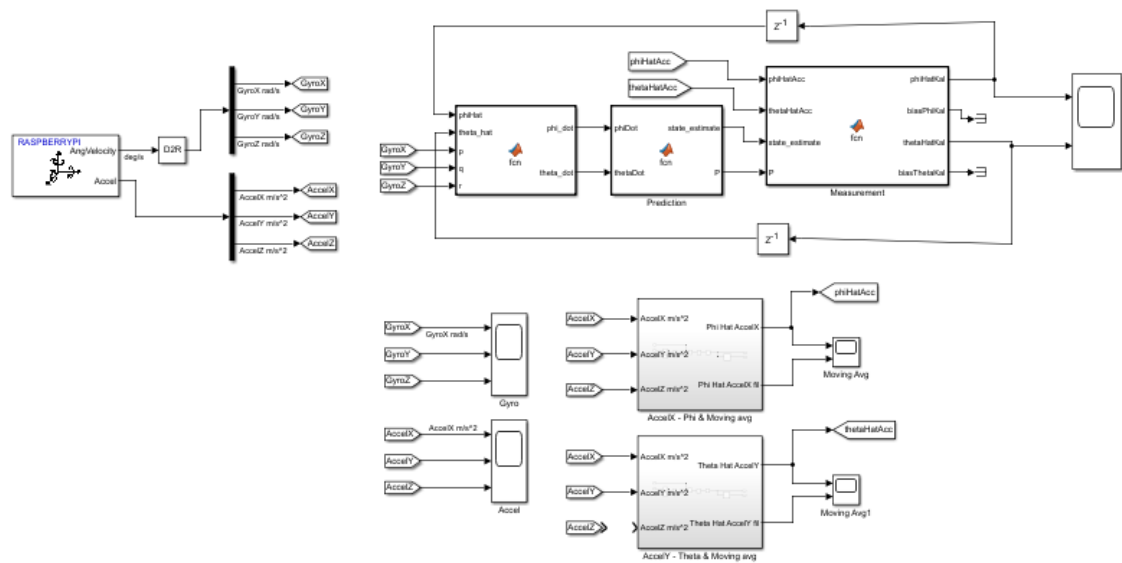


Figure 7-25 - KF model

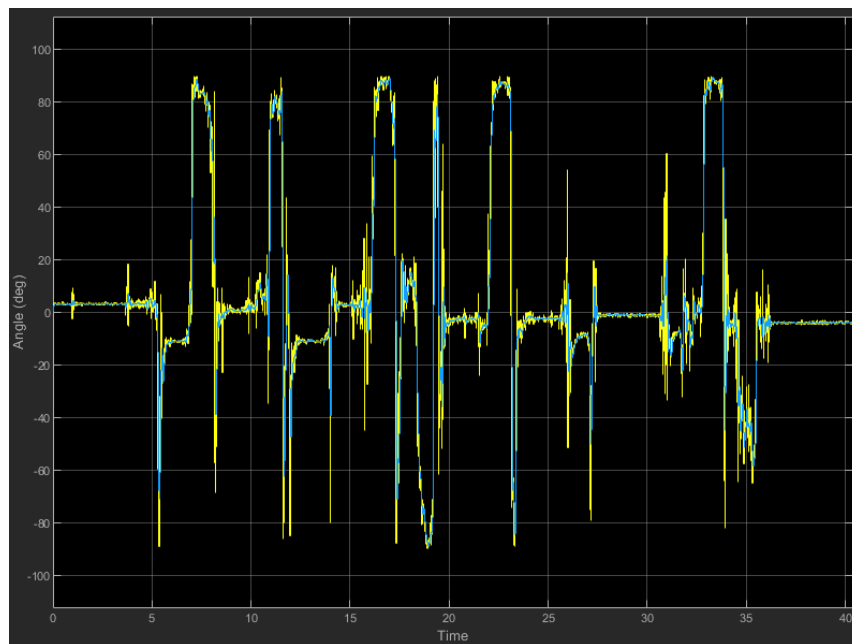


Figure 7-26 - KF result

7.3 Raspberry Pi 3 (RPI 3)

Raspberry Pi (RPI) is a small computer running a Linux-based OS. It is a capable little PC that can be used in electronics projects. For many of those projects, it can act as a standalone computer without needing a separate PC, which makes it very useful for electronics projects such as automated systems or sensors or servers.

The implementation of the model on RPi was straightforward compared to Atmega328p. Thanks to the MATLAB/Simulink support package for RPi. The IMU used on RPi is a snap-on module known as SenseHat.

This section will focus on manipulating a camera as part of sensor fusion and path following. It is computationally challenging to have a camera module on Atmega328p and use it as part of sensor fusion. The approach to achieve lane detection is divided into four; grayscaling the input images, detecting edges, detecting corners, and finally detecting lanes. The tests were conducted on an area marked with white tape. The objective here is to detect edges and corners within this area.



Figure 7-27 – Lane detection test area

7.3.1 Sobel – Edge detection

The edge detector is an integral part of image segmentation. It is used to detect regions of an image that have varying intensity borders rather than homogeneous intensity. This is a standard method to extract areas of an image. Many edge detectors have been used in the field of computer vision, such as Sobel edge detection, Canny edge detection, Laplacian edge detection etc.

The Sobel edge detector by taking an image's pixel intensities and calculates its gradient. Initially, the image is greyscaled, and the kernel convolution operation is executed. The kernel consists of two 3x3 matrices, which represent a filter for edge detection. Vertical gradient approximation (Sobel-Y) is carried out by one of the kernels, and horizontal gradient approximation kernel is for the Sobel-X direction.

Common values used in Sobel kernels are as follows(Horak and Zalud, 2015);

$$X = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \text{ and } Y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad 7.13$$

The greyscaled image is convoluted with Sobel-X and Sobel-Y kernels. In other words, the input image Sobel kernel scans across the entire image and calculates the sum per every Sobel matrix. The Sobel-X calculates the sum across the horizontal axis and Sobel-Y on the vertical axis.

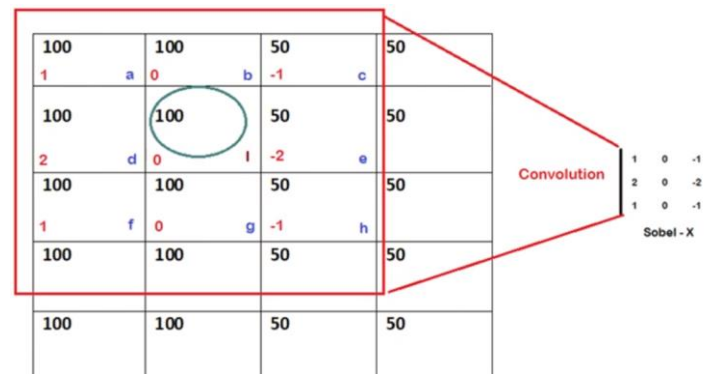


Figure 7-28 - Sobel-X operation (Session 17 - Sobel Edge Detector - A Quick Understanding, 2020)

The greyscaled pixel values are added to the matrices. The resulting gradient approximation is calculated using equation 7.14.

$$G = \sqrt{G_x^2 + G_y^2} \quad 7.14$$

Here G_x and G_y represents the matrix sum, and G represents gradient approximation per Sobel matrix. This G value is compared against a set threshold which determines whether the area in the image is an edge or not.

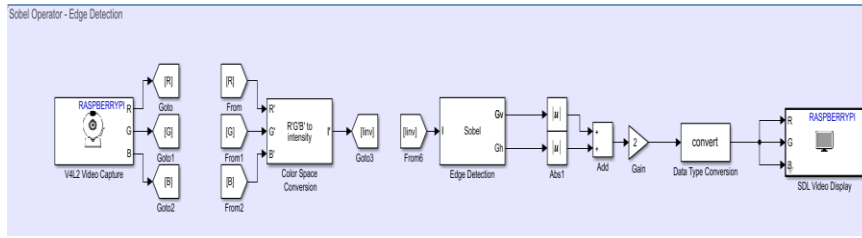


Figure 7-29 - Edge detection model(Horak and Zalud, 2015)



Figure 7-30 - Edge detection result

7.3.2 Harris & Stephen Operator – Corner Detection

In this method, the change in intensity of an image is calculated using the following equation. This equation is derived using the Taylor function for the 2D equation (*Harris Corner Detection - Computer Vision (Python)*, 2019).

$$E(u, v) = \sum [I(x + u, y + v) - I(x, y)]^2 \quad 7.15$$

Here u, v is the shift in intensity and x, y is the original intensity of the image. So, subtracting the shift and original intensities gives E(u, v) change in intensity. The equation can be written in matrix form as;

$$\sum [u, v] \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \times \begin{bmatrix} u \\ v \end{bmatrix} \quad 7.16$$

Consider the shift is very small and if a bilinear approximation is taken of the above equation 7.16, the following is obtained.

$$\sum [u, v] M_{CD} \times \begin{bmatrix} u \\ v \end{bmatrix} \quad 7.17$$

Here M_{CD} is a function where the windowing function (W) has a value of 1.

$$\sum w(x, y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \quad 7.18$$

Like the G value from equation 7.14, here R-value decides whether the image has corners in it or not. To calculate the R, the trace and determinant of M need to be calculated.

$$R = \det M - k(\text{trace } M)^2 \quad 7.19$$

Here determinant M is the result of multiplying both its eigenvalues, trace M is obtained by adding both eigenvalues, and K is a constant established empirically.

The chapter describes how a simple error detection technique is used. It also demonstrates the sensor noise filtering techniques using moving average, complimentary filter and Kalman filter. The chapter reflects on the advantage and disadvantages of different sensor filtering techniques used for sensor fusion. As per the literature review, the Least-square based minimisation model is the most suitable competitor to the Kalman filter. They are used primarily in GNSS (Global Navigation Satellite System). Kalman filters offer state change information, unlike Least-square minimisation. This information will increase estimation accuracy. However, an epoch-by-epoch least-square solution can assess raw data quality better than a Kalman filter. Both approaches may be useful or complementary depending on the application. Another alternative to the Kalman filter is the Double Exponential Smoothing Based Prediction (DESP) which is widely used in business and economic forecasting. The primary disadvantage is that it produces forecasts that lag behind the actual trend and it cannot handle trends well as per the literature review.

A Hough method was attempted for lane detection; however, the lane detection model wasn't fully materialised due to various errors. For lane detection to work, an edge detection technique is required. In this chapter Sobel edge detection method is implemented successfully as it can be seen in the figures above.

Chapter 8 - Conclusions and further work

This chapter presents the overall summary, conclusion, and future recommendations.

8.1 Research Objective

To complete the research project, the following objectives had to be achieved:

- a) Investigate the modelling technique and control theory that will be applied to the quadcopter based on previous work.
- b) Linearise the dynamic model and create a state-space (SS) representation.
- c) Investigate and experiment with different methods to evaluate the aerodynamic and hardware parameters of the quadcopter.
- d) Simulate and verify the results on MATLAB/Simulink.
- e) Develop a rapid prototype 3D model using the popular CAD format for verification and visualisation.
- f) Develop a quadcopter controller model incorporating data fusion/sensor fusion and error correction for a GPS-denied environment. This includes various filters and incorporating a camera for computer vision.
- g) Preliminary tests include deploying the model on the hardware using Atmega328p with Bluetooth communication and basic sensors.
- h) Finally, RPi will be used to add more computationally resourceful sensors for further optimisation and development.

8.2 Research summary and conclusions

As per the research objectives mentioned above, the outcome of the thesis is laid out as follows:

8.2.1 Mathematical modelling of the quadcopter

The essence of Chapter 1 is to provide an overview of types of UAVs from fixed-wing aircraft to multi-rotor aircraft. The importance of the different types of UAVs for different applications is emphasised in this chapter. Applications range from military use to humanitarian relief operations, agriculture, oil and gas exploration, scientific research and surveillance. Examples are highlighted where quadcopters were used in humanitarian relief operations and other tasks. The applications of the quadcopter are followed by a discussion on public safety concerns that may arise from using drones for recreation purposes. This paper provided information about current legislation governing drone use in the UK, who can or cannot own/operate them and what kind of insurance/license will be required to operate one legally. It further discusses research findings related to laws to fly drones over people, premises and distance to maintain while operating.

In Chapter 2 a comprehensive literature review is provided which considers work from early prototypes to today's state-of-the-art controller designs. The literature review reveals a lack of systematic studies on the trade-offs between various controller design parameters in modern controllers as well as obtaining hardware design parameters. This chapter also reveals that small quadcopters are, in fact, more difficult to control than larger aircraft. The major factor contributing to this difficulty lies in the need for an accurate estimation of the quadcopter's components. That is the rotor's angular position, angular velocity (e.g. light/hall effect sensor etc), and quadcopter's position in an inertial frame (LIDAR, Ultrasonic etc). Position estimation is especially challenging on small platforms where the mass of sensors used for position estimation is comparable with or even exceeds the mass of the vehicle itself. This comparison makes it difficult for sensor systems to remain firmly mounted on the vehicle while also providing accurate estimates. Since certain parameters are estimated, a need for another layer of verification (other than graphical) was necessary to evaluate the performance of the quadcopter before being deployed. This paved way to use rapid prototype CAD models (STL and OBJ) for 3D visualisation which gives a dynamic response to system performance.

The core of the whole research involves making a faithful mathematical representation of the quadcopter model. Therefore Chapter 3, Chapter 4 and Chapter 5 go in-depth into deriving these equations, synthesizing quadcopter control laws and validating them in MATLAB/Simulink through open and close loop testing. These chapters carefully consider the derivation of equations of motion, the Newton-Euler method, and the importance of rotational sequence and coordinate frames. Linearisation of the non-linear equation and representation in state-space form. It also goes in-depth regarding motor control, and propeller blade theory, and highlights gyroscopic effects. The results suggest that the quadcopter behaves as expected.

Chapter 5 consists of the linearisation of the equation, implementing a flight controller algorithm. The experiments carried out to obtain the moment of inertia seem to be the best-preferred method. The bifilar pendulum method was examined and compared with Parrot Mambo's known values from the Simulink package. Similarly, motor inductance was established through various experimental techniques. LCR meter seems to return the best results among other practical methods. However, the permanent magnets must be removed while measuring. Finally, a makeshift thrust measurement unit was made using a load cell to determine thrust to weight characteristics of the BLDC motor. The thrust-to-weight unit approximated the BLDC motor and was verified through empirical testing.

