Quadrotor Tailsitter Scalability, Design, and Control: Towards Efficient Aerial Mobility

by

Trevor Phair

Thesis submitted to the University of Ottawa
in partial fulfillment of the requirements for the degree of
Master of Applied Science
in
Mechanical Engineering

Department of Mechanical Engineering
Faculty of Engineering
University of Ottawa

Examining Committee

The following served on the Examining Committee for this thesis.

External Member: Alexander Braun
Professor, Faculty of Engineering and Applied Physics
Queen's University

Internal Member: Davide Spinello
Associate Professor, Department of Mechanical Engineering
University of Ottawa

Supervisors: Eric Lanteigne
Associate Professor, Department of Mechanical Engineering
University of Ottawa

Zekai Hong
Adjunct Professor, Department of Mechanical Engineering
University of Ottawa

Declaration of Authorship

I hereby certify that this thesis is entirely my own original work except where otherwise indicated. I am aware of the University of Ottawa regulations concerning plagiarism, including those regarding consequent disciplinary actions. Any use of the works of any other author, in any form, is properly acknowledged at their point of use.

Abstract

This thesis investigates the scalability, design, and control challenges of tailsitter unmanned aerial systems, which uniquely combine the vertical takeoff and landing capabilities of multirotors with the efficient forward flight characteristics of fixed-wing aircraft. In a tailsitter configuration, multiple rotors provide lift and control during hover that also generate thrust for forward flight, while fixed wings supply additional lift. The transition between hover and forward flight is achieved by rotating the entire vehicle about its pitch axis. The mode-switching operation introduces highly nonlinear dynamics, complicating efforts to accurately model, design, and control such vehicles. To address these challenges, this work develops empirical scalability laws derived from historical data, offering practical guidelines for component mass distribution, wing sizing, and performance estimation during the conceptual design phase of tailsitters. These principles are applied to the design of a 5 kg prototype, serving as a case study. Furthermore, a software-in-the-loop simulation is established to concurrently test the aircraft dynamics and the flight controller. Enhancements to the attitude controller are proposed to improve aircraft performance during the transition and forward flight, with a particular focus on mitigating oscillations in forward flight common to tailsitters with a single flight mode. The controller gains are adapted online using a fuzzy logic scheme based on measured error, error derivative, and aircraft pitch, resulting in improved responsiveness and stability as verified through simulation and test flights with the prototype aircraft. In summary, this thesis outlines empirical correlations for estimating preliminary design parameters of tailsitters and proposes a method to develop control algorithms for such vehicles.

Acknowledgements

I am sincerely grateful to Dr. Zekai Hong, my co-supervisor and mentor throughout this project. Your unwavering support has pushed me through personal and academic growth, and now it is a pillar for my professional aspirations. Thank you for seeing the potential in me and taking the first leap.

To my supervisor, Dr. Eric Lanteigne, I am profoundly grateful for your support, guidance, and patience as I navigated through this thesis. You took an initial concept for the project and steered it into a complete and thorough thesis. The dedication to detail in the prototype design and editing the writing was crucial and appreciated.

To Nathaniel Mailhot, I thank you for becoming my mentor and guide through the unknown world of academia and entrepreneurship. You have shown such immense care and respect to me and my ambition, yet expect nothing in return.

I wish to acknowledge Owen Rainboth for always being there to help, from getting through undergrad to helping with every test flight. From our professional beginnings to a friendship I cherish, thank you. You are not a gambling man, but I appreciate you betting on me.

To my family, blood and chosen, I thank you all for your love, support, and for giving me a win when I needed it. Each of you has done more than you can imagine, and this process could not have been completed without you.

To my Wife. You keep me sane with the continuous show of love, understanding, and support for the path I have chosen. You always remind me there is more in this life to see, and I look forward to seeing it with you. Thank you.

I sincerely thank the Natural Sciences and Engineering Research Council of Canada (NSERC) Collaborative Research and Training Experience (CRATE) Uninhabited aircraft systems Training, Innovation, and Leadership Initiative (UTILI) program for their financial support, which has been instrumental in the completion of this thesis. I also thank the National Research Council (NRC) of Canada for their financial support throughout this project, starting over 4 years ago. Their combined support has allowed me to conduct research in my particular field of interest for my academic career.

Table of Contents

List of Tables

List of Figures

Abbreviations

AoA Angle of Attack

BSC backstepping Control

CAD computer - aided design

CFD computational fluid dynamics

CG centre of gravity

DOB disturbance observer

DOF degrees of freedom

EKF extended Kalman filter

EMP ESC, motor, and propeller

ESC electronic speed controller

FLC fuzzy logic controller

FSPID fuzzy supervised PID

GCS ground control station

GPS global positioning system

HYDRA HYbrid Design and Rotorcraft Analysis

ILC iterative learning controller

ILR integral, lead, and rolloff

INDI incremental nonlinear dynamic inversion

ITSMC integral terminal sliding mode control

LDI linear dynamic inversion

LiPo lithium polymer

LPF low pass filter

LQG linear - quadratic - Gaussian

LQR linear quadratic regulator

LSF linear saturation function

MPC model predictive controller

MTOW maximum takeoff weight

NACA National Advisory Committee for Aeronautics

NDI nonlinear dynamic inversion

NRC National Research Council

P proportional

PD proportional - derivative

PI proportional – integral

PID proportional – integral – derivative

PWM pulse width modulation

QBiT quadrotor biplane tailsitter

RPAS remotely piloted aircraft system

RPM revolutions per minute

SITL software in the loop

SO(3) Special Orthogonal group

UAS uninhabited aircraft system

UIUC University of Illinois Urbana-Champaign

VP + FPID Variable P + Fuzzy PID

VPP variable pitch propeller

VTOL vertical takeoff and landing

WSL Windows Subsystem for Linux

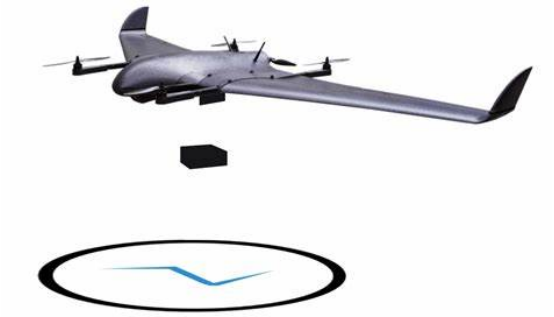
Chapter 1

Introduction

1.1 Background: Vertical Takeoff and Landing Remotely Piloted Aircraft

The utilization of drone technology, also known as uninhabited aircraft systems (UASs), or remotely piloted aircraft systems (RPASs), has had significant impacts on a variety of industries, including payload delivery, surveillance, and disaster management. The use cases are continuing to rise as they are developed for world events, such as the COVID-19 pandemic and the Russia-Ukraine conflict. In particular, vertical takeoff and landing (VTOL) capable RPAS are often used for their inherent ability to operate from nearly any location without the need for a runway, open field, or other infrastructure required to use fixed-wing aircraft. However, multirotors, which are one of the most common VTOL systems, have very short endurance and range when compared to their fixed-wing counterparts due to their inefficient production of lift. The combination of efficient forward flight with VTOL capabilities for RPAS has therefore become an area of interest for many researchers and industries. Two of the most common solutions are quadplanes and tiltrotors that combine components associated with both fixed-wing and multirotor vehicles. Figure 1.1 shows an example of both vehicles specifically designed for payload delivery missions. While both solutions can achieve VTOL and more efficient forward flight than multirotors, they have major drawbacks.

Quadplanes are a RPAS with a fixed-wing aircraft attached to a multirotor, with an example shown in Figure 1.1(a). When in forward flight, the quadplane acts like a fixed-wing aircraft, utilizing a forward propulsion system and typical control surfaces. VTOL is achieved by using the multirotor components, with differential thrust and torque of these rotors enabling the control and motion. Both systems must be used at the same time when gaining speed to transition from hover to forward flight. The components only used for hovering are still carried through the forward flight portion of the mission, and when hovering, the wing and forward propulsion system are unused. The extra components required to complete both flight modes result in excessive takeoff weight, reducing the



(a) The DeltaQuad flying wing quadplane cargo drone with 4 hover motors, 1 forward flight motor, and 2 control surfaces [37].

(b) The Wingcopter 198 tiltrotor delivery drone with 8 rotors, 4 actuators to tilt the rotors, and 4 control surfaces [137].

Figure 1.1: Examples of hybrid RPAS for payload delivery applications.

payload capabilities and range/endurance. Additionally, the multirotor components and associated vehicle structure create additional drag surfaces, reducing the forward flight efficiency. Quadplanes typically use between 8 and 12 actuating parts for the rotors and control surfaces, increasing the chance for part failure when compared to fixed-wing or multirotor RPAS.

Tiltrotors rotate some or all of the lifting rotors used in hover to create the required thrust in forward flight, with an example of a tiltrotor for payload delivery shown in Figure 1.1(b). The forward speed is controlled by adjusting the angle of the rotors, which converts vertical lift into forward thrust as the angle increases. However, mechanisms are required to tilt the rotors, and control surfaces are still needed in many cases for forward flight. Another drawback of this design is that the propellers used are typically not optimized for both hover and forward flight. Tiltrotors typically require at least 10 actuating parts to achieve all aspects of the mission. Similar to quadplanes, the resulting system requires extra components that increase the mass of the system, reducing payload and range, while also making the RPAS more complex and less reliable.

A less common solution to achieve VTOL with efficient forward flight is the tailsitter RPAS, with many examples of these aircraft shown in Figure 1.2. Tailsitters utilize the same systems used in hover and forward flight by rotating the entire aircraft about the pitch axis, with a standard mission profile shown in Figure 1.3. The rotors creating lift in hover are also used to create thrust in forward flight, while fixed wings generate the required lift. There are two primary classes of tailsitters: ones with control surfaces and ones without. When no control surfaces are used, the aircraft operates similarly to a multirotor in all phases of flight, utilizing differential thrust and torque between the motors for the entirety of attitude and position control. If control surfaces are used, a combination of differential thrust, torque, and surface deflection is used for control. Care must be taken in this case to ensure that enough airflow is created by the rotors and flowing over the control surfaces when in hover to have adequate control authority. In either case, only four actuators are

required at a minimum, being four rotors or two rotors and two control surfaces, reducing system complexity and weight while increasing reliability. However, the transition process between hover and forward flight presents a challenging control problem, as the vehicle dynamics are highly nonlinear, and the propeller efficiency between hover and forward flight is contradictory, as seen with tiltrotors.

(a) U.S. Army prototype tailsitter with wings mounted underneath a rigid quadrotor frame [17].

(b) The WingtraOne commercial tailsitter for surveying with two rotors and two control surfaces [139].

(c) Modified SkywalkerX-5 tailsitter with four rotors and two control surfaces [64].

(d) VertiKUL tailsitter for payload delivery with four rotors and a fixed wing [58].

Figure 1.2: Examples of tailsitter RPAS.

1.2 Motivation

Through a co-op placement supervised by Dr. Zekai Hong at the National Research Council of Canada, a small tailsitter RPAS was developed to illustrate the benefits compared to other VTOL options. The aircraft had a takeoff weight of 0.85 kg with no usable payload. The prototype produced is shown in Figure 1.4. A primary result from this project was improving the transition process from other tailsitters that would lose significant altitude. An ideal transition should exhibit full control of the aircraft at any angle of attack without altitude loss or gain if desired. The solution developed was a novel airframe that maximized the blown wing area of the vehicle through positioning stacked wings behind the rotors. The project was successful, showing that a stable transition could be achieved with this

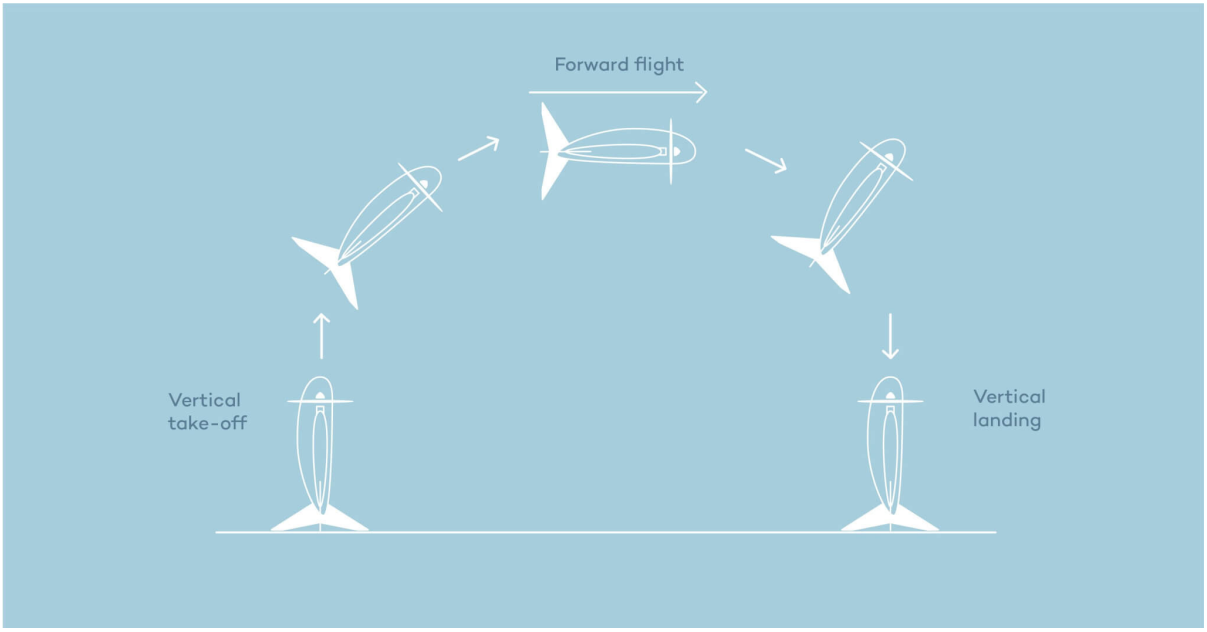
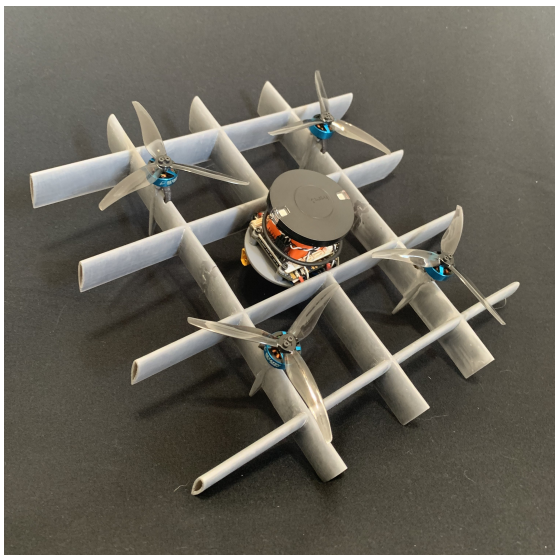


Figure 1.3: Profile sketch of the WingtraOne tailsitter, with vertical, forward, and transition flight modes [138].

airframe, leading to the patenting of the technology [143]. However, the controller used produced significant oscillations in forward flight, and a larger aircraft was desired to prove that the technology could be scaled to a RPAS with a useful payload or mission, motivating the work presented throughout this thesis.



(a) Isometric view of aircraft.

(b) Aircraft in forward flight.

Figure 1.4: A 0.85 kg prototype tailsitter RPAS developed at the National Research Council of Canada with four rotors and a lattice wing structure.

1.3 Thesis Objectives

The purpose of this project is to address gaps in the current literature regarding tailsitter scalability, design, simulation, and control. The project was comprised of the following five primary goals.

1. Develop a database of previously designed tailsitters weighing up to 25 kg to create a set of empirical equations and plots that can be used for empirical conceptual design of future aircraft. Using these results, present examples of the mass distributions for a range of aircraft, showing how features such as the structural and power system masses are expected to scale with takeoff weight.
2. Design a 5 kg tailsitter with a 1 kg payload using the empirical equations as a guide for their validation, while also incorporating the blown wing technology of the smaller prototype. Then construct a prototype of the designed tailsitter for testing.
3. Create a software in the loop (SITL) simulation of the prototype aircraft to test both the flight characteristics and the flight controller.
4. Develop a fuzzy logic-based flight controller to improve on the oscillation issues experienced with the small prototype aircraft. The controller should be compatible with the ArduCopter-based hardware and software previously used.
5. Test the prototype aircraft while using the custom flight controller to validate the use of the empirical data for the conceptual design of a tailsitter, the accuracy of the simulation, and the benefits of the fuzzy logic controller (FLC).

1.4 Thesis Outline

The work presented in this thesis provides a guide for the design, manufacturing, simulation, and control of a quadrotor tailsitter. It is organized with the following structure:

In Chapter 2, the scalability and design of tailsitters is investigated. First, a literature review of the current methodologies used to design tailsitters is presented, focusing on the use of empirical data for weight distribution or utilizing a physics-based design process. The design literature review is followed by a discussion of variable pitch propeller (VPP) and high-angle-of-attack aerodynamics of wings as they pertain to tailsitters. The physical properties of tailsitters discussed are then organized and compared against the takeoff weight to produce a set of empirical equations describing their scaling laws. The scaling laws are then employed for the design of a 5 kg prototype aircraft, with the design process detailed from start to construction.

Chapter 3 presents the procedure used to simulate the prototype vehicle. A SITL simulation is used, allowing for the simultaneous testing of the vehicle dynamics and the flight controller software. The simulation is utilized throughout the project to test the flight characteristics of the vehicle and the custom controller that is implemented.

Chapter 4 begins with a second literature review, focusing on the control of tailsitters. A gap in the literature for the application of utilizing fuzzy logic to adapt the gains of a proportional – integral – derivative (PID) controller for a quadrotor tailsitter is identified. A fuzzy PID controller compatible with ArduCopter is then developed using MATLAB.

Chapter 5 presents the results of the thesis, starting with an explanation of the process taken to test the prototype vehicle. The aircraft performance is then compared to the expected results from the scalability laws derived in Chapter 2, and the simulation from Chapter 3 is verified. The implemented controller from Chapter 4 is then compared against ArduCopter's attitude controller using results from the simulation and with the prototype aircraft, improving the responsiveness and reducing oscillations.

Chapter 6 concludes the work, providing a summary of the findings and the contributions to literature. The future work for this project is discussed, including updating the database, aerodynamic and structural optimization, design and simulation iterations, and flight controller improvements.

Furthermore, details on the design decisions, fabrication techniques, simulation procedures, and literature review required to complete this project that were beyond the scope of the main body of this thesis are presented in the Appendix.

Chapter 2

Scalability and Design

The conceptual design process for both conventional aircraft and RPAS commonly starts with using historical data to estimate many properties [55, 94, 98]. However, a problem associated with designing novel or state-of-the-art aircraft, including tailsitters, is that there is little historical data to work with. The goal of this chapter is to investigate how tailsitter aerodynamic and physical properties scale with maximum takeoff weight (MTOW). Specifications of available commercial and research tailsitter designs will be summarized in charts to aid with the conceptual design process. The results are then used for the conceptual design of a 5 kg prototype aircraft, followed by a summary of the building process.

2.1 Literature Review of Tailsitter Design Techniques and Scalability

The two primary methods used for tailsitter design are empirical and physics-based. When using empirical techniques, historical data is used to guide the conceptual phase, then modifications are made as the design gets more detailed. The empirical data is commonly combined with a conventional design process, which breaks down the aircraft weight into separate components, such as propulsion, payload, structural, and energy masses [55, 94]. Alternatively, physics-based design methods define the performance of vehicles through solvable equations that then set the values for designable parameters.

2.1.1 Conventional and Empirical Design Process

The design method used by Sinha et al. [101] for a single-winged tailsitter was primarily focused on using simulation software to predict the vehicle performance, shown in Figure 2.1(a). The authors estimated the aerodynamic parameters and conducted a stability analysis utilizing a combination of XLFR5 [38] and AVL [40]. While this method did not rely on historical data, it used estimated aerodynamic properties to calculate the required

size of components like the wings.

Similarly, Wang et al. [136] presented the design process of a battery-powered tailsitter with one wing, two control surfaces, and four rotors presented in Figure 2.1(b). The authors used a conventional process for the conceptual design, using weight fractions of the various components to estimate the MTOW. The power requirements for each flight stage were then defined with empirical equations, allowing for the design to be optimized for range or endurance by altering the wing loading and battery weight ratio.

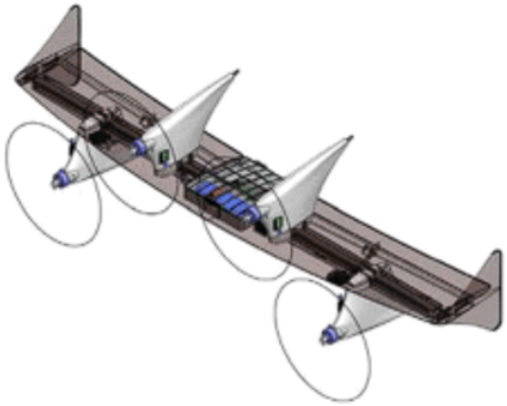
The conventional design process of the quadrotor biplane tailsitter (QBiT) shown in Figure 2.1(c) for 6 kg payload delivery was discussed by Chipade et al [33]. The researchers start with using a modified blade element theory to estimate the rotor performance and the design of variable pitch propellers (VPPs). The wings were designed by comparing the wing loading of different aspect ratios to estimate the power. The power draw, combined with the effects of the induced power of biplanes versus monoplanes, was used to select a wing loading value and aspect ratio, allowing for the other parameters to be quantified. The gasoline engine power plant was selected by estimating the required power in hover, as this is the most demanding flight mode. CATIA was then used to perform a structural analysis for the design of the aircraft structure.

The design of a quadrotor tailsitter with a single wing, shown in Figure 2.1(d), was accomplished using another conventional approach as explained by Sohail et al [102]. The authours estimated the thrust-to-weight ratio required for cruising, climbing, and hovering to create plots for the wing and power loading. Then mass fractions and known masses of components were used to estimate the MTOW of the aircraft.

2.1.2 Physics-Based Design Process

With design techniques that rely on historical data being unreliable for small tailsitters due to the scarcity of published data on this topic, alternative approaches have been used. The alternative design process is to utilize physics-based calculations to determine the size of required components. Since tailsitters act as both multirotors and fixed-wing aircraft, and have highly nonlinear dynamics in the transition, this technique may be complex to implement. However, the results of the designs produced with this method can be utilized to identify the scaling laws of tailsitters.

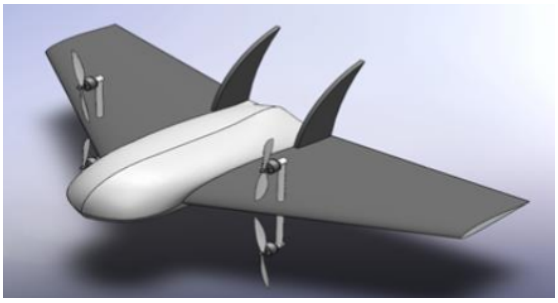
The lack of statistical data for the conceptual design phase of VTOL aircraft under 450 kg was discussed by Govindarajan et al. [52], who present a method of designing such aircraft through physics-based weight models. The authors used the HYbrid Design and Rotorcraft Analysis (HYDRA) tool for their design, which was first presented to design a helicopter with a lift-offset main rotor [106]. The tool estimates the performance of the aircraft in all flight modes using rotorcraft and wake-free analyses. HYDRA was used alongside a



(a) A single-winged tailsitter with four rotors and two control surfaces [101].

(b) A single-winged tailsitter with four rotors and two control surfaces [136].

(c) A biplane tailsitter with four rotors and no control surfaces [33].



(d) A delta wing tailsitter with four rotors and two control surfaces [102].

Figure 2.1: Examples of tailsitters designed using conventional and empirical techniques.

physics-based model to calculate the rotor performance and the sizing of the blades. Iterative finite element analyses are conducted to determine the size and corresponding weight of the components within the aircraft structure. The tool can estimate the aerodynamic performance, propulsion, rotor blades, airfoil, and general aircraft sizing simultaneously through an iterative process, leading to an optimal design based on the initial conditions and requirements. The authors presented the design process of a QBiT for payload delivery as an example of the tool.

Using the HYDRA tool, Reddinger [96] investigated how QBiT performance scales with MTOW while Govindarajan et al. [53] gave more insight into the design process. These works together analyzed MTOWs of 9.1 kg, 22.7 kg, and 453.6 kg to show how tailsitters scale. The FLIGHTLAB software was then used to estimate flight dynamics and analyze each design.

Govindarajan and Sridharan [51] investigated a variety of multirotor platforms, including

QBiTs using the HYDRA tool to compare the conceptual designs for MTOWs of 10, 15, 20, and 25 kg. The authors focused on finding the optimal trim configuration in forward flight for each of the designs considered. The trim settings that minimized the shaft power required to give the most efficient forward flight possible were used. However, each of the designs discussed only had a range requirement of 5 km, making their results difficult to compare to others that can traverse much farther distances, especially as the MTOW increases.

2.1.3 Design Considerations

Two important factors that must be considered when designing tailsitters are the contrasting benefits of propellers primarily used in hover and forward flight and the aerodynamic characteristics of airfoils at very high angles of attack. Propellers are typically optimized for either hover or forward flight, and tailsitters encounter angles of attack between 0 and 90 degrees in a standard mission. Therefore, both the propeller and airfoil performance are discussed in this section.

2.1.3.1 Variable Pitch Propellers

Efficient forward flight is typically achieved with large pitch propellers at lower rotational velocities. Conversely, multirotors tend to use propellers with a smaller pitch to reduce vibrations and turbulence, as well as requiring less power to achieve the peak torque of a motor to improve the controllability of the aircraft. The inclusion of variable pitch propeller (VPP) can solve the efficiency compromise by changing the pitch to be optimal for any flight mode. Therefore, tailsitters equipped with VPPs can achieve the control and hover benefits associated with VTOL aircraft without compromising forward flight efficiency.

The QBiT design presented by Chipade et al. [33] considered the use of VPPs. A modified blade element theory was used to estimate the performance of the rotors and was compared with experimental results, showing a good fit. A single 3.75 hp gasoline engine was utilized to spin all rotors at the same speed, while each rotor had an independent servo to adjust its pitch. It was found that the forward flight mode cruising at 20 m/s requires only 36 percent of the power require for hover when utilizing the VPPs.

The impact of variable pitch propellers was also investigated by Phillips et al. [88] who built a QBiT with wing-mounted motors that each had a servo to adjust the propeller blade pitch. It was shown that in forward flight, an increase in propeller pitch up to 30 degrees both reduces the power draw and increases the cruising speed. In contrast, the lowest power consumption in hover occurs when the blades are at the lowest pitch available, which was 12 degrees. The authors only recorded a 28 percent decrease in power consumption in forward flight versus hover, but this is likely due to the large Angle of Attack (AoA) of 20 degrees used for forward flight. More work on the same system by

Phillips et al. [87] indicates that the power consumption required for hover can be reduced by 50 percent when in forward flight with a lower AoA.

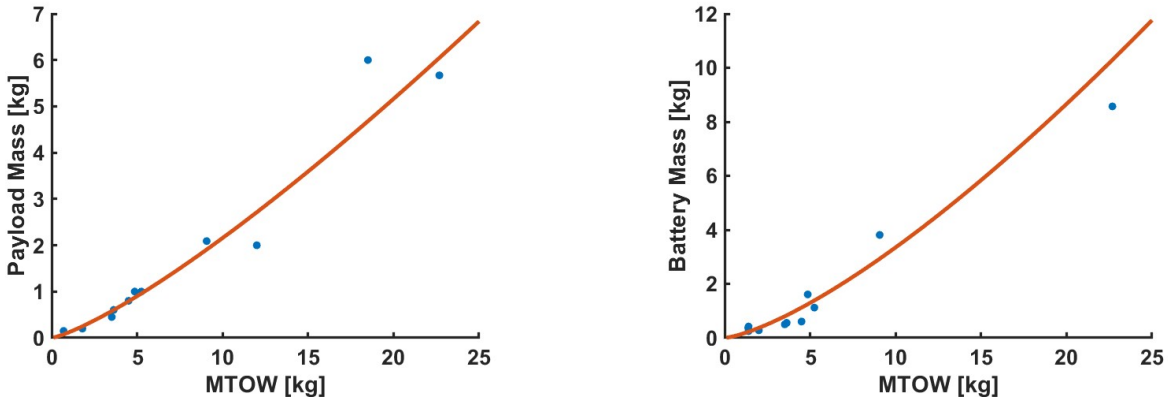
The aircraft presented by Reddinger [96] and Sridharan et al. [53] used VPPs to improve the efficiency in forward flight. The authors determined that increasing the propeller pitch in forward flight allows for a corresponding reduction in the rotational speed of the rotors, which gives a net reduction in power consumption. Using this technique, the cruising rotational rates were seen to be 20 to 40 percent of what was required for hover, resulting in hover-to-cruise power ratios ranging between 5:1 and 6:1.

The work of Simmons [100] illustrates the modelling of variable pitch propellers for use on electric VTOL aircraft in all flight modes. Using a wind tunnel, the rotor's performance was measured as the motor power, pitch collective, incidence angle, and inlet velocity were altered. For each of the flight modes, local surface models were produced using model identification to represent the collected data. The models were validated using alternative propeller data not used to create the models, illustrating the ability to predict the performance of variable-pitch rotors.

2.1.3.2 High Angle of Attack Airfoil Aerodynamics

The aerodynamic coefficients for airfoils at high angles of attack are important information for tailsitters, as they must transition between hover and forward flight. Kubo et al. presented a unique tailsitter combining two fixed-wing aircraft together [62]. The authors used a wind tunnel test to determine the aerodynamic properties of the entire vehicle from 0 to 90 degrees with the slats extended and retracted. An annular wing tailsitter presented by Gill and D'Andrea investigated the aerodynamics of their vehicle to create a theoretical lift model [49]. The modified X-Vert tailsitter presented by Caverly also indicated that lift can be produced throughout the entire transition [31]. However, the drag nearly linearly increased with AoA between forward flight and hover, showing that it is highly inefficient to operate in the transition region. Each of the above sources shows that significant lift can be created after the wing stalls, enabling tailsitters to maintain altitude throughout the transition. However, the analyses involving the full vehicle are less applicable than information focused on airfoil performance.

In contrast, the work of Ma et al. investigates the aerodynamics of an isolated blown wing at all possible angles of attack [76]. It is shown that lift can be created anywhere between hover and forward flight, but this is greatly increased when the rotor thrust is blown over the wings. Lee et al. presented the aerodynamics of a National Advisory Committee for Aeronautics (NACA) 0012 airfoil at all angles of attack to aid with the model identification of their tailsitter [63]. A sparse identification of nonlinear dynamics algorithm was used and compared against a wind tunnel analysis, both showing the lift, drag, and moment coefficients at all angles of attack.



(a) Payload mass versus MTOW.

(b) Battery mass versus MTOW.

Figure 2.2: Tailsitter component masses from literature review.

2.2 Scalability Laws

From the literature review of tailsitter design, there are few examples where empirical equations specific to tailsitters are applied due to the lack of historical data. However, many tailsitters have now been developed under 25 kg, presenting the opportunity to start empirical design. The scaling laws of aircraft are generalized rules that predict the aircraft size and performance based on historical data of similar designs. These laws rely on consistency within the overall design configuration, performance and weight of components used, and the governing laws of physics. The scaling laws of tailsitters presented in this section are determined by the current designs presented in the literature. The data used throughout this section is shown in Table 2.1. When creating the equations to describe the scaling laws, each data point within Table 2.1 was assigned the same weight. However, some studies were not included, such as the work of Govindarajan and Sridharan [51], who presented designs that were relatively large but had a small range of 5 km, and dfasfdasfdsa et al [92]. who had inconsistencies with their presented data.

The first scalability laws investigated are how the component masses scale with MTOW. Instead of using mass fractions, a fitted curve gives more insight into how these parameters scale since the relationships are non-linear. The scaling laws for payload and battery weight are shown in Figure 2.2. These parameters show very close fits to the historical data, though it is limited at larger takeoff weights. However, they can only be applied assuming that the aircraft is not being designed to carry a disproportionately heavy payload relative to the MTOW, or require a much larger or shorter range than comparable tailsitters, such as shown in this work [51].

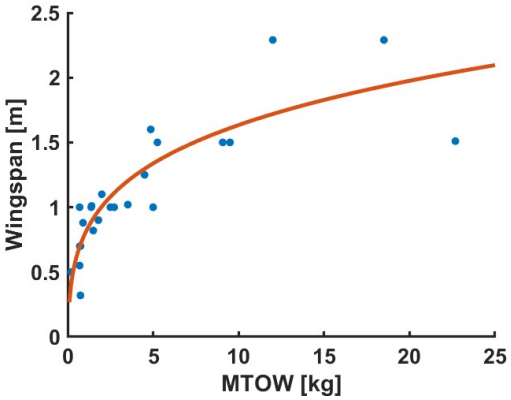
While nearly every tailsitter investigated presented their MTOW, fewer discussed battery and payload weights, with even fewer showing the ESC, motor, and propeller (EMP) or structural weights. In general, the payload and battery weights of electric tailsitters in-

Table 2.1: Data and references used for tailsitter scalability investigation.

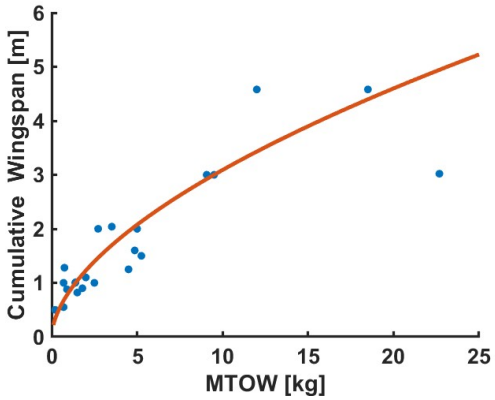
Rotors	Wings	MTOW [kg]	Wingspan [m]	Cumulative Wingspan [m]	Reference Area [m^2]	Chord [m]	Cruise [m/s]	Thrust [kg]	T/W [kg/kg]	Battery Mass [kg]	Payload [kg]	Flight Time [min]	Range [km]	Propeller Diameter [mm]	Wing Loading [N/m^2]	Aspect Ratio	Reference
4	1	0.70	1.00	1.00	0.16	0.16	12.0	2.10	3.00		0.15			203	42.26	6.15	[45, 101]
4	1	4.86	1.60	1.60	0.36	0.23	15.5	10.50	2.16	1.61	1.00	29.00	26.0	229	132.44	7.11	[58]
4	1	1.38	1.00	1.00	0.24	0.21	10.0	3.20	2.32	0.37		40.00	24.0	241	56.41	4.17	[136]
4	1	1.40	1.01	1.01	0.24	0.24	12.0	7.20	5.14	0.42		55.00	24.0	0	57.23	4.25	[73–75]
4	1	2.00	1.10	1.10	0.44	0.40		4.00	2.00	0.28				279	44.59	2.75	[64]
4	1	1.80	0.90	0.90	0.13	0.15	19.0				0.20	26.00	30.0	229	135.83	6.23	[141, 142]
4	2	9.50	1.50	3.00			14.4							610			[77, 96]
4	Annular	0.71	0.70				10.0							203			[49, 50]
4	2	2.72	1.00	2.00	0.44	0.22	14.0	6.10	2.24						60.64	4.55	[93]
4	X	5.00	1.00	2.00	0.50	0.25	22.0	10.20	2.04						98.10	4.00	[140]
4	2	12.00	2.29	4.58	1.51	0.33	20.0				2.00				78.06	6.96	[35]
4	1	5.25	1.50	1.50	0.64	0.43		7.56	1.44	1.12	1.00	9.00			80.47	3.52	[102]
4	2	18.50	2.29	4.58	1.51	0.33	20.0	40.77	2.20		6.00		32.0	760	120.35	6.96	[33]
4	2	1.40						1.83	1.31	0.25							[33]
4	2	3.52	1.02	2.04	0.52	0.25	13.4			0.50	0.45			330	66.64	4.02	[87, 88]
4	2	3.63								0.56	0.60						[53]
4	2	9.07	1.50	3.00	0.75	0.25				3.81	2.09		64.8	597	118.64	6.00	[53]
4	2	22.68	1.51	3.02	0.76	0.25				8.57	5.67			591	292.74	6.00	[53]
2	1	0.90	0.88	0.88			12.0	1.84	2.04					203			[20]
2	1	2.50	1.00	1.00	0.45	0.45								356	54.50	2.22	[135]
2	1	1.50	0.82	0.82													[61]
2	1	0.21	0.50	0.50	0.08	0.17	10.0							114	25.75	3.13	[32]
2	1	0.70	0.55	0.55	0.07	0.13		1.60	2.29					130	97.61	4.30	[117]
2	1	4.50	1.25	1.25	0.60	0.48	16.0			0.604	0.80	54.00	51.8	130	73.58	2.60	[139]

crease slightly exponentially with the MTOW, as shown in Figures 2.2(a) and 2.2(b). These components are typically the only two parts of the design that can be altered after the aircraft design is complete because changes to the avionics, structural elements, and electrical/power components will create significant changes to the overall performance and sizing. However, adjusting the weight of the battery and payload allows for a variety of missions to be completed by the same aircraft, such as longer range with a lighter payload or vice versa.

The properties of tailsitter wingspan related to MTOW are presented in Figure 2.3(a). However, since tailsitters are commonly seen with multiple wings, the cumulative sum of all wingspans can be more useful because this accounts for the total wingspan used, shown in Figure 2.3(b). The cumulative wingspan has a much better fit to the MTOW than just comparing wingspan. Additionally, the cumulative wingspan's relationship is more linear, making it easier to use and less impacted by changes in takeoff weight throughout the design loop.



(a) Tailsitter wingspan versus MTOW.

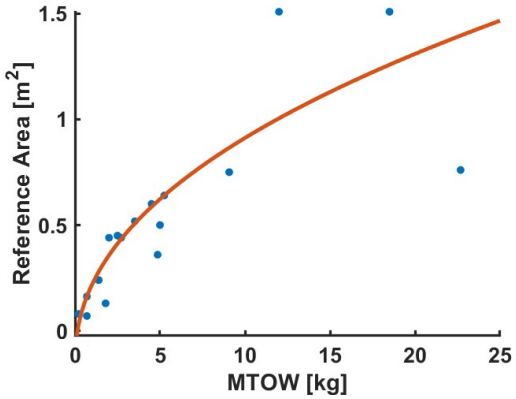


(b) Tailsitter cumulative wingspan versus MTOW.

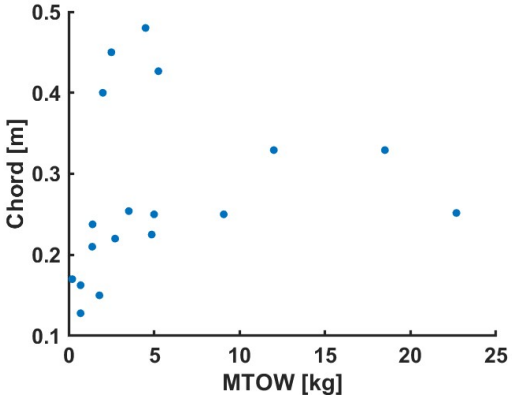
Figure 2.3: Properties of tailsitter wings as MTOW scales.