Chapter 7 is exclusively focused on developing a visualisation platform for 3D models. This uses 3D models made in an open-source Windows 3D builder. Any CAD software can be used to create the quadcopter model. However, the file extension has to be in either STL or OBJ format. The chapter compares two popular file formats used in 3D modelling and compares the size, and time is taken to display flight trajectory on MATLAB. Any changes or disturbances added to the model can be seamlessly observed on this visualisation platform.

Initially, in Chapter 8, the Atmega328p was used for quick prototyping of the model. However, there were many hindrances during implementation. First of all, the quadcopter model inherently produces noises when the motors are running. This noise was corrupting the data, so a robust error correction method had to be implemented. Another major obstacle was encountered during the initial test flight. The ESC was not entering calibration mode, and this was evident visually and through ESC audio feedback. Upon investigation, it was determined that Atmega's internal timers were running at 500Hz and 1Khz frequencies. The MATLAB/Simulink support package for Atmega microcontrollers was in its infancy. After various investigations, these issues were rectified, and the model was demonstrated successfully.

Similarly, being significantly more powerful than Atmega328p, RPi was easy to implement. The RPi had a stable MATLAB/Simulink support package compared with Atmega microcontrollers. This enabled the addition of more computationally challenging components such as a webcam. Chapter 8 presents the initial stage of the lane following technique, and it is demonstrated with corner detection and edge detection method.

8.3 General Discussion

For several years, the quadcopter industry was still in its infancy and had not yet achieved widespread popularity. There have not been any major advancements made in the design of the quadcopter since the early 1900s. Throughout the years, researchers have experimented with a variety of approaches in an effort to develop more effective techniques. Very early models such as Jacques and Louis Breguet Gyroplane Oehmichen George de Bothezat and Ivan Jerome Quadrotor. and D. H. Kaplan model A all had human inputs to keep them airborne (Sigalos *et al.*, 2019). After 1958, quadcopter technology didn't make any big steps forward. Some of the technologies and devices weren't made available to the public until many years later. T.G. Wilson and P.H. Trickey came up with the name "DC motors with solid state commutation" for BLDC motors in the 1960s (Wilson and Trickey, 1962).

The first step of answering the research question was taken in Chapter 3 in which the deterministic part of the system was derived from the Newton-Euler equation in conjunction with a particular rotational movement sequence. The rotational sequence was settled in Chapter 2 in which rigid body moment equations were considered. The research paper Kumar *et al.* (2020) consists of using a quaternion method from Newton-Euler's equation for rotational movements. Another research paper Magnussen, Ottestad and Hovland (2013) uses quaternion to calculate accelerometers and gyroscopes from an inertial measurement unit (IMU) but uses a PD controller at 400Hz for attitude control (inner loop). This method eliminated the gimbal lock issue commonly encountered in Newton-Euler equations-derived UAVs. The gimbal lock principle occurs during rigorous flight manoeuvres only. The singularity phenomena can be avoided by limiting angular rotation to the quadcopter. Quaternion models are complex to create compared to the Newton-Euler equations. Since this research paper is for the rapid prototyping of quadcopters the quaternion method was opted out.

Newtonian mechanics use a predominantly rectangular coordinate system and accounts for all constraining forces. Lagrange's method cleverly avoids issues of constraint forces and employs any set of "generalised coordinates" such as angle, radial distance, etc. that are consistent with constraint relations. This system has the same number of generalised coordinates as degrees of freedom. In any dynamical system, the designer randomly selects some generalised coordinates that are compatible with the system's requirements. In Newtonian mechanics, the so-called Lagrangian is calculated by subtracting the system's kinetic energy from its potential energy. Consequently, the number of degrees of freedom of the system is denoted by the number of different equations derived. This research paper arrives at 6 degrees of freedom quadcopter which consists of 6 non-linear differential equations as seen in Chapter 2.

A nonlinear system is any physical device or process that cannot be precisely characterised by a set of linear equations (algebraic, differential, integral, functional, or a mix of some of these) (Chen, 2016). The motion of a system is regulated by a set of nonlinear equations of motion in a variety of practical problems. The motion equations can be linearized about any arbitrary initial condition. The purpose of linearizing the model is to forecast the outcome. The linearised equations can be connected directly with the state-space equations. Another approach to analysing the stability of a system is using the Transfer function method. State The following are some of the main restrictions of the Transfer function that are encountered

when compared with a state-space system: The system requires to be SISO (Single Input Single Output). Whereas state-space equations can be multiple-in-multiple out, non-linear, time-variant, etc. are all possible. Any initial conditions can be used in SS systems. In contrary TF models have to have initial conditions set to zero. Because of the limitations, there isn't much nuance to be found in the Transfer function models. There are four main representations for the state-space equation: controllable, observable, diagonal, and Jordon canonical forms.

Ziegler-approach Nichol's and root locus analysis were applied in Dong et al. (2013) PID's controller. Dong et al. (2013) achieved outdoor's trajectory tracking flight experiments yielded a 50 cm maximum positional inaccuracy using Zeigler-technique, Nichol's which does not operate in all systems. Velocity tracking was unknown. Castillo et al. (2011)'s research on real-time operating systems (RTOS) with nested saturations for quadcopters was noteworthy. This work showed a saturation-based non-linear quadcopter hovering control law. This followed a 2009 Sanahuja et al. (2010) investigation. Saturation functions in each state stabilise n integrators in a cascade with bounded inputs. The quadcopter chopper appears to have retained stability during abrupt system adjustments. Also, other studies, such as that of Loram, Gawthrop and Lakie (2006), despite demonstrating that a PID controller with a time delay could fit experimental data from a tracking task very well, found that the optimal PID time delay was too short compared to the mean response time delay of the subjects who participated in the task. It is important to mention the PID controllers were tuned by trial and error method. The PID controllers produced commendable outcomes (with slight overshoot) during the simulation of both the inner and outer loop feedback models. The process and sensor filtering was carried out by incorporating a Kalman filter. As per the literature review carried out in this research, other filters (DESP, Least-squared models) were better suited for different applications.

Even though the Kalman Filter was included in the controller design in order to lessen the effect of disturbances in steady-state, the fact that it produces a derivative state of the moment signal makes it potentially risky for the system's ability to maintain its stability in a closed-loop configuration. The use of a disturbance observer, the purpose of which is to estimate the constant disturbances and consider them while designing the controller. An alternate method to the use of an Extended Kalman Filter in the controller design especially for a non-linear system model.

This study by Oktay and Eraslan, (2020) presents the results of a Computational Fluid Dynamics (CFD) analysis of an 11-inch quad-rotor UAV propeller using the overset mesh approach at a rotational speed of 5000 RPM and several different air velocities. The experimental value of the thrust coefficient was found to be in good agreement with the numerical forecast, with a variance of around 14-18%. As was also predicted, the thrust coefficient was observed to decrease with increasing airspeed. In terms of the thrust coefficient, it has been observed that CFD findings follow the same patterns as experimental data. At lower velocities, numerical predictions are found to approach experimental results and results obtained from another study by Czyż *et al.* (2021).

The inspiration to implement CAD modelling, in Chapter 6, came from previous research papers such as Ortega *et al.* (2020) and Haque (2021). Ortega *et al.* (2020) demonstrate the use of STL files and cameras to inspect power plant boilers. Chapter 6 compares the sizes of two prominent file formats used in 3D modelling and simulates them using MATLAB. Unfortunately, Simulink does not support 3D visualisation using these file types, therefore the data are shown in the MATLAB workspace. It displays following a predetermined course. The simulation was successfully implemented and both models (STL and OBJ) work. However, the time taken to execute the model is approximately 10s using the OBJ file when compared to 43 seconds (approximately) due to the file size, which was not highlighted in the study by Ortega *et al.* (2020).

The research carried out by Haque (2021) (as mentioned in the literature review) makes use of SLAMdunk on Parrot Bebop 2, which is out of production and was expensive to purchase. As this research is focused on cheaper alternatives and rapid prototyping RPi with SenseHAT was deemed to be a suitable replacement. The RPi SenseHAT consist of barometer pressure, IMU, LED indicators, temperature sensor, magnetometer, camera integration, onboard joystick etc making it a lot cheaper alternative. Also, MATLAB/Simulink (*Sense HAT - MATLAB & Simulink - MathWorks United Kingdom*, 2021) provided an add-on to easily incorporate. Although this research did not fully materialise the hardware testing and validation, it opens up future opportunities for developers. This research demonstrates successful corner and edge detection using a camera installed on RPi.

The system model is well designed however the controller tuning using the trial and error method was not the best approach because tuning individual subsystems can become time-consuming and tedious. MCP controller, LQR controller etc better suited for a robust steady-

state response. The Atmega328p was not the right choice for this application. The processing power is too limited for a reliable system. Whereas RPi was able to demonstrate reliability for higher computational tasks. However, the temperature of the microprocessor was edging towards 55 degrees celsius without a cooling fan, whilst running edge detection and corner detection.

8.4 Findings and contribution

An easy method for rapid prototyping a 3D model using two popular file formats is made available (STL and OBJ). There are other addons on MATLAB however none of them seems to work in the 2019 MATLAB model. The 3D model can be made in any software, and it can be visualised using written software. The biggest advantage of this technique is rapid prototyping, however, this technique cannot be used for a live view of trajectory on Matlab. There are other applications such as Python which gives better visualisation and can be processed within RPi or on a ground-based system.

During simulation and preliminary tests, it was realised that inaccurate hardware parameters were causing unexpected results. This research explains different methods to extract parameters and compare them against each other. The best way to achieve the moment of inertia especially is by using the bifilar pendulum. It does not require the density of the material or cross-sectional area either. The approximation method is less reliable if the information is scarce. When it comes to motor parameter selection, the motor being used is a BLDC motor however the approximation made is for a DC motor. The internal wiring is significantly different to a normal DC motor. This may raise some issues while testing on a real quadcopter.

While testing the quadcopter it was noticed the motors were not spinning in sync. Upon investigation, it was realised the Atmega pins were running at different frequencies. This research describes a method to change the frequency of pins. This issue was only noticed while conducting a bench test. The only way to prevent this was by picking a different microcontroller for the application. However, Atmega328p was compact and computationally good enough for the purpose.

This research is primarily based on simulation. The limitations of this research were it the control system was never implemented in a real-life quadcopter. Several tests were carried out in-house in a cage setup. However, due to accidents and component failures, the quadcopter was never flown independently.

8.5 Novelty

In summary, the contribution of this research consists of the following,

- This research provides a comprehensive compendium that helps to understand coordinate frames and rotation sequences when selecting a frame of reference. The rotational matrix derived from a sequence of rotation has to be thoroughly understood especially for converting sensor data from the body frame to the inertial frame of a quadcopter. Pictorial representations make it simple and intuitive to follow this difficult concept.
- Visualisation in 3D helps troubleshoot and to evaluate the MATLAB/Simulink model. Any model can be rendered on 3D CAD software, whether it is a sophisticated quadcopter model or a polygon block representing a UAV. This research gives two methods to import STL or OBJ files and manipulate them on MATLAB to see the performance of the quadcopter model.
- Generally, researchers tend to take approximate values of hardware. This is mainly because measuring instruments such as LCR meters, thrust measurement units etc are expensive. In this research, some of this expensive equipment was used and the results were compared against model approximation and alternative methods to extract hardware parameters.
- Full control of each propeller is required for a robust quadcopter model. Atmega microcontroller's PWM timers frequencies are different for different pins. This research demonstrates how to manipulate timer registers to ensure all four motors are running at the correct frequency.

8.6 Future research recommendation

Over the years, control systems in quadcopter design have been developed to the point where they perform functions that were considered to lie in the realm of the impossible in the early days of aviation. There have been several methods considered to develop an efficient, robust quadcopter. This paper presents a solid platform for basic testing and evaluation purposes. A lot of improvements can be made to this developed model, starting with;

- a) Changing the DCM/Newton-Euler system to a quaternion-based model helps eliminate gimbal lock.
- b) Another layer of testing/simulation of the model can be implemented using CFD (Computational Fluid Dynamics) to design a stable control system as well as evaluate stress analysis (FEA – Finite Element Analysis).
- c) Use alternative control systems such as Model Predictive Controller (MPC), Linear Quadratic Regulator (LQR), Neural network etc.
- d) Similarly, noise filtering purposes on a non-linear system can be achieved using, Extended Kalman Filter (EKF), Unscented Kalman Filter (UKF) etc.
- e) Take advantage of the integrated camera for obstacle detection, lane following etc.
- f) Use TCP/IP or radiofrequency instead of UDP for a robust communication network.
- g) Implementing this in real hardware is achievable, especially with RPi.
- h) Implementing a verification framework in practice for stability monitoring of the UAVs
- i) Extending the verification framework by adding software verification to the procedures using theorem proving in addition to model-checking techniques
- j) Using the verification framework in other safety-critical applications such as autonomous cars, surgical robotics, field robotics, etc.
- k) Development of quadcopters in GPS-denied locations by relying on onboard sensors only for navigation and collision avoidance.
- l) Implementation of SLAM technology for GPS-denied location and obstacle avoidance.

8.7 Publications

Ajay K. Josaph, et.al “An Integration of 6DOF Quadcopter MATLAB/Simulink Controller Algorithm onto a PIXHAWK Autopilot”, The 10th International Conference on Power Electronics, Machines and Drives, PEMD, 15 - 17 December 2020 | Online Conference.

Ajay K. Joseph, et.al ” LQR controller design for quad-rotor helicopters” The Journal of Engineering, ISSN 2051-3305, doi: 10.1049/joe.2018.8126 , pp4003-4007, 17th June 2019.

A. K. Joseph, et.al "Design of Self-tuning PID Controller Parameters Using Fuzzy Logic Controller for Quad-rotor Helicopter" Published in International Journal of Trend in Research and Development (IJTRD), ISSN: 2394-9333, Vol. 3, Issue-6 , December 2016.

Ajay K Joseph, et.al “A Wireless communication system for a quadroter helicopter”, 2018 53rd International Universities Power Engineering Conference (UPEC), 4-7 Sept. 2018, Glasgow DOI: 10.1109/UPEC.2018.8542040.

Ajay K. Josaph, et.al “ Experimental Characterisation of quad rotor controller based on Kalman Filter”, 53rd International Universities Power Engineering Conference (UPEC), 4-7 Sept. 2018, Glasgow, DOI: 10.1109/UPEC.2018.8541858.

Reference

'1-0-1-k-j-astrom-pid-controllers-theory-design-and-tuning-2ed.pdf' (no date). Available at: <https://aiecp.files.wordpress.com/2012/07/1-0-1-k-j-astrom-pid-controllers-theory-design-and-tuning-2ed.pdf> (Accessed: 7 July 2021).

3D Builder user's manual (no date). Available at: <https://www.microsoft.com/en-us/3d-print/3d-builder-users-guide> (Accessed: 7 July 2021).

4-Wire Load Cell (1/5/10/20/200kg) with HX711 and Arduino - Circuit Journal (no date). Available at: <https://circuitjournal.com/four-wire-load-cell-with-HX711> (Accessed: 29 June 2021).

6: Euler Angle Definition - Rigid Body Kinematics I | Coursera (no date). Available at: <https://www.coursera.org/lecture/spacecraft-dynamics-kinematics/6-euler-angle-definition-b7yyv> (Accessed: 29 March 2021).

9: Euler Angle Differential Kinematic Equations - Rigid Body Kinematics I (no date) *Coursera*. Available at: <https://www.coursera.org/lecture/spacecraft-dynamics-kinematics/9-euler-angle-differential-kinematic-equations-4r6a8> (Accessed: 29 March 2021).

850Kv Motor AC2830-358 Build Your Own Drone (no date). Available at: <https://www.buildyourowndrone.co.uk/850kv-motor-ac2830-358> (Accessed: 29 June 2021).

'61806036.pdf' (no date). Available at: <https://core.ac.uk/download/pdf/61806036.pdf> (Accessed: 28 March 2021).

A Not-So-Short History of Unmanned Aerial Vehicles (UAV) (2020) Consortiq. Available at: <https://consortiq.com/short-history-unmanned-aerial-vehicles-uavs/> (Accessed: 7 August 2021).

A Simple Method to Measure Unknown Inductors (no date). Available at: <https://www.dos4ever.com/inductor/inductor.html> (Accessed: 14 July 2021).

Abdulkareem, A., Oguntosin, V., Popoola, O.M. and Idowu, A.A. (2022) 'Modeling and Nonlinear Control of a Quadcopter for Stabilization and Trajectory Tracking', *Journal of Engineering*, 2022, p. e2449901. Available at: <https://doi.org/10.1155/2022/2449901>.

admin (2017) *The History of WiFi: 1971 to Today, CableFree*. Available at: <https://www.cablefree.net/wireless-technology/history-of-wifi-technology/> (Accessed: 4 December 2022).

Ajay Joseph (2017) *RD07 MPhil*. University of Derby.

Al, A., Dewi, A.Y. and Hidayat, T. (2018) 'The lifting force relationship of a BLDC motor rotor system to the blade rotation speed', *MATEC Web of Conferences*. Edited by T. Baaken, P. van der Sijde, G. Maas, Zaidir, and M. Yahya, 215, p. 01013. Available at: <https://doi.org/10.1051/mateconf/201821501013>.

Alias/Wavefront OBJ File Format (no date). Available at: <https://people.cs.clemson.edu/~dhouse/courses/405/docs/brief-obj-file-format.html> (Accessed: 29 March 2021).

All About the New Wi-Fi 6 Standard - 802.11ax Explained (Tutorial) (no date) *Tech 21 Century*. Available at: <https://www.tech21century.com/802-11ax-wifi6-explained/> (Accessed: 4 December 2022).

'Anatomy-of-a-Drone0415.pdf' (no date). Available at: <https://rotordronepro.com/wp-content/uploads/2017/04/Anatomy-of-a-Drone0415.pdf> (Accessed: 28 March 2021).

Andriën, A.R.P., Antunes, D., Molengraft, M.J.G. van de and Heemels, W.P.M.H. (2018) 'Similarity-Based Adaptive Complementary Filter for IMU Fusion', in *2018 European Control Conference (ECC)*. *2018 European Control Conference (ECC)*, pp. 3044–3049. Available at: <https://doi.org/10.23919/ECC.2018.8550358>.

ArduCopter ESC 20 Amp for QuadCopter, HexaCopter and OctoCopter use. Build Your Own Drone (no date). Available at: <https://www.buildyourowndrone.co.uk/arducopter-esc-20-amp> (Accessed: 30 June 2021).

Ashour, R., Taha, T., Mohamed, F., Hableel, E., Kheil, Y., Elsalamouny, M., Kadadha, M., Rangan, K., Dias, J., Seneviratne, L. and Cai, G. (2016) 'Site inspection drone: A solution for inspecting and regulating construction sites', in, pp. 1–4. Available at: <https://doi.org/10.1109/MWSCAS.2016.7870116>.

'Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf' (no date). Available at: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf (Accessed: 29 March 2021).

Barbary, M. and ElAzeem, M.H.A. (2021) 'Drones tracking based on robust Cubature Kalman-TBD-Multi-Bernoulli filter', *ISA Transactions*, 114, pp. 277–290. Available at: <https://doi.org/10.1016/j.isatra.2020.12.042>.

Basset, P.-M., Tremolet, A. and Lefebvre, T. (2014) 'Rotary Wing UAV pre-sizing : Past and Present Methodological Approaches at Onera'.

Beck, H., Lesueur, J., Charland-Arcand, G., Akhrif, O., Gagné, S., Gagnon, F. and Couillard, D. (2016) 'Autonomous takeoff and landing of a quadcopter', in *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*. *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 475–484. Available at: <https://doi.org/10.1109/ICUAS.2016.7502614>.

Berna, A., Castillo, P., Sanahuja, G., González, F., García, P. and Albertos, P. (2011) 'Development of a test-bed to implement and validate real-time control strategies for aerial vehicles*', *IFAC Proceedings Volumes*, 44(1), pp. 7660–7665. Available at: <https://doi.org/10.3182/20110828-6-IT-1002.02743>.

Blazakis, J. (2006) 'Border Security and Unmanned Aerial Vehicles', *Connections*, 5(2), pp. 154–159.

Bolandi, H., Rezaei, M., Mohsenipour, R., Nemati, H. and Smailzadeh, S.M. (2013) 'Attitude Control of a Quadrotor with Optimized PID Controller', *Intelligent Control and Automation*, 4(3), pp. 335–342. Available at: <https://doi.org/10.4236/ica.2013.43039>.

Bouabdallah, S. (2007) *Design and control of quadrotors with application to autonomous flying*, *Infoscience*. EPFL. Available at: <https://doi.org/10.5075/epfl-thesis-3727>.

Bouabdallah, S. and Siegwart, R. (2007) 'Full control of a quadrotor', in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems. 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 153–158. Available at: <https://doi.org/10.1109/IROS.2007.4399042>.

Bowditch, N. (no date) '2002 BICENTENNIAL EDITION', p. 882.

Breguet-Richet Gyroplane No.1 helicopter - development history, photos, technical data (no date). Available at: http://www.aviastar.org/helicopters_eng/breguet_gyro.php (Accessed: 29 March 2021).

Brushless Vs. Brushed Motors (2015) *KDE Direct*. Available at: <https://www.kdedirect.com/blogs/news/brushless-vs-brushed-motors> (Accessed: 28 July 2022).

Burns, M. (1993) *Automated fabrication: improving productivity in manufacturing*. Englewood Cliffs, N.J: PTR Prentice Hall.

Capacitance and Inductance Measurements Using an Oscilloscope and a Function Generator (no date). Available at: <https://uk.tek.com/document/application-note/capacitance-and-inductance-measurements-using-oscilloscope-and-function-ge> (Accessed: 14 July 2021).

Carrillo, L.R.G., Rondon, E., Sanchez, A., Dzul, A. and Lozano, R. (2010) 'Position control of a quadrotor UAV using vision *', *IFAC Proceedings Volumes*, 43(15), pp. 31–36. Available at: <https://doi.org/10.3182/20100906-5-JP-2022.00007>.

Chen, G. (2016) 'Stability of Nonlinear Systems'.

Chief, E. in (2016) '5 Advantages and Disadvantages of Exponential Smoothing', *ConnectUS*, 16 October. Available at: <https://connectusfund.org/5-advantages-and-disadvantages-of-exponential-smoothing> (Accessed: 19 December 2022).

Coil Inductance, Measurement and Calculation.wmv (no date). Available at: <https://www.youtube.com/watch?v=Pv6zsQiBzXo> (Accessed: 14 July 2021).

Convert Euler angles to rotation matrix - MATLAB eul2rotm - MathWorks United Kingdom (no date). Available at: <https://uk.mathworks.com/help/robotics/ref/eul2rotm.html> (Accessed: 29 March 2021).

Convertawings Model 'A' Quadrotor at the Cradle of Aviation Museum (2020). Available at: https://www.cradleofaviation.org/history/exhibits/exhibit-galleries/the_jet_age/convertawings_model_a_quadrotor.html (Accessed: 29 March 2021).

Converting to Euler & Tait-Bryan (no date). Available at: <https://marc-b-reynolds.github.io/math/2017/04/18/TaitEuler.html> (Accessed: 29 March 2021).

Create a 6dof IMU with a gyro and accelerometer for (Arduino) multicopters. (no date). Available at: <http://www.brokking.net/imu.html> (Accessed: 31 July 2021).

Crumley, B. (2021) 'Mind control: UK firm develops drone brain piloting tech', *DroneDJ*, 10 November. Available at: <https://dronedj.com/2021/11/10/mind-control-uk-firm-develops-drone-brain-piloting-tech/> (Accessed: 7 December 2022).

Curtiss-Wright VZ-7 (2015) *Military Wiki*. Available at: https://military.wikia.org/wiki/Curtiss-Wright_VZ-7 (Accessed: 29 March 2021).

Czyż, Z., Karpiński, P., Skiba, K. and Wendeker, M. (2021) 'Wind Tunnel Performance Tests of the Propellers with Different Pitch for the Electric Propulsion System', *Sensors (Basel, Switzerland)*, 22(1), p. 2. Available at: <https://doi.org/10.3390/s22010002>.

DiSalle, R. (2020) 'Space and Time: Inertial Frames', in E.N. Zalta (ed.) *The Stanford Encyclopedia of Philosophy*. Winter 2020. Metaphysics Research Lab, Stanford University. Available at: <https://plato.stanford.edu/archives/win2020/entries/spacetime-iframes/> (Accessed: 29 March 2021).

DJI - Official Website (2016) *DJI Official*. Available at: <https://www.dji.com/uk> (Accessed: 4 December 2022).

Dong, W., Gu, G., Zhu, X. and Ding, H. (2013) 'Modeling and Control of a Quadrotor UAV with Aerodynamic Concepts', *undefined* [Preprint]. Available at: <https://www.semanticscholar.org/paper/Modeling-and-Control-of-a-Quadrotor-UAV-with-Dong-Gu/4e3e51351cb2a7a09de8d79c58f64ce07fcafaa7> (Accessed: 8 August 2021).

Dorf, R.C. and Bishop, R.H. (2010) *Modern control systems*. 12th ed. Prentice Hall: Pearson.

'download.pdf' (no date). Available at: <http://lup.lub.lu.se/luur/download?func=downloadFile&recordId=8847641&fileId=8859343> (Accessed: 30 June 2021).

Dr. Kostas Alexis - Dr. Kostas Alexis Personal Website (no date). Available at: <http://www.kostasalexis.com/> (Accessed: 12 April 2021).

'Drone Types: Multi-Rotor vs Fixed-Wing vs Single Rotor vs Hybrid VTOL' (2016) *AUAV*, 8 November. Available at: <https://www.auav.com.au/articles/drone-types/> (Accessed: 28 March 2021).

Etienne Oehmichen Biography - Tourism Châlons (2011) *Châlons en Champagne*. Available at: <https://www.chalons-tourisme.com/en/discover/celebrities/etienne-oehmichen/> (Accessed: 29 March 2021).

'Euler angles' (2021) *Wikipedia*. Available at: https://en.wikipedia.org/w/index.php?title=Euler_angles&oldid=1013072071 (Accessed: 29 March 2021).

Figure 1.3: A common convention for the angles yaw, pitch and roll. (no date) *ResearchGate*. Available at: https://www.researchgate.net/figure/A-common-convention-for-the-angles-yaw-pitch-and-roll_fig3_235412302 (Accessed: 17 October 2022).

Figure 5. Movement of quadcopter, body frame and inertial quadcopter... (no date) *ResearchGate*. Available at: https://www.researchgate.net/figure/Movement-of-quadcopter-body-frame-and-inertial-quadcopter-frame-3-Result-and-Discussion_fig3_335719947 (Accessed: 17 October 2022).

Finite-time Adaptive Integral Backstepping Fast Terminal Sliding Mode Control Application on Quadrotor UAV (2021) *springerprofessional.de*. Available at: <https://www.springerprofessional.de/en/finite-time-adaptive-integral-backstepping-fast-terminal-sliding/17210060> (Accessed: 18 October 2022).

float - Arduino Reference (no date). Available at: <https://www.arduino.cc/reference/en/language/variables/data-types/float/> (Accessed: 31 July 2021).

FORUM, D. (2015) *RC Vs WIFI*. Available at: [//forum.dji.com/thread-124974-1-1.html](http://forum.dji.com/thread-124974-1-1.html) (Accessed: 7 December 2022).

Free Image on Pixabay - Drone, Reverb, Racing, Quadcopter (no date). Available at: <https://pixabay.com/photos/drone-reverb-racing-quadcopter-3758182/> (Accessed: 28 March 2021).

Gandhi, D.A. and Ghosal, M. (2018) 'Novel Low Cost Quadcopter for Surveillance Application', in *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*. 2018 *International Conference on Inventive Research in Computing Applications (ICIRCA)*, pp. 412–414. Available at: <https://doi.org/10.1109/ICIRCA.2018.8597391>.

Guerrero Castellanos, F., Téllez-Guzmán, J.J., Durand, S., Marchand, N., Álvarez-Muñoz, J. and Gonzalez-Diaz, V.R. (2014) 'Attitude Stabilization of a Quadrotor by Means of Event-Triggered Nonlinear Control', *Journal of Intelligent and Robotic Systems*, 73, pp. 123–135. Available at: <https://doi.org/10.1007/s10846-013-9890-y>.

Gustafsson, J. (2009) *English: Illustration of a few shapes and their drag coefficients*. Available at: <https://commons.wikimedia.org/wiki/File:Drag.svg> (Accessed: 19 July 2021).