Similarly, the reference wing area also scales with the MTOW of tailsitters, illustrated by Figure 2.4(a). In this case, the reference area refers to the combined area of all the wings in a design. However, there is not a significant amount of data above 10 kg, making it more difficult to get accurate initial estimates for larger tailsitters currently. The wing chord length compared to MTOW is shown in Figure 2.4(b). In most cases, the chord length is 0.3 m or less, however, there is little correlation showing how the chord scales with mass other than a slight increase. In general, the chord is used to match the desired reference area and wingspan, partially depending on the number of wings.

Similarly, the aspect ratio of tailsitter wings is not as correlated to the MTOW as the wingspan or reference area, but shows a clear increase with weight in Figure 2.5(a). More notable is that the aspect ratio at any takeoff weight does not exceed 7, indicating a limit

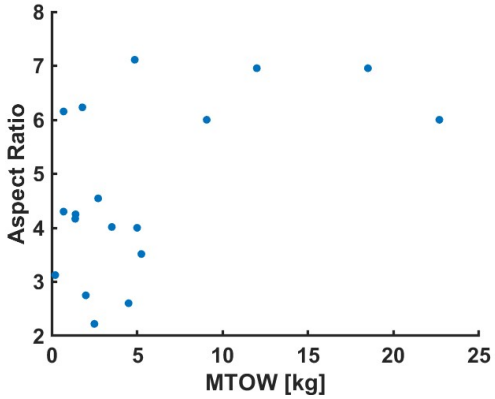


(a) Tailsitter reference area versus MTOW.

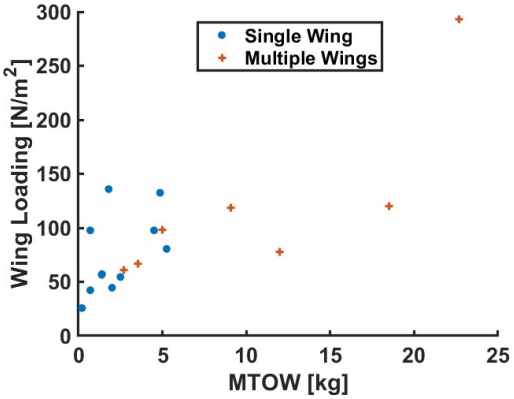


(b) Tailsitter chord length MTOW.

Figure 2.4: Tailsitter wing properties regarding area as MTOW scales.



(a) Tailsitter aspect ratio versus MTOW.

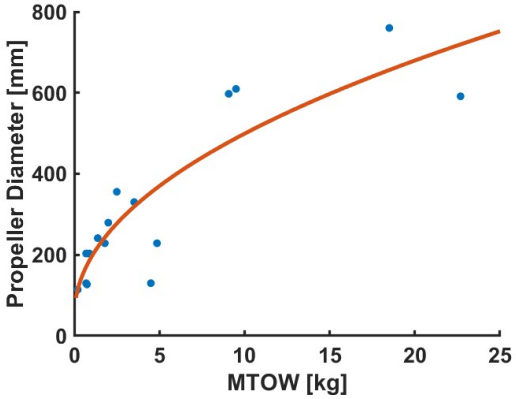


(b) Tailsitter wing loading versus MTOW.

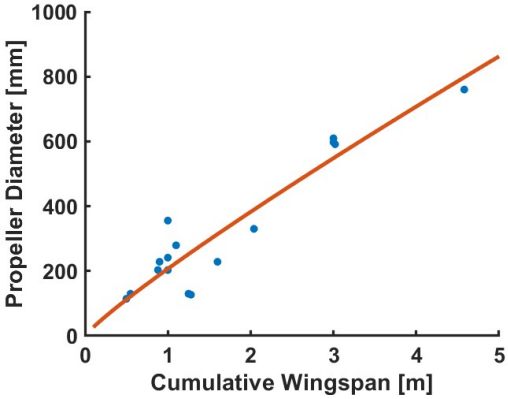
Figure 2.5: Tailsitter wing performance as MTOW scales.

to the weight and strength of structural components currently used on tailsitters. The last parameter regarding wings is the wing loading shown in Figure 2.5(b). The wing loading rarely exceeds 150 N/m^2, and starts to approach this value when the MTOW gets larger. However, for tailsitters around 5 kg or less, there is more variation in the wing loading values. Figure 2.5(b) also shows that as the MTOW exceeds 5 kg, it is much more common to use multiple wings rather than just 1.

The typical propeller diameter of tailsitters scaled with MTOW is shown in Figure 2.6(a), having a close fit throughout the range of takeoff weights investigated. As expected, aircraft with larger masses require larger propellers to ensure enough force is produced to meet the requirements for hover and forward flight. Furthermore, the propeller size also scales very closely with the cumulative wingspan given by Figure 2.6(b). The propeller scaling law is driven by larger propellers being more efficient for forward flight, so it is common to see the largest propeller used when possible. The limitation is commonly set

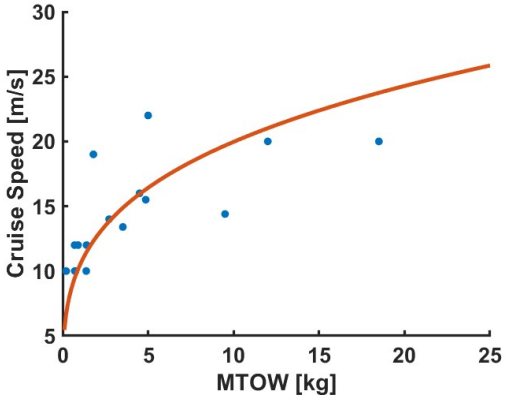


(a) Propeller diameter versus MTOW.

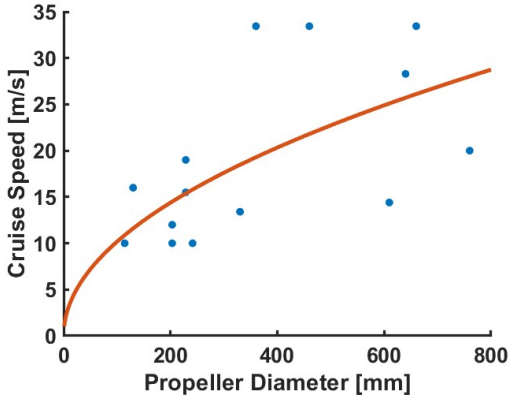


(b) Propeller diameter versus cumulative wingspan.

Figure 2.6: Tailsitter propeller sizes from literature review.



(a) Cruising speed versus MTOW.



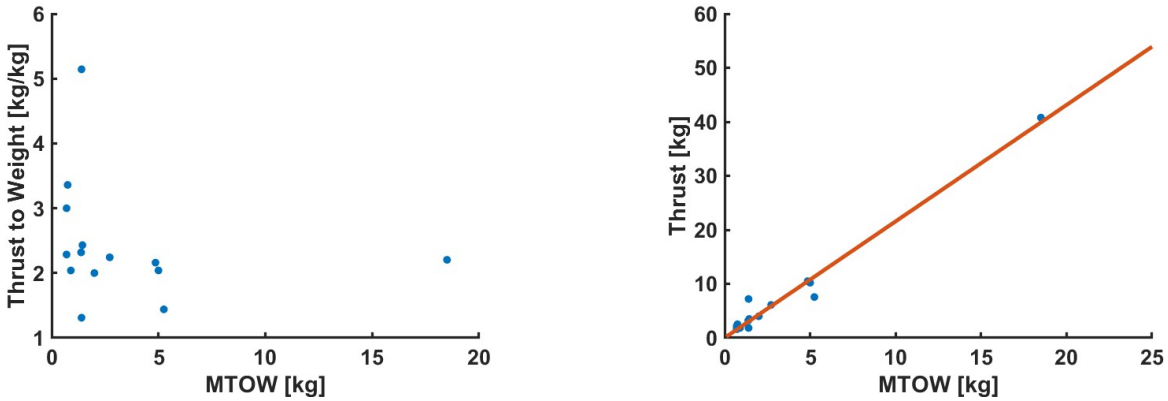
(b) Cruising speed versus propeller diameter.

Figure 2.7: Tailsitter cruising scalability.

by the maximum allowable width of the airframe and rotors, which is usually related to the wingspan.

The cruise speed is primarily a factor of the mission and aircraft size, which in general increases with the MTOW shown in Figure 2.7(a). A reason for the increase in cruising speed is also the increase in propeller diameter and pitch as the aircraft size increases, allowing for dynamic thrust to still be produced at faster velocities. Due to this relationship, the dependency of cruise speed on propeller diameter is shown in Figure 2.7(b). Figure 2.7 shows the limitations of battery propulsion systems for tailsitters as the cruising speeds have diminishing returns when the MTOW and propeller diameter increases.

The thrust-to-weight of the tailsitters investigated is shown in Figure 2.8(a). At a smaller



(a) Thrust-to-weight ratio.

(b) Thrust versus MTOW.

Figure 2.8: Tailsitter thrust properties as MTOW scales.

MTOWs, there is significant variance in the thrust-to-weight ratio. The ratio ranges between 1.3 and 5.1, but the majority are between 2 and 3. The disparity in thrust is likely because tailsitters at smaller scales can be used for a variety of purposes while also having the benefit of being small enough to be tested indoors, enabling the use of lower thrust. However, as the takeoff weight increases, the thrust-to-weight ratio starts to approach a value around 2, since this is typically required to have safe control of the vehicle in hover without being excessively heavy or expensive. The thrust of tailsitters as the MTOW increases is shown in Figure 2.8(b). When the fitted curve is set to have an intercept at the origin, the slope is 2.16 kg of thrust per kg of takeoff weight, showing the average thrust-to-weight ratio for tailsitters.

Each of the scaling laws presented in this section were also given curves of best fit generated using MATLAB's Curve Fitting Toolbox [125]. These curves are all summarized in Table 2.2, including the coefficient of determination R^2, which was calculated using the Curve Fitting Toolbox to give insight into how well the fitted curves represent the data. Each equation takes on the form of

$$Dependant\ Variable = A \times (Independant\ Variable)^B + C$$

2.3 Example Tailsitter Sizing and Weight Distribution

To illustrate how the weights of the various components are expected to scale with takeoff weight, a scalability study with a series of conceptual tailsitters was designed based on the scaling laws determined in Section 2.2. The primary focus of this investigation is to determine the structural weight, which is rarely reported in current literature. The MTOW ranges are from 1 kg to 25 kg to represent tailsitters that are within the typical sizes currently seen in literature. The design methodology for these examples was to first

Table 2.2: Equations for the scalability laws of tailsitters.

Independant Variable	Dependant Variable	A	B	C	R^2
MTOW [kg]	Wingspan [m]	1.394	0.2051	-0.2801	0.7556
MTOW [kg]	Cumulative Wingspan [m]	0.8241	0.5739	0	0.9666
MTOW [kg]	Reference Area [m^2]	0.3719	0.457	-0.1534	0.9057
MTOW [kg]	Chord [m]	-	-	-	-
MTOW [kg]	Aspect Ratio	-	-	-	-
MTOW [kg]	Wing Loading [N/m^2]	-	-	-	-
MTOW [kg]	Propeller Diameter [mm]	149.6	0.4827	44.41	0.7876
Cumulative Wingspan [m]	Propeller Diameter [mm]	206.8	0.8872	0	0.8732
MTOW [kg]	Payload Mass [kg]	0.1188	1.259	0	0.9451
MTOW [kg]	Battery Mass [kg]	0.1425	1.371	0	0.9364
MTOW [kg]	Cruise Speed [kg]	10.43	0.282	0	0.3979
Propeller Diamter [mm]	Cruise Speed [kg]	1.019	0.4996	0	0.3988

define a set of avionics that would be used across all designs to meet the minimum requirements for a quadrotor tailsitter. Then compatible motors, propellers, and electronic speed controllers (ESCs) are chosen, focusing on using voltage requirements and generating a thrust-to-weight ratio of at least 2. Lithium polymer (LiPo) batteries matching this voltage level were then selected, using the expected battery mass to define the capacity. In many cases, multiple motor options could be selected which use different propeller sizes and voltage levels. The motors were first selected that matched the expected propeller size shown in Figure 2.6(a) for the corresponding takeoff weight. For motor options at different voltage levels, the higher voltage was always selected because the total EMP weight was lower, allowing for more mass to be assigned to the battery or structure. However, the higher voltage systems were more expensive, making them ideal if their price is within budget. The remaining mass was assigned to the structural weight, giving useful information for the structural design phase. While this represents the earliest state of conceptual design, the complete process is typically iterated when selecting the electrical components, with a visualization of the process shown in Figure 2.9.

The avionics weight for this study was set to 0.2 kg for all sizes, which represents a standard set of required components. The avionics include the CubePilot Cube Orange flight controller [70], Kore carrier and power distribution board [71], Here3 global positioning system (GPS) [69], Spektrum serial receiver [104] and SiK telemetry radio [122]. The avionics could be made lighter, especially for the smaller concepts, if lighter and smaller flight controllers were used. Conversely, additional sensors for the avionics could be added to larger aircraft, but this can simply be included in the payload weight instead. The motors, propellers, and ESCs were all determined by finding the most appropriately sized combination that resulted in a 2:1 thrust-to-weight ratio, because this is a reasonable value for a tailsitter, as shown in Figure 2.8. All the components were sourced from T-Motor due to the availability of weight and performance data throughout the variety of sizes in-

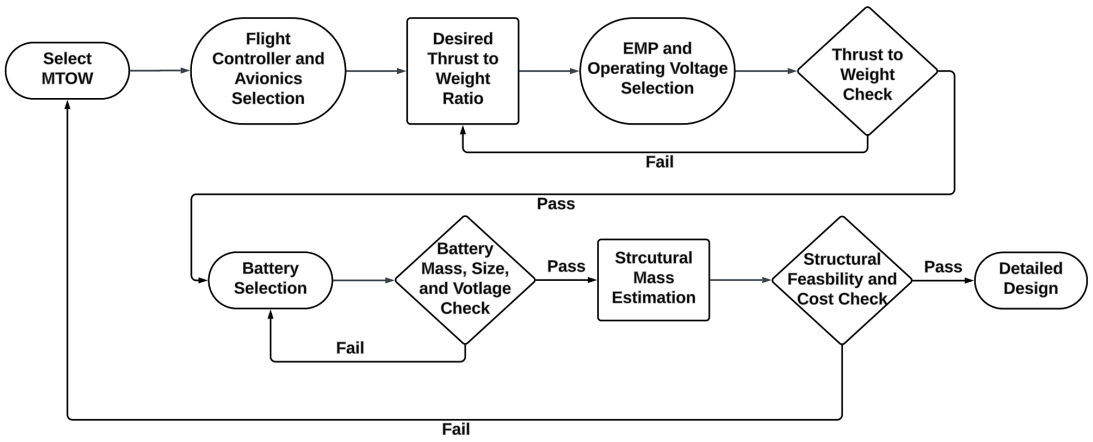


Figure 2.9: Conceptual design loop for component selection using empirical data to define the mass of categories.

vestigated [114]. Although only one brand was used for the power plant of all designs, it is expected that alternative products would offer similar performance, assuming they are operating with current technologies. The battery capacity for this procedure used the energy density of a standard LiPo battery offered by Gens Ace, then used the calculated battery weight to determine the capacity [36]. The mass of the battery for the density calculated was estimated using the battery-to-takeoff weight relationship presented in Table 2.2. The maximum capacity was then reduced by 20 percent to represent the safe usage limitations of LiPo batteries and allow for a short takeoff and landing.

With the avionics weight known, the payload and battery weights calculated from the historical data equations in Table 2.2, and the EMP weights predicted by the selected motors, propellers and ESCs, the structural weight can be determined by subtracting these values from the MTOW for each example. The final weight distributions of each of the tailsitter concepts considered is shown in Figure 2.10 and Table 2.3.

Table 2.3: Example weight distribution of tailsitter RPAS for conceptual designs ranging from 1 to 25 kg.

MTOW [kg]	Payload [kg]	Avionics [kg]	Battery [kg]	EMP [kg]	Structure [kg]
1	0.12	0.2	0.14	0.20	0.34
2.5	0.38	0.2	0.50	0.56	0.86
5	0.90	0.2	1.29	0.82	1.78
10	2.16	0.2	3.35	1.16	3.14
15	3.59	0.2	5.84	1.92	3.45
20	5.16	0.2	8.66	1.90	4.07
25	6.84	0.2	11.76	2.07	4.14

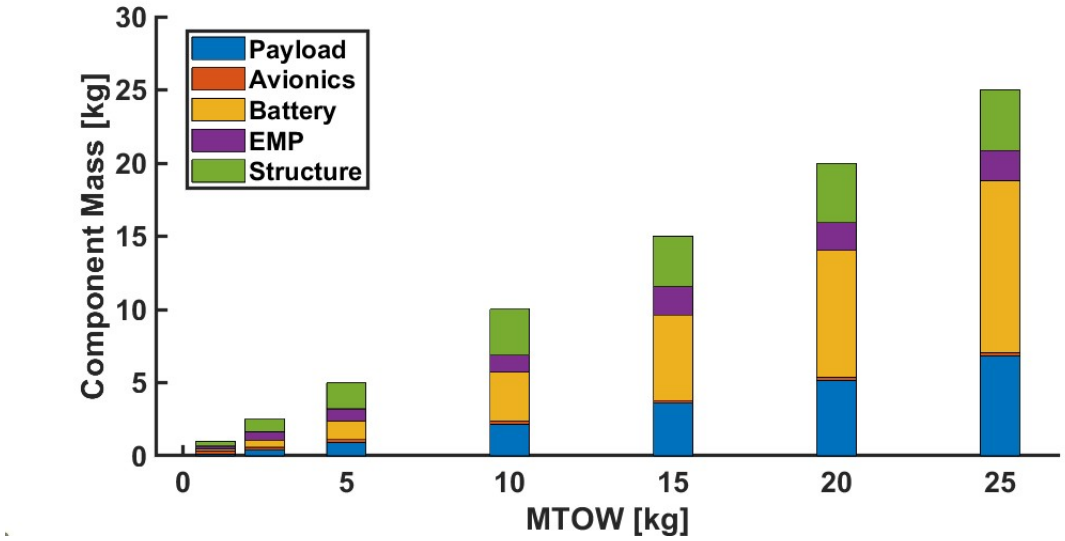
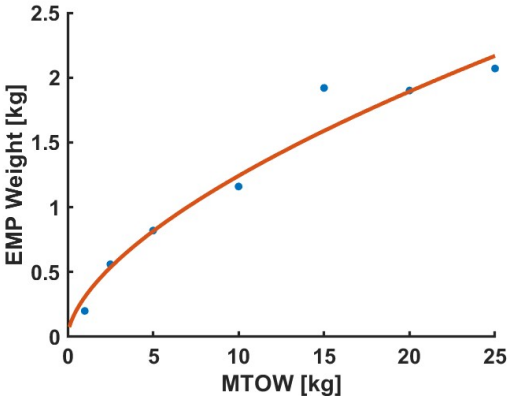


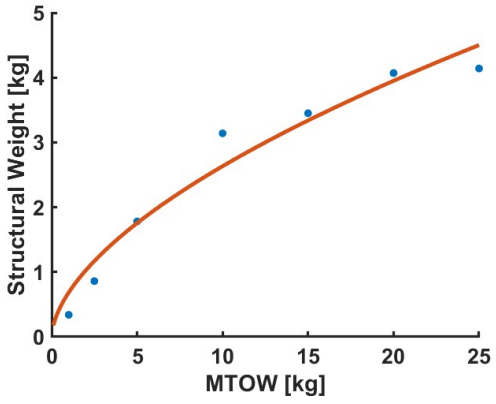
Figure 2.10: Example weight distribution of tailsitter RPAS for conceptual designs ranging from 1 to 25 kg.

2.3.1 Extracted Scaling Laws

From the results shown, for example, tailsitter weight distributions, the combined EMP weight can be plotted versus takeoff weight, as shown in Figure 2.11(a). Additionally, the weights of all components except for the structure are known, which can be subtracted from the MTOW to estimate its scaling law. The structural weight relationship to the MTOW is shown in Figure 2.11(b), allowing for an estimate of the structural weight early in the conceptual design process.



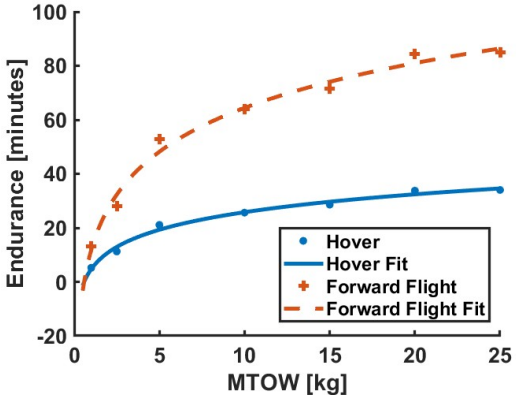
(a) EMP mass scaling with MTOW.



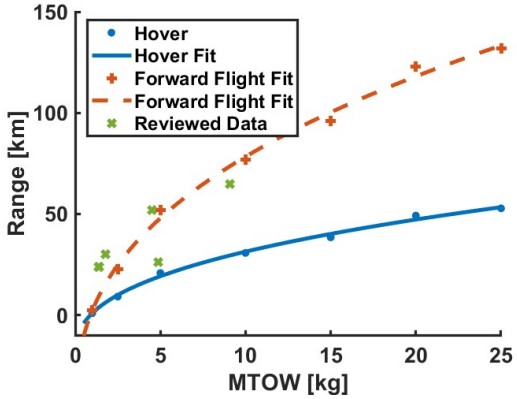
(b) Structural mass scaling with MTOW.

Figure 2.11: Component masses for tailsitters from conceptual design estimates.

The hover times were estimated by using the posted T-Motor data to find how much current is required to produce enough thrust to maintain hover and dividing the available



(a) Hover and forward flight endurance.



(b) Hover and forward flight range.

Figure 2.12: Flight performances of tailsitters based on conceptual design estimates.

current by the current draw. Using the estimated capacity of the battery and the posted performance data for the motors, the endurance and range of the conceptual designs can be calculated. The results shown in Figure 2.12(a) illustrate the flight time that can be achieved by each weight class of tailsitter. The hover efficiency represents the worst scenario and therefore the shortest endurance shown by the blue line. However, forward flight can be used to reduce the energy consumption by 50 to 80 percent, especially with the use of VPP [33, 53, 87, 88, 96]. For this study, a conservative estimate of a 60 percent energy consumption reduction when compared to hover was used. The range for both hover and forward flight shown in Figure 2.12(b) was estimated by multiplying each of the endurance points by the cruising speed for the corresponding takeoff weights previously estimated in Figure 2.7(a). Figure 2.12(b) also has the ranges found when reviewing the literature for tailsitters shown in green, which closely matches what has been predicted, although the data is limited at higher takeoff weights.

The last result obtained from this study was the expected Kv rating for the motors used for the conceptual designs. As shown in Figure 2.13, there is a significant decrease in Kv as the MTOW increases. The result is a rotor system that spins slower for the same input voltage, which is an expected outcome when the propeller size and inertia get larger. The selected rotor system does not necessarily need to follow the trend, but it represents an appropriate source for efficient design and operation specific to tailsitters.

The seven extracted scaling laws are the EMP weight, structural weight, range, endurance, and Kv rating that scale with MTOW. Each of the equations describing their scalability is given in Table 2.4, along with the coefficient of determination to describe the fit. Each of the fitted curves is exponential, taking the form of

$$Dependant\ Variable = A \times (Independant\ Variable)^B + C$$

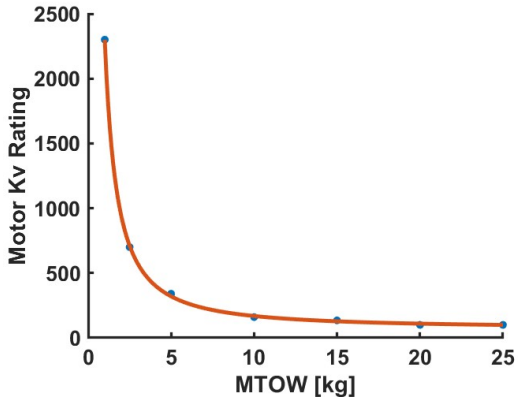


Figure 2.13: Motor Kv used for conceptual tailsitter designs.

Table 2.4: Extracted tailsitter scaling laws for mass distribution and performance estimation.

Independant Variable	Dependant Variable	A	B	C	R^2
MTOW [kg]	EMP Mass [kg]	0.3065	0.6077	0	0.9618
MTOW [kg]	Structural Mass [kg]	0.6841	0.5853	0	0.9561
MTOW [kg]	Hover Endurance [minutes]	306.4	0.0289	-301.7	0.9868
MTOW [kg]	Forward Flight Endurance [minutes]	714.8	0.03084	703	0.9836
MTOW [kg]	Hover Range [km]	21.2	0.3869	-20.34	0.9954
MTOW [kg]	Forward Range [km]	52.97	0.387	-50.83	0.9954
MTOW [kg]	Kv Rating	2227	-1.364	72.18	0.9998

2.4 Prototype Design and Fabrication

To validate the scalability laws presented in Section 2.2, a prototype of about 5 kg was developed as part of this project. The scaling laws presented were used to guide the conceptual design phase, defining key physical dimensions such as battery and structural weight, along with aerodynamic characteristics like wing reference area. A secondary purpose of the thesis is to validate that the blown wing approach used for the 1 kg aircraft presented in Section 1.2 is still effective at larger scales [143].

2.4.1 Requirements

The primary requirement for the developed tailsitter was to utilize the T-Motor MN4014 400 Kv Navigator motors [112], corresponding 15x5 inch propellers [113], and 40 Amp ESCs [111] that were already available at the time of the project. The maximum thrust from this configuration is 2.62 kg per motor, so for a thrust-to-weight ratio of approxi-

mately 2, the MTOW of the vehicle should not exceed 5.24 kg. A vehicle of this size also represents the largest RPAS that can reasonably and safely be developed with the project budget, timeline, and facilities. Additionally, it was decided that the tailsitter should have four rotors and no control surfaces because the author has the most experience with this configuration. Multirotors are very stable and easier to successfully set up for first-time flights than tailsitters, relying on both rotors and control surfaces.

The application that was selected for this prototype was the delivery of medium-sized packages, especially in remote or difficult-to-reach locations. For this reason, it was decided that a large payload bay should be included in the design to accommodate a variety of package sizes. The use of RPAS for medical deliveries is becoming a common trend [105, 108, 149], so the payload bay was designed around a common size for an insulated foam medical cooler with dimensions of 203.2 x 165.1 x 127 mm and a mass of up to 1 kg [132]. Finally, the wingspan was limited to 1 m due to parts such as leading edges or wing spars being commonly sold at lengths of 1 m, and to help with the transportation and logistics of the RPAS.

2.4.2 Conceptual Design

The conceptual design phase starts with sketches that are generally based on aircraft of a similar size or mission type, but modified to include any novel aspects. The novel aspects of this design require the propellers to be positioned between two wings to maximize the blown wing effect. Another factor to consider for tailsitter design is that a rigid multirotor frame is required, and the wings are then mounted to this structure. A rigid multirotor layout is recommended here [9], and is required to ensure the aircraft is stable and controllable while also helping to reduce vibrations. Considering these factors, a basic sketch of the shape of the proposed 5.24 kg aircraft is shown in Figure 2.14, which was made using OnShape [84]. As shown, the wingspan is 1 m to make the aircraft as wide as possible while following the requirements. Additionally, the 15-inch propellers are large enough to significantly cover two wings each, as required by the blown wing claim in the patent being investigated [143]. In the centre is a payload bay with a diameter of 250 mm. This area will have to be designed further, but it is large enough to fit the defined payload in any orientation. The 'X' shape represents carbon fibre tubes that attach the motors securely to the centre area, which will also house the battery, flight controller, sensors, and other electrical components.

With the general shape for the aircraft defined by the sketch, the sizes of major components can be selected by comparing to similar aircraft using the results presented in Section 2.2. Referencing Figure 2.8, the typical thrust-to-weight of tailsitters is around 2, resulting in a maximum thrust of 10.5 kg. By utilizing Figure 2.6(a), which shows the expected propeller size based on takeoff weight, a 5.24 kg tailsitter typically has propellers that are 16 inches in diameter. Therefore, the preexisting configuration of 15 inches will be appropriate for this design, though potentially slightly underpowered. One other design configuration that

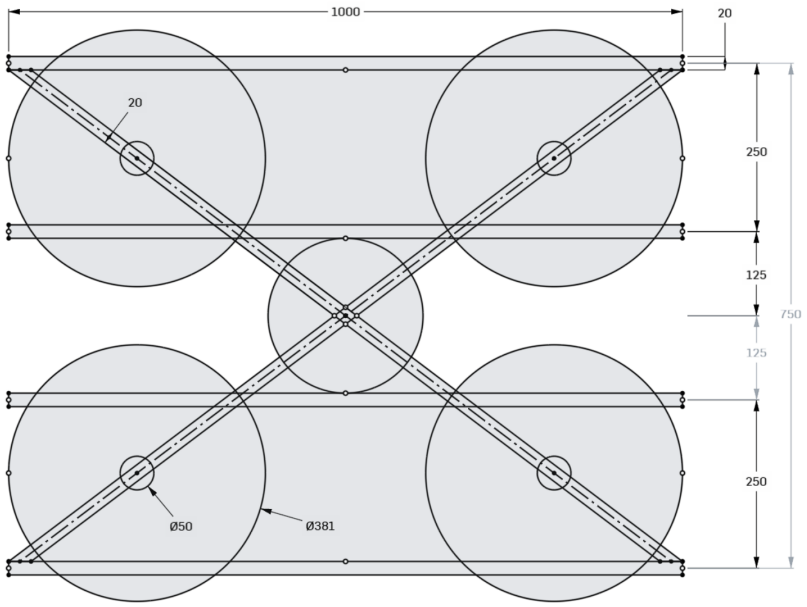


Figure 2.14: Basic sketch showing front view of tailsitter aircraft to be designed further.

is needed early in the process is the estimated cruising speed. From Figure 2.7(a), a 5.24 kg tailsitter is expected to cruise at 16.6 m/s.

2.4.2.1 Wings

The next step in the conceptual design was to define the wing characteristics. A tailsitter with a mass of 5.24 kg will typically have a wingspan of over 1.35 m, obtained from Figure 2.3(a). However, the requirements limit the wings to only 1 m wide, and four wings should be used based on the sketch. The cumulative wingspan for 5.24 kg tailsitters is 2.13 m, given by Figure 2.3(b). Since this design has four wings and most tailsitters have only one or two, the expected cumulative wingspan was doubled to over 4 m, which will require more structural mass to hold the desired shape and smaller chord lengths. The shorter chords are typically a benefit as they result in a higher wing aspect ratio, which is associated with more efficient flight [94]. The expected aspect ratio for a tailsitter of this size is between 6 and 7, as shown in Figure 2.5(a), but the doubling of the cumulative wingspan while keeping the same reference area results in a much larger than expected aspect ratio. However, it is more important to have the proper wing area, as this is required to generate the required lift to achieve flight. From Figure 2.4(a), a 5.24 kg tailsitter should have a reference area of 0.64 m^2. For an aircraft with four 1 m long wings, this results in a wing chord length of 0.16 m, with an aspect ratio of 25. A normalized aspect ratio calculated by using the wingspan and reference area of a single wing rather than the cumulative wingspan and total reference area results in an aspect ratio of 6.25, matching the expected values from Figure 2.5(a).

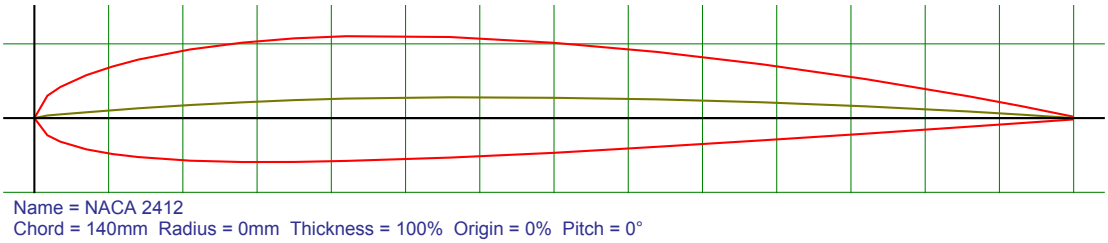


Figure 2.15: Geometry of NACA 2412 airfoil [130].

A simple wing shape configuration was used to make the design and build process easier, similar to other research tailsitters like [17]. The wings had the same chord length at the root and tip and no dihedral, twist, or sweep. However, in the next iteration of this RPAS, an investigation in more effective wing design is recommended. An introduction to wing design is presented in most aircraft or drone design textbooks [55, 94, 98].

A simple design approach was also taken for the airfoils, as detailed aerodynamic analysis was out of the scope and timeline of this project. The major requirements for the airfoils were to be thick enough to enclose the wing spars and any desired electronics. A further requirement for the airfoils was to utilize a NACA shape [26] as they are well defined and understood, have a tool for their shape in the computer - aided design (CAD) software used, OnShape [84], and are supported in the simulation software to be used [95]. The summary of NACA airfoils presented by Cantwell [26] shows that the 4-digit family has good stall characteristics, which is beneficial for the wide range of attack angles experienced by tailsitters, while also being used for general aviation when asymmetric. However, for tailsitters, there is a trade-off in hover and forward flight efficiencies when selecting airfoils: Asymmetric designs provide more lift at smaller AoA, allowing for less drag in forward flight. But the same airfoil will create a lifting force in hover, causing the vehicle to constantly shift in the lifting direction of the wings. The controller will have to account for the drifting effect to maintain position in hover, increasing current draw. Therefore, the camber of the airfoil must be selected based on the mission and the amount of time in hover or forward flight. As this vehicle is meant for payload delivery, most of the mission will be in forward flight, though stable hover and VTOL is required. For all of the above reasons, a NACA 2412 airfoil was selected, whose properties are summarized here [130], with the shape shown in Figure 2.15, and the performance data shown in Figure A.1 in Appendix A.

2.4.2.2 Weight Estimations

The weight of the majority of the aircraft components can also be estimated in the conceptual design phase. Due to the prior selection of the EMP components, their total weight is known to be 0.872 kg. Additionally, the same avionics used in the smaller version of the aircraft will be repurposed here, having a mass under 0.2 kg. The avionics also require

mounting and soldering, so their total mass was increased to 0.2 kg. These components include the CubePilot Cube Orange 2.1 flight controller [70], Kore carrier board [71], Here3 GPS [69], Spektrum serial receiver [104], and SiK telemetry radio [122].

Figure 2.2(a) shows that a 5.24 kg aircraft ideally carries nearly 1 kg, defining the payload weight which corresponds to the desired payload given in the requirements. Additionally, the mass of the battery is estimated to be 1.38 kg from Figure 2.2(b). Therefore, the maximum structural mass is estimated to be the remaining mass of the aircraft:

$$\begin{aligned} M_{structure,max} &= MTOW - M_{battery} - M_{EMP} - M_{avion} - M_{payload} \\ M_{structure,max} &= 5.24\,\text{kg} - 1.38\,\text{kg} - 0.87\,\text{kg} - 0.20\,\text{kg} - 1.00\,\text{kg} \\ M_{structure,max} &= 1.79\,\text{kg} \end{aligned}$$

where M denotes the mass of various components in kg, and the subscripts specify the component. As there is very little data on the structural mass of tailsitters, it is difficult to know if this is a reasonable value. However, with proper design, it seems likely that a structural weight close to 1.79 kg for this aircraft can be achieved, and this result aligns very closely to the structural weight estimated from the extracted scaling laws in Section 2.3.

2.4.2.3 Conceptual Design Review

Before continuing with the detailed design, the basic aerodynamic properties should be reviewed to ensure the results generated by the scalability study have produced a feasible aircraft. First, the required lift of the aircraft can be used to determine the lift coefficient and corresponding AoA for the chosen airfoil.

$$L = \frac{1}{2} C_L \rho v^2 S \tag{2.1}$$

Where L is the total lift in Newtons, C_L is the lift coefficient, ρ is the air density in kg/m^3, v is the airspeed of the aircraft in m/s, and S is the planform wing area. Air density varies significantly with altitude and the resulting pressure, along with temperature. However, a value of 1.225 kg/m^3 is commonly used, which corresponds to 1 atm and 15 °C [129]. The cruising speed of the aircraft is estimated to be 16.6 m/s based on how fast a 5.24 kg tailsitter typically flies in Figure 2.7. Therefore, solving for the lift coefficient gives

$$\begin{aligned} C_L &= \frac{2L}{\rho v^2 S} \\ C_L &= \frac{2(5.24\,\text{kg})(9.81\,\text{m/s}^2)}{(1.225\,\text{kg/m}^3)(16.6\,\text{m/s})^2(0.64\,\text{m})} \\ C_L &= 0.48 \end{aligned}$$

Using the posted data for the NACA 2412 airfoil shown in Figure A.1, a lift coefficient of 0.48 can be achieved at an AoA of about 2 degrees, which is easily achievable for

a standard aircraft. However, for this aircraft, which utilizes the blown wing effect, the resulting increase in airflow over the wings will generate more lift throughout the transition and in forward flight. Therefore, the selected wing area is sufficient to generate the required lift.

2.4.3 Detailed Design

The detailed design involves determining the actual size and shape of specific components required to build the aircraft while also making decisions about the functionality, assembly, and aesthetics. At this stage, it was decided that all parts except for the centre payload bay should be built entirely with removable hardware so the vehicle can be assembled, disassembled, and repaired as required. The payload was positioned at the top of the aircraft when it is placed on the ground so that a customer could easily retrieve the package after landing. Therefore, the bottom of the aircraft was used to house the battery, and the remaining electronics and sensors were positioned surrounding the payload bay. Specifically, the GPS was placed such that it is on the top of the vehicle when in forward flight, so it has a good connection to multiple satellites. The radio receiver and telemetry radios were positioned on the sides and bottom of the aircraft when in forward flight so they have a good connection with the transmitters on the ground. These placements are vital to the operation of the RPAS as carbon fibre attenuates radio waves, reducing the range and performance of the radios. Additionally, separating the GPS, SiK telemetry radio, and receiver with carbon fibre plates helps to isolate their respective signals.

Another design decision at this stage was to make the payload bay an airfoil shape to help reduce the drag of the large area required to house the payload. The middle two wings were also cut where they interfered with the canopy, as there is little usable airflow here. However, this change did not impact the blown wing area or the reference area. Another change was to shorten the top and bottom wings to their respective mounting locations onto the diagonal tubes. This change lowered the structural mass in locations without the propeller downwash, improving airflow. The outcome is a more rounded appearance with better ground clearance when landing vertically. At this stage, the concept sketch was elevated to a more detailed sketch, shown in Figure 2.16, which includes the payload and electronics housing, changes to wing size, and airfoil selection. The design loop presented for the conceptual design was repeated multiple times as decisions were made on different aspects of the aircraft, with the final aircraft parameters listed in Table 2.5. The most significant change is the reduction in wing area, resulting in a higher cruising speed to maintain the required lift.