Haque, A. (2021) 'Autonomous Localization Of A Uav In A 3d Cad Model'.

Harrington, P. (2019) *Autonomous Drone Network: Non-Intrusive Control and Indoor Formation Positioning*. doctoral. Northumbria University. Available at: <https://nrl.northumbria.ac.uk/id/eprint/43394/> (Accessed: 4 December 2022).

Harris Corner Detection - Computer Vision (Python) (2019). Available at: <https://www.youtube.com/watch?v=0z3ci5lak4g> (Accessed: 6 September 2021).

He, Z. and Zhao, L. (2014) 'A Simple Attitude Control of Quadrotor Helicopter Based on Ziegler-Nichols Rules for Tuning PD Parameters', *The Scientific World Journal*, 2014, p. 280180. Available at: <https://doi.org/10.1155/2014/280180>.

'Heba_ElKholy_Thesis_S2014.pdf' (no date). Available at: http://dar.aucegypt.edu/bitstream/handle/10526/3965/Heba_ElKholy_Thesis_S2014.pdf?sequence=1 (Accessed: 12 April 2021).

Hemingway, E.G. and O'Reilly, O.M. (2018) 'Perspectives on Euler angle singularities, gimbal lock, and the orthogonality of applied forces and applied moments', *Multibody System Dynamics*, 44(1), pp. 31–56. Available at: <https://doi.org/10.1007/s11044-018-9620-0>.

Hetényi, D., Gótzty, M. and Blázovics, L. (2016) 'Sensor fusion with enhanced Kalman Filter for altitude control of quadrotors', in *2016 IEEE 11th International Symposium on Applied Computational Intelligence and Informatics (SACI)*. 2016 *IEEE 11th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, pp. 413–418. Available at: <https://doi.org/10.1109/SACI.2016.7507412>.

Hirvonen, V. (no date) 'Lagrangian vs Newtonian Mechanics: The Key Differences', *Profound Physics*. Available at: <https://profoundphysics.com/lagrangian-vs-newtonian-mechanics-the-key-differences/> (Accessed: 17 December 2022).

History of Microcontrollers | Toshiba Electronic Devices & Storage Corporation | Europe(EMEA) (2020). Available at: <https://toshiba.semicon-storage.com/eu/semiconductor/knowledge/e-learning/micro-intro/chapter2/history-microcontroller.html> (Accessed: 28 July 2022).

Hoffmann, G., Huang, H., Waslander, S. and Tomlin, C. (2022) 'Quadrotor Helicopter Flight Dynamics and Control: Theory and Experiment', in *AIAA Guidance, Navigation and Control Conference and Exhibit*. American Institute of Aeronautics and Astronautics. Available at: <https://doi.org/10.2514/6.2007-6461>.

Hoffmann, G., Rajnarayan, D.G., Waslander, S.L., Dostal, D., Jang, J.S. and Tomlin, C.J. (2004) 'The Stanford testbed of autonomous rotorcraft for multi agent control (STARMAC)', in *The 23rd Digital Avionics Systems Conference (IEEE Cat. No.04CH37576)*. The 23rd Digital Avionics Systems Conference, Salt Lake City, UT, USA: IEEE, p. 12.E.4-121-10. Available at: <https://doi.org/10.1109/DASC.2004.1390847>.

Home (no date). Available at: <http://yuneec.uk/> (Accessed: 4 December 2022).

Horak, K. and Zalud, L. (2015) 'Image Processing on Raspberry Pi for Mobile Robotics', in. Available at: <https://doi.org/10.13140/RG.2.1.1440.7281>.

Hou, Y., Lv, M., Liang, X. and Yang, A. (2022) 'Fuzzy adaptive fixed-time fault-tolerant attitude tracking control for tailless flying wing aircrafts', *Aerospace Science and Technology*, 130, p. 107950. Available at: <https://doi.org/10.1016/j.ast.2022.107950>.

Huang, M., Xian, B., Diao, C., Yang, K. and Feng, Y. (2010) 'Adaptive tracking control of underactuated quadrotor unmanned aerial vehicles via backstepping', in *Proceedings of the 2010 American Control Conference. Proceedings of the 2010 American Control Conference*, pp. 2076-2081. Available at: <https://doi.org/10.1109/ACC.2010.5531424>.

① *Encyclopedia - De Bothezat helicopter - rotary-engined aircraft. The de (2020) Blog*. Available at: <https://amp.blog.shops-net.com/27296301/1/de-bothezat-helicopter.html> (Accessed: 29 March 2021).

'IMU Attitude Estimation – Philip Salmony' (no date). Available at: <http://philsal.co.uk/projects/imu-attitude-estimation> (Accessed: 31 July 2021).

Internet Growth Statistics 1995 to 2022 - the Global Village Online (2022a). Available at: <https://www.internetworldstats.com/emarketing.htm#top> (Accessed: 4 December 2022).

Internet Growth Statistics 1995 to 2022 - the Global Village Online (2022b). Available at: <https://www.internetworldstats.com/emarketing.htm#top> (Accessed: 7 December 2022).

Islam, S., Faraz, M., Ashour, R.K., Cai, G., Dias, J. and Seneviratne, L. (2015) 'Adaptive sliding mode control design for quadrotor unmanned aerial vehicle', in *2015 international conference on unmanned aircraft systems (ICUAS)*. IEEE, pp. 34-39.

Jansen, J. (no date) 'Autonomous Localization and Tracking for UAVs using Kalman Filtering', p. 97.

Javir, P.A.V., Pawar, K., Dhudum, S., Patale, N. and Patil, S. (2015) 'Design, Analysis and Fabrication of Quadcopter', *Journal of Advance Research in Mechanical & Civil Engineering (ISSN: 2208-2379)*, 2(3), pp. 16-27.

Johnson, W. (2013) *Rotorcraft Aeromechanics*. Cambridge: Cambridge University Press (Cambridge Aerospace Series). Available at: <https://doi.org/10.1017/CBO9781139235655>.

KDE Direct - Measuring Motor Inductance (no date). Available at: https://www.youtube.com/watch?v=Fyv4_nKbGZI (Accessed: 14 July 2021).

Kokaj, A., Bula, I. and Dermaku, A. (2018) 'Fast Fourier Transform and a Complimentary Filter Based Control of a Robotic System', *IFAC-PapersOnLine*, 51(30), pp. 561–564. Available at: <https://doi.org/10.1016/j.ifacol.2018.11.250>.

Kroo, I., Prinz, F., Shantz, M., Kunz, P., Fay, G., Cheng, S., Fabian, T. and Partridge, C. (2011) 'The Mesicopter: A Miniature Rotorcraft Concept Phase II Final Report', p. 138.

Kumar, R., Bhargavapuri, M., Deshpande, A.M., Sridhar, S., Cohen, K. and Kumar, M. (2020) 'Quaternion Feedback Based Autonomous Control of a Quadcopter UAV with Thrust Vectoring Rotors', in *2020 American Control Conference (ACC). 2020 American Control Conference (ACC)*, pp. 3828–3833. Available at: <https://doi.org/10.23919/ACC45564.2020.9147794>.

Kyuchukova, D., Hristov, G.V., Raychev, J. and Захариев, П. (2019) 'Early Forest Fire Detection Using Drones and Artificial Intelligence', in, pp. 1060–1065. Available at: <https://doi.org/10.23919/MIPRO.2019.8756696>.

LaViola, J.J. (2003) 'Double exponential smoothing: an alternative to Kalman filter-based predictive tracking', in *Proceedings of the workshop on Virtual environments 2003 - EGVE '03. the workshop*, Zurich, Switzerland: ACM Press, pp. 199–206. Available at: <https://doi.org/10.1145/769953.769976>.

Li, S. and Sun, Z. (2022) 'A generalized proportional integral observer–based robust tracking design approach for quadrotor unmanned aerial vehicle', *International Journal of Advanced Robotic Systems*, 19(4), p. 17298806221117052. Available at: <https://doi.org/10.1177/17298806221117052>.

List of Unmanned Aircraft (no date). Available at: <http://publicapps.caa.co.uk/modalapplication.aspx?catid=1&pagetype=65&appid=11&mode=list&type=sercat&id=85&filter=2&reorder=dateOrder> (Accessed: 28 March 2021).

Loram, I.D., Gawthrop, P.J. and Lakie, M. (2006) 'The frequency of human, manual adjustments in balancing an inverted pendulum is constrained by intrinsic physiological factors', *The Journal of Physiology*, 577(Pt 1), pp. 417–432. Available at: <https://doi.org/10.1113/jphysiol.2006.118786>.

Loughran, J. (2020) *Drones to be trialled for search and rescue operations at sea*. Available at: <https://eandt.theiet.org/content/articles/2020/02/drones-to-be-trialled-for-search-and-rescue-operations-at-sea/> (Accessed: 28 March 2021).

Luukkonen, T. (no date) 'Modelling and control of quadcopter', p. 26.

M, A. (2019) 'How a group of drone operators in Kerala rose to the occasion during floods', *The Hindu*, 14 August. Available at: <https://www.thehindu.com/news/national/kerala/how-a-group-of-drone-operators-rose-to-the-occasion-during-floods-and-landslips-in-kerala/article29091339.ece> (Accessed: 28 March 2021).

Macintyre, A.G., Barbera, J.A. and Smith, E.R. (2006) 'Surviving collapsed structure entrapment after earthquakes: a "time-to-rescue" analysis', *Prehospital and Disaster Medicine*, 21(1), pp. 4–17; discussion 18–19. Available at: <https://doi.org/10.1017/s1049023x00003253>.

Magnussen, Ø., Ottestad, M. and Hovland, G. (2013) 'Experimental validation of a quaternion-based attitude estimation with direct input to a quadcopter control system', in *2013 International Conference on Unmanned Aircraft Systems (ICUAS). 2013 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 480–485. Available at: <https://doi.org/10.1109/ICUAS.2013.6564723>.

Martinetti, A., Margaryan, M. and Dongen, L. van (2018) 'Simulating mechanical stress on a micro Unmanned Aerial Vehicle (UAV) body frame for selecting maintenance actions', *Procedia Manufacturing*, 16, pp. 61–66. Available at: <https://doi.org/10.1016/j.promfg.2018.10.160>.

Mass Moment of Inertia (no date). Available at: https://www.engineeringtoolbox.com/moment-inertia-torque-d_913.html (Accessed: 17 October 2022).

Mayank (2011) 'I/O Port Operations in AVR', *maxEmbedded*, 9 June. Available at: <https://maxembedded.wordpress.com/2011/06/10/port-operations-in-avr/> (Accessed: 29 March 2021).

McBride, J. (no date) 'Flight Control System for Small High-Performance UAVs', p. 73.

MEMS (2017) *ETHW*. Available at: <https://ethw.org/MEMS> (Accessed: 28 July 2022).

Milionis, G. (no date) 'A FRAMEWORK FOR COLLABORATIVE QUADROTOR – GROUND ROBOT MISSIONS', p. 174.

Model, Simulate and Control a Drone in MATLAB & SIMULINK (2020) *Udemy*. Available at: <https://www.udemy.com/course/quadcopter-drone-dji-mavic-matlab-simulink/> (Accessed: 29 March 2021).

'Modeling Vehicle Dynamics - Quadcopter Equations of Motion' (2017) *Autonomy in Motion*, 7 June. Available at: <https://charlestytler.com/quadcopter-equations-motion/> (Accessed: 17 October 2022).

'Modelling, Identification and Control - Tommaso Bresciani.pdf' (no date).

Mogili, U.R. and Deepak, B.B.V.L. (2018) 'Review on Application of Drone Systems in Precision Agriculture', *Procedia Computer Science*, 133, pp. 502–509. Available at: <https://doi.org/10.1016/j.procs.2018.07.063>.

MSFC, J.W. (2015) *What Is a Helicopter?*, NASA. Brian Dunbar. Available at: <http://www.nasa.gov/audience/forstudents/5-8/features/nasa-knows/what-is-a-helicopter-58.html> (Accessed: 28 March 2021).

Naidoo, Y., Stopforth, R. and Bright, G. (2011) 'Quad-Rotor Unmanned Aerial Vehicle Helicopter Modelling & Control', *International Journal of Advanced Robotic Systems*, 8(4), p. 45. Available at: <https://doi.org/10.5772/45710>.

New drone technology unveiled, your mind can now instruct a drone to fly | WION (2021). Available at: <https://www.youtube.com/watch?v=RjAE92EC9cA> (Accessed: 7 December 2022).

Nirina, A.A., Auguste, R.A. and Auguste, R.P. (2022) 'MATHEMATICAL MODELLING OF A QUADROTOR', 8(1).

Oktay, T. and Eraslan, Y. (2020) 'Computational Fluid Dynamics (CFD) Investigation of a Quadrotor UAV Propeller', in.

Ortega, A., Muñoz, J., McGee, M., Choudhuri, A. and Flores Abad, A. (2020) 'Drone Inspection Flight Path Generation from 3D CAD Models: Power Plant Boiler Case Study', in. Available at: <https://doi.org/10.2514/6.2020-1091>.

Palmer, A. (2020) *Amazon wins FAA approval for Prime Air drone delivery fleet*, CNBC. Available at: <https://www.cnn.com/2020/08/31/amazon-prime-now-drone-delivery-fleet-gets-faa-approval.html> (Accessed: 28 March 2021).

Parrot (2016). Available at: <https://www.parrot.com/uk/drones/parrot-ardrone-20-elite-edition> (Accessed: 4 December 2022).

Petovello, M.G., Cannon, M.E. and Lachapelle, G. (2003) 'Kalman Filter Reliability Analysis Using Different Update Strategies'.

Powers, C., Mellinger, D. and Kumar, V. (2015) 'Quadrotor Kinematics and Dynamics', in K.P. Valavanis and G.J. Vachtsevanos (eds) *Handbook of Unmanned Aerial Vehicles*. Dordrecht: Springer Netherlands, pp. 307–328. Available at: https://doi.org/10.1007/978-90-481-9707-1_71.

Poyi, G.T. (2014) 'A novel approach to the control of quad-rotor helicopters using fuzzy-neural networks'. Available at: <https://derby.openrepository.com/handle/10545/337911> (Accessed: 28 March 2021).

Pretorius, A. and Boje, E. (2014a) 'Design and Modelling of a Quadrotor Helicopter with Variable Pitch Rotors for Aggressive Manoeuvres', *IFAC Proceedings Volumes*, 47(3), pp. 12208–12213. Available at: <https://doi.org/10.3182/20140824-6-ZA-1003.01586>.