Next, the carbon fibre tubes that secure the motors and carbon fibre spars for the wings were sized, representing the structural frame for the aircraft. To ensure that the selected parts are sufficient, a simple beam analysis was conducted. For the diagonal tubes that connect the motors to the centre hub, the worst scenario is when the motors are at full power, but the aircraft is not moving, so the centre is fixed in place. The wing spars were

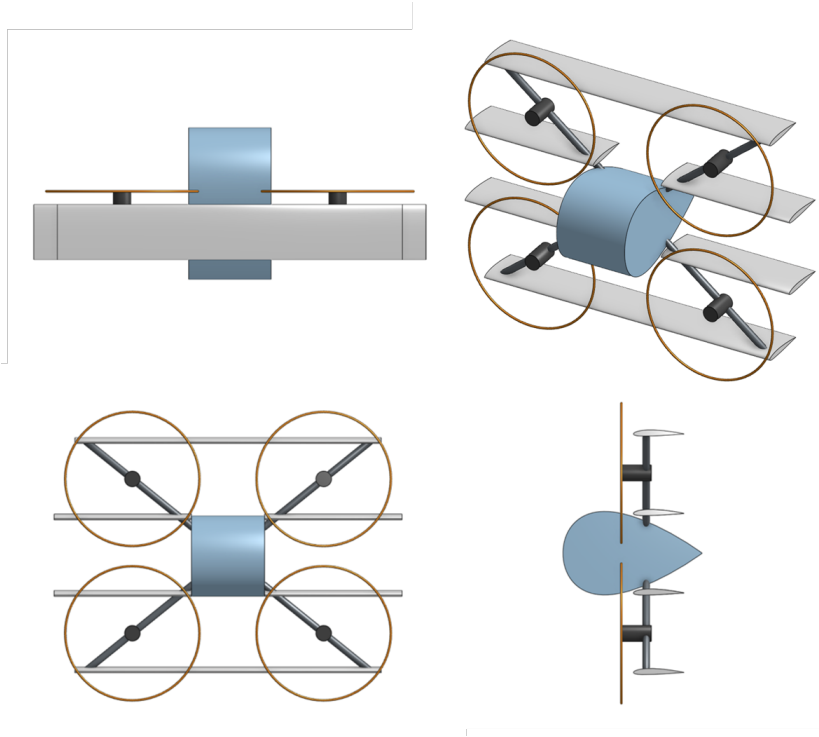


Figure 2.16: Detailed sketch showing front, side, and top views of the tailsitter aircraft designed in this project.

Table 2.5: Final values of prototype aircraft after the detailed design phase.

Parameter	**Value**	**Unit**
MTOW	5.24	kg
Payload Mass	1.00	kg
Battery Mass	1.38	kg
EMP Mass	0.87	kg
Avionic Mass	0.30	kg
Structural Mass	1.69	kg
Wing Chord	0.14	m
Wingspan (top and bottom)	0.88	m
Wingspan (middle)	1.00	m
Cummulative Wingspan	3.76	m
Reference Area (total)	0.53	m$\hat{2}$
Cruiseing Speed	17.60	m/s

also investigated to ensure the materials selected sufficiently prevented deformation. The full analysis for the deformation of both beams is shown in Appendix A.2. An iterative process was used to properly size the beams based on the size, cost, strength, and availability.

For the diagonal mounting tubes, the maximum deformation at the motors was calculated to be 0.71 mm, and 1.2 mm at the wing mounting location. These deformations are very small, especially relative to the 510 mm length of the beam, creating the rigid mount required for the rotors of the aircraft. For the wing spars, the calculated displacement for the long wing is 10 mm, and the maximum deformation of the short wing is 3.9 mm. Both of these wing cases have about 1 percent deflection versus the beam length. Since there are no parts that connect to the wings, these are acceptable results, especially considering their weight. The spars are only 28 g for the long tubes and 13 g for the short tubes. Additionally, when the full wing is constructed, there will be a leading edge, trailing edge, and ribs throughout connecting these to the spar, increasing the moment of inertia and decreasing the deflection further. Another benefit of these spars is that they are thin enough to fit inside the chosen airfoil, even when fully surrounded by a wing rib.

2.4.3.1 Part Selection

To finalize the detailed design phase, the final parts must be selected. At this stage, the dynamic thrust estimations presented in Section 2.4.3.2 indicated that the low-pitch multirotor propellers would not create enough thrust at high angles of attack to achieve the desired airspeed, so they were swapped for higher-pitch APC 10x15 inch electric propellers [90]. For the flight controller, the Orange Cube was used as it has relatively large memory and a fast processor, allowing it to be able to save the additional code required for the custom flight controller that will be used later [70]. The Cube was mounted to a Kore carrier board because it can handle the high-voltage LiPo battery while being lightweight and not requiring an additional power distribution board [71]. The battery selected was a 10 Ah 6S Tattu LiPo battery because it had the largest energy storage while having high enough voltage to operate the required motors and not weighing more than the desired 1.4 kg. The SiK radio and Here3 GPS were both selected as they are relatively light and are common components to use with ArduPilot and the Cube ecosystem [69,122]. The receiver used is the Spektrum SRXL2 DSMX Remote Serial Telemetry Receiver, as it is very light at only 4 g and compatible with existing equipment [104]. The total list of components, their quantity, and their associated weights are listed in Table 2.6. While the total mass of all these components is above the desired empty weight of the aircraft, some steps can be taken in the assembly or future iterations to make components more weight-efficient.

2.4.3.2 Blown Wing Dynamic Thrust and Lift Confirmation

The purpose of this section is to ensure that enough lift and thrust is created by the selected rotor and wing combination to maintain altitude at all angles of attack before

Table 2.6: Masses and quantities of all components required for prototype aircraft.

Item	Quantity	Unit mass [kg]	Total mass [kg]	Source
EMP				
MN 4014 kV 4000	4	0.193	0.772	[112]
APC 15x10E	4	0.045	0.180	[90]
ESC	4	0.026	0.104	[111]
Battery				
Battery	1	1.375	1.375	[48]
Avionics				
Orange Cube Flight Controller	1	0.030	0.030	[70]
Kore Carrier Board	1	0.080	0.080	[71]
Here3 GPS	1	0.052	0.052	[69]
Sik Radio	1	0.025	0.025	[122]
SRXL2 DSMX Receiver	1	0.004	0.004	[104]
Structure				
Diagonal Carbon Fibre Rods	4	0.072	0.288	[78]
Long Wing Spars	2	0.028	0.056	[79]
Short Wing Spars	4	0.013	0.052	[79]
Landing Gear	4	0.005	0.020	[79]
Long Leading Edges	2	0.090	0.180	[18]
Short Leading Edges	4	0.004	0.016	[18]
Long Trailing Edges	2	0.060	0.120	[19]
Short Trailing Edges	4	0.003	0.012	[19]
Long Airfoil Carbon Fibre Tube	4	0.028	0.112	[39]
Short Airfoil Carbon Fibre Tube	4	0.030	0.120	[39]
Carbon Fibre Motor Mount	4	0.004	0.016	N/A
Carbon Fibre Side Plates	2	0.047	0.094	N/A
Carbon Fibre Top/Bot Plates	2	0.039	0.078	N/A
Carbon Fibre Mount Plate 1	1	0.070	0.070	N/A
Carbon Fibre Mount Plate 2	1	0.058	0.058	N/A
Carbon Fibre Rib	36	0.001	0.036	N/A
Carbon Fibre End Rib	4	0.002	0.008	N/A
3D Print Motor Mount	8	0.011	0.090	N/A
3D Print Landing Mount	4	0.013	0.053	N/A
3D Print Landing Foot	4	0.002	0.006	N/A
3D Print Centre Connector	1	0.016	0.016	N/A
3D Print Diagonal Tube Mount	4	0.004	0.016	N/A
3D Print Wing Mounts	8	0.053	0.423	N/A
3D Print Canopy Mounts	12	0.001	0.017	N/A
Kevlar Canopy Parts	4	0.034	0.136	N/A
Total Mass			**4.716**	

moving forward with the design and construction of the prototype. The theory behind the blown wing dynamic thrust calculation is to first estimate how the rotors will adjust the inlet airspeed, using Simple Momentum Theory [56, 57] and Actuator Disk Theory [103]. Then the AoA and Reynolds number are used to estimate the lift coefficient of the blown wing to calculate the produced lift. The lift combined with the upward thrust of the rotors gives the total aircraft lift at all AoA. Note that throughout this section, a 0-degree AoA refers to forward flight, while 90 degrees refers to hover to better align with what is typically presented in literature. In both Simple Momentum Theory and Actuator Disk Theory, if the inlet velocity, v_i, and propeller thrust, T, are known along with the air density, ρ, and disk area, A, the resulting exit velocity, v_e, can be calculated [56, 57, 103].

$$v_e = \sqrt{v_i^2 + \frac{2T}{\rho A}} \tag{2.2}$$

Another benefit of blown wings that must be included in this calculation is their ability to postpone wing stall by encouraging the airflow to stick to the wings at higher AoA [16, 21, 110]. Furthermore, the more attached airflow also increases the lift produced, especially at higher AoA. These impacts are modelled by using trigonometry and component vectors to add the speed of the propelled airflow with the perpendicular airflow to generate a smaller AoA for the blown wing sections.

The only other required component for these calculations is the lift coefficient for wings at AoAs past stall, as these are not typically posted. However, with tailsitters becoming more common vehicles, especially in the research space, multiple examples of work investigate wing performance at high angles either through simulation [63], or wind tunnel tests [31, 62, 76]. In all cases, these reports show that after the initial stall and decrease in lift coefficient, the coefficient then increases again before slowly dropping to zero around 90 degrees. One example of this effect is shown for a NACA 0012 airfoil in Figure 2.17, as presented by [63]. To accommodate for the high-angle effects, the lift coefficient curve for the NACA 2412 airfoil was modified from the data shown in Figure A.1, with the full range of coefficients shown in Figure 2.18. The modifications include extending the represented angles to 90 degrees to represent the lift in all flight modes. The lift coefficients at these angles are set as estimates that follow the same trends given in Figure 2.17 since both are four-digit NACA airfoils. It should be noted that this process is just used to estimate lift at very high angles of attack, but it does not have a significant impact on design because at this point, the majority of lift is produced by the rotors that are facing upwards.

The thrust from the rotors was estimated for every 1000 revolutions per minute (RPM) in the operational range since this was the data available. For this analysis, the AoA was modulated from 90 to 0 degrees, representing hover to forward flight. A linear relationship between the angle of attack and flight speed was used, as shown in Figure 2.19. While the true relationship is non-linear, the linear approximation results in a faster airspeed estimate through the transition. Therefore, the estimated approach results in poorer per-

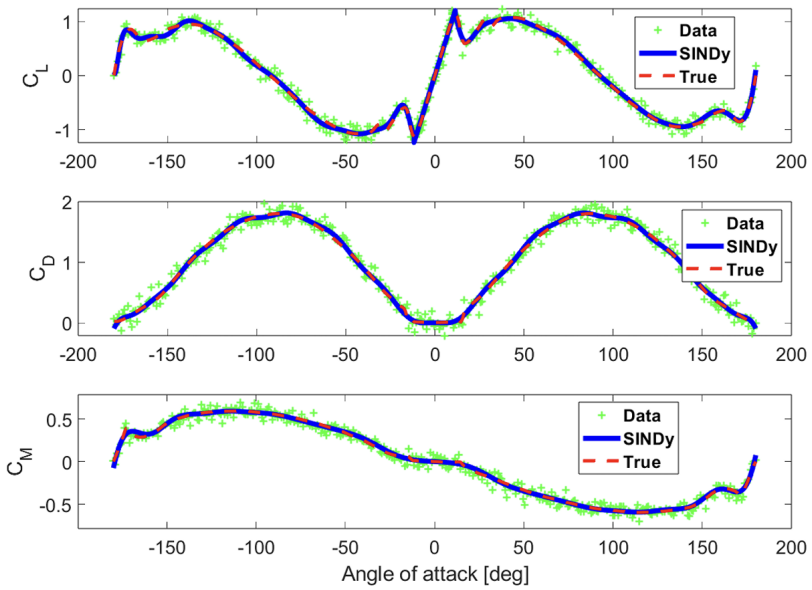


Figure 2.17: Aerodynamic characteristics of NACA 0012 airfoil for all AoA [63].

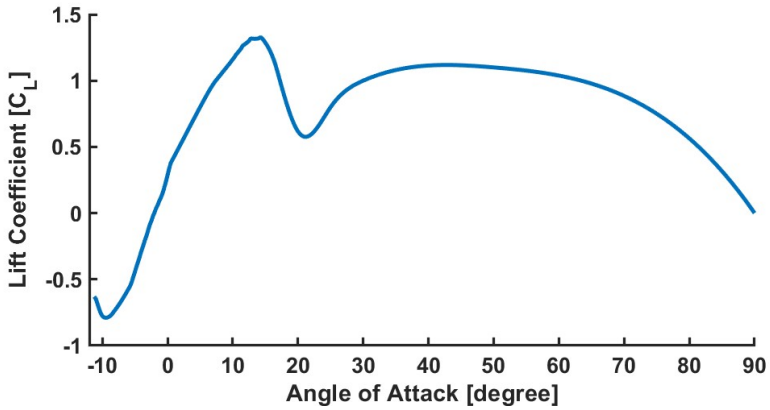


Figure 2.18: Aerodynamic characteristics of NACA 2412 airfoil for AoA between -11.25 and 90 degrees.

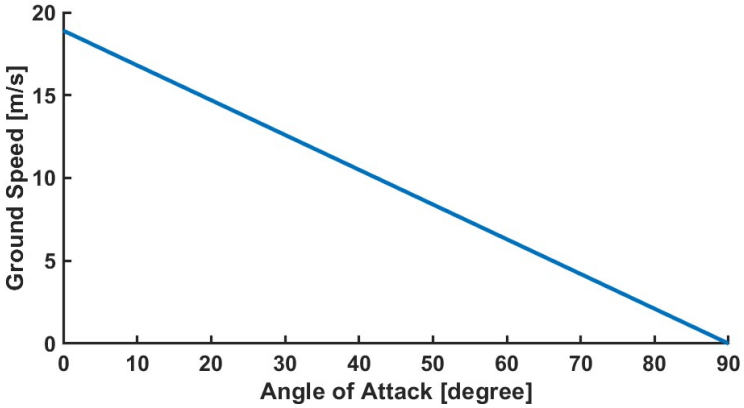


Figure 2.19: Linear relationship between horizontal speed and AoA for dynamic thrust and lift estimation.

formance by the propellers because of the dynamic thrust force losses and more airflow not aligned with the wings, reducing lift. The result is that more thrust from the motors would be required to maintain altitude, giving a safer estimate. For each iteration, the dynamic thrust is used to calculate the maximum achievable forward flight speed, which is then used to calculate the lift of the wings.

The calculation process for the dynamic thrust and resulting lift is as follows. First, the free stream airflow, v_0 is split into components going directly into the rotors, v_{0_i} and the perpendicular airflow that passes by, v_{0_p}, based on the AoA, α.

$$v_{0_i} = v_0 cos(\alpha)$$
$$v_{0_p} = v_0 sin(\alpha)$$

Next, the dynamic thrust, T_d, is estimated by linearly interpolating the posted propeller data based on the current RPM being investigated, and v_{0_i}. With the dynamic thrust known, the airspeed leaving the rotor can be calculated based on Equation 2.2.

$$v_e = \sqrt{v_{0_i}^2 + \frac{2T}{\rho A}}$$

The exit velocity from the propeller is then combined with the pass-through velocity not impacted by the rotors to calculate the resulting rotor airspeed magnitude that blows over the wings, v_b, and direction, θ.

$$v_b = \sqrt{v_e^2 + v_{0_p}^2}$$
$$\theta = sin^{-1}\left(\frac{v_{0_p}}{v_b}\right)$$

Then the Reynolds number is calculated for both the standard and blown wings to help compare to the posted airfoil data [130] and the updated lift coefficients to include high angles of attack in Figure 2.18. Using these charts, the standard wing and blown wing lift coefficients are estimated, as C_{L_s} and C_{L_b} respectively. The equation for the Reynolds number is

$$Re = \frac{vC}{\nu}$$

where v is either the standard or blown velocities, C is the aerodynamic chord, 0.14 m, and ν is the kinematic viscosity, which is set to $1.48 \times 10^{-5}\,\mathrm{m}^2/\mathrm{s}$ for an air density of $1.225\,\mathrm{kg/m}^3$ [129]. Using the standard and blown lift coefficients and airspeeds, the respective standard wing lift, L_s, and blown wing lift, L_b, can be calculated using Equation 2.1. However, the blown wing lift needs to then be adjusted to match the angle of attack of the standard wing and the rest of the aircraft, such that the component of lift is in the right direction:

$$L_s = \frac{1}{2} C_{L_s} \rho v_0^2 S_s$$
$$L_b = \frac{1}{2} C_{L_b} \rho v_0^2 S_b cos(\alpha - \theta)$$

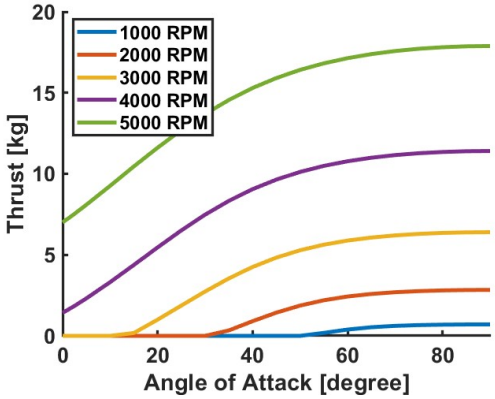
where S_s and S_b are the standard and blown wing areas, based on the leading edge of the wing covered or uncovered by propellers. Finally, the lift of the standard and blown wings can be added to the vertical thrust of the four rotors to find the total lift, L_T, at any angle of attack.

$$L_T = L_s + L_b + 4 \times T_d sin(\alpha)$$

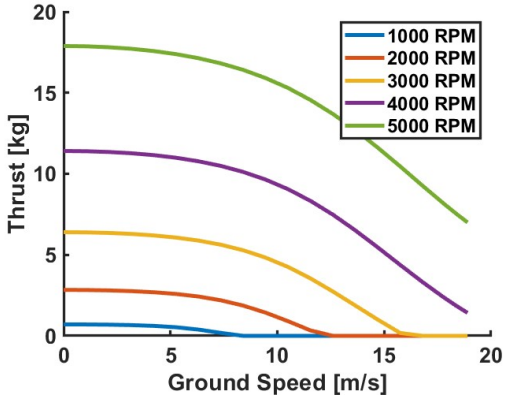
From these calculations, the dynamic thrust of each rotor based on the AoA, and the corresponding horizontal aircraft speed were estimated, as shown in Figures 2.20(a) and 2.20(b) respectively. The RPM required to create enough lift to maintain altitude at all flight angles is shown in Figure 2.21, with the orange line showing the maximum RPM of the motors [112]. The results show that the motors must spin at about 4200 RPM to maintain hover. The required throttle then reduces until 25 degrees, where there is a large increase due to the decrease in wing lift in the 10 degrees after stall. There is also a significant drop in required throttle between 10 and 20 degrees, representing when the standard and blown portions of the wings are in their nominal operating conditions. Figure 2.21 validates that at all AoA between hover and forward flight, enough lift can be created with the current rotor configuration to maintain altitude and control of the aircraft.

2.4.4 Prototype Assembly

With the design phase complete, the next steps were to manufacture and assemble all components. The entire design and assembly process was conducted by the author, with



(a) Dynamic thrust based on AoA.



(b) Dynamic thrust based on ground speed.

Figure 2.20: Dynamic thrust of an APC 15x10 inch propeller at various rotational velocities between 1000 and 5000 RPMs.

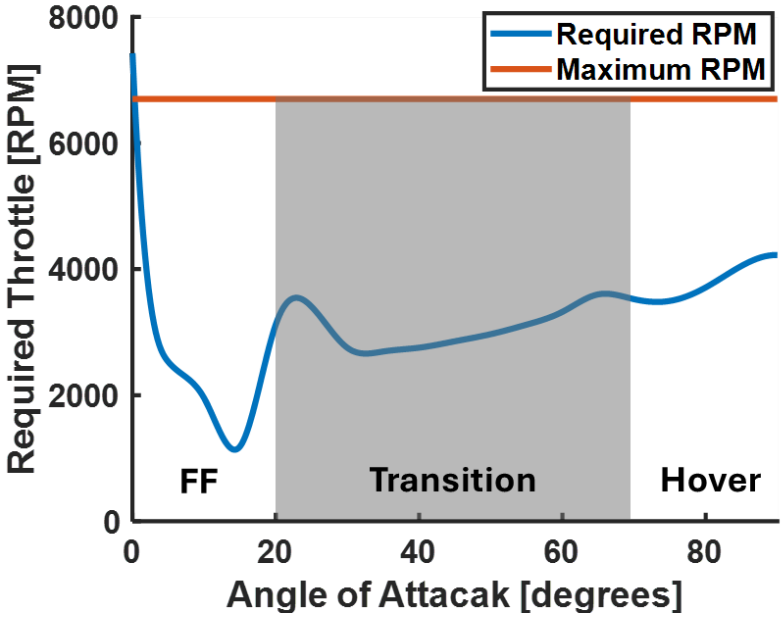


Figure 2.21: RPM required to achieve all AoA in the flight envelope.

guidance from the supervisors and funding from the National Research Council (NRC). A summary of the process is presented in this section, but more detail with images is shown in Appendix B. The majority of parts were sourced as off-the-shelf components, but the carbon fibre plates were cut with a CNC, and the remaining parts were 3D printed using a Bambu Lab X1C [133] with carbon fibre reinforced PLA material [134]. The carbon-reinforced material was used due to its superior mechanical properties while also reducing the effects of layer lines. The parts that were 3D printed include

- Motor mounts

- Landing gear mounts
- Landing gear feet
- Wing mounts
- Diagonal tube holder
- Diagonal tube centre mount, and
- Various mounts for the canopy

The process for assembly was to first build the centre structure, then mount everything required for a multirotor, including all electronic components, as shown in Figure 2.22. The multirotor components were then configured using the first-time setup of the ArduCopter guide as defined here [13], to ensure everything was operating nominally.

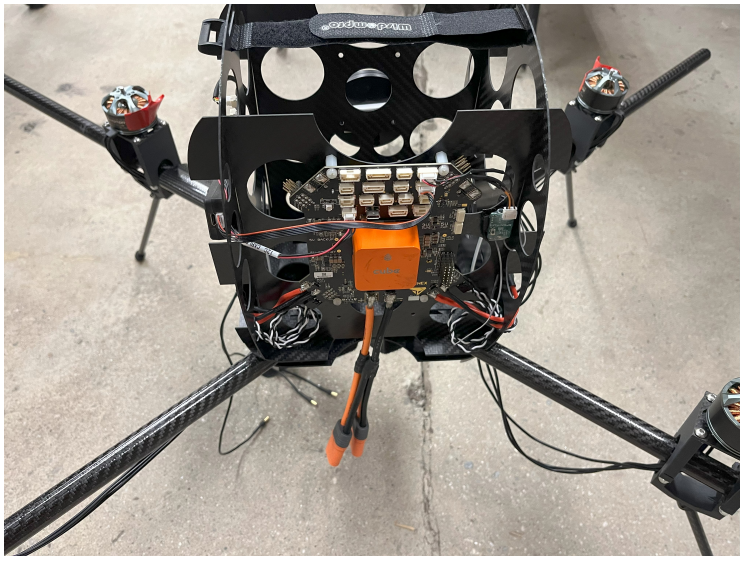


Figure 2.22: Centre hub of prototype aircraft with flight controller, electronics, ESCs, and motors mounted.

The wings were assembled next by gluing the carbon fibre ribs to the carbon fibre spars. Balsa leading and trailing edges were mounted onto the ribs, followed by sanding them to continue the airfoil shape of the ribs. The completed assembly of a short wing segment is shown in Figure 2.23, which was then wrapped with Hanger 9 Transparent Ultracoat polyester covering to create the wing surface [67].

The assembly was completed by mounting the wings to the diagonal tubes and securing them with bolts. The propellers and kevlar canopy cover were also mounted, completing the prototype assembly as shown in Figure 2.24(a). The top section is removable to access the payload bay for delivery missions, as shown in Figure 2.24(b).

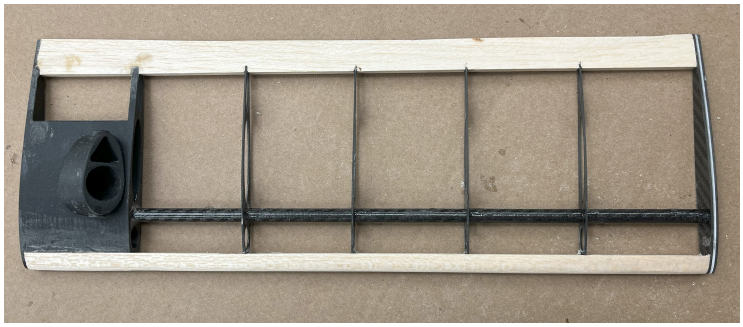


Figure 2.23: Short wing segment showing carbon fibre ribs and spar, balsa leading and trailing edges, and 3D printed mount.

(a) Front view showing wings and canopy cover.

(b) Payload access in opened bay.

Figure 2.24: Photos of fully assembled prototype aircraft.

Chapter 3

Software in the Loop Simulation

The use of simulation environments for RPAS development is beneficial as the system can be tested before prototyping and without damaging the vehicle. This chapter describes the development of a software in the loop (SITL) simulation that combines both the physics and dynamics of the aircraft with the flight controller [123]. The physical environment for the aircraft was simulated using RealFlight 9.5, which is a model aircraft simulator with an industry-leading physics engine [95]. The flight controller simulation was conducted using Mission Planner [4] and used the ArduCopter flight controller software [6]. ArduPilot is an open-source autopilot that has been developed to work for a variety of remotely controlled vehicles, including multirotors and fixed-wing aircraft [8]. It can be used on the hardware previously described to control the real aircraft, while SITL allows the modifications to the flight controller to be simulated.

3.1 Simulation Validation

The first step to verifying the designed RPAS through the simulation is to ensure that the methodology used is valid. When using Mission Planner, the exact same code that would be implemented on the prototype is used in the SITL simulation, ensuring that the flight controller would respond the same way to the same situations. Therefore, if the RealFlight simulation can accurately represent the physics of the prototype, the simulated and real flight controllers will produce the same results.

RealFlight has also been used to verify the design of aircraft, such as in [54]. In one case, it was used to determine the difference in performance of an aircraft as the wing length was adjusted. [89] illustrates how the physics model can be used to evaluate change the aircraft operation based on parametric inputs. Another example shows how the same method of RealFlight simulation in combination with Mission Planner for SITL was used to verify the design of a VTOL tiltrotor RPAS, which has a very similar functionality to the aircraft discussed in this thesis [30].

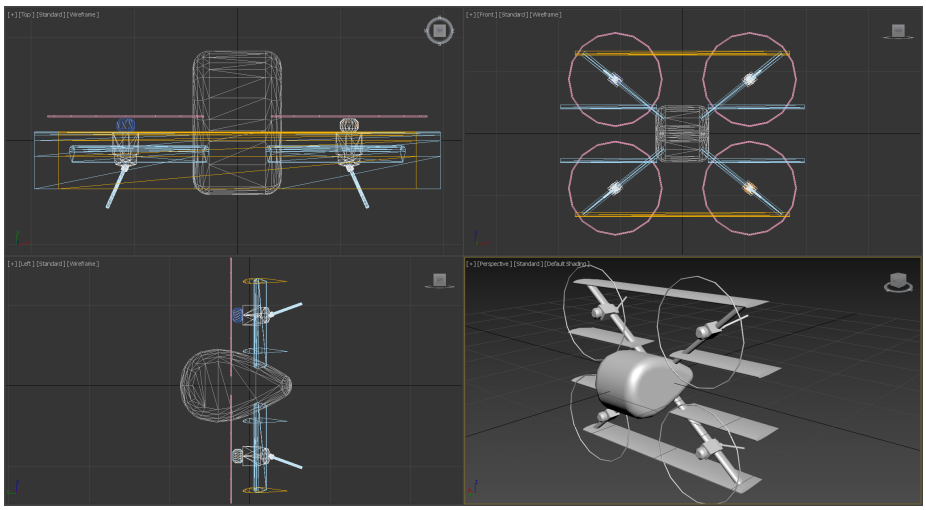


Figure 3.1: Geometry and orientation used for visualization in the SITL simulation.

3.2 RealFlight Simulation Configuration

A RealFlight 9.5 simulation requires two main elements to function correctly. The first is a visual model of the aircraft, which is displayed in the physics environment so the aircraft motion and overall size can be observed. A physics model is also required, which uses the visual model to determine the scale of the components. In addition, all the interactions between the aircraft and the simulated world are calculated based on the physics model. The simulated electronics are also configured within the physics to send the desired control signals to each rotor. A recent post on the ArduPilot forums detailing this procedure for a quadplane and using alternative software is given here [150], and Appendix D gives the specific details used for this thesis.

3.2.1 Visual Model

The CAD for the prototype aircraft was developed using OnShape, then the major aerodynamic sections were imported into 3DS Max 2020 as an ACIS file for creating the visual model [60]. 3DS Max was used instead of OnShape as a program is required that can assign a TGA colour file to the geometry, along with the linking of the components, which is defined in RealFlight how each section of the vehicle eventually connects to the fuselage. The visual model for the aircraft in 3DS Max 2020 is shown in Figure 3.1. The vehicle is intentionally oriented as shown to align with what RealFlight 9.5 and ArduCopter expect for the SITL simulation.

3.2.2 Physical Model

The FBX (From Filmbox) file of the aircraft was then imported into RealFlight by navigating to Simulation > Import > 3D Model (FBX, KEX). The physics of the simulated

vehicle was then configured, starting with the airframe, allowing for the adjustment of the centre of gravity (CG). The battery was configured as a 10000 mAh LiPo, matching what was used on the prototype [36]. The aerodynamic components that represent most of the vehicle were then added to the model, using the visual model to help define their shapes, and masses were set from manufacturer data or measurements taken from the prototype. The wings were then each added as a child component to the fuselage and shaped to match their visual frame. The large wings had a mass of 0.18 kg, while the small wings had a mass of 0.095 kg, matching the weight distribution from the CAD model. Additionally, the wing root and tip were both set to the NACA 2412 airfoil with a constant chord of 140 mm to represent the prototype aircraft. The final components to configure are the engines, which represent the motors and propellers. Each engine has a throttle servo assigned, which is used to send the pulse width modulation (PWM) signal from the flight controller to the motors. Additionally, the propeller diameter, pitch, and type are assigned here, which were set to match the APC 15x10 inch electric propellers used [90]. A custom torque generator was used to match the performance presented by [91] and [112]. The rotation direction was configured so that motors 1 and 2 spun clockwise, while 3 and 4 spun counterclockwise, as expected by ArduCopter [119]. The vehicle tab was used to confirm that the total mass, wing loading, and stall speed are close to the expected values throughout the setup procedure.

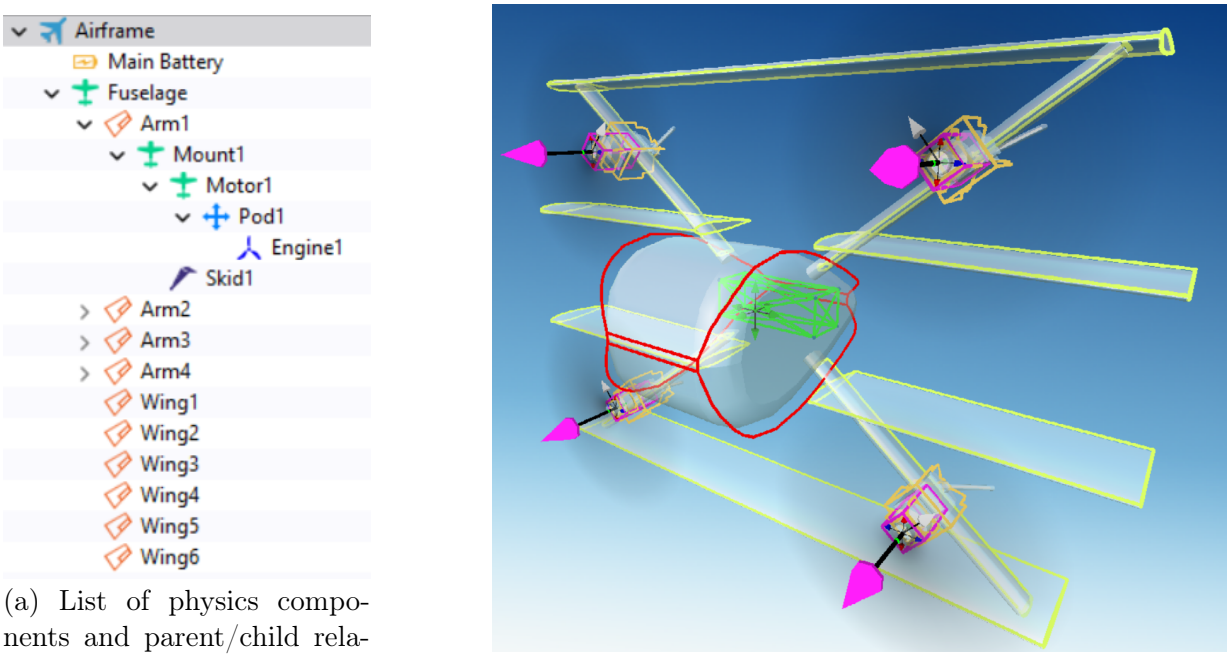
The list of components and their parent and child relationships is shown in Figure 3.2(a). The physical simulation includes the payload bay, diagonal airfoil arms, wings, landing gear, and motors. Additionally, the visualization for the physics of the configured components within RealFlight 9.5 is shown in Figure 3.2(b).

3.3 Software in the Loop Simulation

The ArduCopter flight controller software used and modified in Chapter 4 can also be simulated using SITL [123]. The SITL feature enables the ability to thoroughly test the flight controller capabilities by running the exact same code as implemented on a real aircraft. In this section, a summary of the setup procedure for the SITL simulation used to test the autopilot and physics of the vehicle configured in RealFlight 9.5. While the ability to test SITL just using Mission Planner and RealFlight 9.5 is possible, the presented method allows for an efficient workflow that includes modification of the ArduPilot code. The complete procedure for creating a SITL simulation is described in Appendix D.2.

3.3.1 Requirements

The methods for modifying the ArduPilot code discussed and presented in the forums and tutorials are mainly accomplished using Linux. To utilize the same system in Windows 10 or 11, the Windows Subsystem for Linux (WSL) is required [81]. This allows for the Windows functionality to be used while still being able to access Linux applications. A Linux



(a) List of physics components and parent/child relationships.

(b) Visualization of simulated physics.

Figure 3.2: RealFlight 9.5 completed physical model.

distribution is also required, such as Ubuntu [68]. ArduPilot recommends using Eclipse, NotePad++, VSCode, or Atom to modify the code as they have been utilized previously with the ArduPilot project [12]. A recommendation of the author is to use VSCode due to the ability to connect directly to WSL and utilize the Ubuntu terminal within the editor [80].

To verify the usage and performance of any modified code in a SITL simulation, the use of Ubuntu following the documentation to set up SITL on Windows [123] is required. Connecting the SITL simulation to a ground control station (GCS) such as Mission Planner enables testing of the software that is used when flying the real aircraft [4]. The use of a visualization and physics-based simulation programs such as RealFlight improves the usability and realism of the simulation [12, 95].

3.3.2 MATLAB and Simulink attitude controller

The C++ code generation tool in MATLAB and Simulink [126] was utilized to create new flight controllers for ArduPilot, following the method described in [7]. The Simulink model must be configured to discrete fixed steps of 0.0025 seconds (400 Hz), representing the refresh rate of ArduCopter's main code. The device vendor in hardware implementation must be changed to ARM Compatible, and the device type set to ARM Cortex-M. Each of the inputs, outputs, and blocks throughout the controller must have their data type set to single for the code to compile. It should be noted that only discrete time blocks can

be used. In the case of PID controllers, the PID block can be changed to discrete time by opening the block parameters and selecting discrete time under the time domain section.

With the code for the custom controller generated, it needs to be added to the ArduCopter codebase so it can be used in the simulation and on the prototype aircraft. This process was crucial to the SITL simulation workflow and testing various controllers. However, the actual changes made to the code to create the custom controller are presented in Chapter 4. The process for adding a custom attitude flight controller to ArduCopter is outlined here [7], and described in detail for this thesis in Appendix D.2.5.3. The steps include copying the code generated by MATLAB and Simulink into ArduPilot's code, then enabling the use of custom controllers, and compiling the new code to be used in the SITL simulation.

3.3.3 Connecting Mission Planner and RealFlight to SITL

The benefits of using a SITL with a ground control station, such as Mission Planner and a physics model and visualizer, such as RealFlight, are twofold. First, the use of a ground control station in the simulation allows the user to learn how to set up physical hardware, since the user interface and techniques are the same. This includes setup procedures, configuring the ArduPilot settings, and interacting with the heads-up display. Second, the ability to visualize how the aircraft is flying and then compare it with the heads-up display will allow the user to learn how they are related. Additionally, physics-based simulations like RealFlight can be used to simulate very specific aircraft, granting the ability to test the devices that will be implemented. To use RealFlight 9.5 as the physics simulation for SITL with Mission Planner, a few settings must be changed as outlined in [15] and described in detail in Appendix D.2.6. A screenshot showing the RealFlight 9.5 and Mission Planner SITL simulation of the prototype tailsitter is shown in Figure 3.3.

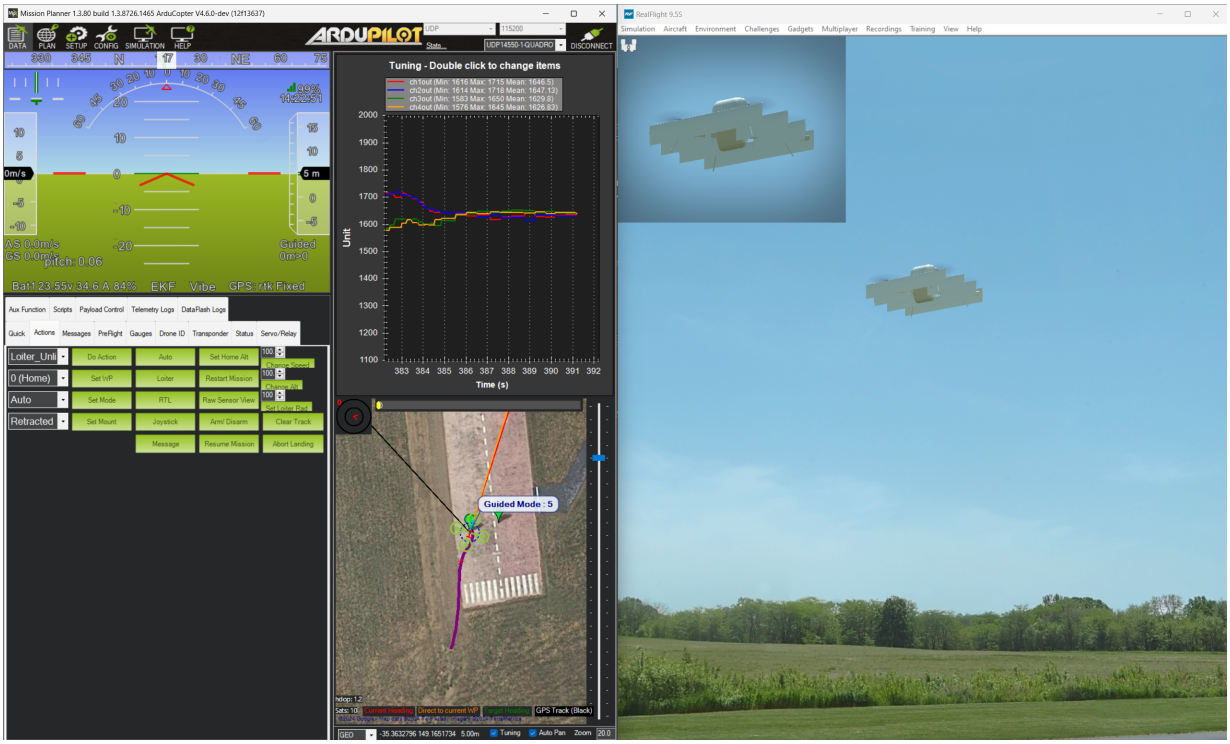


Figure 3.3: SITL simulation with the Mission Planner app on the left and RealFlight 9.5 on the right.