Pretorius, A. and Boje, E. (2014b) 'Design and Modelling of a Quadrotor Helicopter with Variable Pitch Rotors for Aggressive Manoeuvres', *IFAC Proceedings Volumes*, 47(3), pp. 12208–12213. Available at: <https://doi.org/10.3182/20140824-6-ZA-1003.01586>.

'Quadcopter Dynamics, Simulation, and Control.pdf' (no date). Available at: <https://andrew.gibiansky.com/downloads/pdf/Quadcopter%20Dynamics,%20Simulation,%20and%20Control.pdf> (Accessed: 7 July 2021).

readObj - File Exchange - MATLAB Central (no date). Available at: <https://www.mathworks.com/matlabcentral/fileexchange/18957-readobj> (Accessed: 29 March 2021).

Ruwaimana, M., Satyanarayana, B., Otero, V., Muslim, A.M., A, M.S., Ibrahim, S., Raymaekers, D., Koedam, N. and Dahdouh-Guebas, F. (2018) 'The advantages of using drones over space-borne imagery in the mapping of mangrove forests', *PLOS ONE*, 13(7), p. e0200288. Available at: <https://doi.org/10.1371/journal.pone.0200288>.

Sanahuja, G., Castillo, P. and Sanchez, A. (2010) 'Stabilization of n integrators in cascade with bounded input with experimental application to a VTOL laboratory system: INTEGRATORS IN CASCADE WITH BOUNDED INPUT', *International Journal of Robust and Nonlinear Control*, 20(10), pp. 1129–1139. Available at: <https://doi.org/10.1002/rnc.1494>.

sandeep jagdevFollow (no date) *Changing Baud Rate of HC-05 (Bluetooth)*, *Instructables*. Available at: <https://www.instructables.com/Changing-Baud-Rate-of-HC-05-Bluetooth/> (Accessed: 29 March 2021).

Satici, A., Poonawala, H. and Spong, M.W. (2013) 'Robust Optimal Control of Quadrotor UAVs', *IEEE Access*, 1, pp. 79–93. Available at: <https://doi.org/10.1109/ACCESS.2013.2260794>.

Sebesta, K.D. and Boizot, N. (2014) 'A Real-Time Adaptive High-Gain EKF, Applied to a Quadcopter Inertial Navigation System', *IEEE Transactions on Industrial Electronics*, 61(1), pp. 495–503. Available at: <https://doi.org/10.1109/TIE.2013.2253063>.

Şenkul, F. and Altuğ, E. (2013) 'Modeling and control of a novel tilt — Roll rotor quadrotor UAV', in *2013 International Conference on Unmanned Aircraft Systems (ICUAS). 2013 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 1071–1076. Available at: <https://doi.org/10.1109/ICUAS.2013.6564796>.

Sense HAT - MATLAB & Simulink - MathWorks United Kingdom (2021). Available at: <https://uk.mathworks.com/help/supportpkg/raspberrypiio/sense-hat.html> (Accessed: 23 December 2022).

Session 17 - Sobel Edge Detector - A Quick Understanding (2020). Available at: https://www.youtube.com/watch?v=H4kKKU2_tJM (Accessed: 6 September 2021).

Shaqura, M. and Shamma, J.S. (2017) 'An Automated Quadcopter CAD based Design and Modeling Platform using Solidworks API and Smart Dynamic Assembly', in *Proceedings of the 14th International Conference on Informatics in Control, Automation and Robotics. 14th International Conference on Informatics in Control, Automation and Robotics*, Madrid, Spain: SCITEPRESS - Science and Technology Publications, pp. 122–131. Available at: <https://doi.org/10.5220/0006438601220131>.

Sigalos, A., Papoutsidakis, M., Chatzopoulos, A. and Piromalis, D. (2019) 'DESIGN OF A FLIGHT CONTROLLER AND PERIPHERALS FOR A QUADCOPTER', *International Journal of Engineering Applied Sciences and Technology*, 4, pp. 463–470. Available at: <https://doi.org/10.33564/IJEAST.2019.v04i05.067>.

Skydio 2+™ and X2™ – Skydio Inc. (no date). Available at: <https://www.skydio.com//> (Accessed: 4 December 2022).

Srinivasan K, V. (2016) *Implementation of an unmanned aerial vehicle for new generation Peterbilt trucks*. Available at: <http://adsabs.harvard.edu/abs/2016Mst.....83S> (Accessed: 12 April 2021).

'Static Thrust Calculation' (2011) *quadcopterproject*, 12 September. Available at: <https://quadcopterproject.wordpress.com/static-thrust-calculation/> (Accessed: 30 June 2021).

Stevens, B.L. and Lewis, F.L. (2003) *Aircraft control and simulation*. 2nd ed. Hoboken, N.J: J. Wiley.

STL File Reader (no date). Available at: <https://www.mathworks.com/matlabcentral/fileexchange/22409-stl-file-reader> (Accessed: 29 March 2021).

STL (STereoLithography) File Format, ASCII (2019). Available at: <https://www.loc.gov/preservation/digital/formats/fdd/fdd000506.shtml> (Accessed: 29 March 2021).

Šustek, M. and Úředníček, Z. (2018) 'The Basics of Quadcopter Anatomy', *MATEC Web of Conferences*, 210, p. 01001. Available at: <https://doi.org/10.1051/matecconf/201821001001>.

test1 (no date). Available at: [http://ffden-2.phys.uaf.edu/webproj/211_fall_2018/J-Rod_Maltos/physics_4th.html#:~:text=They%20create%20lift%20by%20rotating,quadcopter%20dro ne%20in%20the%20air.&text=While%20motors%202%20and%204,a%20body%20should%20be%20z ero](http://ffden-2.phys.uaf.edu/webproj/211_fall_2018/J-Rod_Maltos/physics_4th.html#:~:text=They%20create%20lift%20by%20rotating,quadcopter%20dro ne%20in%20the%20air.&text=While%20motors%202%20and%204,a%20body%20should%20be%20z ero.). (Accessed: 29 March 2021).

The Economist (2017) 'A brief history of Wi-Fi', 22 November. Available at: <https://www.economist.com/technology-quarterly/2004/06/12/a-brief-history-of-wi-fi> (Accessed: 4 December 2022).

The Importance of 3D Visualisation In The Aerospace Industry (2017) *Dean Group International*. Available at: <https://www.deangroup-int.co.uk/the-importance-of-3d-visualisation-in-the-aerospace-industry/> (Accessed: 29 March 2021).

The STL Format | *fabbers.com* (no date). Available at: https://www.fabbers.com/tech/STL_Format#:~:text=Each%20facet%20is%20uniquely%20identified,numbers%20stored%20for%20each%20facet (Accessed: 29 March 2021).

'Transfer function Definition - Advantages & Disadvantages | Electricalvoice' (2017), 15 December. Available at: <https://electricalvoice.com/transfer-function-definition-advantages-disadvantages/> (Accessed: 19 December 2022).

UK Drone Laws: An In-depth Guide (no date a) *heliguy™*. Available at: <https://www.heliguy.com/blogs/posts/uk-drone-laws-where-can-i-fly> (Accessed: 28 March 2021).

UK Drone Laws: An In-depth Guide (no date b) *heliguy™*. Available at: <https://www.heliguy.com/blogs/posts/uk-drone-laws-where-can-i-fly> (Accessed: 28 March 2021).

Understanding Euler Angles | *CH Robotics* (no date). Available at: <http://www.chrobotics.com/library/understanding-euler-angles> (Accessed: 29 March 2021).

Valenti, M., Bethke, B., Fiore, G., How, J. and Feron, E. (2006) 'Indoor Multi-Vehicle Flight Testbed for Fault Detection, Isolation, and Recovery', in *AIAA Guidance, Navigation, and Control Conference and Exhibit. AIAA Guidance, Navigation, and Control Conference and Exhibit*, Keystone, Colorado: American Institute of Aeronautics and Astronautics. Available at: <https://doi.org/10.2514/6.2006-6200>.

Veroustraete, F. (2015) 'The Rise of the Drones in Agriculture', p. 4.

'Victor Hayes | IEEE Computer Society' (no date). Available at: <https://www.computer.org/profiles/victor-hayes> (Accessed: 4 December 2022).

Waslander, S.L., Hoffmann, G.M., Jung Soon Jang and Tomlin, C.J. (2005) 'Multi-agent quadrotor testbed control design: integral sliding mode vs. reinforcement learning', in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems. 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Edmonton, Alta., Canada: IEEE, pp. 3712–3717. Available at: <https://doi.org/10.1109/IROS.2005.1545025>.

Watterson, W. (2016) *WiFi and the coming IoT invasion*, *TNW | Insider*. Available at: <https://thenextweb.com/news/wifi-coming-iot-invasion> (Accessed: 4 December 2022).

Wavefront OBJ toolbox - File Exchange - MATLAB Central (no date). Available at: <http://uk.mathworks.com/matlabcentral/fileexchange/27982-wavefront-obj-toolbox> (Accessed: 7 May 2017).

What are the requirements under the subcategories of the 'open' category? (no date a) *EASA*. Available at: <https://www.easa.europa.eu/faq/116452> (Accessed: 28 March 2021).

What are the requirements under the subcategories of the 'open' category? (no date b) EASA. Available at: <https://www.easa.europa.eu/faq/116452> (Accessed: 28 March 2021).

What is Kinematics? Definition, Formula, Derivation, Sample Problems - GeeksforGeeks (no date). Available at: <https://www.geeksforgeeks.org/what-is-kinematics-definition-formula-derivation-sample-problems/> (Accessed: 17 October 2022).

Wilson, T.G. and Trickey, P.H. (1962) 'D-C machine with solid-state commutation', *Electrical Engineering*, 81(11), pp. 879–884. Available at: <https://doi.org/10.1109/EE.1962.6446586>.

Yıldırım, Ş., Çabuk, N. and Bakırcıoğlu, V. (2019) 'Design and trajectory control of universal drone system', *Measurement*, 147, p. 106834. Available at: <https://doi.org/10.1016/j.measurement.2019.07.062>.

Yumpu.com (no date) *Cartesian, Cylindrical Polar, and Spherical Polar Coordinates*, *yumpu.com*. Available at: <https://www.yumpu.com/en/document/read/35671901/cartesian-cylindrical-polar-and-spherical-polar-coordinates> (Accessed: 17 October 2022).

Zhang, J., Ren, Z., Deng, C. and Wen, B. (2019) 'Adaptive fuzzy global sliding mode control for trajectory tracking of quadrotor UAVs', *Nonlinear Dynamics*, 97(1), pp. 609–627. Available at: <https://doi.org/10.1007/s11071-019-05002-9>.

(No date a). Available at: <https://www.airforce-technology.com/projects/pc-1-multipurpose-quadcopter/:~:text=Capable%20of%20carrying%20a%20variety,and%20search-and-rescue.> (Accessed: 28 March 2021).

(No date b). Available at: <https://www.airforce-technology.com/projects/pc-1-multipurpose-quadcopter/:~:text=Capable%20of%20carrying%20a%20variety,and%20search-and-rescue.> (Accessed: 28 March 2021).

APPENDIX A - Civil Aviation Authority Rules and Regulations

Civil Aviation Authority



List of Unmanned Aircraft

The table below details the CAA Publication results for your search. For further details, please click the "Title" of the publication you are interested in.

[Search Again](#)

Filter: ☐ All ☒ Current and Future ☐ Cancelled, Superseded, Withdrawn and Reference Only
Order By: ☐ Reference ☒ Date

[Refresh List](#)

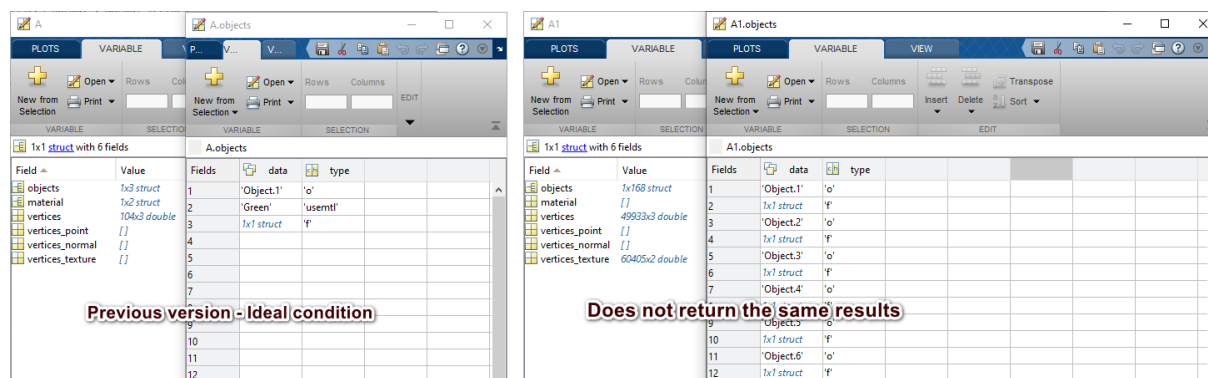
Reference	Title	Status	as at Date
CAP1789A	The UAS Implementing Regulation; UK consolidated text	Current	8 January 2021
CAP1789B	The UAS Delegated Regulation: UK consolidated text	Current	8 January 2021
CAP1974	Drone and Model Aircraft Registration and Education System (DMARES) Enforcement Guidance	Current	22 December 2020
CAP1361	Small Unmanned Aircraft (SUA) operators holding a valid CAA permission	Current	18 December 2020
CAP2013	Air Navigation Order 2020 Amendment – Guidance for unmanned aircraft system users	Current	17 December 2020
CAP2003	Drone Rules: FLYING FOR FUN	Current	16 December 2020
CAP2004	Drone Rules: FLYING AS A HOBBY AND AT A CLUB	Current	16 December 2020
CAP2005	Drone Rules: USING A DRONE FOR WORK	Current	16 December 2020
CAP2006	Drone Rules: FLYING IN THE COUNTRYSIDE	Current	16 December 2020
CAP2007	Drone Rules: FLYING IN TOWNS AND CITIES (OR BUSY AREAS)	Current	16 December 2020
CAP2008	Drone Rules: THE NEW UAS REGULATIONS	Current	16 December 2020
CAP2012	Drone Rules: REQUIREMENTS FOR FLYING IN THE OPEN CATEGORY	Current	16 December 2020
CAP722C	UAS Airspace Restrictions Guidance and Policy	Current	10 December 2020
CAP 722	Unmanned Aircraft System Operations in UK Airspace - Guidance	Current	5 November 2020
CAP1804	Charge Proposal for the UK Drone and Model Aircraft Registration and Education Scheme Consultation - CAA Response Document	Current	7 November 2019
CAP1827	Beyond Visual Line of Sight (BVLOS) operations of unmanned aircraft systems (UAS) in unsegregated airspace: Sandbox brief	Current	1 August 2019
CAP1827A	Sandbox Brief: Memorandum of Understanding	Current	30 July 2019
CAP1827B	Sandbox Brief: NDA	Current	30 July 2019
SRG1320	Application for Exemption or Permission for Operation of Unmanned Aircraft Systems (UAS) In UK Airspace	Current	June 2018
SRG1321	Application for Operation of an Unmanned Aircraft (UA) over 20kg in UK Airspace	Current	25 August 2016

APPENDIX B – Read Wavefront OBJ File

The Wavefront OBJ toolbox provided by Dirk-Jan Kroon `read_wobj()` should return vertices and faces to be manipulated.