Chapter 4

Control

The precursor to the prototype developed as part of this thesis is the LaBWing Mini, developed at the NRC of Canada, and is shown in Figure 4.1. The project was completed 1 year before the start of the work presented in this thesis. The aircraft utilizes multiple horizontal wings, which are positioned to maximize the wing area in the downwash of propellers. The blown wings are used to prevent the wings from stalling, so they produce more lift during the transition period. However, the vehicle was controlled using the proportional (P)+PID attitude controller from ArduCopter, which resulted in severe oscillations when in forward flight. Therefore, the goal of this chapter is to implement a flight controller that improves performance in forward flight while using a single flight mode to achieve all attitudes of tailsitters to fully utilize the benefits of the blown wing technology.

There exists significant literature on the control and testing of multirotor aircraft, but much less applied to quadrotor tailsitters. Common multirotor control techniques that can be applied to tailsitters are first reviewed, such as those designed for greater agility, since the physical structure and actuators are similar to the aircraft developed in this thesis. Next, control methodologies applied to tailsitters with two rotors and control surfaces are investigated, as these vehicles have similar mission profiles, although they achieve their control with different actuators. The control of quadrotor tailsitters is then investigated, highlighting gaps in the current literature. The findings presented for the controllers in this section are a summary of the detailed explanation presented in Appendix E.

4.1 Multirotor Control Review

For multirotor UASs, many off-the-shelf solutions exist for their control. One prominent example is the open-source ArduCopter suite produced by ArduPilot [6]. The benefit of this type of flight controller is that it can be adapted based on the specific needs of each aircraft. For example, the number of motors, their orientation, sensor usage, speed and attitude limits, and more can all be activated and utilized depending on the parameters set in a GCS such as Mission Planner or QGroundControl [4,41]. However, since this needs to

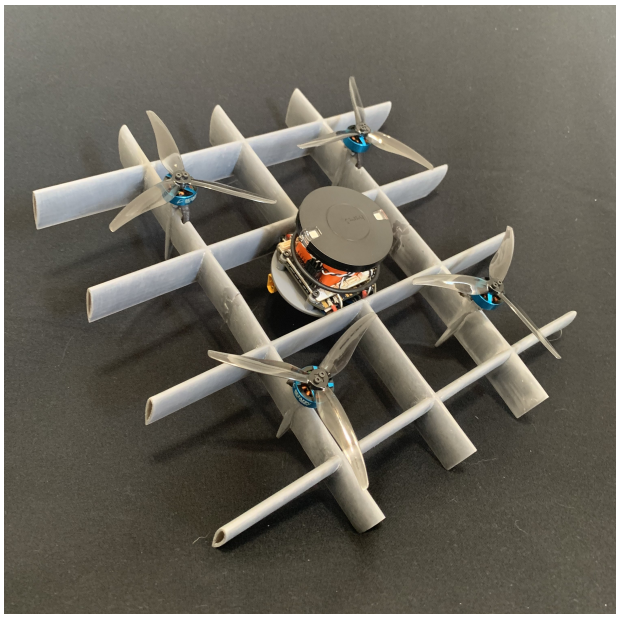


Figure 4.1: The LaBWing Mini quadrotor tailsitter developed at the NRC of Canada.

work on a vast range of aircraft, it uses a generic P+PID attitude control method to send PWM signals to the motors for their actuation [3]. Similar off-the-shelf flight controller hardware and firmware combinations include the Dronecode PX4 Pixhawk system, KISS by Flyduino, Betaflight, Cleanflight, and Arduino [2, 23, 34, 42, 46].

Many methods have been used to improve the performance of multirotor vehicles compared to standard PID techniques that are often applied. Huang et al. combined the aerodynamic effects of quadcopters, including blade flapping and thrust variation due to flight speed, with a PID controller [59]. The added dynamics improved performance at high attitude angles and flight speeds, similar to the transition and forward flight portions of a tailsitter's mission. A 6-degrees of freedom (DOF) state space model of a quadcopter was made by Tahir et al., controlled by a linear quadratic regulator (LQR) utilizing feedback regulation [115, 116]. The authors note that the gains of the controller can be adjusted to make the aircraft more or less agile, and also found that in noisy environments, the implemented controller is impractical. A state space model considering quadcopter kinematics, dynamics, and the Coriolis effect was developed by Tengis and Batmunkh [124]. The aircraft was controlled using full state feedback with pole placement to define the feedback gain matrix, and simulated using MATLAB. Srinivasan developed a detailed state space model of their quadcopter by relating the battery voltage and PWM signal of the ESC to the rotational velocity of the rotor [107]. Srinivasan measured the thrust and torque at different rotational velocities, along with measuring the moments of inertia of the vehicle. A cascaded feedback system with an optimized LQR was used for the attitude and path-following control. The work presented by Abbasi et al. also improved upon the use of PID controllers for multirotor attitude control with the addition of fuzzy logic to adapt the gains [1]. The authors used MATLAB to develop their FLC to help the aircraft reach

the hover state more quickly while remaining stable.

A basic mission requirement for a tailsitting aircraft is to rotate 90 degrees around the pitch axis to transition from hover to forward flight, which can result gimbal lock if using the standard Euler angle representation. One solution to this problem is to represent the aircraft attitude using Quaternions [82, 86]. Fresk and Nikolakopoulos presented a flight controller for quadcopter attitude that utilizes Quaternions [47]. A simple model was used for a rigid body quadcopter with Euler-Newton dynamic equations. The aircraft was then controlled through a feedback loop with two proportional controllers (P^2), and was able to manoeuvre the aircraft through a flip without experiencing gimbal lock. Zhao et al. used quaternions for trajectory and yaw control of a quadcopter which utilized a command filtered backstepping technique [146]. The authors comment on how linearizing around hover results in good control at the designed point, but exhibits worse performance as the aircraft's attitude gets further from hover. Cariño et al. also designed a quaternion-based control scheme that focused on the position of a quadrotor rather than attitude [29]. The controller developed was a LQR that was proven to be linearly and exponentially stable, even when in the presence of wind.

4.2 Tailsitter Control Review

The tailsitter controller review is separated into three primary categories. Controllers that can be implemented for a wide range of tailsitters, controllers that focus on aircraft with control surfaces, and control strategies specifically applied to copter tailsitters, which is the same aircraft type described in this thesis. The section concludes with a comparison of the strengths and weaknesses, along with highlighting potential gaps in the literature. A detailed summary of each method is presented in Appendix E.2.

4.2.1 Universal Tailsitter Controllers

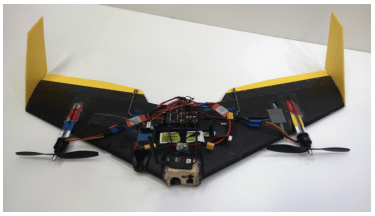
There are multiple commercial solutions available for tailsitting RPAS control, although fewer for specifically copter tailsitters. One example includes the ArduPlane flight controller by ArduPilot, which can be applied to tailsitters by using the Q_ENABLE parameter to set the vehicle as a quadplane and the Q_TAILSIT_ENABLE parameter to configure it as a tailsitter [5]. Moreover, options within ArduPlane enable its use for control surface or copter tailsitters. For copter tailsitters, the motors are controlled using ArduCopter using the same method explained in Section 4.1. An example of an aircraft using the ArduPilot tailsitter controller is discussed by Sohail et al. on an aircraft with four rotors and one wing [102].

The work presented by Bulka and Nahon is not limited to a specific vehicle type; rather, it can be used on a variety of platforms, including tailsitters [28]. The controller is based on a

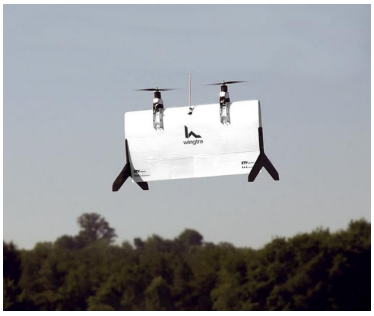
mathematical model of the aircraft and is never linearized, so the strategy can be employed across the entire flight envelope of a variety of aircraft through a range of speeds. The controller requires the desired and measured linear and angular positions and velocities to function, and then the difference is computed as errors in quaternions. The position controller was used to orient the aircraft such that the thrust force reduces the position error using a proportional - derivative (PD) controller. From the desired orientation, the attitude controller calculates the moment required for correction using angular errors in quaternions, which is also regulated using a PD controller. The desired force is used to achieve the correct altitude and speed based on thrust from the rotors and the lift and drag of the wings. A proportional controller was used to regulate the velocity, while a PD controller was utilized for altitude. A mixer was then used to convert the forces and moments calculated by the controllers into commands for all actuators based on vehicle type.

4.2.2 Dual Rotor and Control Surface Tailsitter Controllers

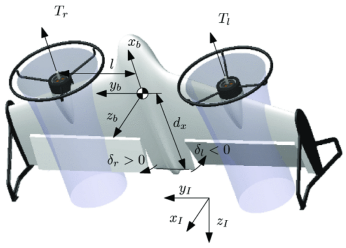
While the LaBWing relies solely on rotors for manoeuvring, there are many control strategies only applied to tailsitters with two rotors and control surfaces that could be expanded to copter tailsitters. Bapst et al. presented the TBS Caipirinha, a flying wing tailsitter with two rotors, shown in Figure 4.2(a) [20]. It used a single controller for all flight modes and quaternions for attitude representation. The controller is a first-order feedback loop using quaternions to calculate the desired body rates. The Pacflyer S100 tailsitter is shown in Figure 4.2(b), which is a similar aircraft investigated by Verling et al [135]. A Pixhawk autopilot was implemented, with its extended Kalman filter (EKF) for attitude estimations. The authors conducted detailed aerodynamic testing of the aircraft in a wind tunnel to characterize the forces and moments created depending on the angle of attack, elevon deflection, motor speed, and inlet airspeed. The aircraft was then modelled with this data, using a coordinate-free representation known as the Special Orthogonal group (SO(3)) to avoid singularities. A state space model using the wind tunnel data, along with the error and error derivative, was used to control the aircraft actuators with success on a prototype aircraft. The error and error derivative signals were each multiplied by a gain matrix, essentially operating as a state space PD controller. The position was controlled using the Pixhawk L1 navigation controller [85]. A similar aircraft called the X-VERT VTOL shown in Figure 4.2(c) using a single flight mode was presented by Chiappinelli and Nahon [32]. The controller developed was a cascaded PID loop utilizing quaternion algebra, based on the work of Bulka and Nahon for agile fixed-wing aircraft [27]. Tal and Karaman developed a global controller for agile trajectory tracking shown in Figure 4.2(d) [117]. The flight controller presented utilized incremental control action known as an incremental nonlinear dynamic inversion (INDI) controller to predict the changes in linear and angular acceleration utilizing quaternions for attitude representation. A flying wing tailsitter was also investigated by Zhang et al., shown in Figure 4.2(e) [144]. The authors implemented a dual fuzzy logic PID controller with a single flight mode, attempting to reduce oscillations and improve performance when compared to a generic PID controller. One of the fuzzy



(a) TBS Caipirinha [20].

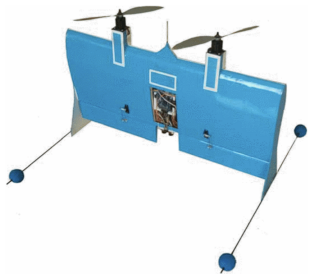


(b) Paciflyer S100 [135].



(c) X-VERT [32].

(d) Prototype by Tal and Karaman [117].



(e) Prototype by Zhang et al [144].

Figure 4.2: Tailsitters with two rotors, a single wing, and control surfaces presented in control literature.

controllers sets the universal gains based on airspeed and attitude error. The second was used to independently tune the proportional, integral, and derivative gains based on the error and error derivative.

4.2.3 Uniquely Configured Control Surface Tailsitter Controllers

The X-Hound shown in Figure 4.3(a) is a quadrotor tailsitter with four control surfaces, an upside-down V-wing, and a V-tail creating an 'X' structure presented by Zhaoying Li et al [65]. The authors presented a robust nonlinear dynamic inversion (NDI) controller with a single flight mode utilizing quaternion tracking errors to calculate the error rates. From the error rates, the time derivatives are calculated, which are used to define the control input and thrust based on dynamic inversion. The resulting controller used state feedback to define a transfer function for the linear error system, allowing for the design of a robust compensator.

A unique fighter-style tailsitter called the THU1500, shown in Figure 4.3(b), was presented by Kuang et al [61]. It utilized thrust vectoring, a stand for vertical takeoffs, and high AoA forward landings, classifying it as a belly-landing thrust-vectoring tailsitter. The authors did not use quaternions, citing that they can generate large heading errors, and instead opted for Euler angles combined with multiple flight modes that are associated with hor-

(a) X-Hound [65].

(b) THU1500 [61].

(c) Liu et al. prototype [66].

Figure 4.3: Unique tailsitters with two rotors and control surfaces presented in control literature.

izontal and vertical orientations. To avoid oscillation when switching between forward and hover flight modes, a hysteresis method is used so that the switching points overlap. The overall controller structure is a feedback loop with a P - PID controller, similar to ArduCopter's attitude controller. The controller was tested on a physical aircraft, and results show that there was roll oscillation due to rapid angle of attack changes, giving an angular error of 10 degrees.

The unique tailsitter presented by Liu et al. is shown in Figure 4.3(c) [66]. It had an 'X'-arranged quadrotor attached through 4 wings with control surfaces to a large fuselage. Also attached to the fuselage was an additional rotor at the nose, plus a main wing with two control surfaces. The controller developed had a single flight mode and used quaternions for attitude tracking errors. The control scheme presented contained an H_∞ state feedback controller for nominal error tracking and an adaptive controller to limit the impact created by disturbances.

4.2.4 Quadrotor tailsitter control

The remaining controllers were all tested on quadrotor copter tailsitters, with some having a single wing, and others are classified as a QBiT. The Quadshot had a single wing to generate lift, and utilized two control surfaces to assist the four rotors with manoeuvring, as shown in Figure 4.4(a). Sinha et al. presented a flight control that had a separate mode for hover, forward flight, and acrobatic flight, where the pilot inputs on a transmitter are used to determine the desired angular rates [101]. Quaternions were used to represent the heading, while Euler angles were used for the roll and pitch. The three different flight modes each used the same PID control strategy with different gains, and the transition was completed through a linear change in pitch, resulting in errors up to 35 degrees. Flores et al. also used the Quadshot platform to test their controller, but only in a 2D simulation to investigate the transition [45]. The presented controller utilized a linear saturation function

(LSF) using time-scale separation to track the linear and angular acceleration dynamics separately, where the desired aircraft pitch was the primary result.

The VertiKUL UAS shown in Figure 4.4(b) by Hochstenbach et al. was one of the first tailsitters that had no control surfaces [58]. It was designed for use in parcel delivery using a numerical method to optimize the range for up to a 1 kg payload. The control for the VertiKUL was implemented using ArduPilot as described in Section 4.1 and 4.2, using the P+PID attitude controller with feed forward gains [3]. The aircraft was tested in all flight modes, but had significant pitch errors since its control was primarily reactive.

Wang et al. presented a single wing copter tailsitter that could fly at almost 1.5 times the speed for almost 1.5 times longer than a similarly sized quadcopter, which is shown in Figure 4.4(c) [136]. The flight controller implemented was a PID controller for both the inner and outer loops for attitude and position control, respectively, operating on the SO(3) coordinate frame. However, there were significant errors, oscillations, and delays observed between the desired and measured positions and attitudes. Zhang et al. worked on the same aircraft, but modified the controller [145]. The attitude was controlled using a PD loop with attitude represented by quaternions, while the altitude was controlled with a PID controller, and a single flight mode was used. However, the simulation results exhibit a throttle drop at the beginning of the transition, which results in altitude loss.

A modified X5 Flying Wing quadrotor tailsitter developed by Lyu et al. is shown in Figure 4.4(d) [75]. There are two elevons in addition to the four rotors to increase the moment in the pitch axis. The Pixhawk flight controller used has three separate modes, being hover, forward flight, and the transition [42]. The attitude was controlled with a cascaded structure where the attitude was regulated using a proportional controller, then the angular velocity was regulated with a PID, similar to ArduCopter [6]. A "Quaternion linear" method was used for the attitude error decomposition to avoid singularities while achieving linear error rates. A mixer was utilized to compute the required thrust for each motor and deflection of the two elevons based on the calculated thrust and moments. The controller was tested on the aircraft, tracking the desired attitude with errors of 20 degrees, while experiencing oscillations with and without the use of elevons. Zhou et al. worked on the same vehicle without using the control surfaces, and developed a unified control method so only one controller was required for all flight modes [148]. The desired attitude was tracked using a proportional controller, and the angular error was corrected using a PID controller, both utilizing quaternions for attitude representation. The attitude controllers was capable of tracking throughout the flight regime, but had overshoot and significant errors up to 15 degrees. Lyu et al [73] further investigated this aircraft utilizing the SO(3) coordinate frame for a cascaded attitude controller that was comprised of a proportional controller for the attitude, and a PID controller with feedforward for the angular velocity. The authors also added a transition controller whose purpose was to maintain altitude through the transition process, using a PID loop. Lyu et al. furthered this work by using a velocity disturbance observer (DOB) for the position controller by estimating and compen-

sating for any wind disturbances by utilizing C optimal control theory [74]. The DOB used the output of the plant and subtracted the control signal to estimate the disturbance of the system, which was then filtered and compensated for. The final addition to this work was an attitude controller designed in the frequency domain presented by Zhou et al [147]. A frequency analysis using a series of sinusoidal inputs was added to the controller to get data used for creating a full spectrum model. Plotting the frequency response shows that the aircraft can be described at lower frequencies, representing the main aircraft dynamics, and at higher frequencies, representing the dynamic modes. Transfer functions to represent this data were created, then the controller was designed using a loop shaping technique which utilized integral, lead, and rolloff (ILR) transfer functions.

The Skywalker X-5 shown in Figure 4.4(e) was a modified flying wind copter tailsitter with a plus motor configuration and two control surfaces presented by Boyang Li et al. by [64]. The inner loop of the control scheme utilized a Pixhawk attitude controller with 3 flight modes using the SO(3) to represent rotations [42]. The outer loop was controlled with a model predictive controller (MPC), where the model was estimated from wind tunnel tests, and estimations of the rotorwash. The prediction model assumes the form of a discrete-time state space model, and disturbances that can be measured, such as wind, are used for prediction through a feed-forward signal, allowing for faster responses. The controller was also tested on a physical vehicle and showed a good ability to reject disturbances to the system, such as crosswind.

The control of another quadrotor tailsitter with no control surfaces but large mounting angles for the rotors was investigated by Xu et al., and is shown in Figure 4.4(f) [141]. The controller had a single flight mode and utilized quaternions for the attitude representation. The attitude controller was designed using loop shaping by adding a sweep of sinusoidal inputs to the system, then measuring the output. A spectral analysis was then used to find the gain and phase delay of the system, and using these results, a fitted model was used to represent the system. A notch filter was also implemented to decrease the impact of modes caused by motor vibrations. In this case, the loop shaping controller was designed with a PID compensator. Xu and Zhang continued this work by adding a feedback iterative learning controller to regulate the position error for the Pugachev's Cobra manoeuvre [142]. To add the iterative learning controller (ILC), first a lifted domain model was created, which adds the learning correction to the attitude controller. The continuous disturbance of the system, which represents the unmodelled aircraft dynamics, was then estimated in an iteration domain Kalman filter. Both versions of the controller were implemented using the Pixhawk flight controller.

Reddinger et al. created a flight controller for the CRC-20 QBiT shown in Figure 4.4(g) using a scheduled linear dynamic inversion (LDI) controller that required the linearization of the vehicle dynamics, represented by a state space model [97]. A wind frame representation of the state space model was used, which allowed for control of the aircraft during the transition. However, the model was not developed for hover or forward flight, and

the planned transition path had a large altitude variation. Differential flatness was utilized by McIntosh et al. to reduce the computational load required for trajectory planning for the CRC-20 QBiTs transition based on the aircraft dynamics previously defined using nonlinear dynamic inversion [77, 109]. The differential flatness was due to the assumption that the out-of-plane (not in the direction of transition) variables are constant. Two flight modes are used with the dynamics defined and simplified around their orientation, being hover and forward flight. Then a NDI control architecture was used to calculate outputs for both modes, and they are blended based on the pitch to create the output signal.

A quadrotor tailsitter with an annular outer wing, shown in Figure 4.4(h), was investigated by Gill and D'Andrea [49, 50]. The presented controller was cascaded with a PID outer loop for position based on the frame acceleration instead of position or velocity. The inner loop was used to align the rotors to generate the forces in the direction required for the outer loop, and utilizes a feedforward component from the outer loop with a PD controller. A controller for the motor voltage was also described, where the inner control allocation calculated the required rotational speeds of the rotors based on the commanded values for thrust and torque, which were then sent through an iterative algorithm initialized by the voltage required for hover.

An alternative method to a feedforward ILC on a QBiT shown in Figure 4.4(i) was presented by Raj et al [93]. For the aerodynamics model, the lift and drag were the sum of the wing sections that were impacted, or unaffected by the rotorwash, with the aerodynamic coefficients based on simple models rather than wind tunnel testing. However, wind tunnel tests are used to parameterize the propeller performance as the thrust produced reduces significantly as the inlet airspeed increases. The data obtained from these tests was arranged using a neural network, which was later used by the iterative learning algorithm for the feedforward thrust demanded. For the ILC, the feedforward trajectories for the pitch and thrust were given fourth-order polynomials with scalars to be determined. A terminal error cost function was then defined to optimize the fourth-order polynomials for the transition. A PD controller was used for the lateral direction regulation, and a PID controller was used with the feedforward ILC signals to manage attitude.

A unique quadrotor tailsitter with an X wing arrangement that was angled 45 degrees off of level ground, shown in Figure 4.4(j), was investigated by Xin et al [140]. Three separate controllers were used to account for hover, forward flight, and the transition flight modes. Dual Euler angles were used for attitude representation, with one aligned for hover and the second aligned for forward flight to avoid singularities. The aircraft dynamics were modelled using Newton-Euler equations, and the aerodynamic forces and moments were obtained using a set of equations defined in [22] and coefficients solved using computational fluid dynamics (CFD). To control the aircraft, NDI was utilized, which attempts to linearize the nonlinear dynamics of the aircraft to regulate the angles and angular rates.

Dalwadi et al. investigated the control of a QBiT shown in Figure 4.4(k) using backstepping

Control (BSC) for the attitude regulation and an integral terminal sliding mode control (ITSMC) for the position [35]. The authors also tested using the BSC and ITSMC for both the inner and outer loops, but found a hybrid gave the best results. The controllers have hover, forward flight, and transition flight modes, where the transition was performed by commanding a 90-degree pitch rotation, and had no control over position other than altitude. The use of the three flight modes enables the use of Euler angles while avoiding singularities.

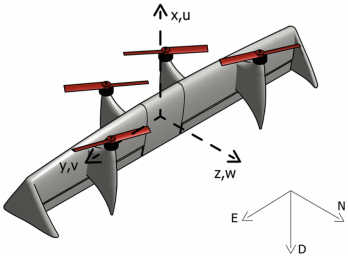
4.3 Control Review Summary

The different control strategies on a variety of aircraft discussed in Section 4.2 are summarized in Table 4.1, following the same order presented. The majority of the controllers used quaternions [28, 49, 58, 73–75, 141, 142, 145, 147, 148] to represent the attitude to avoid singularities when the tailsitter transitions. If this is not the case, then there were either multiple flight modes with limits on the angles implemented [35, 93, 97, 136], or the controller was only used for hover [64].

Very little of the literature commented on the computational requirements of the controllers implemented. However, the following works indicated that their methods required calculations that were relatively difficult to compute within the refresh rate of flight controllers [64, 77, 148]. The ILC discussed here [93] was described as being at least as computationally light as typical PID methods while also being easier to implement due to the simplified aerodynamic model. The acceleration-based model used by the ILC method presented by [142] was also computationally similar to PID control methods. In both cases, the models are simpler since the controller learns the aerodynamics and required feedforward inputs iteratively.

It should also be noted that four of the controllers were only tested through simulations rather than implemented on real aircraft [28, 35, 77, 145]. These controllers all had relatively small errors due to the lack of disturbances in most cases. In contrast, the controllers implemented on real aircraft were tested outdoors, so they had to endure the impact of wind, along with other aerodynamic factors that were not considered in the mathematical models.

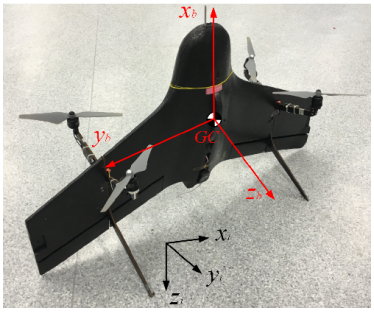
Controllers that utilize parts of PID regulators are very common for RPASs since they are relatively simple to implement and can correct for disturbances or dynamics without the requirement of a model. The work of [27, 28, 32], and [61] utilizes this strategy but exhibits poor responses with large errors or significant delays. To improve this, feedforward strategies were added by [49, 73, 74, 145, 148], and were all implemented on real aircraft. With feedforward signals, the controller can predict how changes in the input will impact the system, allowing it to respond to new inputs faster. The aircraft in these cases were able to track position with maximum errors between 1 and 2 m. The attitude errors have more variance, but there was no unified testing method, with some tests using step inputs while



(a) Quadshot [45, 101].

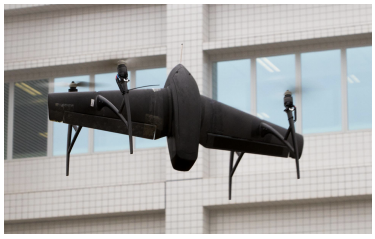
(b) VertiKUL [58].

(c) Wang et al. prototype [136, 145].



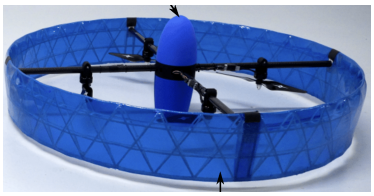
(d) Modified X5 Flying Wing [73–75, 147, 148].

(e) Skywalker X-5 [64].

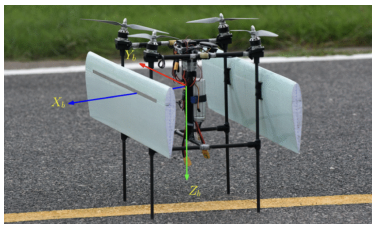


(f) Xu et al. prototype [141, 142].

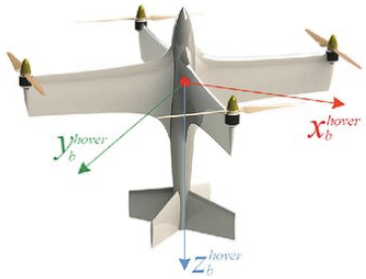
(g) CRC-20 QBiT [77, 96]



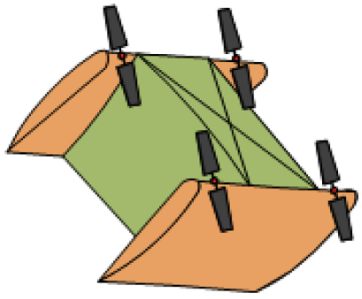
(h) Gill and D'Andrea prototype [49, 50].



(i) Raj et al. QBiT [93].



(j) Xin et al. prototype [140].



(k) Dalwadi et al. QBiT [35].

Figure 4.4: Quadrotor tailsitters presented in control literature.

others had smooth transitions. Loop shaping techniques were applied by [147] and [141] to tune the PID systems rather than using manual tuning methods. These techniques were tested on aircraft and were shown to be at least as good as the standard tunes. [74] added a DOB for crosswind compensation using H_∞ synthesis techniques for their PID controller. It was only tested in hover but showed great position stabilization in the presence of wind. The use of fuzzy logic to adapt the gains of PID controllers resulted in faster responses

while also reducing oscillation and required a single flight mode on the flying wing tailsitter as presented by [144].

Alternate methods include the MPC presented by [64], which when used with a linearized state space model of the system, can measure disturbances and correct for factors not considered. This method was tested on an aircraft in windy conditions and showed a good ability to hold position and track trajectories. A linearized state space model was also used by [97] to develop a LDI controller. A NDI controller and trajectory planner was shown by [77] that utilized a simplified 3 - DOF model. This controller was used for both the inner and outer control loops, and resulted in good tracking and stability for both attitude and position. [35] used a BSC for the inner loop and an ITSMC for the outer loop of their controller. When simulated, the attitude error was as much as 8 degrees, and the altitude error was between 1 and 2 m.

The final control strategy implemented for tailsitters is the use of ILC to determine how much feedforward signal is required for a manoeuvre. This method was used for the inner and outer loops by [93] and [142] respectively, and resulted in very small position errors for real aircraft tests, even when performing advanced techniques such as Pugachev's Cobra manoeuvre.

4.3.1 Recommendation

From the review of current literature, many controller techniques can be applied to improve the performance of tailsitters. However, there is a gap in the literature where fuzzy logic PID controllers have been implemented on multirotors and a flying wing tailsitter with control surfaces, but have not been used on copter tailsitters. The literature of fuzzy logic as applied to adapting the gains of multirotors can be used as the multirotors and copter tailsitters use differential thrust and torque of rotors for manoeuvring. Moreover, the information from control surface tailsitters can also be used to guide how gains should be adjusted based on pitch and airspeed, since both vehicle types generate lift with fixed wings in forward flight. Therefore, the control scheme that will be designed in this thesis is a PID based controller that will utilize fuzzy logic to adapt the gains based on flight conditions.

A major benefit of fuzzy logic is that it can be used to describe complex systems without the requirement of having an accurate model, making it very useful for model-free control systems [43]. Crisp measurements are taken from the system, usually regarding the error and error rate, and are then fuzzified. Fuzzifying the data is required for decision-making and converts the crisp measurement into descriptive terms. These fuzzified inputs are then compared to a set of rules and a fuzzy inference system to determine a fuzzified output, representing how the output of the plant should behave. Opposite to the inputs, the fuzzified outputs have their descriptive words converted into crisp values that the system can

Table 4.1: Summary of control methods for tailsitter UAS.

Vehicle	Rotors	Wings	Control Surfaces	Hardware	Testing	Flight modes	Inner Controller	Attitude Error [deg]	Outer Controller	Position Error [m]	Coordinate frame	Reference
TBS Caipirinha	2	1	2	Pixhawk 4	Both	1	1st Order Feedback	25	Pixhawk PID	0.5	Quaternion	[20]
Pacflyer S100	2	1	2	Pixhawk	Real	1	State Space PD	25	L1 Navigation		SO(3)	[135]
X-VERT VTOL	2	1	2		Simulation	1	Cascaded PID	20	Cascaded PID	5	Quaternion	[27, 32]
	2	1	2		Real	1	INDI	12	PD	0.3	Quaternion	[117]
	2	1	2		Both	1	Dual Fuzzy PID	4			Z-X-Y Euler	[144]
X-Hound	2	X	4		Simulation	1	NDI		NDI	0.5	Quaternion	[65]
THU1500	2	1	2	ATMEGA2560	Real	2	P + PID	10	P + PID	3	Dual Euler	[61]
	5	1+X	6		Simulation	1	Adaptive H_∞ state feedback	0.5			Quaternion	[66]
Quadshot	4	1	2	Paparazzi	Real	3	PID	35	PID		Combined	[45, 101]
VertiKUL	4	1	0	Pixhawk	Both	3	PI + FF	5	PI + FF	2	Quaternion	[58]
	4	1	2	Pixhawk	Real	3	PID + FF	50	PID	1	SO(3)	[136]
	4	1	0		Simulation	1	PD	0.1	PID	0.1	Quaternion	[145]
Modified X5	4	1	2	Pixhawk	Real	3	P, PID + FF	20	PID + FF	1.5	Quaternion linear	[75]
Modified X5	4	1	0	Pixhawk	Both	1	P, PID	15	PID + FF		Quaternion	[148]
Modified X5	4	1	0	Pixhawk	Both	2	P, PID + FF	5	PID	2	Quaternion + SO(3)	[73]
Modified X5	4	1	0	Pixhawk	Real	1	P, PID		P, PID, H_∞ DOB	0.1	Quaternion	[74]
Modified X5	4	1	0	Pixhawk	Real	1	Loop shaping ILR	5	P		Quaternion	[147]
Skywalker X-5	4	1	2	Pixhawk	Both	3	Pixhawk PID	13	Linearized MPC	0.2	SO(3)	[64]
	4	1	0	Pixhawk 4	Real	1	P, Loop shaping PID	5	PI + FF	2	Quaternion + Z-X-Y Euler	[141]
	4	1	0	Pixhawk 4	Real	1	P, Loop shaping PID	10	PD + ILC FF	0.45	Quaternion	[142]
CRC-20	4	2	0		None	3	LDI SS		LDI SS		Z-X-Y Euler	[96]
CRC-20	4	2	0		Simulation	2	NDI	25	NDI	0.5	Z-X-Y Euler	[77]
	4	Annular	0	STM32 F4	Both	1	PD + FF	1	PID	1.5	Quaternion	[49, 50]
	4	2	0	Pixhawk 4	Both	3	IL FF + PID		PD	0.5	Z-X-Y Euler	[93]
	4	X	0		Both	3	NDI	15	NDI	10	Dual Euler	[140]
	4	2	0		Simulation	3	BSC	8	ITSMC	20	Z-Y-X Euler	[35]

use to take a control action.

While fuzzy logic can be used directly to control a plant, it is common to see a fuzzy supervised PID (FSPID) controller. The benefit of this design is the ability to control a nonlinear process over a wide range of operating conditions that can not be achieved using just a PID controller [99]. The same approach was presented by Fekik et al [44]. and by Abbasi et al [1]. for quadrotors, and by Zhang et al [144]. for a flying wing tailsitter. Using these works as inspiration, the ArduCopter attitude controllers will be modified to include gain adjustment using fuzzy logic based on the current state of the aircraft.

4.4 Fuzzy Logic Attitude Controller Design

The controller designed for the prototype tailsitter aircraft presented throughout must be compatible with ArduCopter since the SITL simulation capabilities of ArduPilot enable vigorous testing of the implemented controller without risking damage to the actual system. ArduCopter also has many beneficial features, including a robust Kalman filter, motor mixer, and position controller. Moreover, ArduCopter is a commonly used open-source platform, so using this as a starting point gives good competition for comparison of the controller developed. For the remainder of this thesis, the standard ArduCopter attitude controller will be referred to as the base or the P+PID controller, while the fuzzy logic controller presented will be referred to as the custom or Variable P + Fuzzy PID (VP + FPID) controller. Additionally, four types of errors were discussed throughout the controller implementation. Each of these are presented in Figures 4.5 and 4.6 as applicable, and are each defined as

- Error: the difference between the desired attitude and the measured attitude (roll, pitch, and yaw) of the aircraft as determined by the flight controller. The measured error is the value that the flight controller tries to minimize.
- Error derivative: The differentiation of the error with respect to time. It is utilized by the FLC to adapt the gains of the PID controller discussed in this chapter.
- Error rate: the difference between the target rate of change of attitude and the measured rate of change of attitude of the aircraft. The target rate is generated by the error signal being multiplied by the proportional gain. The measured rate is estimated by the sensors and the Kalman filter in the Orange Cube flight controller.
- Error rate derivative: The differentiation of the error rate with respect to time. In practice, this is the angular acceleration, but it is obtained through differentiation rather than measurement. It is a value utilized by the derivative term in the PID controller.

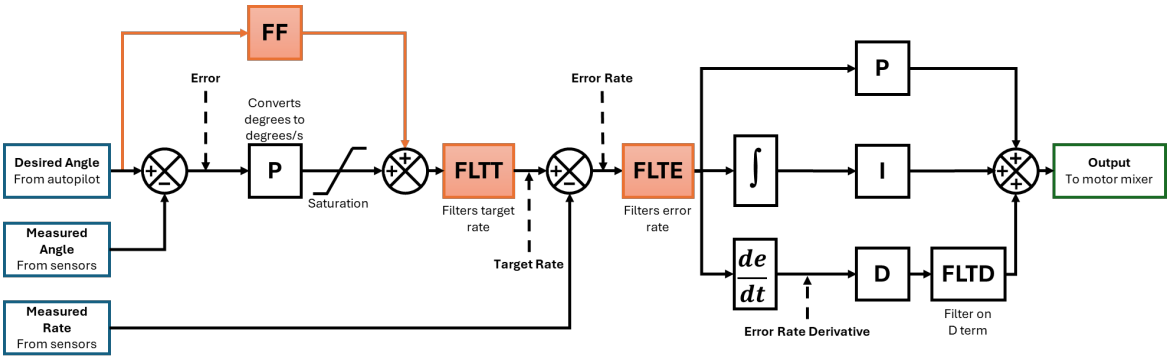


Figure 4.5: Base ArduCopter P+PID attitude controller diagram for a single axis. Sections in orange represent components of the controller that were unused or disabled.

4.4.1 ArduCopter Attitude Controller

The ArduCopter base attitude controller contains a P+PID loop for each axis, shown in Figure 4.5 for a single axis [11]. The inputs to the system are the desired angle as generated by the autopilot or controller input from the pilot, the measured angle of the aircraft taken from the sensor suite, and the measured angular rate of the aircraft. The output of the controller is the desired angular rate of the aircraft for each axis. First, the angular error is sent through the first proportional controller to calculate the target rate. The target rate is then saturated to prevent very large signals and added to the feedforward gain before being low-pass filtered to remove noise. Next, the error rate is generated by subtracting the measured rate from the target rate signal before being filtered again to remove the measurement noise. The error rate is then sent through the PID controller, whose output is the final signal in the attitude controller. The output is sent to the motor mixer to calculate the thrust required for each motor.

The feedforward signal and filter components (labelled as FLTT and FLTE) represented by the orange sections shown in Figure 4.5 were not used. The FLTT and FLTE filters had no noticeable impact on performance when tested in simulation and verified in real flights. However, the filter on the derivative term (labelled as FLTD) in the PID controller was required for stable operation by removing jitter from the derivative signal.