Dirk-Jan Kroon (2021). Wavefront OBJ toolbox (<https://www.mathworks.com/matlabcentral/fileexchange/27982-wavefront-obj-toolbox>), MATLAB Central File Exchange. Retrieved 25th January, 2021.

However, the example provided by MATLAB should return vertices and faces as shown in Figure 11-1. The example shows by calling 3rd structure from `A.Objects` variable should return `Object.1`, 'Colour' and 1x1 struct of "faces".



```
% Example,  
% OBJ=read_wobj('examples\example10.obj');  
% FV.vertices=OBJ.vertices;  
% FV.faces=OBJ.objects(3).data.vertices;  
% figure, patch(FV,'facecolor',[1 0 0]); camlight
```

Alternative to `read_wobj()` is `readobj` by Bernard Abayowa – last update 28th February 2008.

Bernard Abayowa (2021). `readObj` (<https://www.mathworks.com/matlabcentral/fileexchange/18957-readobj>), MATLAB Central File Exchange. Retrieved 25th January, 2021.

`Readobj()` file is significantly slower than `read_wobj()`, however, to overcome this issue the CAD file is loaded as a .mat format to MATLAB's current folder. In this case it is saved as `A2.mat`.

```
load (A2.mat)
Vert = A2.v;
Fac = A2.f.v;
```

After saving as A2.mat use the command shown above to access CAD file's vertices and faces.

The `stlread()` is provided by Eric Johnson was last updated on 20th July 2011.

Eric Johnson (2021). STL File Reader (<https://www.mathworks.com/matlabcentral/fileexchange/22409-stl-file-reader>), MATLAB Central File Exchange. Retrieved 25th January 2021.

```
>> tic;ParrotSTLMambo = stlread('ParrotMambo.stl');toc
Elapsed time is 0.150284 seconds.
>> ParrotSTLMambo

ParrotSTLMambo =

    triangulation with properties:

        Points: [49933x3 double]
    ConnectivityList: [99980x3 double]

>> tic;ParrotSTLMambo = stlreadAJ('ParrotMambo.stl');toc
Elapsed time is 0.850920 seconds.
>> ParrotSTLMambo

ParrotSTLMambo =

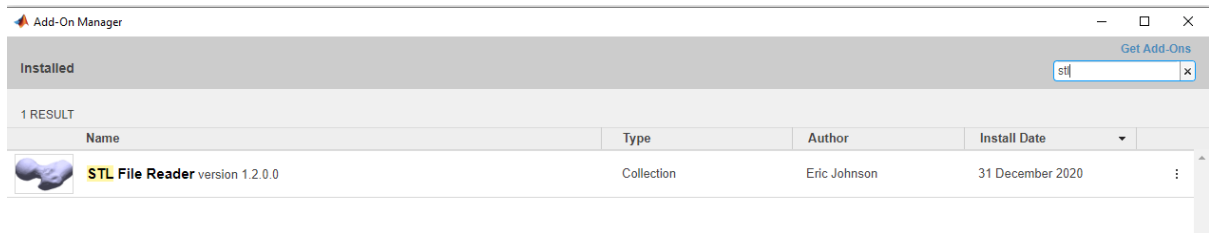
    struct with fields:

        faces: [99980x3 double]
    vertices: [299940x3 double]
```

The error was it returns Points and ConnectivityList rather than vertices and faces as provided in the example file.

```
% STLREAD imports geometry from an STL file into MATLAB.
% FV = STLREAD(FILENAME) imports triangular faces from the ASCII or binary
% STL file indicated by FILENAME, and returns the patch struct FV, with fields
% 'faces' and 'vertices'.
%
% [F,V] = STLREAD(FILENAME) returns the faces F and vertices V separately.
%
% [F,V,N] = STLREAD(FILENAME) also returns the face normal vectors.
%
% The faces and vertices are arranged in the format used by the PATCH plot
% object.
```

Also, the only STL toolbox package installed is the same mentioned above. The `stlread()` file was renamed to overcome this issue.



Steps to consider

Ideally Simulink model, STL/OBJ and M files should be in the same folder

Three parts of M file

Compressing data – Data is compressed by taking every 100th sample

Collecting data – M file named Main.m

DCM and plotting process – AnimationQuad.m

Transferring *To Workspace* and saving as *Array* format can be seen in Figure 0-1 below.

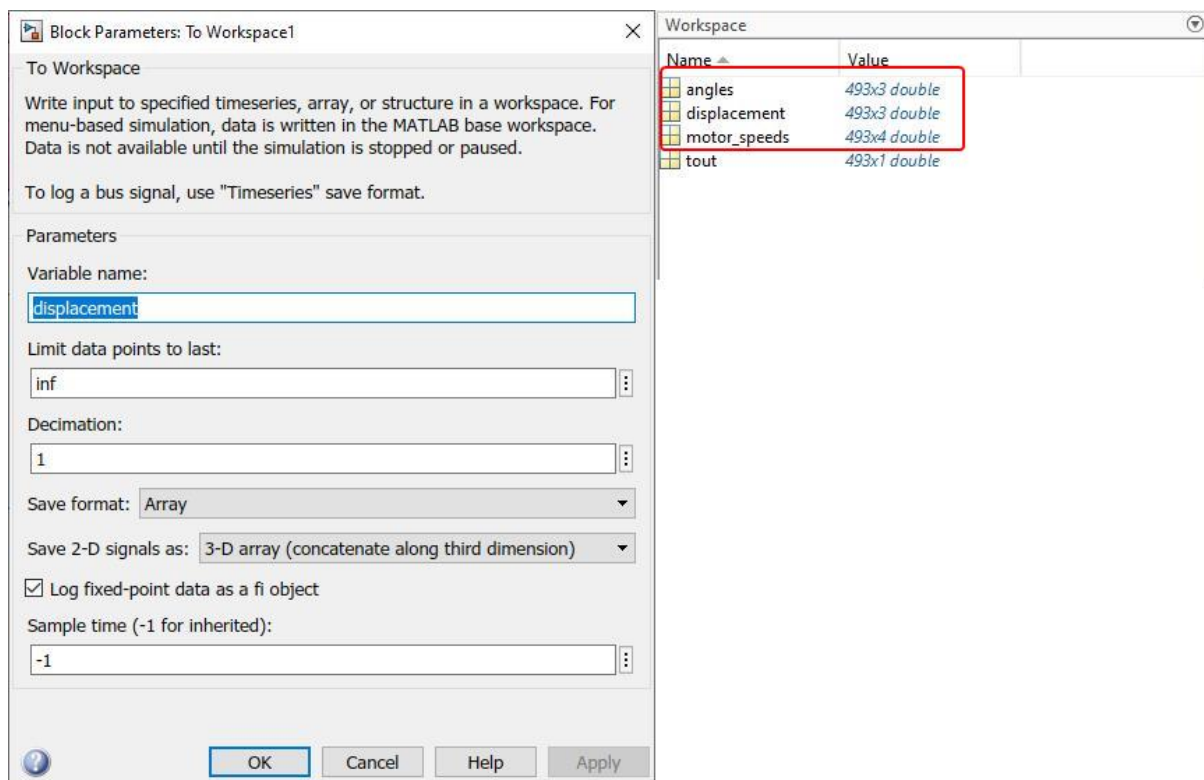


Figure 0-1 - Saving to the Array format

The '*viewang*' is the viewing angle, taken as the input to MATLAB's *view ()* command as shown in Figure 7-4 below.

```

14 %% Gathering data from workspace variables
15 - x = displacement(:,1);
16 - y = displacement(:,2);
17 - z = displacement(:,3);
18
19 - pitch = angles(:,1);
20 - yaw = angles(:,2);
21 - roll = angles(:,3);
22
23 - scale_factor = 1; %Scale of quadrotor = 1:1
24 - step = 2; %Step size for the plot|
25
26 - viewang = [120 20]; %View angle for video
27

```

Figure 0-2 - Setting up viewing angle and step size

The *stlread()* file was renamed to overcome this error, as shown in Figure 0-3.

```

>> tic;ParrotSTLMambo = stlreadAJ('ParrotMambo.stl');toc
Elapsed time is 0.850920 seconds.
>> ParrotSTLMambo

ParrotSTLMambo =

    struct with fields:

        faces: [99980x3 double]
        vertices: [299940x3 double]

```

Figure 0-3 - *stlreadAJ()* renamed

Displacement and angle for loop.

```

for i=1:step:(ii-resto)
    if mod(i,1) == 1
        clf;
        %plot3(x,y,z);
        plot3(-1.5,0,0); hold on;
        plot3(1.5,0,0); hold on;
        set(gca,'XDir','reverse') ;
        xlabel('X [m]');
        ylabel('Y [m]');
        zlabel('Z [m]');
        grid on;
        hold on;
        light;
    end

```

Figure 0-4: For loop for displacement and angle

Note after compressing data, run main file to execute AnimationQuad.m function for visualisation.

```
1. % Compress data – Save as m.file
2. int_i = 1;
3. int_i2 = 1;
4. anglesComp = zeros(length(angles), 3);
5. dispComp = zeros(length(displacement), 3);
6. iter = (length(angles) - 1)/100;
7. for i = 1:iter
8.     if i == 1
9.         anglesComp(int_i, 1) = angles(i,1);
10.        anglesComp(int_i, 2) = angles(i,2);
11.        anglesComp(int_i, 3) = angles(i,3);
12.
13.        dispComp(int_i, 1) = displacement(i, 1);
14.        dispComp(int_i, 2) = displacement(i, 2);
15.        dispComp(int_i, 3) = displacement(i, 3);
16.    else
17.        i = 100*i;
18.        int_i2 = 100*i;
19.        if i < (length(angles) - 1)
20.            anglesComp(int_i, 1) = angles(int_i2,1);
21.            anglesComp(int_i, 2) = angles(int_i2,2);
22.            anglesComp(int_i, 3) = angles(int_i2,3);
23.
24.            dispComp(int_i, 1) = displacement(int_i2, 1);
25.            dispComp(int_i, 2) = displacement(int_i2, 2);
26.            dispComp(int_i, 3) = displacement(int_i2, 3);
27.        end
28.    end
29.    int_i = int_i + 1;
```

```
30. % i
31. % int_i
32. end
33.
```

	Quad	Trajectory
--	------	------------

```

1. %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2.
3. %                               clc;
4. %                               clear all;
5. %                               close all;
6. %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
7. %testing svn
8. % Load simulink model using sim() function
9.
10. % sim('RD07'); % Name of the file between "
11. % load_system('OldOpenLoop2021'); 'New_SemiFinal_Quad_Model_19A'
12. tic
13. sim(""); % desired simulink model with .slx extension
14.
15. % Gather data from workspace variable
16. % x = displacement(:,1); %(:,1) represents all rows and only 1st coloum
17. % y = displacement(:,2); %(:,2) represents all rows and only 2nd coloum
18. % z = displacement(:,3);
19. %
20. % pitch  = angles(:,1);
21. % yaw    = angles(:,2);
22. % roll   = angles(:,3);
23.
24.
25.
26. x = dispComp(:,1); %(:,1) represents all rows and only 1st coloum
27. y = dispComp(:,2); %(:,2) represents all rows and only 2nd coloum
28. z = dispComp(:,3);
29.
30. pitch  = anglesComp(:,1);
31. yaw    = anglesComp(:,2);

```

```
32. roll = anglesComp(:,3);
33.
34.
35. scale_factor = 1; % Scale of quadrotor 1:1
36. step = 1; % Step size for the plot
37.
38. viewang = [120 20] % View angle for the video
39. % close_system('OldOpenLoop2021');
40.
41. % Plot quadrotor trajectory from displacement and angle values
42. M = AnimationQuad(x, y, z, pitch, roll, yaw, scale_factor, step, viewang);
43. % t;
44.
```



```

%
% % Load simulink model using sim() function
%
% sim('RD07'); % Name of the file between "
%
% mov = 1;
% cur_dir=pwd;
%
% % Gather data from workspace variable
% x = displacement(:,1); %(:,1) represents all rows and only 1st coloum
% y = displacement(:,2); %(:,2) represents all rows and only 2nd coloum
% z = displacement(:,3);
%
% pitch = angles(:,1);
% yaw = angles(:,2);
% roll = angles(:,3);
%
% scale_factor = 1; % Scale of quadrotor 1:1
% step = 1; % Step size for the plot
%
% viewang = [120 20] % View angle for the video

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Load the quadcopter model and manipulate vertex positions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Method 1 - OBJ
% load QModel_simp.mat;

```

```

% load A.mat;
% V = A1.vertices;

% % Method 2 - OBJ
% % wavefront obj start
% % tic;ParrotOBJMambo = readObj('Mambo.mtl');toc
% %   ParrotOBJMambo = readObj('Mambo.mtl');
%   tic
%   load ParrotOBJMambo.mat;
%   V = ParrotOBJMambo.v;
%   F = ParrotOBJMambo.f.v;
%   toc
% % wavefront obj end


% % Method 3 - STL
% % STL obj start
% % tic;ParrotSTLMambo = stlreadAJ('ParrotMambo.stl');toc %- This extracts Windows 3D
file and can be saved into a matlab ".mat" file
% %   [F,V] = stlreadAJ('ParrotMambo.stl');
% % STL obj end


% Select only one - 2nd or 3rd method!
V1 = V;
V1(:,1);

% Normalise vertex values
V(:,1)=V(:,1)-round(sum(V(:,1))/size(V,1)); % round it to nearly integer - (sum(V(:,1))-add the
first column and all rows /(no of rows [size(V, 1)]))
V(:,2)=V(:,2)-round(sum(V(:,2))/size(V,1));
V(:,3)=V(:,3)-round(sum(V(:,3))/size(V,1));

correction=max(abs(V(:,1)));
V=V./(scale_factor*correction);