4.4.2 Fuzzy System for Adapting Gains of Attitude Controller

To improve the performance of the P+PID attitude controller used by ArduCopter, an fuzzy logic controller (FLC) was added to each of the proportional and PID controllers of each axis to adjust the gains. The PID controller gains were modified based on the error and error derivative, which is a technique commonly applied to multirotors [1, 44]. Additionally, the gains of the proportional controller are modified based on the aircraft pitch, which is directly related to the ground speed, and follow a strategy similar to what Zhang et al. implemented for a control surface tailsitter [144]. The resulting controller is

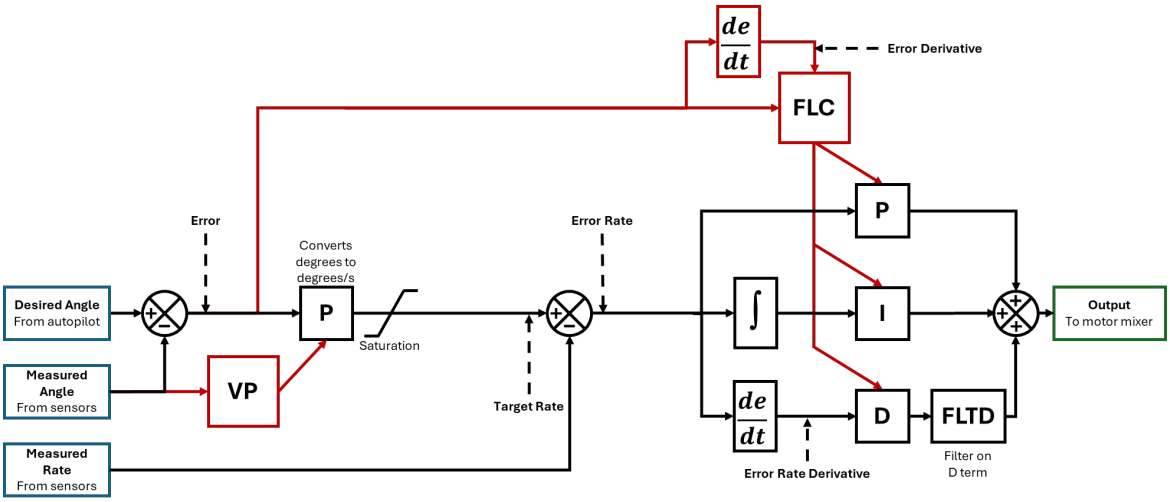


Figure 4.6: Custom VP + FPID attitude controller diagram for a single axis. The red components indicate changes from the base ArduCopter attitude controller, and unused components have been removed.

a VP + FPID controller, as shown in Figure 4.6 with the changes from the base controller presented in red.

4.4.2.1 Variable Pitch Controller

The purpose of the variable pitch controller is to adjust the gains for the proportional controller that generates the target rate of the aircraft based on the current pitch. The controller was influenced by the work of [144], who reduced the global gains of their fuzzy controller as the tailsitter transitioned from hover to forward flight. Additionally, it was determined through simulation testing that a reduction in the proportional gains as the angle of attack increased towards forward flight was beneficial to reduce oscillations. However, using a small constant gain would result in a system that responded too slowly when in hover, producing large errors that are especially dangerous during takeoff and landing. Additionally, it was observed that the gain reduction benefits were primarily related to the yaw axis, so the adaptive gains were only applied to the yaw proportional controller to reduce the computational load. When hovering, the maximum gains are used, but when in forward flight, the proportional controller gain is reduced to half the maximum value. A smooth transition is used between hover and forward flight to prevent erratic responses. The resulting variable pitch controller was created using MATLAB, and the map for the proportional gain adjustment is shown in Figure 4.7.

The inputs, outputs, and rules used to create the map for the proportional controller are listed in Tables 4.2, 4.3, and 4.4, respectively. These values were used to configure the fuzzy logic controller within MATLAB, using the Fuzzy Logic Designer application [127]. Each of the rules had a weight of 1 applied and used the "And" connection. The resulting

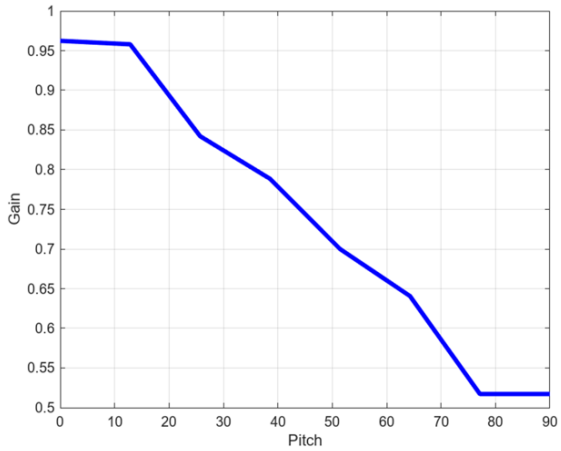


Figure 4.7: Proportional controller gain adjustment map determined by aircraft pitch.

input and output membership plots created in MATLAB are shown in Figure 4.8. The input and output are both described as having values of VS, S, M, B, or VB, which means Very Small, Small, Medium, Big, or Very Big, respectively. Tables 4.2 and 4.3 show how these parameters are defined, while Table 4.4 describes how the rules are related. The input and output signals can be multiplied by a gain and have an offset added to further tune the controller.

Table 4.2: Input signal configuration for FLC proportional gain adjustment.

Pitch	**Type**	**Parameters**
VS	Triangular	[0 0 30]
S	Triangular	[15 35 50]
M	Triangular	[35 50 65]
B	Triangular	[50 60 65]
VB	Trapezoidal	[60 70 90 90]

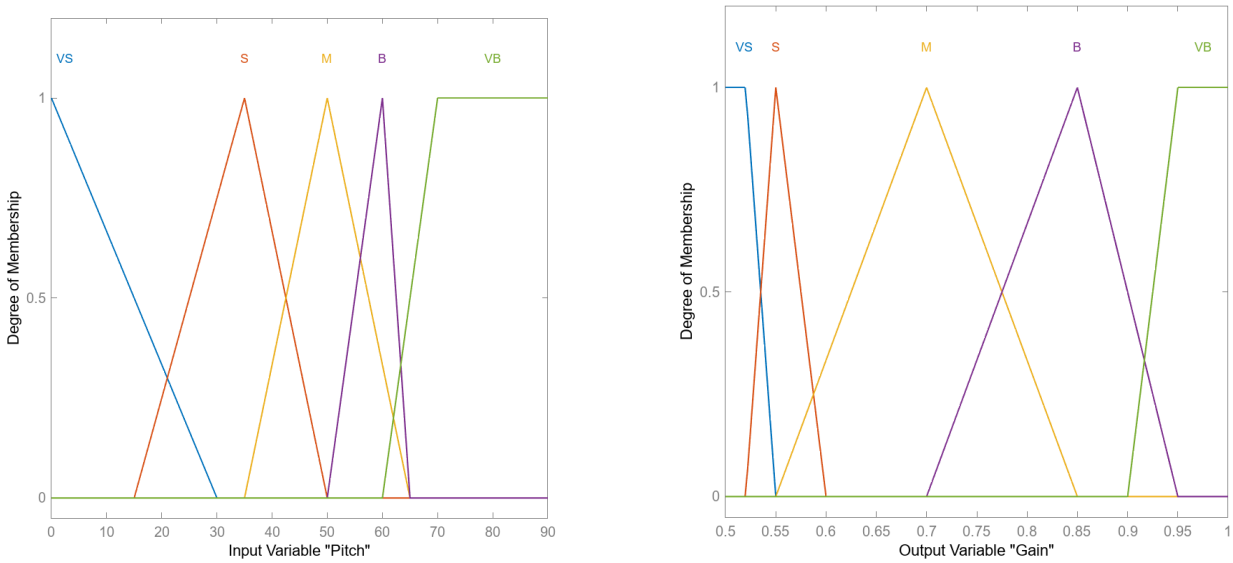
Table 4.3: Output signal configuration for FLC proportional gain adjustment.

Gain	**Type**	**Parameters**
VS	Trapezoidal	[0.5 0.5 0.52 0.55]
S	Triangular	[0.52 0.55 0.6]
M	Triangular	[0.55 0.7 0.85]
B	Triangular	[0.7 0.85 0.95]
VB	Trapezoidal	[0.9 0.95 1 1]

While the profile for the proportional gain is a line connecting the two constant gains for hover and forward flight, the use of fuzzy logic enables testing a variety of different

Table 4.4: Rule set configuration for FLC proportional gain adjustment.

Rule	Pitch	Gain
1	VS	VB
2	S	B
3	M	M
4	B	S
5	VB	VS



(a) Input pitch membership functions.

(b) Output gain membership functions.

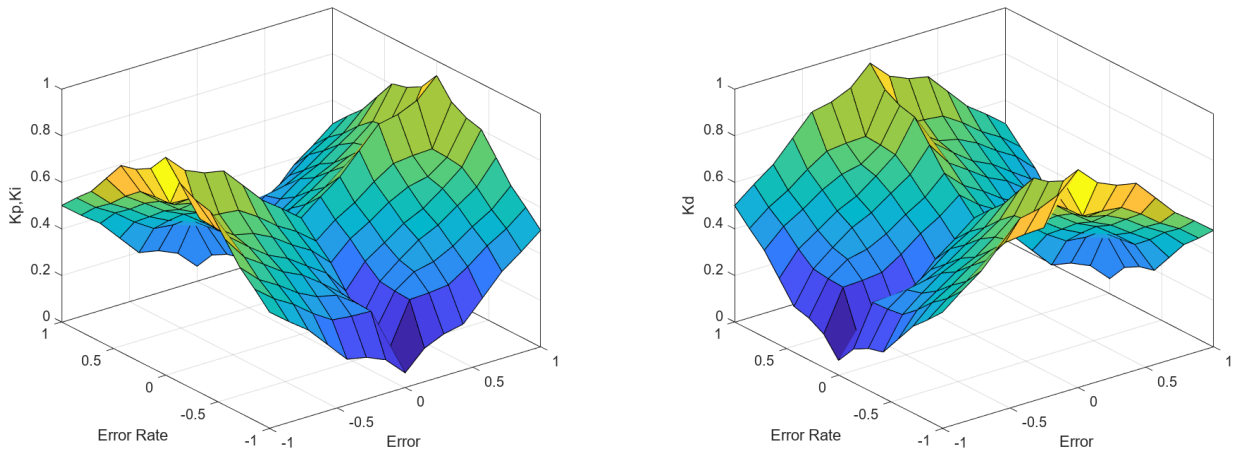
Figure 4.8: Membership function plots are created using MATLAB's Fuzzy Logic Designer for the proportional gain adjustment.

configurations. Examples of this include adjusting the slope of the line for different ranges of angles of attack through the membership functions or adding other parameters like airspeed to have an impact on the proportional gain. However, through simulation testing, it was determined for this specific vehicle that changing the gain nearly linearly based on only the angle of attack produced good performance while remaining simple to implement. Since the final version of the gain adjustment for the proportional controller is reliant on only one variable, it is labelled as a variable proportional controller.

4.4.2.2 Fuzzy Proportional-Integral-Derivative Controller

The purpose of the fuzzy PID controller was to adapt the gains that regulated the rate of the aircraft to both improve responsiveness and stability. The strategy applied has been previously used for multirotors, as presented by [1], and applies to the tailsitter presented, as the actuation structure is the same. For the fuzzy PID controller, the gains were adapted based on the error and error derivative in the axis to be controlled. The surfaces

were created using the MATLAB Fuzzy Logic Designer application [127], and are shown in Figure 4.9. The resulting membership functions for the inputs and output are shown in Figure 4.10. Other methods were attempted, but in the simulation, this approach gave the best results. The purpose of the surface is to increase the proportional and integral gains as the measured error increases, representing long-term and large error correction. However, the proportional and integral gains are reduced when an increase in the error derivative is observed, helping to reduce oscillations and instability. In contrast, the derivative gains have the opposite response, where they decrease for larger errors but increase for large error derivatives to stabilize the system.



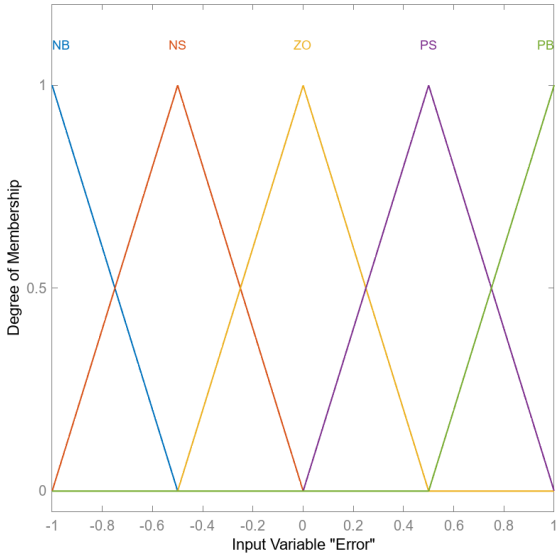
(a) Proportional and integral gains.

(b) Derivative gains.

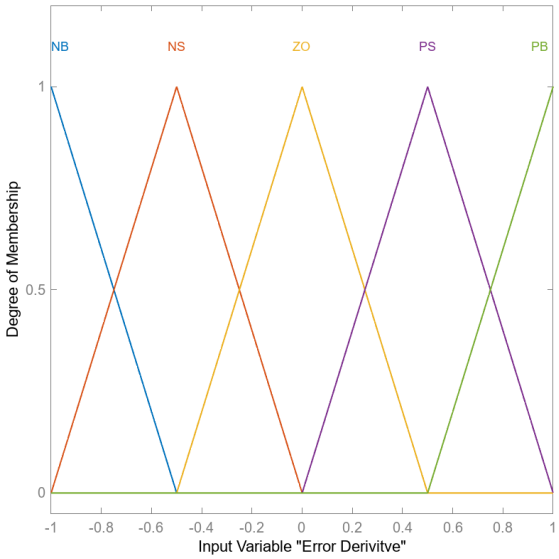
Figure 4.9: FLC gain adjustment for the PID controllers based on error and error derivative.

The inputs for the PID gain controller can be anywhere between -1 and 1, and are described as NB, NS, ZO, PS, and PB, which mean Negative Big, Negative Small, Zero, Positive Small, and Positive Big, respectively. The definition of these parameters are described in Table 4.5. The output of the FLC is the gain, which can be VS, S, M, B, or VB, meaning Very Small, Small, Medium, Big, or Very Big, respectively, and each of the outputs are defined in Table 4.6. The rules connecting the two inputs and output are then shown in Table 4.7, defining how the fuzzy controller adjusts the PID gains.

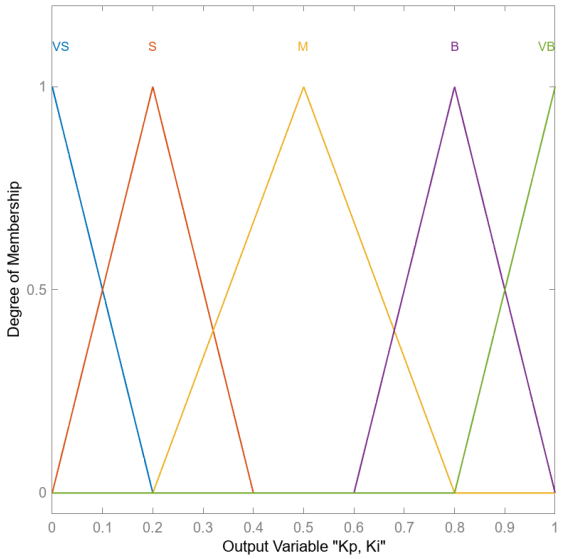
The derivative gains are the opposite of what was used for the proportional and integral gains. Since both of these maps, which are shown in Figure 4.9, are normalized to 0 and 1, the derivative map can be obtained by subtracting the proportional and integral gain map from 1. Therefore, all three gains for each axis can be calculated with a single lookup table, reducing the computational load. The normalized output from each map can then be modified using a multiplier and offset for better tuning.



(a) Input error rate membership functions.



(b) Input error derivative membership functions.



(c) Output gain membership functions.

Figure 4.10: Membership function plots created using MATLAB's Fuzzy Logic Designer for the PID controller gain adjustment.

Table 4.5: Input signal configuration for FLC to adjust the gains of the PID controller.

Error	**Error Derivative**	**Type**	**Parameters**
NB	NB	Triangular	[-1 -1 -0.5]
NS	NS	Triangular	[-1 -0.5 0]
ZO	ZO	Triangular	[-0.5 0 0.5]
PS	PS	Triangular	[0 0.5 1]
PB	PB	Triangular	[0.5 1 1]

Table 4.6: Output signal configuration for FLC to adjust the gains of the PID controller.

Gain	Type	Parameters
VS	Triangular	[0 0 0.2]
S	Triangular	[0 0.2 0.4]
M	Triangular	[0.2 0.5 0.8]
B	Triangular	[0.6 0.8 1]
VB	Triangular	[0.8 1 1]

Table 4.7: Rule set configuration for FLC to adjust the gains of the PID controller.

Rule	Error	Error Derivative	Gain
1	NB	NB	M
2	NB	NS	S
3	NB	ZO	VS
4	NB	PS	S
5	NB	PB	M
6	NS	NB	B
7	NS	NS	M
8	NS	ZO	S
9	NS	PS	M
10	NS	PB	B
11	ZO	NB	VB
12	ZO	NS	B
13	ZO	ZO	M
14	ZO	PS	B
15	ZO	PB	VB
16	PS	NB	B
17	PS	NS	M
18	PS	ZO	S
19	PS	PS	M
20	PS	PB	B
21	PB	NB	M
22	PB	NS	S
23	PB	ZO	VS
24	PB	PS	S
25	PB	PB	M

4.4.2.3 Variable Pitch + Fuzzy Proportional-Integral-Derivative Equation

With each of the maps defined, the equation to describe the VP + FPID attitude controller for each axis is:

$$U = k_p\left(e, \dot{e}\right) e_r + k_i\left(e, \dot{e}\right) \int e_r dt + k_d\left(e, \dot{e}\right) \frac{de_r}{dt}$$

where k_p, k_i, and k_d are the PID gains depending on the fuzzy logic controller and U is the output signal sent to the motor mixer. e is the attitude error, $\dot{e}$ is its derivative, and e_r is the error rate given by:

$$e_r = k(p)e - r$$
$$\dot{e} = \frac{de}{dt}$$

where k is the proportional rate controller gain dependent on the aircraft pitch, p, and r is the measured angular rate. The same equations can then be updated to include offsets and multipliers for tuning the normalized FLC maps:

$$\begin{aligned} U = {} & K_p\left(C_p\left(C_{flc}\left(\frac{e}{K_e}, \frac{\dot{e}}{K_{ed}}\right) - O_p\right) + 1\right) e_r \\ & + K_i\left(C_i\left(C_{flc}\left(\frac{e}{K_e}, \frac{\dot{e}}{K_{ed}}\right) - O_i\right) + 1\right) \int e_r dt \\ & + K_d\left(C_d\left(O_d - C_{flc}\left(\frac{e}{K_e}, \frac{\dot{e}}{K_{ed}}\right)\right) + 1\right) \frac{de_r}{dt} \end{aligned}$$

where K_p, K_i, and K_d are the base PID gains K_e and K_{ed} are the gains to adjust the range of error and error derivatives used by the FLC map, limited to a range of -1 and 1. C_{flc} is the output of the FLC dependent on the error and error rate divided by the relevant gains. O_p, O_i, and O_d are the offsets applied to the FLC for the proportional, integral, and derivative gains, respectively. These are then multiplied by the C_p, C_i, and C_d gains for each section of the PID controller before being normalized around 1. Finally, these values are multiplied by the base gains before being sent through the PID controller. The error rate and error rate derivative are then given by

$$e_r = CK(p)e - r$$
$$\dot{e_r} = \frac{de_r}{dt}$$

where C is the multiplier gain for each axis, defined by C_R, C_P, and C_Y for roll, pitch, and yaw, respectively, and K is the base controller angular gain for each axis, defined as K_R, K_P, and K_Y. Table 4.8 summarizes the gains used for each axis. Note that there is no C gain for the roll and pitch axes, as the fuzzy map to adjust the rate was only applied to

the roll axis.

Table 4.8: Gains for custom VP + FPID controller variables in roll, pitch, and yaw axes.

Parameter	Roll Value	Pitch Value	Yaw Value
C	-	-	2
K	4.5	4.5	4.5
K_p	0.135	0.135	0.18
K_i	0.135	0.135	0.018
K_d	0.0036	0.0036	0.0001
C_p	0.5	0.25	0.5
C_i	0.5	0.25	0.15
C_d	0.5	0.25	0.5
O_p	0.7	0.7	0.7
O_i	0.7	0.7	0.7
O_d	0.5	0.5	0.5
K_e	20	10	20
K_{ed}	5	1	5

Therefore, the final equation for the VP + FPID controllers for each axis is:

$$
\begin{aligned}
U = {} & K_p \left(C_p \left(C_{flc} \left(\frac{CK(p)e - r}{K_e}, \frac{\frac{d(CK(p)e-r)}{dt}}{K_{ed}} \right) - O_p \right) + 1 \right) (CK(p)e - r) \\
& + K_i \left(C_i \left(C_{flc} \left(\frac{CK(p)e - r}{K_e}, \frac{\frac{d(CK(p)e-r)}{dt}}{K_{ed}} \right) - O_i \right) + 1 \right) \int (CK(p)e - r)dt \\
& + K_d \left(C_d \left(O_d - C_{flc} \left(\frac{CK(p)e - r}{K_e}, \frac{\frac{d(CK(p)e-r)}{dt}}{K_{ed}} \right) \right) + 1 \right) \frac{d\left(CK(p)e - r\right)}{dt}
\end{aligned}
$$

Chapter 5

Results and Testing

The prototype development was a significant portion of this thesis, helping to validate the work completed. A comparison between the prototype aircraft and the scaling laws used to develop it is first presented. Then, the testing process for the prototype aircraft is presented to give insight into how the results were obtained for the simulation and controller comparison presented thereafter. The flight characteristics of the simulation model and the prototype aircraft are then compared to validate the results obtained using the simulation. Finally, the results for the base and custom controller obtained in both simulation and with the prototype are presented.

5.1 Sizing Comparison

The expected weight of the components, actual weights, and percent difference of the scaling laws and prototype aircraft are shown in Table 5.1. The largest increase in mass, both relative and absolute, was for the EMP. The increase in mass came from the hardware required to mount the propellers to the motors, which was not included in the posted motor weight. However, the remaining sections very closely matched the actual masses to the designed masses, showing that the conceptual design obtained from the scaling laws are achievable and reasonable at this scale. It is especially impressive that the structural weight was so close to the design point, with only a 4 percent increase. The small weight difference could easily be reduced with future design iterations.

Performance-wise, the vehicle had sufficient thrust to rapidly increase altitude and control against the significant wing disturbances. Additionally, it reached a cruising speed of about 17 m/s, depending on wind directions. While this is slightly lower than the design cruising speed of 17.6 m/s, it is very close to the 16.6 m/s cruising speed based on the scaling law for a 5.24 kg tailsitter. Additionally, faster cruising speeds could have been achieved at a lower AoA, but this was tested due to the 30 km/h wind gusts.

Table 5.1: Difference in conceptual design masses and real measured mass of prototype tailsitter.

Section	**Design Mass [kg]**	**Real Mass [kg]**	**Difference [%]**
EMP	0.870	1.052	20.92
Avionics	0.200	0.191	-4.50
Battery	1.380	1.375	-0.36
Payload	1.000	1.000	0.00
Structure	1.790	1.854	3.58
Total	5.240	5.472	4.43

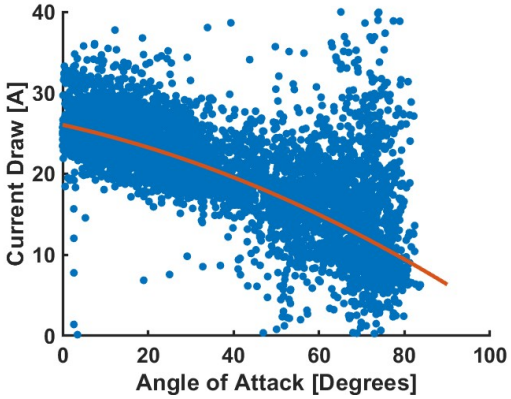
The current draw in hover was measured to be 26 A, which correlates to the posted data from both T-motor [112] and APC Propellers [91]. Forward flight at a 10-degree AoA only required 35 percent of the power needed for hover at 9.15 A, shown in Figure 5.1. The current draw could be reduced to only 7.88 A when at an AoA of 5 degrees, showing how tailsitters can have much more efficient cruising than multirotors. These results also correlate with the expected reduction in power draw from these sources [33, 53, 87, 88, 96]. Each data point in Figure 5.1 is a time instance where the pitch, ground speed, and current draw were simultaneously measured. The large variation in the measurements is due to the severe wind disturbance during the test flight, the non-rigid motion, the inertial effect of the aircraft, and measurement error. Both plots then had a second-order polynomial fit to the data using bisquare robustness to show the average trend using MATLAB's curve-fitting toolbox [125]. The equations are summarized in Table 5.2 along with the coefficient of determination R^2 to represent the quality of fit, both having the form of

$$Dependant\ Variable = A \times (Independant\ Variable)^2 + B \times (Independant\ Variable) + C$$

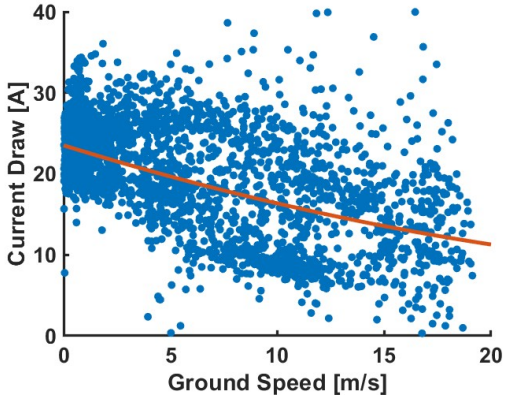
Table 5.2: Second-order equations of best fit for current draw of prototype tailsitter obtained using MATLAB's curve-fitting toolbox [125].

Independant Variable	Dependant Variable	A	B	C	R^2
Current Draw [A]	Angle of Attack [deg]	-1.134E-03	-0.1174	26.06	0.7673
Current Draw [A]	Ground Speed [m/s]	0.01064	-0.8245	23.53	0.5036

Additionally, the possible hover time was estimated by comparing the used battery capacity measured while recharging after flights with the actual flight time. The maximum hover time that could be achieved with the presented prototype aircraft is 19.3 minutes, which is very close to the predicted 21.1 minutes presented in Figure 2.12(a) for the extracted scaling laws. The current draw ratio between hover and forward flight was 0.35, so the maximum forward flight time is 55.1 minutes, which is close to but slightly larger than the 52.7 minutes predicted in Figure 2.12(a).



(a) Current draw versus pitch.



(b) Current draw versus ground speed.

Figure 5.1: Current draw of prototype tailsitter during test flights. Each data point represents a measurement of the state of the aircraft at that instance. The fitted lines are obtained using MATLAB's curve-fitting toolbox to represent the average of the measurements at each AoA [125].

5.2 Test Flight Procedure

The first step to testing was the configuration of the flight controller following the guide presented here [13]. The controller had to be configured for both the simulation and on the prototype, with the changes made to the parameters listed in Appendix C. A total of 11 flights were conducted to setup the prototype aircraft and collect the data for the comparison of the simulation and the controllers. The strategy applied was to slowly increase the maximum allowable pitch angle of the vehicle so it could systematically prove its capabilities. The purpose and result for each flight is described below.

- Flight 0: Test arm and disarm. The aircraft was armed, and then a small amount of throttle was given to force the propellers to rotate. Once they were spinning, the throttle was reduced, and the vehicle was disarmed with the transmitter.
- Flight 1: First hover and pitch tests. The vehicle was armed in Altitude Hold Mode, where it was given the signal to increase altitude to about 5 m, then maintain [83]. The aircraft was then put into Loiter mode to verify the performance of the GPS and control system by attempting to hold its position and heading [121]. After proving it could hover properly, the aircraft was given pitch and roll commands. The commands started with small inputs of about 5-10 degrees and increased to 45 degrees by the end of the flight. Flight 1 concluded the first day of testing due to time.
- Flight 2: Testing 60-degree pitch. After ensuring the vehicle was still in working order and no severe vibrations were present in the data from Flight 1, the second day of testing started by increasing the limits of the controller to allow for pitching up to 60 degrees.

- Flight 3: Testing 70-degree pitch. The third flight was to determine if the vehicle could successful reach a pitch angle of 70 degrees. The increases in angles were getting smaller at this stage since the simulation showed that tailsitters can get unstable around this attitude when poorly tuned. The top speed of the aircraft was also increased to 15 m/s because the aircraft was encroaching on the previous limit in Flight 2.

- Flight 4: Testing 80-degree pitch. The allowable pitch angle was increased to 80 degrees, with the maximum ground speed set to 20 m/s. However, due to the use of low-pitch 15x5 inch multirotor propellers, only a speed of 16 m/s at a pitch of 75 degrees was reached. After this test, the throttle output was investigated, showing that it was nearly saturated. Therefore, it was deiced to switch to the 15x10 inch APC propellers to achieve better forward flight performance. The battery was also changed after this flight.

- Flight 5: Testing 80-degree pitch. No parameters were changed for this flight, but the higher-pitch propellers were used. In hover, it was observed that the throttle levels were similar, but the aircraft now achieved 17 m/s top speeds at 79 degrees of pitch without saturating the throttle.

- Flight 6: Testing 80-degree pitch. No parameters were changed for this flight either, but more aggressive commands were given to test the limits of the aircraft. A max speed of 18 m/s was reached with a pitch of 81 degrees. After this test, the battery was changed again.

- Flight 7: Test custom controller. This was the first test where the custom controller was used. The vehicle performance was very similar to when the base controller was used, with both being tested in the same flight. Testing for day 2 concluded with this flight. The data was then analyzed, showing that the FLC was not being utilized well to adjust the PID gains. Modifications to both the gains and the controller were made, which now represent what was presented in Chapter 4.

- Flight 8: Day 3 baseline. The purpose of this test was to get a baseline flight for the base controller with the wind conditions for day 3. Wind at this time was 12 km/h with 24 km/h gusts. The flight consisted of a takeoff, two complete passes, and then a landing.

- Flight 9: Custom controller test. Flight 9 was an attempt to copy flight 8 but with the custom controller enabled. Wind increased to 13 km/h with 26 k/h gusts. The flight consisted of a takeoff, two complete passes, and then a landing.

- Flight 10: Testing 85-degree pitch. In flight 10, the pitch limit was increased to 85 degrees, representing full forward flight. Multiple passes were done with both the base and custom controller. However, for an unknown reason, the data was not saved to the flight controller, so it could not be investigated or presented. Visually, the aircraft had better pitch control with reduced oscillations when using the custom

controller. After this test, the aircraft was fully shut down, and the battery was changed to attempt to prevent data loss in the next test.

- Flight 11: Testing 85-degree pitch. The purpose of this flight was to attempt to recreate the visual success of Flight 10 and save the data for investigation. The wind for this test was 15 km/h with up to 30 km/h gusts. The flight path for this test is shown in Figure 5.2, with the red path showing manual operation in Loiter mode and the orange showing GPS waypoint navigation in Guided mode. The test took place on a narrow farm field, and the aircraft was flown above the trees. This was the last complete test flight for the thesis. The results of Flight 11 are those presented in the comparative studies in Sections 5.3 and 5.4.

- Flight 12: Not completed. Another test flight was planned to gather more data for the comparison of the flight controllers. However, when increasing the altitude, the wind gusts were making it difficult to control and safely fly the aircraft, so it was landed, completing the testing for the day. Due to weather and logistics, this was the last day of testing.

5.3 Prototype and Simulated Aircraft Comparison

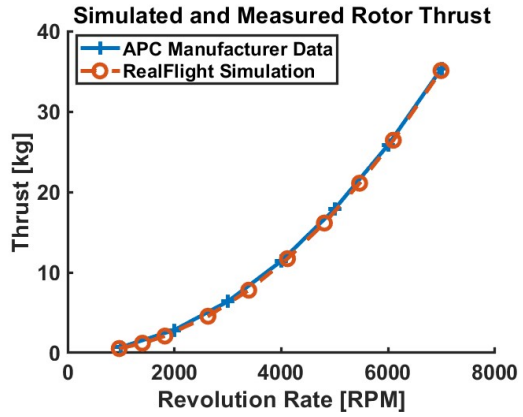
The purpose of the simulation is to create an accurate model of the prototype RPAS designed and tested throughout the thesis. An essential aspect of the aircraft to replicate precisely is the rotors, since they produce the lift in hover and the thrust in forward flight, and their combined thrust and torque are responsible for the control in all modes. The known parameters include the propeller diameter, pitch, and type, which were set to 381 mm, 254 mm, and APC electric, respectively, to match the propeller used on the prototype [90]. To better match the expected thrust at the correct rotational velocity and current draw, the simulated motor settings were adjusted so the motor had a resistance of $0.15\,\Omega$, and an idle current of $1.1\,\text{A}$ at $10\,\text{V}$. With these settings, the simulated rotors are compared to the APC 15x10 inch electric propeller data [90], shown in Figure 5.3. The APC propeller data from the manufacturer was simulated using vortex theory over a range of RPM, then verified with real thrust tests [91]. The measurements from the RealFlight simulation were obtained by modulating the thrust signal and using the thrust, torque, power draw, and RPM readings within the simulation to determine the rotor performance. The results from each of the plots in Figure 5.3 indicate that the simulation accurately represents the rotor configuration used on the prototype.

Flight 11 was used for the comparison between the prototype and simulated aircraft due to its abundance of data and completion of multiple transitions. A test was conducted in the simulation environment, matching the wind speeds and giving similar inputs. The purpose of these tests are to compare the vehicle's behaviour from the test flights and in simulation, as summarized in Figures 5.4 and 5.5. Each of the data points presented from

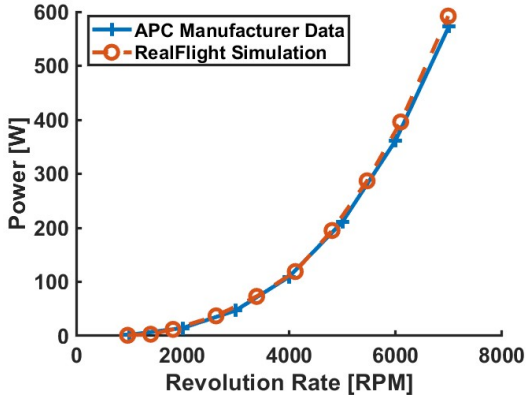
Figure 5.2: Flight path and location for flight 11 testing. The red lines represent passes made in loiter mode with manual control, and the orange lines are GPS waypoint passes in guided mode.

these tests represents an instance in time where a measurement of the aircraft properties was taken, such as velocity, current draw, and throttle level. Just as with Figure 5.1, the variable in the data presented is due to severe wind disturbance during the test flight, the non-rigid motion, the inertial effect of the aircraft, and measurement error. The fitted lines in each of the plots are a second-order polynomial created using MATLAB's curve fitting toolbox with bisquare robustness [125]. The fitted curves for each case in Figures 5.4 and 5.5 are given in Table 5.3. The overall observations are that each of the plots has the same trends with similar magnitudes, indicating that the complex dynamics of the tailsitter are represented well in the simulation.

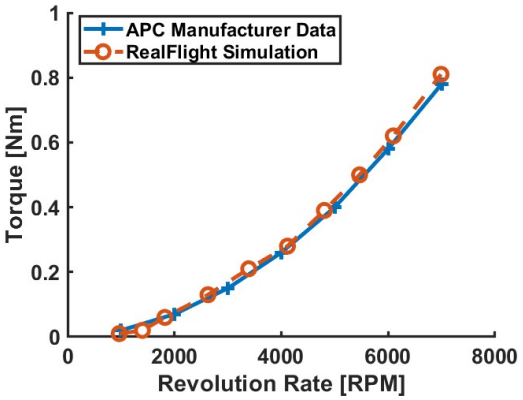
The current draw for the prototype and simulated model are shown in Figures 5.4(a) and 5.4(b) respectively. In both cases, the current draw is significantly reduced as the angle of attack increases, with a similar range of disparity from the average line shown. As the aircraft gets closer to forward flight, the disparity increases, especially around 75-90 degrees, representing how the vehicle suddenly requires large inputs from the motors to correct for the disturbances. These results show that the simulated aircraft produces a



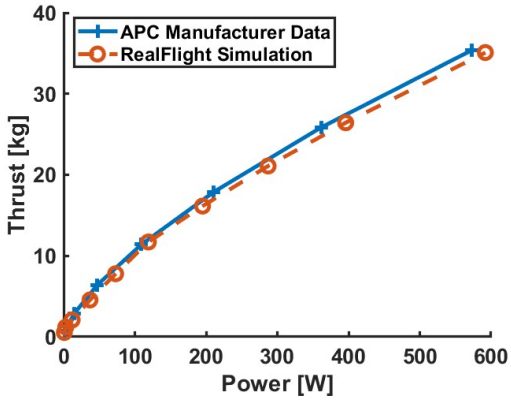
(a) Rotor thrust versus rotational speed.



(b) Output power versus rotational speed.

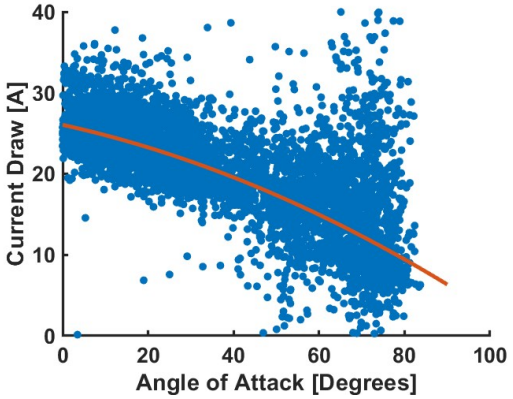


(c) Rotor torque versus rotational speed.

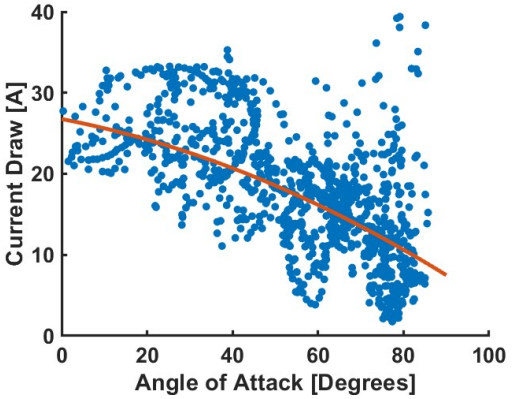


(d) Rotor thrust versus output power.

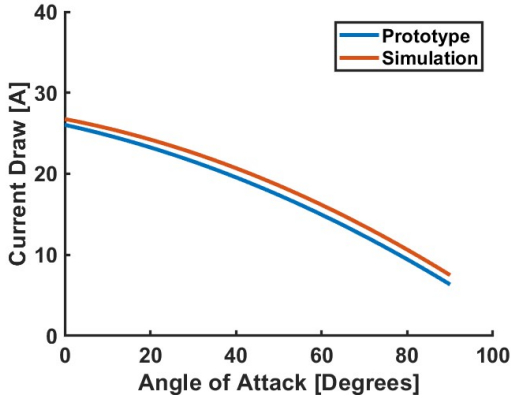
Figure 5.3: RealFlight 9.5 simulated and posted rotor performance data [90].



(a) Prototype current draw versus angle of attack from Flight 11.



(b) Simulated current draw versus angle of attack.

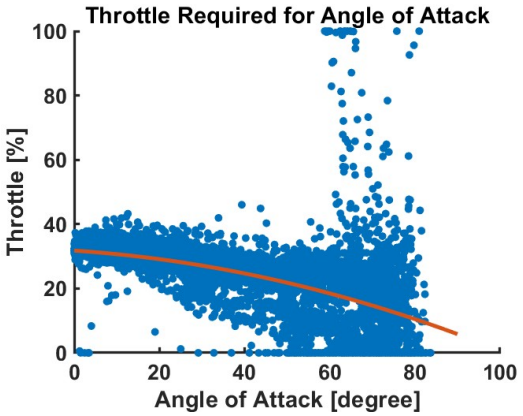


(c) Fitted lines representing real and simulated current draw.

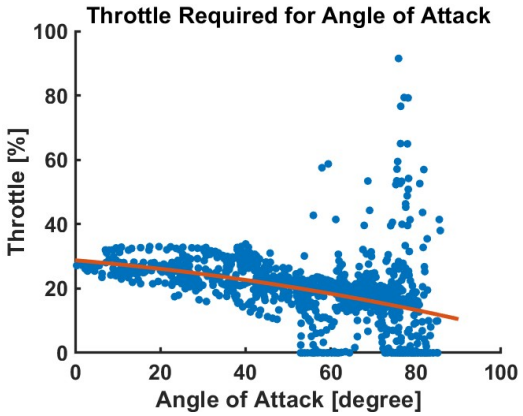
Figure 5.4: Comparison of real and simulated current draw to validate the accuracy of the simulated model.

similar amount of thrust and lift from the motors and wings from hover to forward flight, so a similar current is used. The simulated current draw is always slightly higher than what was measured for the prototype vehicle, which would result in shorter flight times in the modelled environment shown by the average fit of both in Figure 5.4(c).

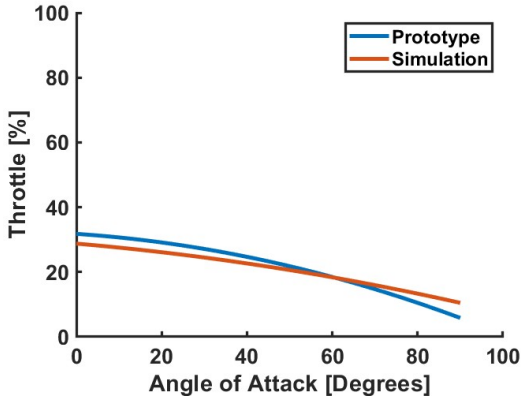
The simulated throttle level also had very similar properties to the prototype, based on the AoA shown in Figures 5.5(a) and 5.5(b). Both the amount of variance around the fitted curve and the increase in variation when the AoA is above 60 degrees are similar. The fitted curve for the prototype and simulated throttle level are compared directly in Figure 5.5(c). A similar amount of current draw throughout the transition shows that the simulated blown wings are creating lift, validating the blown wing effect in the simulation environment. However, the throttle is not reduced as much once forward flight is achieved, indicating that the drag in this orientation is too large and should be reduced in future iterations. Additionally, requiring more throttle but utilizing the same amount of current



(a) Prototype throttle output versus angle of attack.



(b) Simulated throttle output versus angle of attack.



(c) Fitted lines representing real and simulated throttle output.

Figure 5.5: Comparison of real and simulated throttle level to validate the accuracy of the simulated model.

could indicate that the simulated propeller has different dynamic thrust properties than the actual propeller.

The last comparison for the prototype and simulation is the airspeed and AoA relationship shown in Figure 5.6. The simulated vehicle can reach similar ground speeds as the prototype aircraft but tends to fly slower for the same AoA. Therefore, the simulated body drag in forward flight is likely too large, correlating with the results discussed for the throttle above. Also, there is an apparent hysteresis effect, which is due to the inertia of the vehicle as it is increasing or decreasing its speed by modulating pitch along with the impact of wind when flying upstream or downstream. These results further validate the ability of the simulation to model the aerodynamics of the tailsitter.

The results presented in this section show that both the rotor configuration and the aerodynamics of the vehicle are accurate representations of the prototype aircraft. However, there

Table 5.3: Second-order equations of best fit comparing prototype and simulated tailsitter performance obtained using MATLAB's curve-fitting toolbox [125].

Independent Variable	Dependant Variable	A	B	C	R
Prototype Current Draw [A]	Angle of Attack [deg]	-1.134E-03	-0.1174	26.06	0.7673
Simulated Current Draw [A]	Angle of Attack [deg]	-1.248E-03	-0.1018	26.76	0.6589
Prototype Throttle [%]	Angle of Attack [deg]	-2.220E-03	-0.0885	31.78	0.2984
Simulated Throttle [%]	Angle of Attack [deg]	-1.001E-03	-0.1130	28.77	0.1578

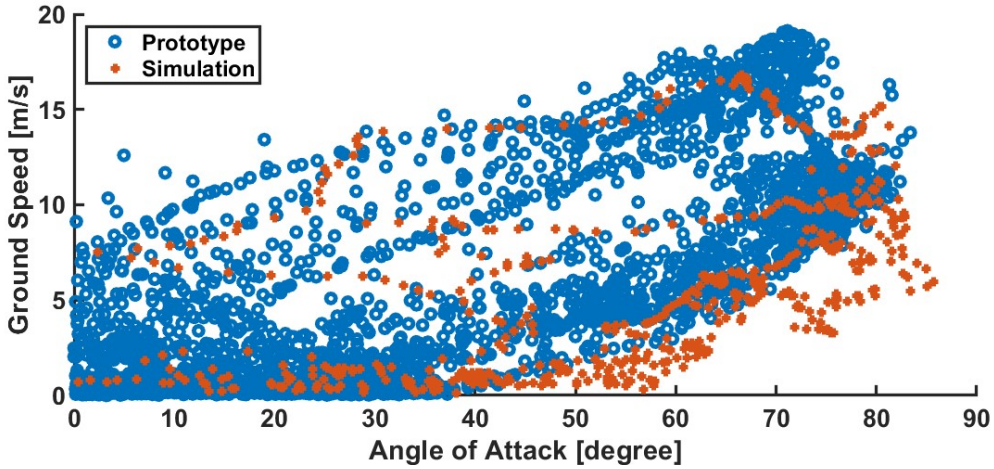
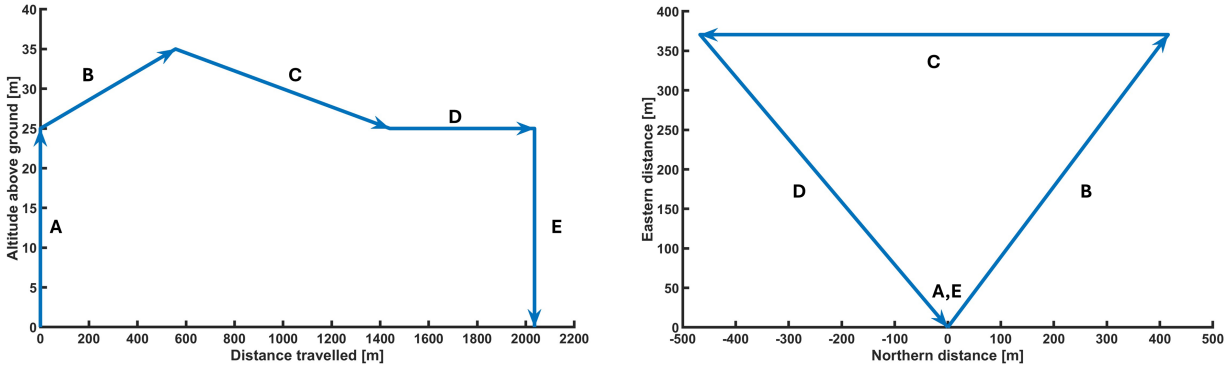


Figure 5.6: Comparison of speed versus AoA relationship for prototype and simulated aircraft. Each data point represents a measurement of the state of the aircraft at that instance.

are areas for improvement in future iterations, especially regarding the drag of the aircraft. Therefore, the use of this simulation for testing the controller developed should give results that reasonably represent what would occur when testing the prototype aircraft.

5.4 Controller Comparison

To validate the custom flight controller implemented, testing was carried out with both the simulation and the prototype aircraft. The results from the simulation tests are used to show the capabilities of the custom controller in ideal conditions and with repeatable test flights. The results from the prototype tests are to prove that the custom control is viable and effective on real hardware.



(a) Vertical path.

(b) Planar path.

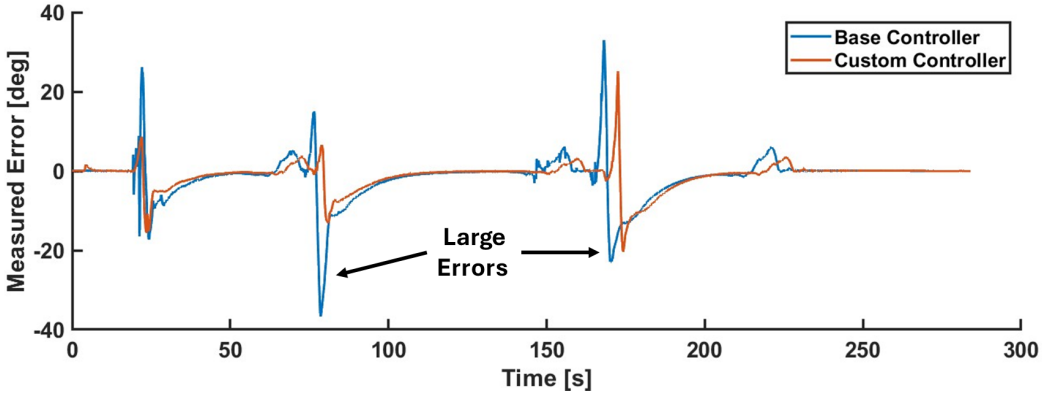
Figure 5.7: Simulated test flight path for flight controller comparison.

5.4.1 Simulation Testing using RealFlight 9.5

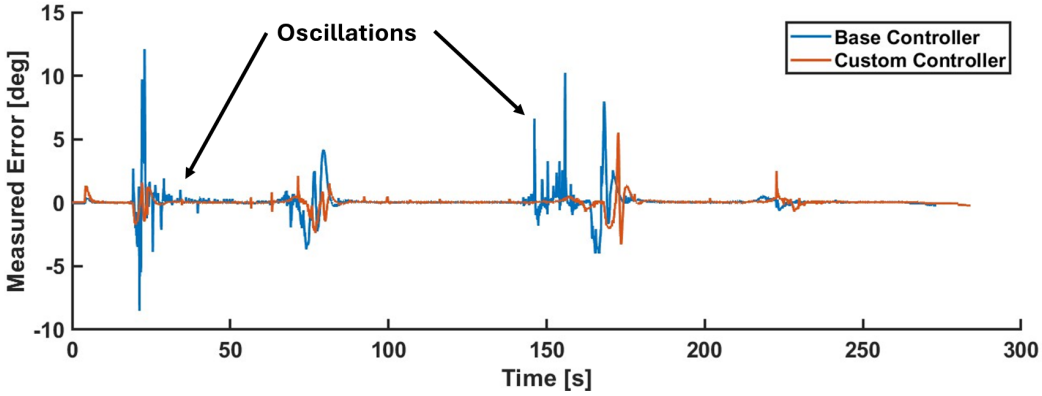
The simulated test flight starts with a vertical takeoff, then a triangular flight pattern is covered by three legs shown in Figure 5.7. Each leg includes a forward transition, straight-line forward flight, and a transition back to hover. The test concludes with a vertical landing at the takeoff location. The entire flight path for the test is summarized in Figure 5.7, with Section A indicating the takeoff, Sections B, C, and D representing the forward flight legs, and Section E showing the vertical landing. In takeoff, the aircraft rises to an altitude of 25 m. Then, for Sections B and C of the test, the altitude is increased and decreased by 10 m, respectively. Forward flight in Section D has a constant altitude, followed by a landing in Section E by reducing the altitude by 25 m. For each test, four trials were conducted for both controllers to ensure the validity of the results. These tests were different from the straight-line tests used on the prototype, so the wind was blowing on the aircraft from different directions. However, the testing used in the simulation and with the prototype both consisted of straight-line transitions and turning while in hover.

The first trial had no wind, simulating ideal conditions, with the results shown in Figure 5.8. The measured error for each axis is presented, showing how the custom VP + FPID controller outperforms the base P+PID controller. Figure 5.8(a) shows cases where the base controller experiences much larger errors than the custom controller in the roll axis. Another benefit of the custom controller is the reduction of oscillations experienced in forward flight and the transition, especially in the pitch axis shown in Figure 5.8(b). The larger gains available in hover, combined with the higher gains accessed by the fuzzy logic PID portion of the custom controller when the errors are increasing, enable the system to respond much faster. The faster response produces a faster recovery to errors, as shown by the yaw response in Figure 5.8(c).

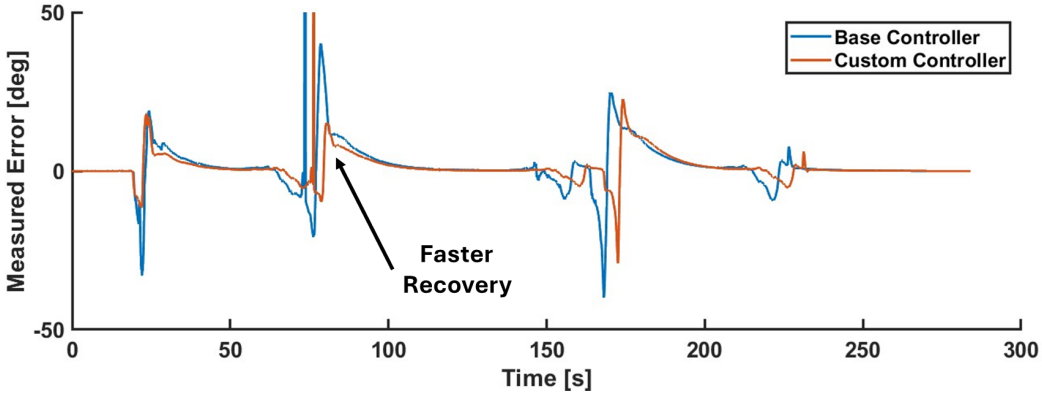
The gains used for the variable proportional portion of the custom controller are based on the aircraft pitch, with the calculated values shown in Figure 5.9. When in hover, larger gains are output by the FLC, but smaller gains are required to prevent the oscillations in



(a) Roll error.



(b) Pitch error.



(c) Yaw error.

Figure 5.8: Measured errors for both controllers in the roll, pitch, and yaw axes for a simulated flight test with no wind.

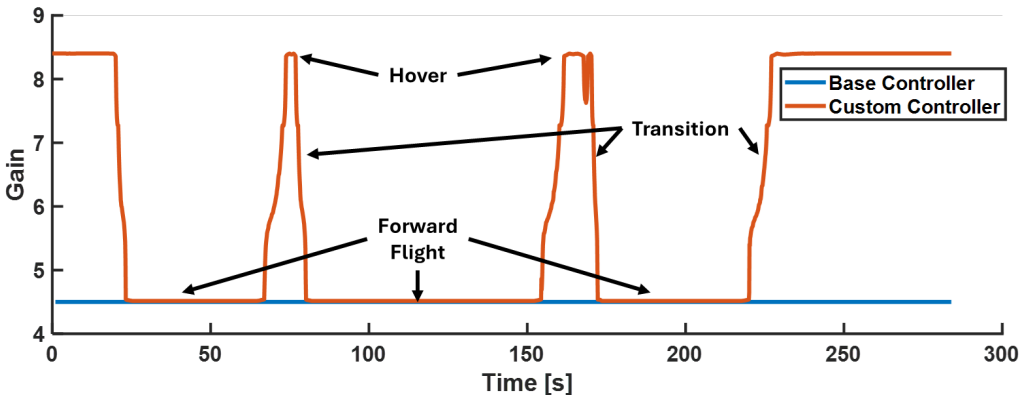
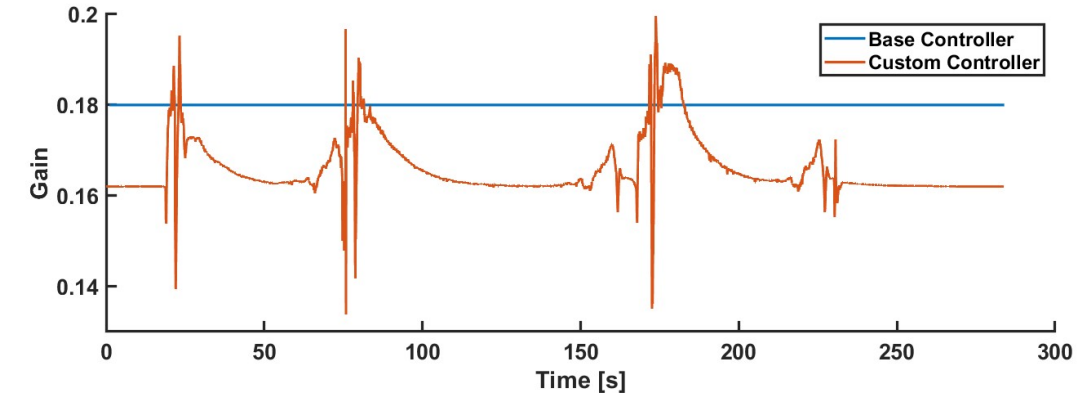


Figure 5.9: Comparing the gain of the base proportional controller with the calculated gain of the custom variable proportional controller for the yaw axis in a simulated test with no wind.

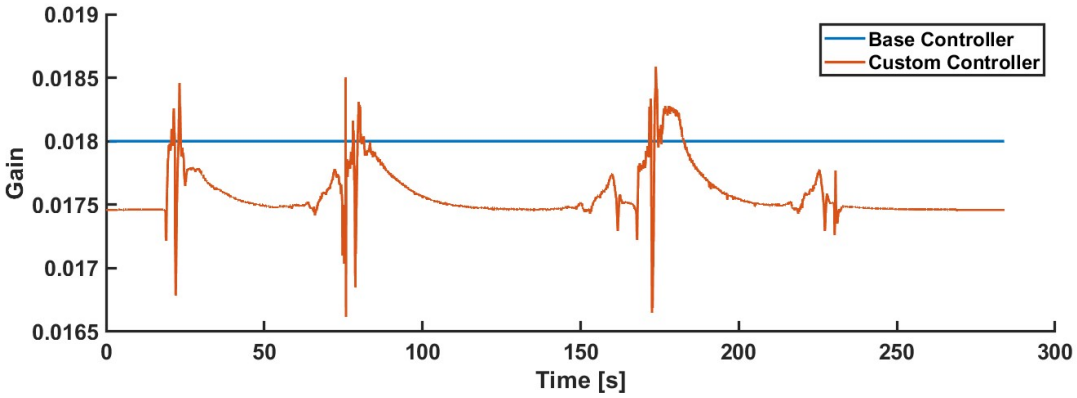
forward flight. The transition periods between hover and forward flight result in changes in the outputted gains, represented by the diagonal lines. The base controller always used the lowest gain of the custom controller because even though a larger constant value would improve responsiveness, it also causes oscillations and instability if used in forward flight.

The gains calculated by the fuzzy logic system for the yaw PID controller for the simulated test are shown in Figure 5.10. The proportional and integral signals have the same trend but are scaled differently to match the base controller gains. The derivative gain has the opposite response as expected by the opposite rule set. In general, the proportional and integral gains are lower to make the system more stable and robust by preventing oscillation and overshoot, but when high errors are detected, the controller uses larger gains to correct faster. Conversely, the derivative gains are reduced in this case to also improve responsiveness, but large derivative gains are seen when the error derivative is large to stabilize the system.

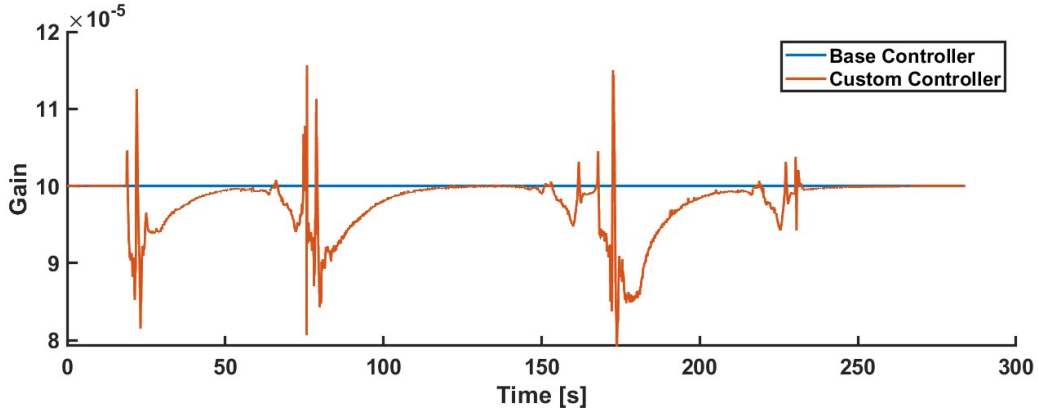
The same test was repeated for a total of four trials to improve the reliability of the results. Then, wind was added in the simulation environment, with the speeds rated at 5 km/h with 10-12 km/h gusts. Four trials were also used, but the direction was changed to North, East, South, and West for each test. The same process was repeated for wind speeds of 10 km/h with 20-25 km/h gusts. The maximum and average measured errors for each of the cases are presented in Table 5.4. In nearly every axis in every test, both the maximum and average errors are improved when using the custom VP + FPID controller. Therefore, in simulation, the custom controller implemented can control the tailsitter aircraft throughout the entire flight regime in ideal conditions and in the presence of significant disturbances like wind.



(a) Proportional gain.



(b) Integral gain.



(c) Derivative gain.

Figure 5.10: Comparing the gains of the base PID controller with the calculated gains for the custom fuzzy PID controller for the yaw axis in a simulated flight test with no wind.

Table 5.4: Measured error in simulated test flights for the base and custom controllers.

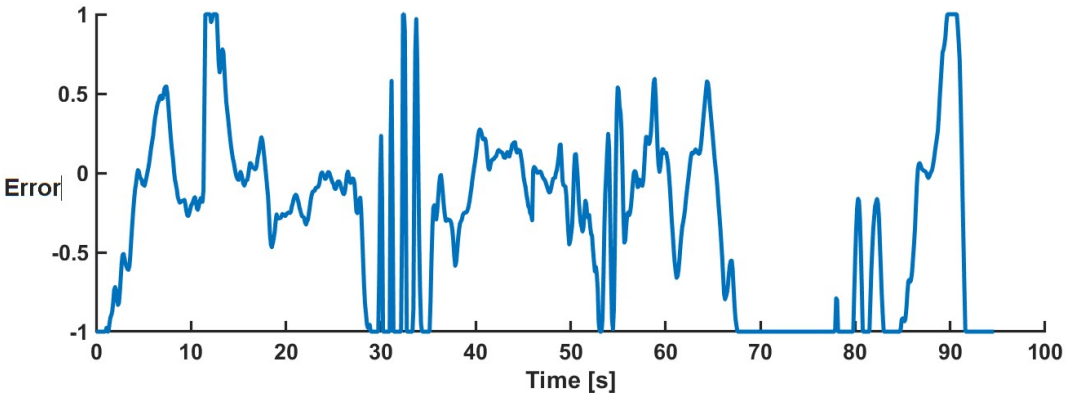
Test Configuration		Measured Errors [degrees]					
		Roll		Pitch		Yaw	
Controller	Direction	Average	Maximum	Average	Maximum	Average	Maximum
No Wind							
P+PID	NA	0.33	20.76	0.56	13.24	1.28	15.49
P+PID	NA	0.38	20.21	1.02	35.76	1.33	12.88
P+PID	NA	0.21	10.06	0.37	7.34	1.25	11.96
P+PID	NA	0.48	15.88	0.95	35.23	1.36	12.71
VP + FPID	NA	0.17	1.94	0.26	2.54	0.80	9.86
VP + FPID	NA	0.18	5.25	0.27	4.36	0.78	9.97
VP + FPID	NA	0.20	5.38	0.33	3.84	0.82	9.33
VP + FPID	NA	0.16	3.07	0.27	5.70	0.80	9.81
5 km/h Wind							
P+PID	North	0.50	6.51	1.06	22.68	2.07	14.23
P+PID	East	0.51	7.36	1.03	24.24	2.14	16.85
P+PID	South	0.45	5.58	0.76	10.73	2.46	13.73
P+PID	West	1.04	61.88	1.55	60.39	2.70	21.70
VP + FPID	North	0.42	8.10	0.74	19.45	1.37	11.76
VP + FPID	East	0.43	8.25	0.73	11.82	1.40	12.95
VP + FPID	South	0.35	3.09	0.59	10.95	1.61	9.79
VP + FPID	West	0.49	14.22	0.93	22.36	1.39	10.81
10 km/h Wind							
P+PID	North	1.47	35.32	1.95	38.29	3.37	20.76
P+PID	East	1.36	8.29	0.99	18.65	3.22	16.11
P+PID	South	0.81	6.49	1.43	21.16	4.27	17.83
P+PID	West	1.05	10.60	1.79	44.17	3.71	18.38
VP + FPID	North	0.67	5.09	0.94	13.80	2.30	11.10
VP + FPID	East	0.73	5.70	0.99	24.79	2.06	13.63
VP + FPID	South	0.72	4.15	0.88	10.17	3.20	17.29
VP + FPID	West	0.96	25.25	1.25	23.53	2.71	13.12

5.4.2 Prototype Aircraft Testing

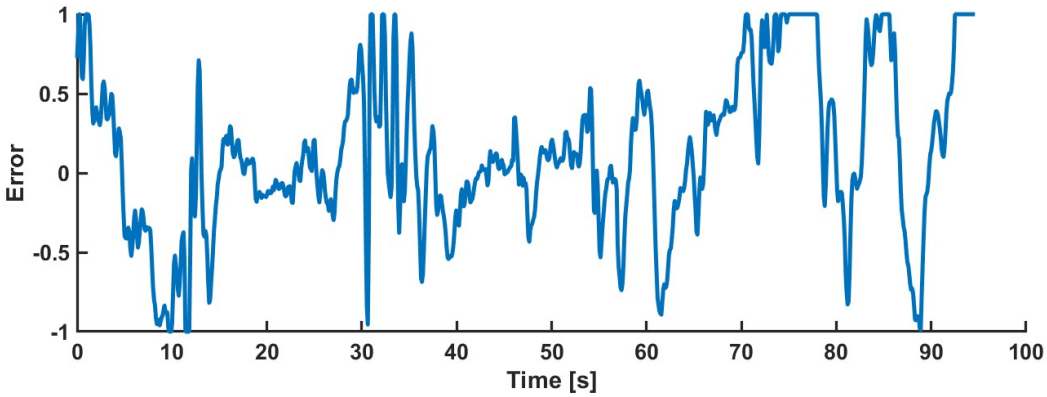
Testing was then conducted on the prototype aircraft following the procedure defined in Section 5.2 to compare the performance of the two controllers. First, the gains that were presented in Table 4.8 were tuned using data from flight 7. The goal was to set the scaling gains for the measured error such that a standard flight regime in reasonable winds uses the entire fuzzy logic map. If the gains set the input values too large, the FLCs will get easily saturated and mostly output the minimum or maximum values. In contrast, if the inputs become too small, the maximum output signals will not be utilized. The attitude errors measured by the autopilot throughout the flight, after being divided by the K_e gain defined in Table 4.8 and saturated between -1 and 1, are shown in Figure 5.11. The measured errors shown for each axis indicate that the entire range of input values is used while still being between the saturation limits of -1 and 1 in each axis for most of the test.

The error derivative signals shown in Figure 5.12 are significantly more erratic than the error signals themselves. The increased variability is expected when differentiating a noisy and rapidly changing signal, such as the aircraft's error. However, the error derivative was intentionally left unfiltered to preserve key features that indicate when the signal begins to change direction or oscillate, which could be lost with filtering. Additionally, the use of filters commonly delays the resulting signal, which would make this system less responsive to the oscillations and rapid attitude changes it is trying to correct. The error derivative signal was also saturated between -1 and 1 after being divided by the K_{ed} gains. The brief duration of the peaks at the minimum and maximum values suggests that the scaling gains for the input signals are appropriately set.

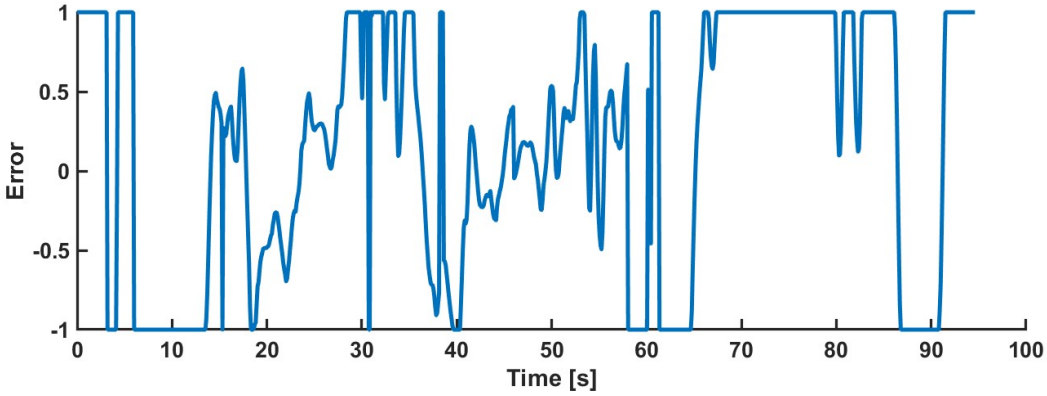
The test flight conducted on the prototype aircraft involved multiple passes where the vehicle would start in hover, then fly forward as it transitioned into forward flight. After a short time in forward flight, the vehicle would then transition back into hover. Then the aircraft would turn around in hover and repeat the transitions and forward flight. After each test, the flight controller was switched between the base and custom configurations to ensure the wind was as similar as possible for both cases. Two tests with two passes each were conducted for each controller in flight 11. The wind for these tests was 15 km/h with gusts exceeding 30 km/h. The errors measured by the flight controller of the base and custom controllers are shown in Figure 5.13. In the roll axis, the second forward flight pass from 40 to 60 seconds in Figure 5.13(a) highlights the reduction in measured errors, reducing from a maximum value of 5.6 to 3.1 degrees. The forward flight for pitch in Figure 5.13(b) also reduces the large error experienced by the base controller from 45 to 50 seconds. For the yaw axis in Figure 5.13(c), the magnitude of the measured errors is not significantly improved, but both the frequency and gain of oscillations are reduced throughout the test. These reductions are especially noticeable from 20 to 30 seconds and from 40 to 55 seconds in the test.



(a) Error measured in roll axis.

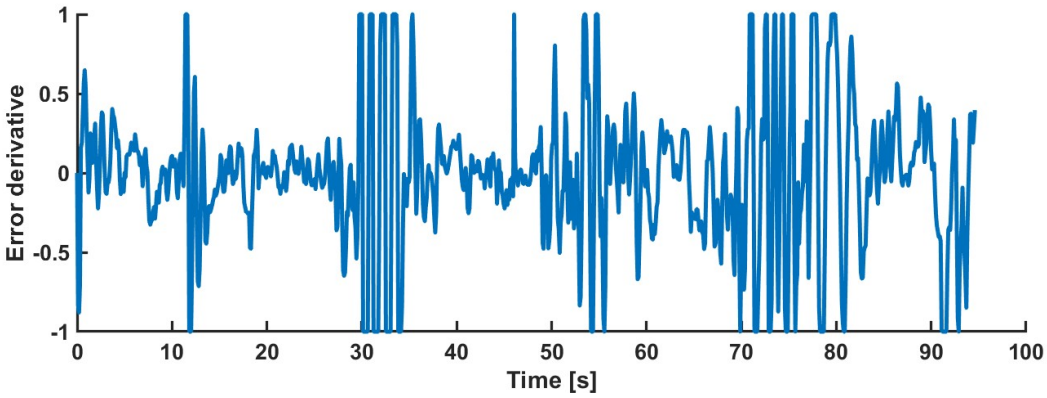


(b) Error measured in pitch axis.

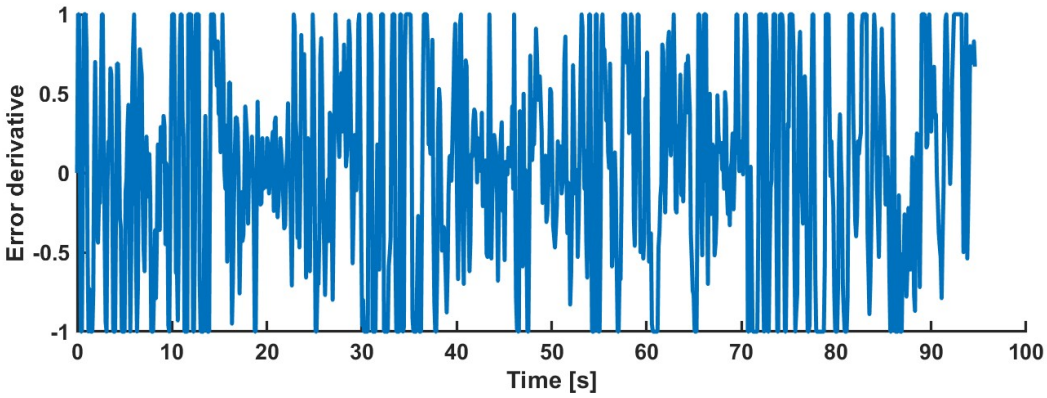


(c) Error measured in yaw axis.

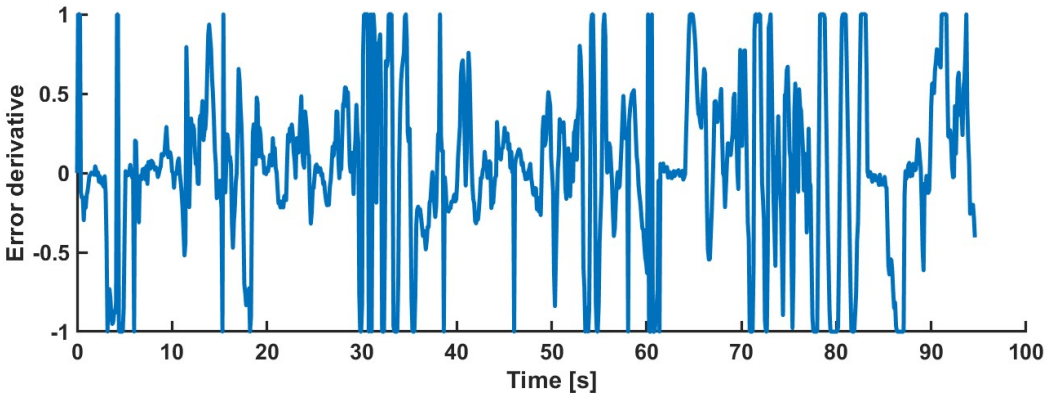
Figure 5.11: Error after being divided by K_e gain for each axis and saturated between -1 and 1 on a test flight with the prototype aircraft.



(a) Error derivative measured in roll axis.

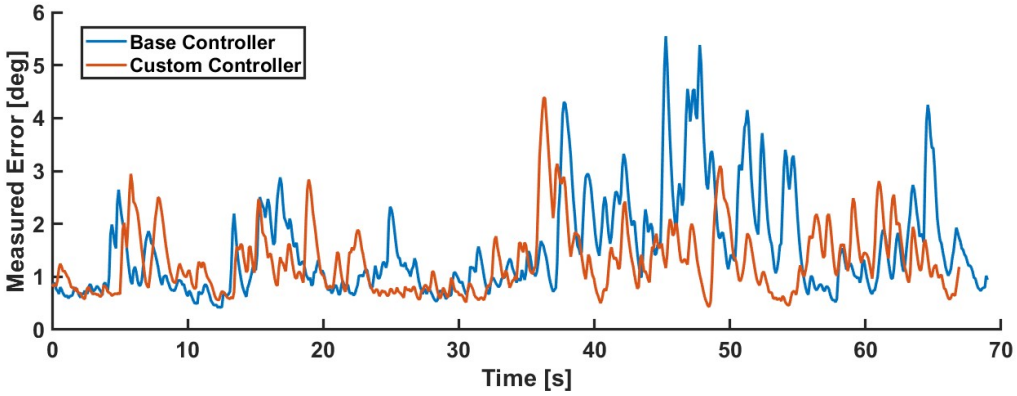


(b) Error derivative measured in pitch axis.

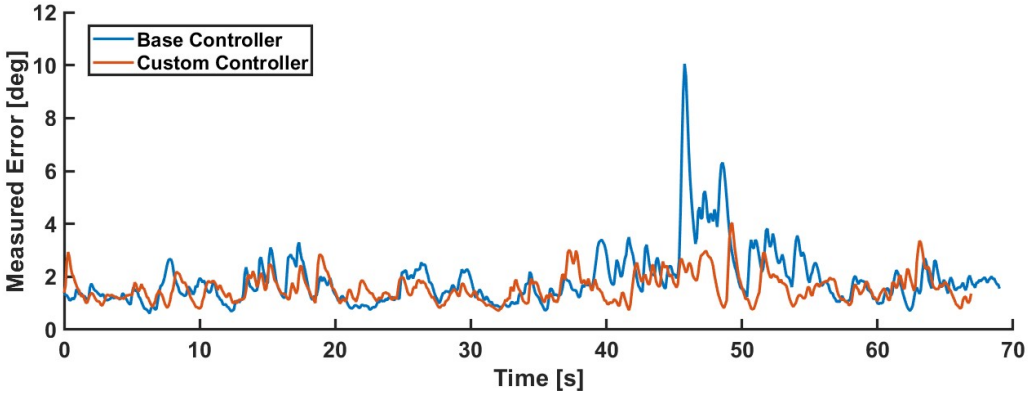


(c) Error derivative measured in yaw axis.

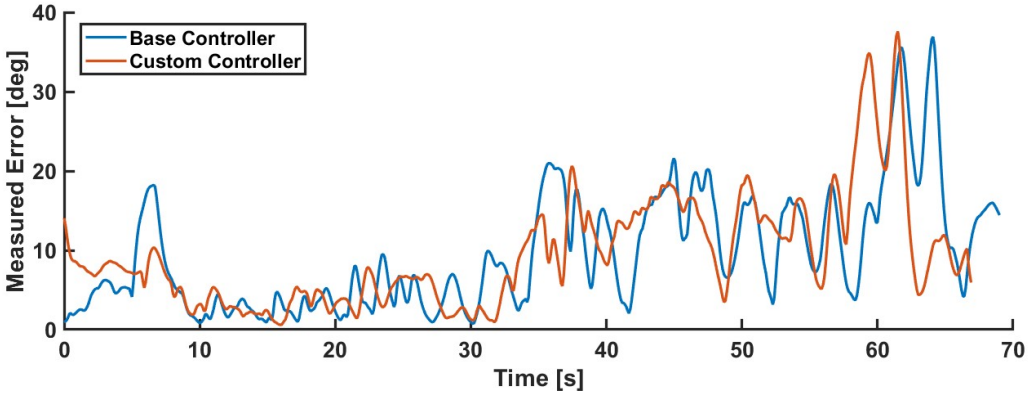
Figure 5.12: Error derivative after being divided by K_{ed} gain for each axis and saturated between -1 and 1 on a test flight with the prototype aircraft.