```

```

ii=length(x);

resto=mod(ii,step);

frame = 0;

for i=1:step:(ii-resto)
    if mod(i,1) == 1
        clf;
        %plot3(x,y,z);
        plot3(0,0,0,'X'); hold on;
        set(gca,'XDir','reverse') ;
        xlabel('X [m]');
        ylabel('Y [m]');
        zlabel('Z [m]');
        grid on;
        hold on;
        light;
    end
    theta=pitch(i);
    phi=roll(i);
    psi=yaw(i);
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% https://www.symbolab.com/
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% \begin{pmatrix} \cos\left(\psi \right)\cdot \cos\left(\theta \right) \\
% & -\sin\left(\psi \right)\cdot \cos\left(\theta \right) & \sin\left( \right. \\
% \left. \theta \right) \backslash \cos\left(\psi \right)\cdot \sin\left(\theta \right. \\
% \left. \right)\cdot \sin\left(\phi \right)+\sin\left(\psi \right)\cdot \cos \\
% \left(\phi \right) & -\sin\left(\psi \right)\cdot \sin\left(\theta \right. \\
% \left. \right)\cdot \sin\left(\phi \right)+\cos\left(\psi \right)\cdot \cos \\
% \left(\phi \right) & -\cos\left(\theta \right)\cdot \sin\left(\phi \right. \\
% \left. \right) \backslash -\cos\left(\psi \right)\cdot \sin\left(\theta \right)\cdot \\
% \cos\left(\phi \right)+\sin\left(\psi \right)\cdot \sin\left(\phi \right)

```

```

% \right)&\sin\left(\psi \right)\cdot \sin\left(\theta \right)\cdot
% \cos\left(\phi \right)+\cos\left(\psi \right)\cdot \sin\left(\phi
% \right)&\cos\left(\theta \right)\cdot \cos\left(\phi \right)\end{pmatrix}
%
% Rotation sequence = Rx Ry Rx - Z facing up
% R = [\cos(\psi)*\cos(\theta), -\sin(\psi)*\cos(\theta), \sin(\theta);
%
% \cos(\psi)*\sin(\theta)*\sin(\phi)+\sin(\psi)*\cos(\phi) ...
% -\sin(\psi)*\sin(\theta)*\sin(\phi)+\cos(\psi)*\cos(\phi) ...
% -\cos(\theta)*\sin(\phi);
%
% -\cos(\psi)*\sin(\theta)*\cos(\phi)+\sin(\psi)*\sin(\phi) ...
% \sin(\psi)*\sin(\theta)*\cos(\phi)+\cos(\psi)*\sin(\phi) ...
% \cos(\theta)*\cos(\phi)
% ];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%CL clockwise rotation matrix - NED - RHR - Z facing down
% Rotation sequence = Rx Ry Rz
% R = [ \cos(\psi)*\cos(\theta), \cos(\theta)*\sin(\psi), -\sin(\theta);
%
% \sin(\phi)*\sin(\theta)*\cos(\psi) - \cos(\phi)*\sin(\psi)...
% \sin(\phi)*\sin(\theta)*\sin(\psi) - \cos(\phi)*\cos(\psi)...
% \sin(\phi)*\cos(\theta);
%
% \cos(\phi)*\sin(\theta)*\cos(\psi) + \sin(\phi)*\sin(\psi)...
% \cos(\phi)*\sin(\theta)*\sin(\psi) - \sin(\phi)*\cos(\psi)...
% \cos(\phi)*\cos(\theta);
% ];

```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%TP Counter-clockwise rotation matrix - RHR - Z facing UP
```

```
% Rotation sequence = Rz Ry Rx
```

```
R = [ cos(psi)*cos(theta)...
      cos(psi)*sin(theta)*sin(phi) - sin(psi)*cos(phi)...
      sin(psi)*sin(phi) + cos(psi)*sin(theta)*cos(phi);

      sin(psi)*cos(theta)...
      cos(psi)*cos(phi) + sin(psi)*sin(theta)*sin(phi)...
      sin(psi)*sin(theta)*cos(phi) - cos(psi)*sin(phi)...

      -sin(theta), cos(theta)*sin(phi), cos(theta)*cos(phi)
    ];
```

```
%New vertices formed after rotation
```

```
Vnew=V*R;
```

```
rif=[x(i) y(i) z(i)]; %displacement
```

```
X0= repmat(rif,size(Vnew,1),1); %bring it to correct size
```

```
Vnew=Vnew+X0;
```

```
% Patch the new quadrotor vertices on the axes
```

```
% p = patch('faces',A1.objects(3).data.vertices,'vertices',Vnew); hold on;
```

```
p = patch('faces',F,'vertices',Vnew); hold on;
```

```
set(p,'facec',[0.24,0.65,0.24]); hold on;
```

```
set(p, 'EdgeColor','none');
```

```
xlabel('X [m]');
```

```
ylabel('Y [m]');
```

```

xlabel('Z [m]');

origin = [0, 0, 0];
Xaxis = [5, 0, 0];
Yaxis = [0, -5, 0];
Zaxis = [0, 0, 5];

%
% %Draw body-attached frame X-Axis
% mArrow3(origin, Xaxis, 'facealpha', 0.5, 'color', 'red', ...
%         'stemWidth', 0.05); %, 'EraseMode','normal');
% hold on;
% %Draw body-attached frame Y-Axis
% mArrow3(origin, Yaxis, 'facealpha', 0.5, 'color', 'green', ...
%         'stemWidth', 0.05); %, 'EraseMode','normal');
% hold on;
% %Draw body-attached frame Z-Axis
% mArrow3(origin, Zaxis, 'facealpha', 0.5, 'color', 'blue', ...
%         'stemWidth', 0.05); %, 'EraseMode','normal');
% hold on;

if mov || (i == 1)
    view(theView);
    axis equal;
end

%Capture video
if mov
    lighting gouraud;
    frame = frame + 1;
    M(frame) = getframe;
end

```

```

%   t = datetime('now'); % - startTime;
%   % Add points to animation
% %   addpoints(h,datenum(t),roll);
% %   addpoints(h2,datenum(t),pitch);
%   addpoints(h,datenum(t),p);
% %   addpoints(h4,datenum(t),thetaHat);
%   % Update axes
%   ax.XLim = datenum([t-seconds(50) t]);
%   datetick('x','keeplimits')
%   drawnow

end

hold on;

%Plot the displacement trajectory
plot3(x,y,z,'--','LineWidth',2);
lighting gouraud;
grid on;
view(theView);

daspect([1 1 1]) ;

xlabel('X [m]');
ylabel('Y [m]');
zlabel('Z [m]');
title('Euler Angle/DCM');
% title(['Time elapsed = ' num2str(toc) 's'])

cd (cur_dir);
toc

```

Chapter 8-246 of 265

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
Parrot Mambo Bifilar Pendulum Test																		
No:	Pendulum Test	Period T/10	Time for 10 Osc	Period	10x Osc	Period	Omega (w)	= 2pi f	= 2.74831	>> Vehicle.Airframe.inertia ans = 1.0e-04 * 0.5829 0 0 0.7169 0 1.0000								
1	Time for 10 Osc	22.38	2.238	24.85	2.485	25.62	2.562	g = w^2 L	4pi^2 / w^2 * L	9.795472								
2		23.36	2.336	24.73	2.473	25.35	2.535											
3		22.8	2.28	25.36	2.536	25.27	2.527											
4		22.75	2.275	25.32	2.532	25.32	2.532											
5		22.83	2.283	25.32	2.532	25.24	2.524											
6		22.98	2.298	25.12	2.512	25.28	2.528											
7		22.73	2.273	25.07	2.507	25.27	2.527											
8		23.23	2.323	25.34	2.534	25.31	2.531											
9		22.91	2.291	25.45	2.545	25.25	2.525											
10		22.65	2.265	25.37	2.537	25.03	2.503											
Avg Time @0.15m		2.2862	Avg Time @0.10m	2.5193	Avg Time	2.5294												
$\hat{f}_x \gg$																		
No:	Moment of Inertia Z	x10 sec	x20 sec	x10	x20	L = 1.5621m D = 0.215m L String length D is the distance between 2 strings												
1		11.2	22.14		11.48	22.62												
2			21.91		11.31	22.56												
3		10.73	22.03		11.29	22.42												
4		11.01	22.41		11.53	22.56												
5		11.16	22.33		11.37	22.31												
6		11.09	22.49		11.15	22.47												
7					11.36	22.65												
8					11.22	22.57												
9					10.95	22.27												
10					11.13	22.62												
				Avg T	1.1279	1.12525												
$\frac{(0.0068 \times 9.81 \times 0.125^2 \times 1.2133^2)}{(16 \times \pi^2 \times 1.2192)} = 0.0000796965$																		
X																		
Moment of Inertia Y		No of Tests	No of Oscillations	Time Taken	Period													
1		10	11.77	1.177		No of Test	No of Osc	Time Taken	Period									
2		10	12.05	1.205		1	10	12.23	1.223									
3		10	11.85	1.185		2	10	12.13	1.213									
4		10	11.74	1.174		3	10	12.2	1.22									
5		10	11.83	1.183		4	10	12.16	1.216									
6		10	11.53	1.153		5	10	12.31	1.231									
7		10	12.17	1.217		6	10	12.25	1.225									
8		10	11.55	1.155		7	10	11.97	1.197									
9		10	11.59	1.159		8	10	12.08	1.208									
10		10	11.62	1.162		9	10	11.94	1.194									
						10	10	12.06	1.206									
						Avg			1.2133									
						11	20	24.6	1.23									


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

//Declaring some global variables
int gyroX, gyroY, gyroZ;
long accelX, accelY, accelZ, acc_total_vector;
int temp;
long gyroX_cal, gyroY_cal, gyroZ_cal;
long loop_timer;
float angle_pitch, angle_roll;
int angle_pitch_buffer, angle_roll_buffer;
boolean set_gyro_angles;
float angle_roll_acc, angle_pitch_acc;
float angle_pitch_output, angle_roll_output;

/* Ultrasonic variables - Begin*/
const int echo = 7;
const int trig = 8;

long dur;
float dis;
/*Ultrasonic Variables - End */

/* Receiveing data with Header '<' = 0x3C and Footer '>' = 0x3E - Begin*/
const byte numBytes = 5;
byte receivedBytes[numBytes]; // 0, 1, 2, 3, 4?
byte numReceived = 0;

boolean newData = false;
/* Receiveing data with Header '<' = 0x3C and Footer '>' = 0x3E - End*/
```

```

int dSpace;

float fromSimulink;

//#define M1Pin 3
//#define M2Pin 9
//#define M3Pin 10
//#define M4Pin 11

//Servo esc5, esc6, esc9, esc10;    //Creating a servo class name with ESC
int M1 = 0, M2 = 0, M3 = 0, M4 = 0;

void setup() {
    Wire.begin();                    //Start I2C as master
    Serial.begin(115200);            //Use only for debugging - 1382400 (tried
    but due to limitation of Arduino) highest on hc-05 - set to 230400
    Serial.println("<Arduino is ready>");

    // esc5.attach(5);                // Motor 1
    // esc6.attach(6);                // Motor 2
    // esc9.attach(9);                // Motor 3
    // esc10.attach(10);              // Motor 4

    // Timer - Pin 9 and pin 10 - 16bit
    TCCR1A = 0b10100001; TCCR1B = 0b00000011;
    TCNT1 = 0;

    // Timer 2 - Pin 3 and Pin 10 - 8 bit

    pinMode(3, OUTPUT); pinMode(9, OUTPUT); pinMode(10, OUTPUT); pinMode(11,
    OUTPUT);

    pinMode(13, OUTPUT);

```

```

pinMode(echo, INPUT);  pinMode(trig, OUTPUT);

onetime_IMU_registerSetup();

digitalWrite(13, HIGH);

for (int cal_int = 0; cal_int < 2000 ; cal_int ++){

    conti_IMU_data();
    gyroX_cal += gyroX;
    gyroY_cal += gyroY;
    gyroZ_cal += gyroZ;
    delay(3);
}

gyroX_cal /= 2000;
gyroY_cal /= 2000;
gyroZ_cal /= 2000;


digitalWrite(13, LOW);

loop_timer = micros();
}

void loop(){

    recvBytesWithStartEndMarkers();
    showNewData();

    conti_IMU_data();

    gyroX -= gyroX_cal;
    gyroY -= gyroY_cal;

```

```

gyroZ -= gyroZ_cal;

angle_pitch += gyroX * 0.0000611;
angle_roll += gyroY * 0.0000611;

angle_pitch += angle_roll * sin(gyroZ * 0.000001066);
angle_roll -= angle_pitch * sin(gyroZ * 0.000001066);

acc_total_vector = sqrt((accelX*accelX)+(accelY*accelY)+(accelZ*accelZ));
angle_pitch_acc = asin((float)accelY/acc_total_vector)* 57.296;
angle_roll_acc = asin((float)accelX/acc_total_vector)* -57.296;

angle_pitch_acc -= 0.0;
angle_roll_acc -= 0.0;

if(set_gyro_angles){
    angle_pitch = angle_pitch * 0.9996 + angle_pitch_acc * 0.0004;
    angle_roll = angle_roll * 0.9996 + angle_roll_acc * 0.0004;
}
else{
    angle_pitch = angle_pitch_acc;
    angle_roll = angle_roll_acc;
    set_gyro_angles = true;
}

angle_pitch_output = angle_pitch_output * 0.9 + angle_pitch * 0.1;
angle_roll_output = angle_roll_output * 0.9 + angle_roll * 0.1;

Serial.write(40);
Serial.write(41);

serialFloat(angle_pitch_output);

```

```

    serialFloat(angle_roll_output);

    serialFloat(ultrasonic());

    while(micros() - loop_timer < 4000);
    loop_timer = micros();
}