(a) Roll error.

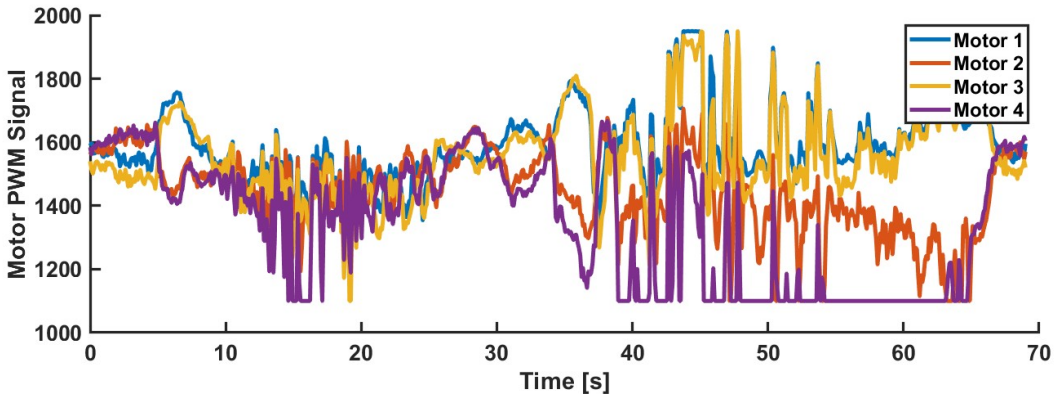


(b) Pitch error.

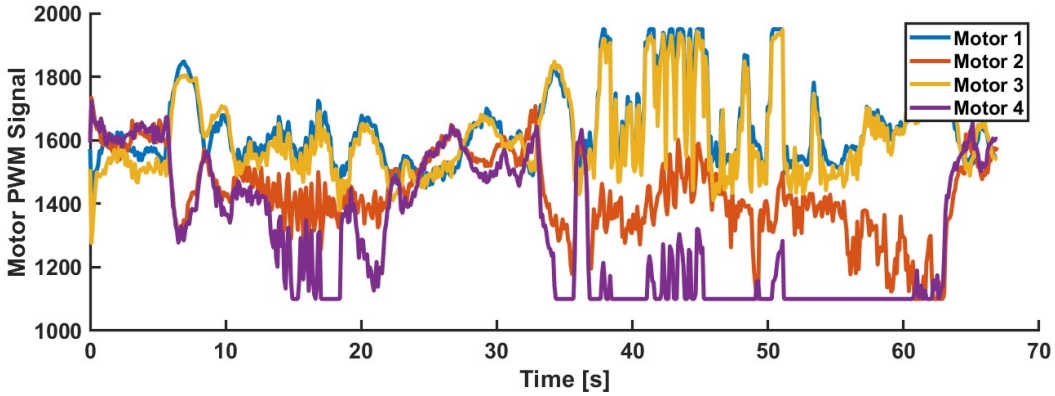


(c) Yaw error.

Figure 5.13: Measured errors for base and custom controllers in roll, pitch, and yaw axes for flight 11 with high winds.



(a) Base P+PID controller.



(b) Custom VP + FPID controller.

Figure 5.14: Motor signals for base and custom controller for real test flight of two transition passes with high winds from flight 11.

An alternative way to observe oscillations is to look at the motor outputs, which are responsible for creating oscillations in a poor controller or, ideally, correcting them. The four motor outputs for both controllers in the above test are shown in Figure 5.14. In the first forward flight pass from 10 to 25 seconds, both the base and custom controllers have oscillating motor signals, but the magnitude of oscillation is significantly reduced for the custom controller. The same trend is also observed for the second pass, from 35 to 55 seconds. Alternatively, the power requirement of the aircraft can be measured to show oscillation in the motors since the motors are the primary source of power draw. The current draw for the same test is shown in Figure 5.15. While oscillations are observed for both controllers, the magnitude and frequency are significantly reduced for the custom controller, indicating that the motors and vehicle are oscillating less.

In Figure 5.16, the change in proportional gain as the aircraft pitches between hover and forward flight is shown. As expected from the simulation, the custom controller accesses higher gains when in hover and lower gains in forward flight to improve repressiveness and stability. The proportional, integral, and derivative gains in the yaw axis PID controller

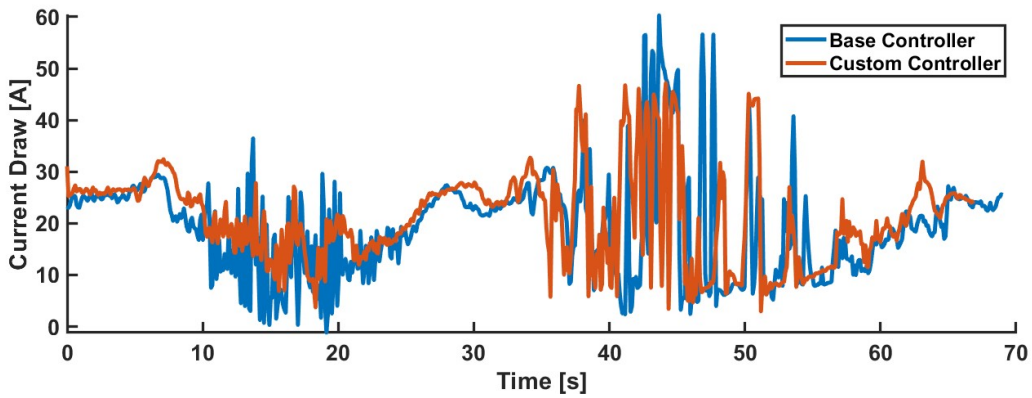


Figure 5.15: Combined current draw of motors of base and custom controller for flight 11.

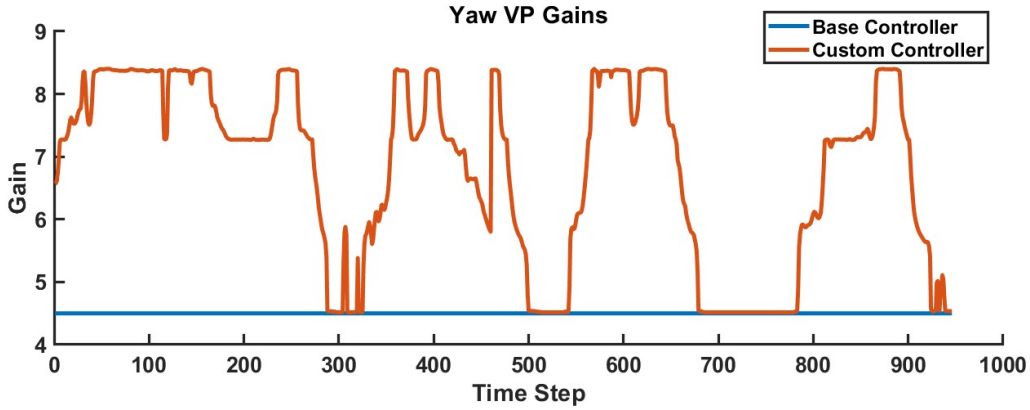
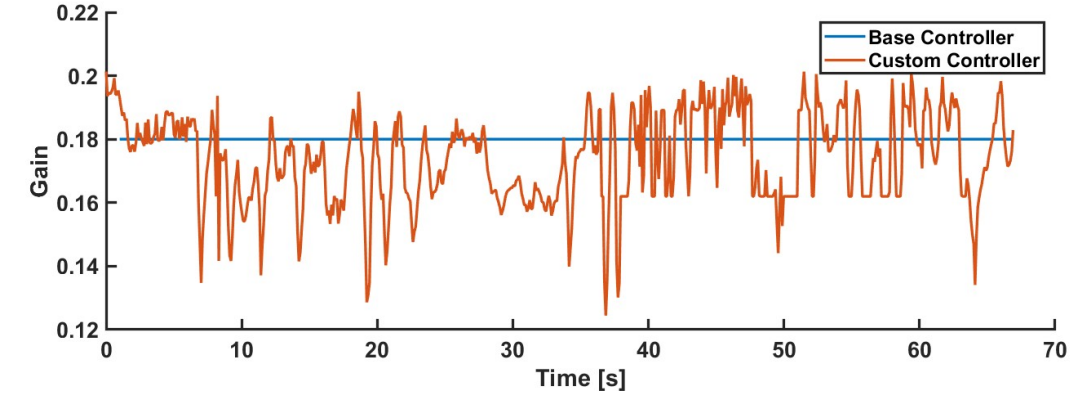


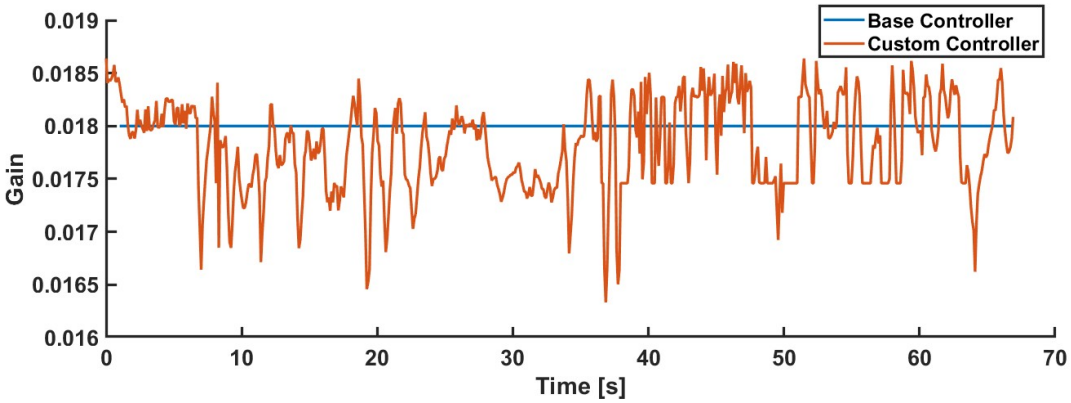
Figure 5.16: Comparing the gain of the base proportional controller with the calculated gain of the custom variable proportional controller for the yaw axis in two passes of flight 11 with high winds.

based on the error and error derivative of the vehicle are shown in Figure 5.17. Each of these responses has gains that are similar to the simulated results shown in Figure 5.9, illustrating that the controller is working nominally.

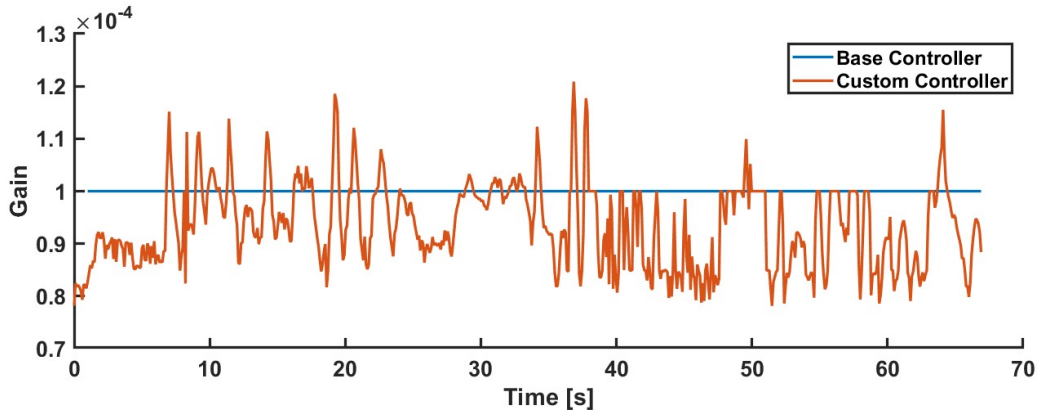
The results from the four tests in flight 11 are summarized in Table 5.5. The first round of testing used manual control of the aircraft in Loiter mode, resulting in larger errors measured for both controllers. Loiter mode is used to maintain the current heading, altitude, and position, but the joysticks of the controller can be used to change the desired aircraft motion [121]. The second round of tests used waypoint navigation in Guided mode, which improved the errors experienced by the aircraft. Guided mode is activated through the ground control station (Mission Planner) by selecting a point on the map and telling the aircraft to fly there [120]. In both test cases, the average and maximum error was reduced for each axis, showing how the custom controller is capable of responding faster while also reducing oscillations and increasing stability.



(a) Proportional gain.



(b) Integral gain.



(c) Derivative gain.

Figure 5.17: Comparing the gains of the base PID controller with the calculated gains for custom fuzzy PID controller for the yaw axis in two passes of flight 11 with high winds.

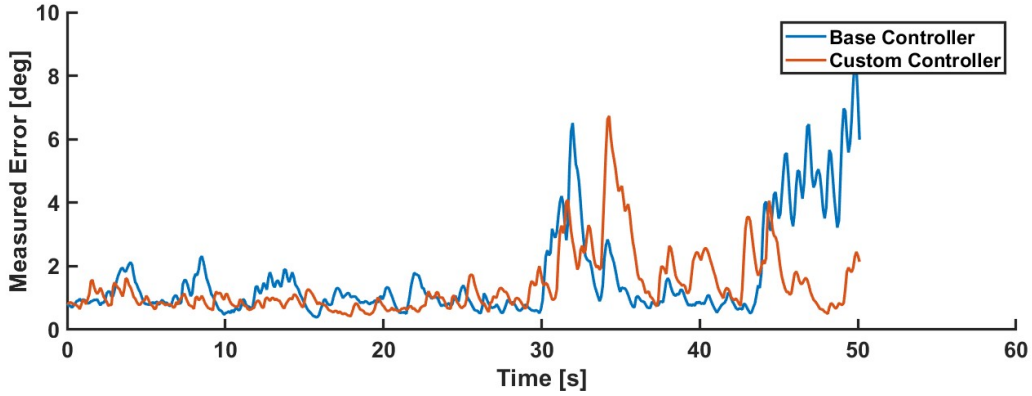
Table 5.5: Measured errors from the flight controller in prototype aircraft test flights for the base and custom controllers from flight 11.

Test Configuration		Measured Errors [degrees]					
		Roll		Pitch		Yaw	
Controller	Mode	Average	Maximum	Average	Maximum	Average	Maximum
15 km/h Wind							
P+PID	Loiter	2.11	20.69	1.99	7.78	12.56	38.10
VP + FPID	Loiter	1.95	11.71	1.61	6.02	10.43	38.10
P+PID	Guided	1.27	5.55	1.61	10.05	9.44	41.34
VP + FPID	Guided	1.17	4.42	1.48	4.04	8.59	37.60

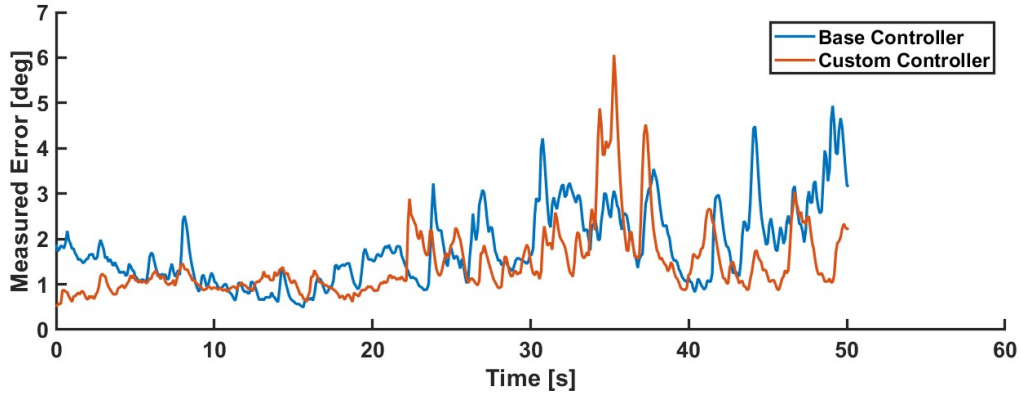
Flights 8 and 9 were meant to get a baseline flight for the test day and ensure that the new custom controller was stable and tuned appropriately by checking for oscillations. However, since there is less data than desired, these tests are also being compared. A full pass lasting 50 seconds was conducted using both controllers, which were similar to each other and can be used for comparison. The measured error in each direction is shown in Figure 5.18, and the results are summarized in Table 5.6. In all axes, the average of the measured error is reduced due to the faster response and reduction in oscillation of the system. In roll, the average error is reduced by 17 percent, with the maximum value measured reduced by 21 percent. In yaw, the average and minimum errors were reduced by over 60 percent each, vastly improving the positional control of the aircraft. The pitch axis showed a decrease in average error by 19 percent, but a large gust caused a big pitch error, increasing the maximum value measured by 23 percent. The large spike indicates that further tuning of the error derivative in the pitch axis to improve the response to disturbances would be beneficial.

Table 5.6: Comparison of average and maximum errors of base and custom controllers in a straight pass from flights 8 and 9.

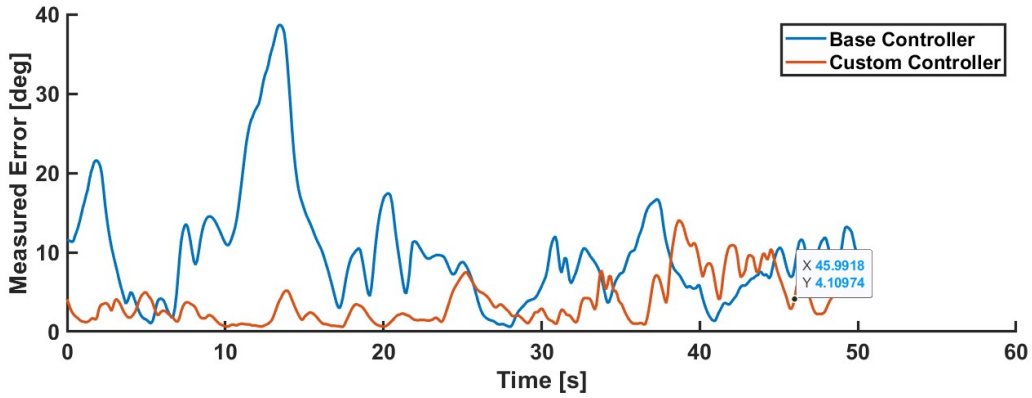
Test Configuration		Measured Errors [degrees]					
		Roll		Pitch		Yaw	
Controller	Mode	Average	Maximum	Average	Maximum	Average	Maximum
12 - 13 km/h Wind							
P+PID	Loiter	1.66	8.52	1.81	4.94	10.09	38.68
VP + FPID	Loiter	1.38	6.73	1.46	6.06	3.85	14.05



(a) Roll error.



(b) Pitch error.



(c) Yaw error.

Figure 5.18: Measured errors for both controllers in the roll, pitch, and yaw axes for real test flights 8 and 9.

5.4.3 Controller Results Summary

The average improvement of the three different simulated tests and the prototype tests is shown in Table 5.7. The average and maximum measured errors are reduced in all cases, especially with low and no wind, as demonstrated by the simulation cases. When the wind speed increases, the average improvement of the measured errors decreases, which is illustrated through the high wind simulations and prototype testing. From the tests using the prototype aircraft, the maximum errors are reduced by 24 to 34 percent in each axis, and the average error is reduced by 11 to 29 percent in each axis. The average and maximum measured errors are improved in all axes at all wind speeds tested, showing that the use of fuzzy logic to adapt the controller gains has significantly enhanced the attitude regulation of the tailsitter.

Table 5.7: Average error improvement of custom versus base controller from all test flights.

Test Configuration		Average Improvement on Measured Errors [%]					
		Roll		Pitch		Yaw	
Test	Wind [km/h]	Average	Maximum	Average	Maximum	Average	Maximum
Simulation	0	-49.18	-76.64	-61.40	-82.04	-38.71	-26.54
Simulation	5	-32.25	-58.63	-32.07	-45.30	-38.41	-31.86
Simulation	10	-34.18	-33.80	-34.03	-40.88	-29.52	-24.55
Prototype	12-15	-10.83	-34.23	-15.84	-29.24	-28.73	-24.03

Chapter 6

Conclusion

The work presented in this thesis has expanded upon the current literature regarding tailsitter scalability, design, prototyping, simulation, and control using fuzzy logic to adapt PID attitude controller gains to make tailsitter design and development more attainable for future contributions. This thesis shows in detail how a tailsitter can be designed, simulated, and controlled through a prototype aircraft.

6.1 Contributions

The following contributions to maturing the literature regarding tailsitters are presented in this thesis:

- Development of a set of empirical scalability laws, illustrating how characteristics such as wingspan, reference area, payload and battery mass, and cruising speed relate to the MTOW. These scaling laws are then used to conduct a scalability study for the conceptual design of quadrotor tailsitters ranging from 1 to 25 kg. From this study, additional laws are extracted for tailsitters, expressing how the structural and EMP masses, range, and endurance each scale with MTOW. Finally, the scaling laws are utilized for the empirical conceptual design of a quadrotor tailsitter to validate their effectiveness in the design process. Design considerations specific to tailsitters and the fabrication of a prototype vehicle are also presented.
- A prototype aircraft was modelled using RealFlight 9.5 and ArduCopter for a SITL simulation. The simulation allows for the simultaneous testing of both the aircraft flight characteristics and the custom controller presented.
- The development and implementation of a controller to adapt the gains of ArduCopter's P+PID controller using fuzzy logic is then presented. It was tested both in simulation and on the prototype aircraft to validate its performance.

- The scalability laws, simulation, and controller are each validated through the development of a 5 kg prototype tailsitter. The mass distribution of the resulting prototype closely matched the scalability laws used for its design, and the resulting range and endurance aligned with what was predicted from the laws developed in the scalability study. The simulation was validated by comparing the results of test flights conducted in the simulation environment and on the prototype aircraft, showing that the rotors and aerodynamics are accurately represented. The custom controller implemented was tested using the simulation and prototype aircraft, proving the performance gains achieved are still present when exposed to disturbances such as wind and are practical to implement on real systems. The custom controller reduced the average and maximum attitude errors in all axes by 11 to 82 percent from ideal conditions to 15 km/h winds with 30 km/h gusts.

6.2 Discussion

Throughout the work conducted in this thesis, many lessons were learned and observations were made. The following list summarizes these results.

- The utilization of the blown wing discussed in the patent applied to the prototype aircraft successfully ensured the altitude could be controlled throughout the flight regime [143]. In the simulation, the altitude was increased, decreased, and maintained corresponding to the three legs of flight in the test pattern. For the prototype, the altitude was successfully maintained with minimal error throughout Flight 11 as the pitch was modulated, showing that accurate altitude control can be achieved for tailsitters, even in the presence of high wind.

- The wing design for the prototype was meant to maximize the wing area positioned behind the rotors to benefit from their airflow. However, the use of larger propellers, or more rotors positioned laterally, would enable the use of larger wingspans while still benefiting from the rotorwash. In contrast, a larger wingspan used on the same aircraft would result in sections of the wings that are not blown, which perform worse through the transition, but would have cleaner airflow for forward flight.

- The wings designed for the prototype are primarily a result of weight, time, and fabrication limitations. While more complex wings improve performance, they require much more detailed analysis and are more difficult to implement structurally, resulting in a larger structural mass. However, this vehicle would benefit from a wing design that mimics what is typically seen with flying wing aircraft, including wing twist and airfoil reflex for pitch stability, dihedral for roll stability, and wing sweep combined with variable chord lengths for more efficient forward flight.

- The scaling laws developed and utilized throughout this thesis are purely a result of the available data. Since there was little data available for tailsitters larger than 25 kg, the scaling laws presented should not be applied to RPAS much larger. Additionally,

the scaling laws are not proven to apply to very small and micro tailsitters, such as those under 250 g. Another point of caution is to avoid further populating the data with tailsitter designs using only the scaling laws, as this would force the aircraft to be confined to the fitted curves, artificially validating the laws. Instead, the data should be populated with designs made independently from the scaling laws or only utilize the laws for a reference.

- The larger pitch propeller vastly outperformed the lower pitch propeller in forward flight by being able to achieve higher cruising speeds at lower current draw. Additionally, the current draw in hover was nearly identical, so the only drawback of the larger pitch propeller is that the range of usable RPM is smaller since more thrust is generated at lower speeds, limiting the control actuation. Additionally, the higher-pitch propeller has a larger mass, increasing the inertia and decreasing the responsiveness, although this was not very noticeable in practice.

6.3 Future Work

The work conducted in this thesis represents the starting point for this project. Therefore, many avenues can be investigated in the future, depending on the interests of the researcher. The future work of this project should begin with an update to the scalability laws by adding any new or unused tailsitter data. Similarly, the scaling laws extracted from the conceptual design study should be updated with current technology, such as improved battery performance, and ensure that all components, like the propeller mounting hardware, are included.

A next step for this project is the implementation of the same controller with ArduPlane instead of ArduCopter [118]. The benefit of this procedure could be improved positional control due to ArduPlane's use of a total energy control system for altitude and pitch maintenance, and the L1 navigation controller. However, ArduPlane expects a tailsitter to use three flight modes, which is why ArduCopter was used for this project. The use of ArduPlane with a single flight mode for tailsitter control would be a significant challenge to overcome.

The prototype aircraft and the simulated model could both go through a series of iterations to improve their performance. For the prototype, a more detailed structural analysis would enable the optimization of the weight and strength of the structural components used, resulting in extra mass that can be given to the payload or battery. Aerodynamically, the wings could be improved through CFD, and utilizing wing characteristics typically seen on flying wings, such as dihedral for roll stability, wing twist and sweep for pitch stability, and wing tips to reduce vortex drag. The simulation can be improved by using flight data to compare the simulated results against and then adjusting simulation parameters to match the data. Specific test flights can be conducted to acquire more data, such as a horizontal

sliding test to estimate the body drag in hover.

The attitude controller implemented could also be improved in many ways. A self-learning fuzzy logic controller would ensure that the gains of the fuzzy controller are optimized for the control problem presented. Alternatively, more test flights or wind tunnel testing with the prototype could be conducted to identify a model for the aircraft based on parameters like the airspeed, sideslip speed, AoA, rotor speed, and battery level. The model could then be used alongside a model reference adaptive controller or a model predictive controller to improve performance by estimating how the aircraft will manoeuvre in the future, then selecting an optimal approach. The use of reinforcement learning or artificial intelligence to learn the behaviours of the aircraft based on the simulated and test data could also produce an improved controller.

References

[1] Ehsan Abbasi, Mohammad Mahjoob, and Reza Yazdanpanah Abdolmalaki. Controlling of quadrotor uav using a fuzzy system for tuning the pid gains in hovering mode. 09 2013. [Accessed: 16/11/2024]. URL: https://www.researchgate.net/publication/317385853_Controlling_of_Quadrotor_UAV_Using_a_Fuzzy_System_for_Tuning_the_PID_Gains_in_Hovering_Mode.

[2] Arduino. *What is Arduino?*, 2022. [Accessed: 03/10/2022]. URL: https://www.arduino.cc/.

[3] ArduPilot Dev Team. *Copter Attitude Control*, 2021. [Accessed: 23/09/2022]. URL: https://ardupilot.org/dev/docs/apmcopter-programming-attitude-control-2.html.

[4] ArduPilot Dev Team. *Mission Planner Home*, 2021. [Accessed: 03/10/2022]. URL: https://ardupilot.org/planner/.

[5] ArduPilot Dev Team. *Tailsitter Planes*, 2021. [Accessed: 21/09/2022]. URL: https://ardupilot.org/plane/docs/guide-tailsitter.html.

[6] ArduPilot Dev Team. *Copter Home*, 2022. [Accessed: 03/10/2022]. URL: https://ardupilot.org/copter/.

[7] ArduPilot Dev Team. *Adding Custom Attitude Controller to Copter*, 2024. [Accessed: 17/11/2024]. URL: https://ardupilot.org/dev/docs/copter-adding-custom-controller.html.

[8] ArduPilot Dev Team. *ArduPilot - Versitile, Trusted, Open*, 2024. [Accessed: 03/10/2024]. URL: hhttps://ardupilot.org/.

[9] ArduPilot Dev Team. *Building a QuadPlane*, 2024. [Accessed: 30/10/2024]. URL: https://ardupilot.org/dev/docs/apmcopter-programming-attitude-control-2.html.

[10] ArduPilot Dev Team. *Building the code*, 2024. [Accessed: 14/11/2024]. URL: https://ardupilot.org/dev/docs/building-the-code.html.

[11] ArduPilot Dev Team. *Copter Attitude Control*, 2024. [Accessed: 27/11/2024]. URL: https://ardupilot.org/dev/docs/apmcopter-programming-attitude-control-2.html.

[12] ArduPilot Dev Team. *Editors and IDEs*, 2024. [Accessed: 14/11/2024]. URL: https://ardupilot.org/dev/docs/code-editing-tools-and-ides.html.

[13] ArduPilot Dev Team. *First Time Setup*, 2024. [Accessed: 11/11/2024]. URL: https://ardupilot.org/copter/docs/initial-setup.html.

[14] ArduPilot Dev Team. *Setting up the Build Environment (Linux/Ubuntu)*, 2024. [Accessed: 14/11/2024]. URL: hhttps://ardupilot.org/dev/docs/building-setup-linux.html.

[15] ArduPilot Dev Team. *Using SITL with RealFlight*, 2024. [Accessed: 16/11/2024]. URL: https://ardupilot.org/dev/docs/sitl-with-realflight.html.

[16] Pooneh Aref, Mehdi Ghoreyshi, Adam Jirasek, Matthew J. Satchell, and Keith Bergeron. Computational study of propeller–wing aerodynamic interaction. *Aerospace*, 5(3), 2018. URL: https://www.mdpi.com/2226-4310/5/3/79, doi:10.3390/aerospace5030079.

[17] U.S. Army. *Quadrotor Biplane*, 2020. [Accessed: 30/10/2024]. URL: https://www.defense.gov/Multimedia/Photos/igphoto/2002828874/.

[18] National Balsa. *1/2 x 1 x 36 Conventional Leading Edge*, 2024. [Accessed: 11/11/2024]. URL: https://www.nationalbalsa.com/en-ca/products/12136le.

[19] National Balsa. *1/4 x 1 x 36 Balsa Wood Trailing Edge*, 2024. [Accessed: 11/11/2024]. URL: https://www.nationalbalsa.com/en-ca/products/14136te.

[20] Roman Bapst, Robin Ritz, Lorenz Meier, and Marc Pollefeys. Design and implementation of an unmanned tail-sitter. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1885–1890, 2015. doi:10.1109/IROS.2015.7353624.

[21] Samuel Barnhart, Barath Narayanan, and Sidaard Gunasekaran. Blown wing aerodynamic coefficient predictions using traditional machine learning and data science approaches. 01 2021. doi:10.2514/6.2021-0616.

[22] Randal W. Beard and Timothy W. McLain. *Small Unmanned Aircraft Theory and Practice*. Princeton University Press, Princeton, 2012. [Accessed: 05/01/2023]. URL: https://doi.org/10.1515/9781400840601, doi:doi:10.1515/9781400840601.

[23] Betaflight. *Are you ready to fly?*, 2022. [Accessed: 03/10/2022]. URL: https://betaflight.com/.

[24] John Brandt and Michael Selig. Propeller performance data at low reynolds numbers. 01 2011. URL: https://arc.aiaa.org/doi/abs/10.2514/6.2011-1255, doi:10.2514/6.2011-1255.

[25] John B. Brandt, Robert W. Deters, Gavin K. Ananda, Or D. Dantsker, and Michael S. Selig. *UIUC Propeller Database*, 2022. [Accessed: 08/10/2022]. URL: https://m-selig.ae.illinois.edu/props/propDB.html.

[26] Stanford University Brian Cantwell. *The NACA airfoil series*, 2013. [Accessed: 04/11/2024]. URL: hhttps://web.stanford.edu/~cantwell/AA200_Course_Material/The%20NACA%20airfoil%20series.pdf.

[27] Eitan Bulka and Meyer Nahon. Autonomous control of agile fixed-wing uavs performing aerobatic maneuvers. In *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 104–113, 2017. doi:10.1109/ICUAS.2017.7991437.

[28] Eitan Bulka and Meyer Nahon. A universal controller for unmanned aerial vehicles. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4171–4176, 2018. URL: https://ieeexplore-ieee-org.proxy.bib.uottawa.ca/document/8593878, doi:10.1109/IROS.2018.8593878.

[29] Jossué Cariño Escobar, Hernán Abaunza González, and Pedro Castillo Garcia. Quadrotor quaternion control. 06 2015. doi:10.1109/ICUAS.2015.7152367.

[30] Stephen J. Carlson and Christos Papachristos. The minihawk-vtol: Design, modeling, and experiments of a rapidly-prototyped tiltrotor uav. In *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 777–786, 2021. doi:10.1109/ICUAS51884.2021.9476731.

[31] Dylan Caverly, Meyer Nahon, and Jovan Nedić. *Aerodynamic characterization and control of a Tail-Sitter Aircraft taking off in a crosswind.* McGill University, 2020. [Accessed: 10/11/2024]. URL: https://escholarship.mcgill.ca/concern/theses/ks65hh693?locale=en.

[32] Romain Chiappinelli and Meyer A. Nahon. Modeling and control of a tailsitter uav. *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 400–409, 2018.

[33] Vishnu S. Chipade, Abhishek, Mangal Kothari, and Rushikesh R. Chaudhari. Systematic design methodology for development and flight testing of a variable pitch quadrotor biplane vtol uav for payload delivery. *Mechatronics*, 55:94–114, 2018. URL: https://www.sciencedirect.com/science/article/pii/S0957415818301351, doi:10.1016/j.mechatronics.2018.08.008.

[34] Cleanflight Team. *Open-Source flight controller software for modern flight boards*, 2017. [Accessed: 03/10/2022]. URL: http://cleanflight.com/.

[35] Nihal Dalwadi, Dipankar Deb, and Jagat Jyoti Rath. Biplane trajectory tracking using hybrid controller based on backstepping and integral terminal sliding mode control. *Drones*, 6(3), 2022. URL: https://www.mdpi.com/2504-446X/6/3/58, doi:10.3390/drones6030058.

[36] Gensace DE. *Industrial Drone Battery*, 2024. [Accessed: 12/11/2024]. URL: https://gensace.de/collections/industrial-drone.

[37] DeltaQuad. *Pro #CARGO*, 2024. [Accessed: 07/01/2025]. URL: https://www.deltaquad.com/products/pro-cargo/.

[38] Andre Deperrois. *XLFR5*, 2021. [Accessed: 05/01/2023]. URL: http://www.xflr5.tech/xflr5.htm.

[39] DragonPlate. *Streamlined Symmetrical Airfoil Carbon Fiber Tube - 1.63" ID x 0.74" ID x 48"*, 2024. [Accessed: 11/11/2024]. URL: https://dragonplate.com/streamlined-symmetrical-airfoil-carbon-fiber-tube-163-id-x-074-id-x-48.

[40] Mark Drela and Harold Youngren. *AVL Overview*, 2022. [Accessed: 05/01/2023]. URL: http://web.mit.edu/drela/Public/web/avl/.

[41] Dronecode Project, Inc. *QGroundControl*, 2019. [Accessed: 03/10/2022]. URL: http://qgroundcontrol.com/.

[42] Dronecode Project, Inc. *Software Overview*, 2021. [Accessed: 03/10/2022]. URL: https://px4.io/software/software-overview/.

[43] Clarence De Silva Fakhreddine O. Karray. *Soft Computing and Intelligent Systems Design Theory, Tools and Applications*. Pearson Education Limited, Essex, 2004.

[44] Arezki Fekik, Ahmad Azar, Hamida Mohamed Lamine, H. Denoun, Mohandsaidi Sabrina, A. Bousbaine, Nashwa Ahmad Kamal, Ibraheem Ibraheem, Amjad Humaidi, Ammar Al Mhdawi, and Alaa Khamis. *Modeling and Simulation of Quadcopter Using Self-tuning Fuzzy-PI Controller*, pages 231–251. 07 2023. doi:10.1007/978-3-031-26564-8_8.

[45] A. Flores, A. Montes de Oca, and G. Flores. A simple controller for the transition maneuver of a tail-sitter drone. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 4277–4281, 2018. doi:10.1109/CDC.2018.8619303.

[46] Flyduino. *The KISS Experience*, 2019. [Accessed: 03/10/2018]. URL: http://kiss.flyduino.net/.

[47] Emil Fresk and George Nikolakopoulos. Full quaternion based attitude control for a quadrotor. In *2013 European Control Conference (ECC)*, pages 3864–3869, 2013. doi:10.23919/ECC.2013.6669617.

[48] GensTattu. *Tattu G-Tech 6S 10000mAh 30C 22.2V Lipo Battery Pack with EC5 Plug for UAV Drone*, 2024. [Accessed: 11/11/2024]. URL: https://genstattu.com/ta-30c-10000-6s1p-ec5.html.

[49] R. Gill and R. D'andrea. An annular wing vtol uav: Flight dynamics and control. *Drones*, 4(2):1–34, 2020. URL: https://www.mdpi.com/2504-446X/4/2/14, doi:10.3390/drones4020014.

[50] Rajan Gill and Raffaello D'Andrea. Computationally efficient force and moment models for propellers in uav forward flight applications. *Drones*, 3(4), 2019. URL: https://www.mdpi.com/2504-446X/3/4/77, doi:10.3390/drones3040077.

[51] Bharath Govindarajan and Ananth Sridharan. Conceptual sizing of vertical lift package delivery platforms. *Journal of Aircraft*, 57(6):1170–1188, 2020. arXiv:https://doi.org/10.2514/1.C035805, doi:10.2514/1.C035805.

[52] Bharath Govindarajan, Ananth Sridharan, and Michael Avera. Integration of physics based weight models into rotorcraft design sizing. 09 2017.

[53] Bharath Govindarajan, Ananth Sridharan, and Inderjit Chopra. A scalability study of the multirotor biplane tailsitter using conceptual sizing. *Journal of the American Helicopter Society*, 65, 01 2019. doi:10.4050/JAHS.65.012009.

[54] Nickolas Grady, Michael Frye, and Chunjiang Qian. *The Instrumentation and Flight Testing of a Rotorcraft Vehicle for Undergraduate Flight Control Research*. AIAA, 2012. URL: https://arc.aiaa.org/doi/abs/10.2514/6.2006-6739, arXiv:https://arc.aiaa.org/doi/pdf/10.2514/6.2006-6739, doi:10.2514/6.2006-6739.

[55] Jay Gundlach. *Designing unmanned aircraft systems: a comprehensive approach*. AIAA education series, Virginia, 2012.

[56] Nancy Hall. *Propeller Analysis*, 2021. [Accessed: 08/11/2024]. URL: https://www.grc.nasa.gov/WWW/k-12/airplane/propanl.html.

[57] Nancy Hall. *Propeller Thrust*, 2021. [Accessed: 08/11/2024]. URL: https://www.grc.nasa.gov/WWW/k-12/airplane/propth.html#:~:text=A%20spinning%20propeller%20sets%20up%20a%20pressure%20lower,because%20the%20propeller%20does%20work%20on%20the%20airflow.

[58] M. Hochstenbach, C. Notteboom, B. Theys, and J. De Schutter. Design and control of an unmanned aerial vehicle for autonomous parcel delivery with transition from vertical take-off to forward flight - vertikul, a quadcopter tailsitter. *International Journal of Micro Air Vehicles*, 7(4):395–405, 2015. doi:10.1260/1756-8293.7.4.395.