```

```

/* -----
_*/
/* ----- Functions -----
----*/
/* -----
_*/

```

```

void serialFloat(float f)
{
    byte * b = (byte *) &f;

    Serial.write(b[0]);
    Serial.write(b[1]);
    Serial.write(b[2]);
    Serial.write(b[3]);
} // serialFloat(1)

```

```

void conti_IMU_data(){
    Wire.beginTransaction(0x68);
    Wire.write(0x3B);
    Wire.endTransmission();
    Wire.requestFrom(0x68,14);
    while(Wire.available() < 14);
}

```

```

    accelX = Wire.read()<<8|Wire.read();
    accelY = Wire.read()<<8|Wire.read();
    accelZ = Wire.read()<<8|Wire.read();
    temp = Wire.read()<<8|Wire.read();
    gyroX = Wire.read()<<8|Wire.read();
    gyroY = Wire.read()<<8|Wire.read();
    gyroZ = Wire.read()<<8|Wire.read();
}

```

```

void onetime_IMU_registerSetup(){
    Wire.beginTransmission(0x68);
    Wire.write(0x6B);
    Wire.write(0x00);
    Wire.endTransmission();
    Wire.beginTransmission(0x68);
    Wire.write(0x1C);
    Wire.write(0x10);
    Wire.endTransmission();
    Wire.beginTransmission(0x68);
    Wire.write(0x1B);
    Wire.write(0x08);
    Wire.endTransmission();
}

```

```

float ultrasonic()
{
    digitalWrite(trig, LOW);
    delayMicroseconds(2);

    digitalWrite(trig, HIGH);
    delayMicroseconds(10);
    digitalWrite(trig, LOW);

    dur = pulseIn(echo, HIGH);
}

```

```

delay(50);

//Serial.println(dur);

dis = ((dur * 0.034)/2 )/100;

// Serial.println(dis);

return dis;
}

void recvBytesWithStartEndMarkers()
{
    static boolean recvInProgress = false;
    static byte ndx = 0;
    byte startMarker = 0x3C;
    byte endMarker = 0x3E;
    byte rb;

    while (Serial.available() > 0 && newData == false) {
        rb = Serial.read();
        Serial.println(rb);

        if (recvInProgress == true) {
            if (rb != endMarker) {
                receivedBytes[ndx] = rb;
                ndx++;
                if (ndx >= numBytes) {
                    ndx = numBytes - 1;
                }
            }
        }
        else {
            receivedBytes[ndx] = '\0'; // terminate the string

```

```

        recvInProgress = false;
        numReceived = ndx; // save the number for use when printing
        ndx = 0;
        newData = true;
    }
}

else if (rb == startMarker) {
    recvInProgress = true;
}
}
}

void showNewData()
{
    if (newData == true) {

        for (byte n = 0; n < numReceived; n++) {
            Serial.println(receivedBytes[n]);

        }

//    Received data from Simulink

        Serial.println(receivedBytes[0]);                Serial.println(receivedBytes[1]);
Serial.println(receivedBytes[2]); Serial.println(receivedBytes[3]);

        M1 = receivedBytes[0]; M2 = receivedBytes[1] ; M3 = receivedBytes[2]; M4 =
receivedBytes[3];

//    esc5.write(M1); //    esc6.write(M2); //    esc9.write(M3); //    esc10.write(M4);
//    Serial.println();

```

```

    analogWrite(3, M1); analogWrite(9, M2); analogWrite(10, M3); analogWrite(11, M4);

    newData = false;
}
else
{
    M1 = 0, M2 = 0, M3 = 0, M4 = 0;
    //    esc5.write(M1);
    //    esc6.write(M2);
    //    esc9.write(M3);
    //    esc10.write(M4);

    analogWrite(3, M1); analogWrite(9, M2); analogWrite(10, M3); analogWrite(11, M4);
}
}

void PWM ()
{
    dSpace = Serial.read();
//    Serial.print("dSpace: ");
//    Serial.println(dSpace);
//    esc5.write(dSpace);
//    esc6.write(dSpace);
//    esc9.write(dSpace);
//    esc10.write(dSpace);
}

```

```
% a = arduino;
% imu = mpu6050(a,'SampleRate',100,'SamplesPerRead',5,'ReadMode','latest');

% Error using Complementary_Filter_Working (line 27)
% Error occurred while reading data from the I2C device. Check the connections between the
hardware and device, clear the device object and create a new one.
% Code works but further investigation needed.
% https://uk.mathworks.com/help/fusion/ref/complementaryfilter-system-
object.html#d123e114333
% https://github.com/SayanSeth/MPU-6050-MATLAB-Toolbox

figure(1)
h = animatedline;
h.Color = 'r';
% h.Marker = '.'
h2 = animatedline;
h2.Color = 'g';

legend('Roll', 'Pitch')
xlabel('Time (s)')
ylabel('Angle (Degrees)')
title('Roll')

dt = 0.0185;

ax = gca;
ax.YGrid = 'on';
count = 0;
stop = false;
startTime = datetime('now');
```

```
while 1
```

```
    dt = 0.0185;
```

```
    G = readAngularVelocity(imu);
```

```
    GyroX = G(1,1);
```

```
    GyroY = G(1,2);
```

```
    GyroZ = G(1,3);
```

```
    A = readAcceleration(imu);
```

```
    AccelX = A(1,1);
```

```
    AccelY = A(1,2);
```

```
    AccelZ = A(1,3);
```

```
    % Convert gyroscope measurements to radians
```

```
    GyroX_rad = GyroX * pi / 180.0;
```

```
    GyroY_rad = GyroY * pi / 180.0;
```

```
    GyroZ_rad = GyroZ * pi / 180.0;
```

```
    % 1) Accelerometer only
```

```
    phiDot_acc = atan2(AccelY, sqrt(AccelX.^2 + AccelZ.^2));
```

```
    thetaDot_acc = atan2(-AccelX, sqrt(AccelY.^2 + AccelZ.^2));
```

```
    % 2) Complimentary Filter only
```

```
    alpha = 0.1;
```

```
    p = GyroX_rad;
```

```
    q = GyroY_rad;
```

```
    r = GyroZ_rad;
```

```
    phiDotGyrRoll = zeros(1, 2);
```

```
    thetaDotGyrPitch = zeros(1, 2);
```

```

phi_hat_complimentary = zeros(1, 2);
thetaDot_complimentary = zeros(1, 2);

phiDot = phiDotGyrRoll(1);
thetaDot = thetaDotGyrPitch(1);

% Transfer matrix where body rates p,q,r is used to calculate inertial
% matrix. Note psidot is not calculated due to absence of magnetometer in
% MPU6050
phiDotGyrRoll(2) = phiDot + dt * (p + sin(phiDot) * tan(thetaDot) * q + cos(phiDot) *
tan(thetaDot) * r);
thetaDotGyrPitch(2) = thetaDot + dt * (cos(phiDot) * q - sin(phiDot) * r);

% Fusion both accelerometer and Gyroscope by taking 90% of gyroscopes
% reading and 10% accelerometer reading (alpha).
phiDot_comp(2) = (1 - alpha) * phiDotGyrRoll(2) + alpha * phiDot_acc;
thetaDot_complimentary(2) = (1 - alpha) * thetaDotGyrPitch(2) + alpha * thetaDot_acc;

phiDotGyrRoll(1) = phiDotGyrRoll(2);
thetaDotGyrPitch(1) = thetaDotGyrPitch(2);

phiDot_comp(1) = phiDot_comp(2);
thetaDot_comp(1) = phiDot_comp(2);

% Convert all estimates to degrees
phiDot_acc = phiDot_acc * 180.0 / pi; thetaDot_acc = thetaDot_acc * 180.0 / pi;
phiDotGyrRoll = phiDotGyrRoll * 180.0 / pi; thetaDotGyrPitch = thetaDotGyrPitch * 180.0
/ pi;
phiDot_comp(2) = phiDot_comp(2) * 180.0 / pi;
thetaDot_comp(2) = thetaDot_comp(2) * 180.0 / pi;
phi_hat_kalman = phi_hat_kalman * 180.0 / pi;
theta_hat_kalman = theta_hat_kalman * 180.0 / pi;

```

```

roll_comp = phiDot_comp(2);
pitch_comp = thetaDot_comp(2);

% Get current time
t = datetime('now'); % - startTime;
% Add points to animation
addpoints(h,datenum(t),roll_comp);
addpoints(h2,datenum(t),pitch_comp);
% Update axes
ax.XLim = datenum([t-seconds(15) t]);
datetick('x','keeplimits')
drawnow
end

```

```
#include <ESP8266WiFi.h>
#include <WiFiUdp.h>

const char* ssid = "*****";
const char* password = "*****";

WiFiUDP Udp;

unsigned int localUdpPort = 4210; // local port to listen on
char incomingPacket[255]; // buffer for incoming packets
char replyPacket[] = "Hi there! Got the message :-)"; // a reply string to send back

void setup()
{
  Serial.begin(115200);
  Serial.println();

  Serial.printf("Connecting to %s ", ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }
  Serial.println(" connected");

  Udp.begin(localUdpPort);
  Serial.printf("Now listening at IP %s, UDP port %d\n", WiFi.localIP().toString().c_str(),
localUdpPort);
}

void loop()
{
```

```

int packetSize = Udp.parsePacket();
if (packetSize)
{
    // receive incoming UDP packets
    Serial.printf("Received %d bytes from %s, port %d\n", packetSize,
Udp.remoteIP().toString().c_str(), Udp.remotePort());
    int len = Udp.read(incomingPacket, 255);
    if (len > 0)
    {
        incomingPacket[len] = 0;
    }
    Serial.printf("UDP packet contents: %s\n", incomingPacket);

    // send back a reply, to the IP address and port we got the packet from
    Udp.beginPacket(Udp.remoteIP(), Udp.remotePort());
    Udp.write(replyPacket);
    Udp.endPacket();
}
}

```

```
#include <HX711.h>

#define DOUT A1
#define CLK A0

HX711 scale(DOUT, CLK);

long errorValue = -415.534; // -180 & -220/frame, 427g = 180012 => 180012/427 = 421.534

void setup() {
  Serial.begin(9600);

  scale.set_scale(errorValue); //
  scale.tare();

  Serial.print("Place a known weight "); //(14563.00 / known weight 35g)
// while(!Serial.available()) {};
//
// float x = scale.get_units(10);
// Serial.print("Measured Weight ");
// Serial.println(x, 1);
}

void loop()
{
  scale.set_scale(errorValue);

  float x = scale.get_units(10);
  Serial.print("Measured Weight ");
  Serial.println(x, 1);
}
```

APPENDIX J - Motor parameter extraction table

Vrms (V)	PWM (V)	Weight (g)	Ang RPM	W (rad/sect)	Thrust (N)	W squared	Ke or Kt	Force F	Power KW	V/W	I/W	V/I	W/I
4.3	4.183	-67.9	1050	109.9557429	-	12090.26539	0.009095	-0.6661	1.29	0.039107	0.002728	14.33333	366.5191
5.04	4.252	-36.6	1404	147.0265362	-	21616.80234	0.006801	-0.35905	3.024	0.03428	0.004081	8.4	245.0442
5.68	4.301	-4.7	1816	190.1710753	-	36165.03788	0.005258	-0.04611	5.112	0.029868	0.004733	6.311111	211.3012
5.68	4.325	-0.9	1799	188.3908395	-	35491.10839	0.005308	-0.00883	5.68	0.03015	0.005308	5.68	188.3908
5.83	4.355	22	2027	212.2669436	0.21582	45057.25536	0.004711	0.21582	6.413	0.027465	0.005182	5.3	192.9699
6.26	4.385	50.2	2311	242.0073541	0.492462	58567.55943	0.004132	0.492462	8.764	0.025867	0.005785	4.471429	172.8624
6.46	4.426	64.8	2435	254.9926037	0.635688	65021.22795	0.003922	0.635688	10.336	0.025334	0.006275	4.0375	159.3704
6.72	4.449	94.8	2676	280.2300647	0.929988	78528.88916	0.003568	0.929988	12.096	0.02398	0.006423	3.733333	155.6834
6.94	4.482	118.8	2875	301.069296	1.165428	90642.72098	0.003321	1.165428	14.574	0.023051	0.006975	3.304762	143.3663
7.14	4.516	142.3	3028	317.0914185	1.395963	100546.9677	0.003154	1.395963	16.422	0.022517	0.007253	3.104348	137.8658
7.32	4.555	160.8	3173	332.275783	1.577448	110407.196	0.00301	1.577448	18.3	0.02203	0.007524	2.928	132.9103
7.62	4.586	205.8	3501	366.6238627	2.018898	134413.0567	0.002728	2.018898	21.336	0.020784	0.007637	2.721429	130.9371
7.72	4.622	222.6	3560	372.8023282	2.183706	138981.5759	0.002682	2.183706	23.16	0.020708	0.008047	2.573333	124.2674
7.96	4.65	256.4	3151	329.9719484	2.515284	108881.4867	0.003031	2.515284	27.064	0.024123	0.010304	2.341176	97.05057
8.31	4.675	313.3	4023	421.2875748	3.073473	177483.2207	0.002374	3.073473	31.578	0.019725	0.00902	2.186842	110.8652
8.4	4.701	321.6	4104	429.769875	3.154896	184702.1455	0.002327	3.154896	32.76	0.019545	0.009075	2.153846	110.1974
8.73	4.739	356.5	4350	455.5309348	3.497265	207508.4325	0.002195	3.497265	37.539	0.019164	0.00944	2.030233	105.9374
9.01	4.77	393.9	5416	567.1621937	3.864159	321672.954	0.001763	3.864159	42.347	0.015886	0.008287	1.917021	120.6728
9.13	4.808	415.2	4621	483.9098884	4.073112	234168.8769	0.002067	4.073112	44.737	0.018867	0.010126	1.863265	98.75714
9.42	4.819	474.1	4901	513.2315198	4.650921	263406.593	0.001948	4.650921	50.868	0.018354	0.010522	1.744444	95.04287
9.53	4.855	492	4981	521.6091003	4.82652	272076.0535	0.001917	4.82652	54.321	0.01827	0.010928	1.67193	91.51037
9.52	4.884	535.2	5132	537.4217833	5.250312	288822.1731	0.001861	5.250312	57.12	0.017714	0.011164	1.586667	89.5703
9.51	4.92	549.4	5238	548.5220773	5.389614	300876.4693	0.001823	5.389614	58.962	0.017337	0.011303	1.533871	88.4713