[59] Haomiao Huang, G.M. Hoffmann, Steven L. Waslander, and Claire J. Tomlin. Aerodynamics and control of autonomous quadrotor helicopters in aggressive maneuvering. *2009 IEEE International Conference on Robotics and Automation*, pages 3277–3282, 2009.

[60] Autodesk Inc. *Autodesk 3ds Max: Create massive worlds and high-quality designs*, 2024. [Accessed: 13/11/2024]. URL: https://www.autodesk.com/products/3ds-max/overview?term=1-YEAR&tab=subscription.

[61] Minchi Kuang, Jihong Zhu, Wufan Wang, and Yunfei Tang. Flight controller design and demonstration of a thrust-vectored tailsitter. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5169–5174, 2017. doi:10.1109/ICRA.2017.7989605.

[62] Daisuke Kubo, Koji Muraoka, and Noriaki Okada. High angle of attack flight characteristics of a wing-in-propeller-slipstream aircraft. *International Council of the Aeronatical Sciences*, 2010. [Accessed: 10/11/2024]. URL: https://www.icas.org/ICAS_ARCHIVE/ICAS2010/PAPERS/237.PDF.

[63] Hosun Lee, Jayden Dongwoo Lee, and Hyochoong Bang. Aerodynamic model idendication of a vtol tailsitter uav using sparse identification of nonlinear dynamics. *International Council of the Aeronatical Sciences*, 2024. [Accessed: 10/11/2024]. URL: https://www.icas.org/ICAS_ARCHIVE/ICAS2024/data/papers/ICAS2024_0653_paper.pdf.

[64] Boyang Li, Weifeng Zhou, Jingxuan Sun, Chih-Yung Wen, and Chih-Keng Chen. Development of model predictive controller for a tail-sitter vtol uav in hover flight. *Sensors*, 18(9), 2018. URL: https://www.mdpi.com/1424-8220/18/9/2859, doi:10.3390/s18092859.

[65] Zhaoying Li, Wenjie Zhou, and Hao Liu. Robust controller design for a tail-sitter uav in flight mode transitions. In *2018 IEEE 14th International Conference on Control and Automation (ICCA)*, pages 763–768, 2018. URL: https://ieeexplore.ieee.org/document/8444168, doi:10.1109/ICCA.2018.8444168.

[66] Deyuan Liu, Hao Liu, Jiansong Zhang, and Frank L Lewis. Adaptive attitude controller design for tail-sitter unmanned aerial vehicles. *Journal of Vibration and Control*, 27(1-2):185–196, 2021. doi:10.1177/1077546320925350.

[67] Horizon Hobby LLC. and Hangar 9. *UltraCote Lite, Transparent Clear*, 2020. [Accessed: 11/11/2024]. URL: https://www.horizonhobby.com/product/ultracote-lite-transparent-clear/HANU964.html.

[68] Canonical Ltd. *Canonical Ubuntu*, 2024. [Accessed: 16/11/2024]. URL: https://ubuntu.com/.

[69] CubePilot Pty. Ltd. *Here 3 Manual*, 2022. [Accessed: 04/11/2024]. URL: https://docs.cubepilot.org/user-guides/here-3/here-3-manual.

[70] CubePilot Pty. Ltd. *The Cube Orange Standard Set*, 2024. [Accessed: 04/11/2024]. URL: https://www.cubepilot.org/#/cube/features.

[71] CubePilot Pty. Ltd. *Kore Carrier Board*, 2024. [Accessed: 04/11/2024]. URL: https://docs.cubepilot.org/user-guides/carrier-boards/kore-carrier-board.

[72] F. H. Lutze. *Lecture aoe 3104 vehicle performance*, 2014. [Accessed: 06/10/2022]. URL: http://www.dept.aoe.vt.edu/~lutze/AOE3104/thrustmodels.pdf.

[73] X. Lyu, H. Gu, J. Zhou, Z. Li, S. Shen, and F. Zhang. Simulation and flight experiments of a quadrotor tail-sitter vertical take-off and landing unmanned aerial vehicle with wide flight envelope. *International Journal of Micro Air Vehicles*, 10(4):303–317, 2018. URL: https://journals.sagepub.com/doi/10.1177/1756829318813633, doi:10.1177/1756829318813633.

[74] X. Lyu, J. Zhou, H. Gu, Z. Li, S. Shen, and F. Zhang. Disturbance observer based hovering control of quadrotor tail-sitter vtol uavs using h_∞bran synthesis. *IEEE Robotics and Automation Letters*, 3(4):2910–2917, 2018. URL: https://ieeexplore.ieee.org/document/8385166, doi:10.1109/LRA.2018.2847405.

[75] Ximin Lyu, Haowei Gu, Jinni Zhou, Zexiang Li, Shaojie Shen, and Fu Zhang. A hierarchical control approach for a quadrotor tail-sitter vtol uav and experimental verification. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5135–5141, 2017. URL: https://ieeexplore.ieee.org/document/8206400, doi:10.1109/IROS.2017.8206400.

[76] Ziqing Ma, Ewoud J.J. Smeur, and Guido C.H.E. de Croon. Wind tunnel tests of a wing at all angles of attack. *International Journal of Micro Air Vehicles*, 14:17568293221110931, 2022. arXiv:https://doi.org/10.1177/17568293221110931, doi:10.1177/17568293221110931.

[77] K. McIntosh, J.-P. Reddinger, D. Zhao, and S. Mishra. Optimal trajectory generation for transitioning quadrotor biplane tailsitter using differential flatness. 2021. URL: https://move.rpi.edu/publications/conference-papers/optimal-trajectory-generation-transitioning-quadrotor-biplane.

[78] McMaster-Carr. *Rigid Lightweight Carbon Fiber Round Tube*, 2024. [Accessed: 06/11/2024]. URL: https://www.mcmaster.com/8069N16/.

[79] McMaster-Carr. *Ultra-Strength Lightweight Carbon Fiber Tube*, 2024. [Accessed: 06/11/2024]. URL: https://www.mcmaster.com/5287T72/.

[80] Microsoft. *Visual Studio Code*, 2024. [Accessed: 16/11/2024]. URL: https://code.visualstudio.com/.

[81] Microsoft. *Windows Subsystem for Linux Documentation*, 2024. [Accessed: 16/11/2024]. URL: https://learn.microsoft.com/en-us/windows/wsl/.

[82] E.E.L. Mitchell and A.E. Rogers. Quaternion parameters in the simulation of a spinning rigid body. *SIMULATION*, 4(6):390–396, 1965. arXiv:https://doi.org/10.1177/003754976500400610, doi:10.1177/003754976500400610.

[83] Altitude Hold Mode. *Guided Mode*, 2024. [Accessed: 07/01/2025]. URL: https://ardupilot.org/copter/docs/altholdmode.html.

[84] Onshape. *The Perfect CAD & PDM Solution for Education*, 2024. [Accessed: 30/10/2024]. URL: https://www.onshape.com/en/74/.

[85] Sanghyuk Park, John Deyst, and Jonathan How. *A New Nonlinear Guidance Logic for Trajectory Tracking*. URL: https://arc.aiaa.org/doi/abs/10.2514/6.2004-4900, arXiv:https://arc.aiaa.org/doi/pdf/10.2514/6.2004-4900, doi:10.2514/6.2004-4900.

[86] Hardik Parwana and Mangal Kothari. Quaternions and attitude representation. 08 2017. doi:10.48550/arXiv.1708.08680.

[87] Brandyn Phillips, Vikram Hrishikeshavan, Derrick Yeo, and Inderjit Chopra. Flight performance of a package delivery quadrotor biplane. 01 2017.

[88] Brandyn Phillips, Derek Safieh, Vikram Hrishikeshavan, and Inderjit Chopra. Performance and control of variable pitch proprotors for multi-scale quadrotor biplane tail-sitters (qbits). 05 2019.

[89] Chung-Kiak Poh, Chung-How Poh, Mei-Ling Yeh, and Tien-Yin Chou. Conceptual design of a tropical cyclone uav based on the ar-6 endeavor aircraft, 2014. URL: https://arxiv.org/abs/1407.8454, arXiv:1407.8454.

[90] APC Propellers. *15x10E*, 2024. [Accessed: 11/11/2024]. URL: https://www.apcprop.com/product/15x10e/.

[91] APC Propellers. *Performance Data*, 2024. [Accessed: 08/11/2024]. URL: htthttps://www.apcprop.com/technical-information/performance-data/.

[92] Zheng Qiao, Dong Wang, Jiahui Xu, Xinbiao Pei, Wei Su, Dong Wang, and Yue Bai. A comprehensive design and experiment of a biplane quadrotor tail-sitter uav. *Drones*, 7(5), 2023. URL: https://www.mdpi.com/2504-446X/7/5/292, doi:10.3390/drones7050292.

[93] N. Raj, A. Simha, M. Kothari, Abhishek, and R.N. Banavar. Iterative learning based feedforward control for transition of a biplane-quadrotor tailsitter uas. pages 321–327, 2020. URL: https://ieeexplore.ieee.org/document/9196671, doi:10.1109/ICRA40945.2020.9196671.

[94] Daniel P. Raymer. *Aircraft design: a conceptual approach*. AIAA education series, California, 2018.

[95] RealFlight. *The #1 RC Flight Simulator in the World!*, 2024. [Accessed: 04/11/2024]. URL: https://www.realflight.com/.

[96] J.-P. Reddinger. Performance and controls of a scalable quadrotor biplane tailsitter. 2019. doi:10.2514/6.2019-1287.

[97] Jean-Paul Reddinger, Kristoff McIntosh, Di Zhao, and Sandipan Mishra. Modeling and trajectory control of a transitioning quadrotor biplane tailsitter. 2019. URL: https://move.rpi.edu/publications/conference-papers/modeling-and-trajectory-control-transitioning-quadrotor-biplane.

[98] Jan Roskam. *Airplane Design Part I : Preliminary Sizing of Airplanes*. Design, Analysis and Research Corporation, Kansas, 2018.

[99] Kapil Dev Sharma, Mohammad Ayyub, Sumit Saroha, and A. Faras. Advanced controllers using fuzzy logic controller (flc) for performance improvement. URL: https://api.semanticscholar.org/CorpusID:40605021.

[100] Benjamin M. Simmons. Efficient variable-pitch propeller aerodynamic model development for vectored-thrust evtol aircraft. In *AIAA AVIATION 2022 Forum*, 2022. URL: https://arc.aiaa.org/doi/abs/10.2514/6.2022-3817, arXiv:https://arc.aiaa.org/doi/pdf/10.2514/6.2022-3817, doi:10.2514/6.2022-3817.

[101] Pranay Sinha, Piotr Esden-Tempski, Christopher A. Forrette, Jeffrey K. Gibboney, and Gregory M. Horn. Versatile, modular, extensible vtol aerial platform with autonomous flight mode transitions. In *2012 IEEE Aerospace Conference*, pages 1–17, 2012. doi:10.1109/AERO.2012.6187313.

[102] S.Q. Sohail, F. Akram, N. Hussain, and A. Shahzad. Design and transition of a quad rotor tail-sitter vtol uav with experimental verification. pages 244–251, 2021. URL: https://ieeexplore.ieee.org/document/9393187, doi:10.1109/IBCAST51254.2021.9393187.

[103] Z. S. Spakovszky. *11.7 Performance of Propellers*, 2007. [Accessed: 08/11/2024]. URL: https://web.mit.edu/16.unified/www/FALL/thermodynamics/notes/node86.html.

[104] Spektrum. *SRXL2 DSMX Remote Serial Telemetry Receiver*, 2024. [Accessed: 04/11/2024]. URL: https://www.spektrumrc.com/product/srxl2-dsmx-remote-serial-telemetry-receiver/SPM4651T.html.

[105] Eric Spitznagel. *Medical Delivery by Drone Is Happening. How Good Is It?*, 2023. [Accessed: 30/10/2024]. URL: https://www.webmd.com/first-aid/news/20231221/medical-delivery-drone-what-to-know.

[106] Ananth Sridharan, Bharath Govindarajan, V. T. Nagaraj, and Inderjit Chopra. Design considerations of a lift-offset single main rotor compound helicopter. 01 2016.

[107] Anshuman Srinivasan. Modeling, design and control of a 6 d-o-f quadcopter fleet with platooning control. 01 2021. Master's Thesis.

[108] Amazon Staff. *Get medications faster with drone delivery from Amazon Pharmacy*, 2023. [Accessed: 30/10/2024]. URL: https://www.aboutamazon.com/news/retail/amazon-pharmacy-amazon-air-prescription-drone-delivery.

[109] Swati Swarnkar, Hardik Parwana, Mangal Kothari, and Abhishek Abhishek. *Development of Flight Dynamics Model and Control of Biplane-Quadrotor UAV*. URL: https://arc.aiaa.org/doi/abs/10.2514/6.2018-1851, arXiv:https://arc.aiaa.org/doi/pdf/10.2514/6.2018-1851, doi:10.2514/6.2018-1851.

[110] Peter J. Swatton. John Wiley & Sons, Ltd, 2010. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470710944.fmatter, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470710944.fmatter, doi:10.1002/9780470710944.fmatter.

[111] T-Motor. *AIR 40A 2-6S Multi-Rotor UAV Drone ESC*, 2024. [Accessed: 30/10/2024]. URL: https://store.tmotor.com/product/air-40a-6s-esc.html.

[112] T-Motor. *MN4014 Navigator Type UAV Multi-Motor KV400*, 2024. [Accessed: 30/10/2024]. URL: https://store.tmotor.com/product/mn4014-kv400-motor-navigator-type.html.

[113] T-Motor. *P15*5 Prop-2PCS/PAIR*, 2024. [Accessed: 30/10/2024]. URL: https://store.tmotor.com/product/polish-carbon-fiber-15x5-prop.html.

[114] T-Motor. *T-Motor Store*, 2024. [Accessed: 12/11/2024]. URL: https://store.tmotor.com/.

[115] Zaid Tahir, Mohsin Jamil, Saad Liaqat, Lubva Mubarak, Waleed Tahir, and Syed Gilani. Design and development of optimal control system for quad copter uav. *Indian Journal of Science and Technology*, 9:1–11, 07 2016. doi:10.17485/ijst/2016/v9i25/96611.

[116] Zaid Tahir, Mohsin Jamil, Saad Liaqat, Lubva Mubarak, Waleed Tahir, and Syed Gilani. State space system modeling of a quad copter uav. *Indian Journal of Science and Technology*, 9, 07 2016. doi:10.48550/arXiv.1908.07401.

[117] Ezra Tal and Sertac Karaman. Global incremental flight control for agile maneuvering of a tailsitter flying wing. *Journal of Guidance, Control, and Dynamics*, 45(12):2332–2349, 2022. doi:10.2514/1.G006645.

[118] ArduPilot Dev Team. *ArduPilot Plane*, 2024. [Accessed: 04/11/2024]. URL: https://ardupilot.org/plane/.

[119] ArduPilot Dev Team. *Connect ESCs and Motors*, 2024. [Accessed: 13/11/2024]. URL: https://ardupilot.org/copter/docs/connect-escs-and-motors.html.

[120] ArduPilot Dev Team. *Guided Mode*, 2024. [Accessed: 07/01/2025]. URL: https://ardupilot.org/copter/docs/ac2_guidedmode.html.

[121] ArduPilot Dev Team. *Loiter Mode*, 2024. [Accessed: 07/01/2025]. URL: https://ardupilot.org/copter/docs/loiter-mode.html.

[122] ArduPilot Dev Team. *SiK Telemetry Radio*, 2024. [Accessed: 04/11/2024]. URL: https://ardupilot.org/copter/docs/common-sik-telemetry-radio.html.

[123] ArduPilot Dev Team. *SITL Simulator (Software in the Loop)*, 2024. [Accessed: 13/11/2024]. URL: https://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html.

[124] Ts. Tengis and A. Batmunkh. State feedback control simulation of quadcopter model. In *2016 11th International Forum on Strategic Technology (IFOST)*, pages 553–557, 2016. doi:10.1109/IFOST.2016.7884178.

[125] Inc. The MathWorks. *Curve Fitting Toolbox*, 2024. [Accessed: 16/12/2024]. URL: hhttps://www.mathworks.com/products/curvefitting.html.

[126] Inc. The MathWorks. *Embedded Coder*, 2024. [Accessed: 16/11/2024]. URL: https://www.mathworks.com/products/embedded-coder.html.

[127] Inc. The MathWorks. *Fuzzy Logic Designer*, 2024. [Accessed: 03/12/2024]. URL: https://www.mathworks.com/help/fuzzy/fuzzylogicdesigner-app.html.

[128] Inc. The MathWorks. *gensurf*, 2024. [Accessed: 16/11/2024]. URL: https://www.mathworks.com/products/embedded-coder.html.

[129] The Engineering ToolBox. *Air - Density, Specific Weight and Thermal Expansion Coefficient vs. Temperature and Pressure*, 2003. [Accessed: 04/11/2024]. URL: https://www.engineeringtoolbox.com/air-density-specific-weight-d_600.html?vA=10&units=C#.

[130] Airfoil Tools. *NACA 2412 (naca2412-il)*, 2024. [Accessed: 04/11/2024]. URL: http://airfoiltools.com/airfoil/details?airfoil=naca2412-il.

[131] Airfoil Tools. *Reynolds number calculator*, 2024. [Accessed: 04/11/2024]. URL: http://airfoiltools.com/calculator/reynoldsnumber?MReNumForm%5Bvel%5D=17.3&MReNumForm%5Bchord%5D=.14&MReNumForm%5Bkvisc%5D=1.5111E-5&yt0=Calculate.

[132] Uline. *Insulated Foam Shipping Kit - 6 x 4 1/2 x 3″*, 2024. [Accessed: 30/10/2024]. URL: https://www.uline.ca/Product/Detail/S-7887/Insulated-Shippers-and-Supplies/Insulated-Foam-Shipping-Kit-6-x-4-1-2-x-3.

[133] Bambu Lab US. *Bambu Lab X1C 3D Printer*, 2024. [Accessed: 11/11/2024]. URL: https://us.store.bambulab.com/products/x1-carbon.

[134] Bambu Lab US. *BPLA-CF*, 2024. [Accessed: 11/11/2024]. URL: hhttps://us.store.bambulab.com/collections/fiber-reinforced/products/pla-cf.

[135] Sebastian Verling, B. Weibel, M. Boosfeld, Kostas Alexis, Michael Burri, and Roland Siegwart. Full attitude control of a vtol tailsitter uav. pages 3006–3012, 05 2016. doi:10.1109/ICRA.2016.7487466.

[136] Ya Wang, Lyu Ximin, Haowei Gu, Shaojie Shen, Zexiang Li, and Fu Zhang. Design, implementation and verification of a quadrotor tail-sitter vtol uav. 06 2017. URL: https://ieeexplore.ieee.org/document/7991419, doi:10.1109/ICUAS.2017.7991419.

[137] Wingcopter. *The Wingcopter 198*, 2024. [Accessed: 07/01/2025]. URL: https://wingcopter.com/wingcopter-198.

[138] Wingtra AG. *Tailsitters vs. quadplanes – why a VTOL tailsitter is the best surveying drone for your mapping missions*, 2018. [Accessed: 28/12/2022]. URL: https://wingtra.com/tailsitters-vs-quadplanes-why-a-vtol-tailsitter-is-the-best-surveying-drone-for-your-mapping-missions/.

[139] Wingtra AG. *The WingtraOne VTOL Drone for Mapping and Surveying*, 2018. [Accessed: 21/09/2022]. URL: https://wingtra.com/why-wingtra/vtol-drone/.

[140] Hongbo Xin, Yujie Wang, Xianzhong Gao, Qingyang Chen, Bingjie Zhu, Jianfeng Wang, and Zhongxi Hou. Modeling and control of a quadrotor tail-sitter unmanned aerial vehicles. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 236(3):443–457, 2022. doi:10.1177/09596518211050466.

[141] W. Xu, H. Gu, Y. Qing, J. Lin, and F. Zhang. Full attitude control of an efficient quadrotor tail-sitter vtol uav with flexible modes. pages 542–550, 2019. URL: https://arxiv.org/abs/1903.06393, doi:10.1109/ICUAS.2019.8797947.

[142] W. Xu and F. Zhang. Learning pugachev's cobra maneuver for tail-sitter uavs using acceleration model. *IEEE Robotics and Automation Letters*, 5(2):3452–3459, 2020. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9013021, doi:10.1109/LRA.2020.2976323.

[143] Trevor Phair Zekai Hong. *Blown wing proprotor-driven powered-lift aerodyne - WO2024150155A1*, 2024. [Accessed: 30/10/2024]. URL: https://patents.google.com/patent/WO2024150155A1/en.

[144] Dizhou Zhang, Zili Chen, and Leiping Xi. Adaptive dual fuzzy pid control method for longitudinal attitude control of tail-sitter uav. In *2016 22nd International Conference on Automation and Computing (ICAC)*, pages 378–382, 2016. doi:10.1109/IConAC.2016.7604949.

[145] F. Zhang, X. Lyu, Y. Wang, H. Gu, and Z. Li. Modeling and flight control simulation of a quad rotor tail-sitter vtol uav. 2017. URL: https://www.researchgate.net/publication/312109050_Modeling_and_Flight_Control_Simulation_of_a_Quadrotor_Tailsitter_VTOL_UAV, doi:10.2514/6.2017-1561.

[146] Sheng Zhao, Wenjie Dong, and Jay A. Farrell. Quaternion-based trajectory tracking control of vtol-uavs using command filtered backstepping. In *2013 American Control Conference*, pages 1018–1023, 2013. doi:10.1109/ACC.2013.6579970.

[147] J. Zhou, X. Lyu, X. Cai, Z. Li, S. Shen, and F. Zhang. Frequency domain model identification and loop-shaping controller design for quadrotor tail-sitter vtol uavs. pages 1142–1149, 2018. URL: https://ieeexplore.ieee.org/document/8453475, doi:10.1109/ICUAS.2018.8453475.

[148] Jinni Zhou, Ximin Lyu, Zexiang Li, Shaojie Shen, and Fu Zhang. A unified control method for quadrotor tail-sitter uavs in all flight modes: Hover, transition, and level flight. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4835–4841, 2017. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8206359.

[149] Zipline. *Zipline for Healthcare*, 2024. [Accessed: 30/10/2024]. URL: https://www.flyzipline.com/solutions/healthcare.

[150] George Zogopoulos. *Creating a RealFlight model with Blender*, 2024. [Accessed: 13/11/2024]. URL: https://discuss.ardupilot.org/t/creating-a-realflight-model-with-blender/116967.

Appendices

Appendix A

Detailed Prototype Design Notes

The design of the prototype tailsitter involved many decisions and analyses that were beyond the main focus of the thesis, but are further explained in this appendix. First, the airfoil selection is shown, followed by an analysis of the structural strength of the wings and diagonal rods of the vehicle. Many parts that were designed using CAD are then presented and explained.

A.1 Airfoil Selection

The airfoil selection is an important part in the design process for any aircraft, but with tailsitters, the symmetry of the airfoil is more significant. A non-symmetric airfoil typically results in more efficient forward flight, creating more lift at lower angles of attack. However, when a tailsitter utilizes blown wings, the vehicle will drift due to the forces created by the airflow over the non-symmetric airfoil. The flight controller then has to compensate for this drift, resulting in a tilt to the vehicle in hover to maintain position, resulting in less efficient hover. In the case of the aircraft designed for this thesis, it was decided to use a non-symmetric airfoil to achieve more efficient forward flight since this was a much larger portion of the mission profile.

When selecting an airfoil, it is also crucial to know the Reynolds number the wings will experience, as airfoil performance data is commonly categorized based on this value, such as the data presented by Airfoil Tools [130] for the chosen NACA 2412 airfoil. The Reynolds number is based on the speed of the aircraft flowing through the air, v, the characteristic chord length of the airfoil, c, and the kinematic viscosity of the air, ν. At $-10\,^{\circ}\mathrm{C}$, the kinematic viscosity of air is $1.2462 \times 10^{-5}\,\mathrm{m}^2/\mathrm{s}$, and at $20\,^{\circ}\mathrm{C}$, the kinematic viscosity of air is $1.5111 \times 10^{-5}\,\mathrm{m}^2/\mathrm{s}$ [131]. Therefore, the Reynolds number for this application when

cruising at 17.6 m/s will range between

$$Re = \frac{vc}{\nu}$$
$$Re = \frac{(17.6\,\mathrm{m/s})(0.14\,\mathrm{m})}{1.2462 \times 10^{-5}\,\mathrm{m^2/s}}$$
$$Re = 197721$$

and

$$Re = \frac{(17.6\,\mathrm{m/s})(0.14\,\mathrm{m})}{1.5111 \times 10^{-5}\,\mathrm{m^2/s}}$$
$$Re = 163060$$

Therefore, the data at 200000 Reynolds number will be used to characterize the airfoil since this is the closest to the calculated values. Section 2.4.2.1 listed the reasons why the NACA 2412 airfoil was selected, citing simplicity, compatibility with OnShape and the RealFlight simulation, abundance of data, good stall characteristics, and a small camber for efficient forward flight without creating excessive lateral force in hover. The performance of the NACA 2412 airfoil at 200000 Reynolds number is shown in Figure A.1.

A.2 Structural Strength and Deformation

For any multirotor vehicle, it is critical to have rigid mounts for the motors; otherwise, the aircraft will be very difficult to control, leading to vibrations and oscillations. To confirm that the parts selected are rigid enough, a simple beam deformation analysis was conducted on the carbon fibre beams used on the prototype aircraft. For the diagonal tubes that connect the motors to the centre hub, the worst-case scenario is when the motors are a full power, but the aircraft is not moving, so the centre is fixed. In this case, the diagonal tube is considered a cantilever beam, fixed at one end with the propeller force acting 350 mm from the mounting point, shown schematically in Figure A.2. The modulus of elasticity for the carbon fibre tube, E, was set to 155 GPa, obtained from the manufacturer [78]. The moment of inertia for the tube is calculated as follows.

$$I_x = \pi \left(r_{OD}^4 - r_{ID}^4\right)/4$$
$$I_x = \pi \left((19.050\,\mathrm{mm}/2)^4 - (15.875\,\mathrm{mm}/2)^4\right)/4$$
$$I_x = 3347\,\mathrm{mm}^4$$

Then the displacement of the beam can be calculated at the point a for the rotor location and at the point b where the diagonal connecting rod attaches to the top or bottom wing.

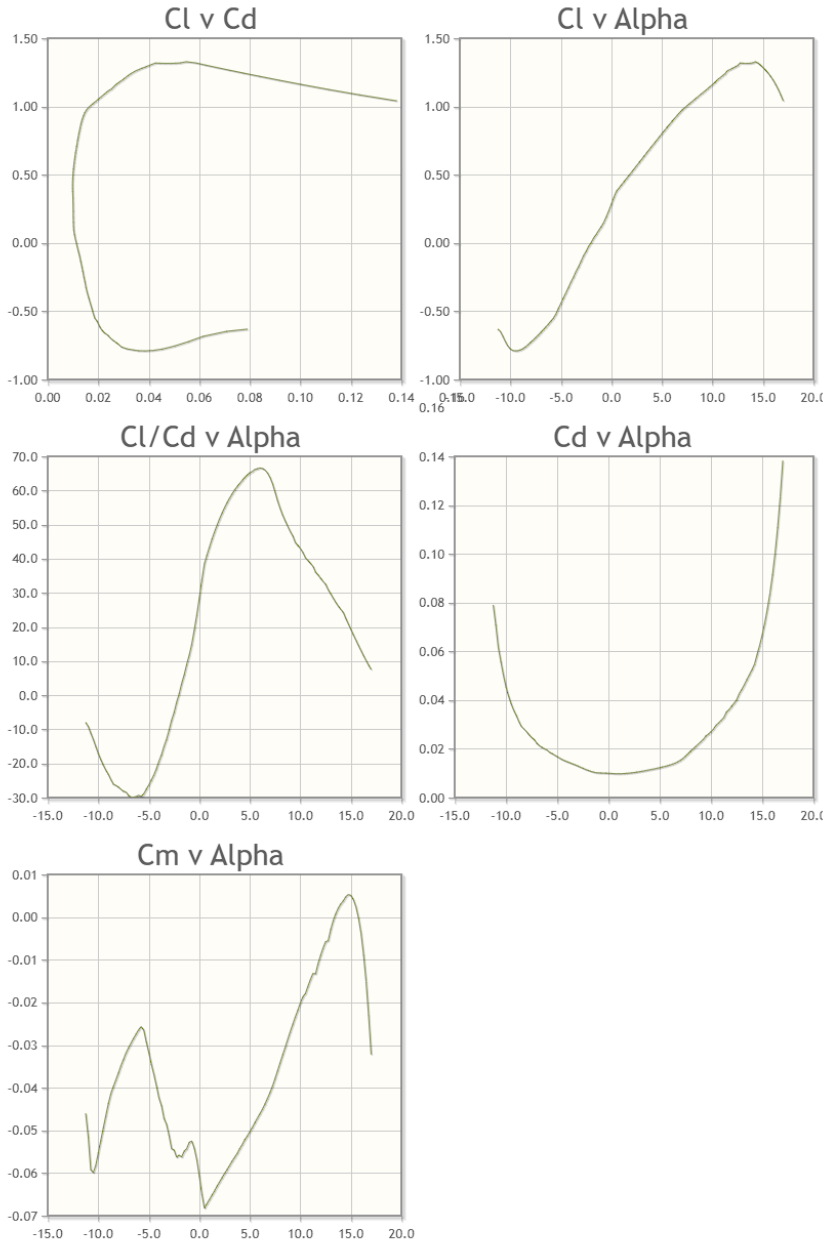


Figure A.1: Aerodynamic performance data of NACA 2412 airfoil at 200000 Reynolds Number [130].

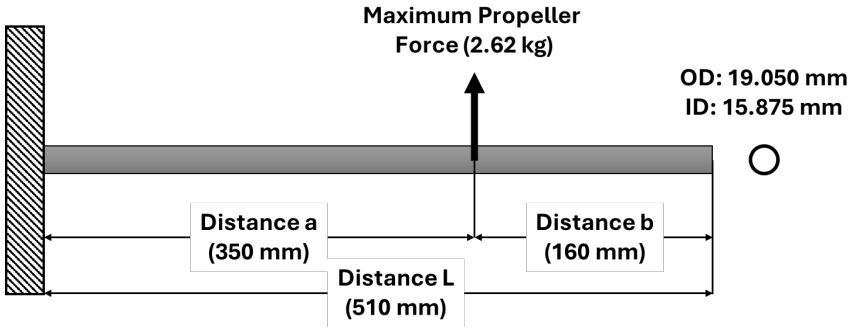


Figure A.2: Schematic of beam deflection calculation for carbon fibre diagonal rods connecting motors to centre hub or aircraft.

First, the displacement at the rotor with an applied point force, P, of 2.62 kg is

$$\begin{aligned}
\delta_{a,max} &= \frac{Pa^3}{3EI_x} \\
\delta_{a,max} &= \frac{(2.62\,\text{kg})(9.81\,\text{m/s}^2)(0.35\,\text{m})^3}{3(155\,\text{GPa})(3347\,\text{mm}^4)} \\
\delta_{a,max} &= 0.708\,\text{mm}
\end{aligned}$$

Similarly, the displacement at the wing mount at location b is

$$\begin{aligned}
\delta_{b,max} &= \frac{Pa^2(3L-a)}{6EI_x} \\
\delta_{b,max} &= \frac{(2.62\,\text{kg})(9.81\,\text{m/s}^2)(0.35\,\text{m})^2(3(0.51\,\text{m})-0.35\,\text{m})}{6(155\,\text{GPa})(3347\,\text{mm}^4)} \\
\delta_{b,max} &= 1.19\,\text{mm}
\end{aligned}$$

The tube's inner diameter, outer diameter, and material were selected iteratively until the resulting displacement reached a small enough value to prevent warping and misalignment of the motors at a reasonable price. The calculations shown represent the decided configuration.

A similar process was also conducted for the wing spars, which are made from smaller carbon fibre tubes with a modulus of elasticity of 87.6 GPa [79]. For the spars, there are two cases: the first is supported nearly rigidly on both ends with the diagonal tubes, making up approximately 26 percent of the total lift per wing; the second is a cantilever beam supported on one end and makes up approximately 12 percent of the total lift per wing. Both of those cases are shown in Figure A.3, along with the corresponding beam

dimensions. The wing spar has the following moment of inertia.

$$
\begin{aligned}
I_x &= \pi \left(r_{OD}^4 - r_{ID}^4\right)/4 \\
I_x &= \pi \left((8.128\,\text{mm}/2)^4 - (6.350\,\text{mm}/2)^4\right)/4 \\
I_x &= 134.4\,\text{mm}^4
\end{aligned}
$$

For the first case, the maximum beam displacement is calculated to be

$$
\begin{aligned}
\delta_{max} &= \frac{5wa^4}{384EI_x} \\
\delta_{max} &= \frac{5Pa^3}{384EI_x} \\
\delta_{max} &= \frac{5(1.36\,\text{kg})(9.81\,\text{m/s}^2)(0.88\,\text{m})^3}{384(87.6\,\text{GPa})(134.4\,\text{mm}^4)} \\
\delta_{max} &= 10.1\,\text{mm}
\end{aligned}
$$

For the second case, the maximum beam displacement is

$$
\begin{aligned}
\delta_{max} &= \frac{Pb^3}{8EI_x} \\
\delta_{max} &= \frac{(0.63\,\text{kg})(9.81\,\text{m/s}^2)(0.88\,\text{m})^3}{8(87.6\,\text{GPa})(134.4\,\text{mm}^4)} \\
\delta_{max} &= 3.9\,\text{mm}
\end{aligned}
$$

Both of these cases have about 1 percent deflection versus the beam length. Since there are no parts that connect to the wings, these are acceptable results, especially considering how light the spars are at only 28 g for the long tubes and 13 g for the short tubes. Additionally, when the full wing is constructed, there will be a leading edge, trailing edge, and ribs throughout that connect these to the spar, increasing the moment of inertia and decreasing the deflection further. Another benefit of these spars is that they are thin enough to fit inside the chosen airfoil, even when fully wrapped by a wing rib.

A.3 3D Printed Parts

The 3D printed parts were used to connect the structural elements. Each part was printed using carbon fibre-reinforced PLA with an X1 Carbon 3D Printer from Bambu Labs. An isometric view of the CAD used to design the prototype aircraft is shown in Figure A.4(a), which includes the 3D printed parts, carbon fibre rods, spars, ribs, and plates, balsa leading and trailing edges, motors, and the landing gear. A motor mount positioned on the diagonal spar, with a carbon fibre plate to hold the motor, and the start of the landing gear is shown in Figure A.4(b). The top right mount that attaches the wing to the diagonal carbon fibre

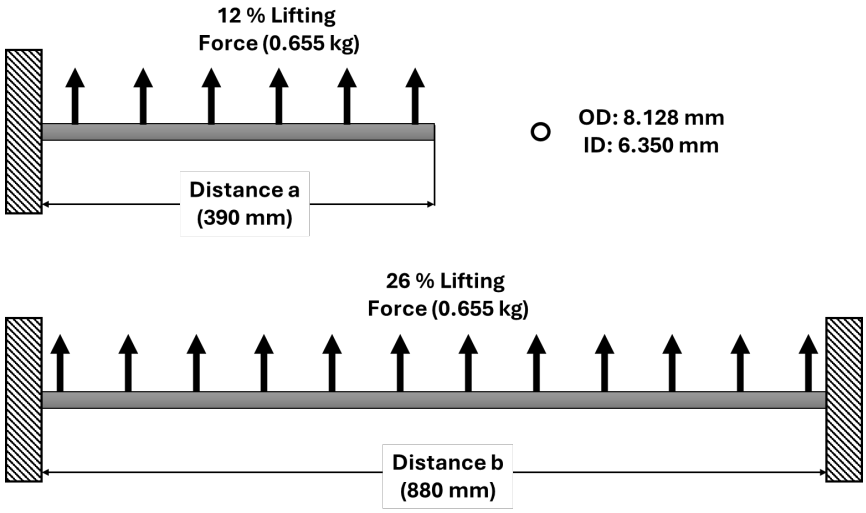
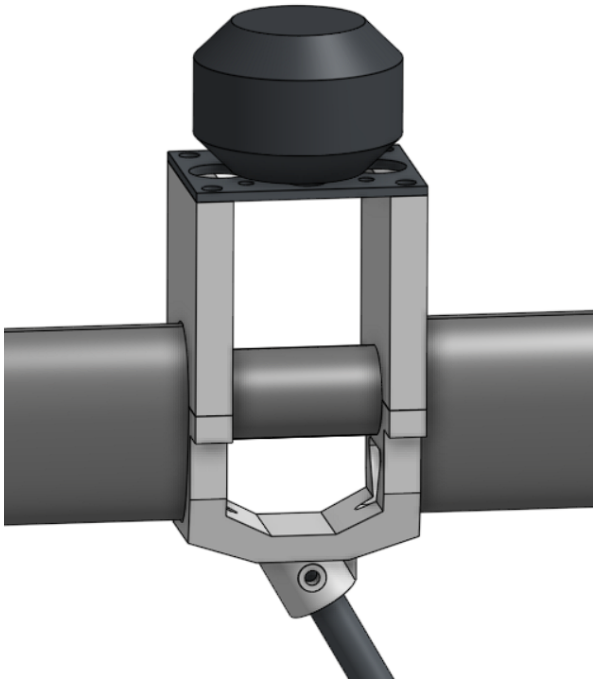


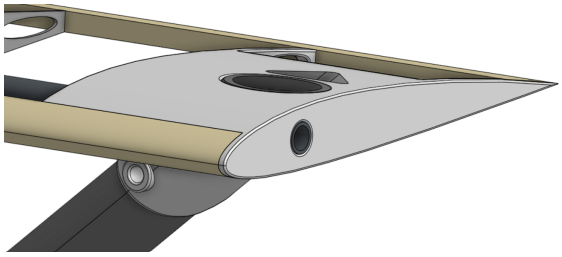
Figure A.3: Schematic of beam deflection calculation for both wing configurations.

tubes is shown in Figure A.4(c). The 8 wing mounts are all unique since the airfoil is non-symmetric, but they all have the same style and purpose. The back of the centre carbon fibre hub is shown in Figure A.4(d) with the back plate hidden to expose the mounting parts for the diagonal tubes.

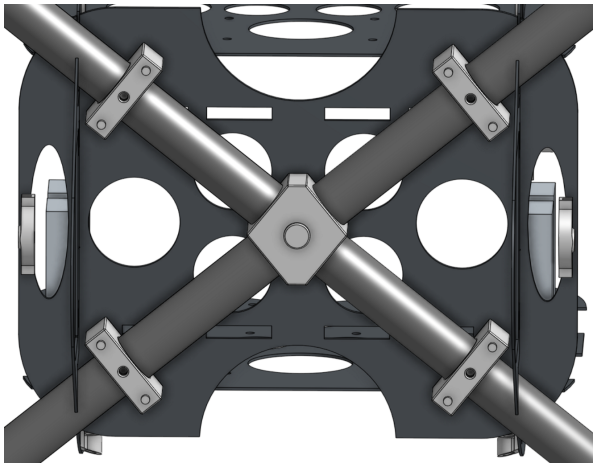
(a) Isometric view.



(b) Motor Mount.



(c) Top wing mount.



(d) Top wing mount.

Figure A.4: CAD of tailsitter prototype aircraft.

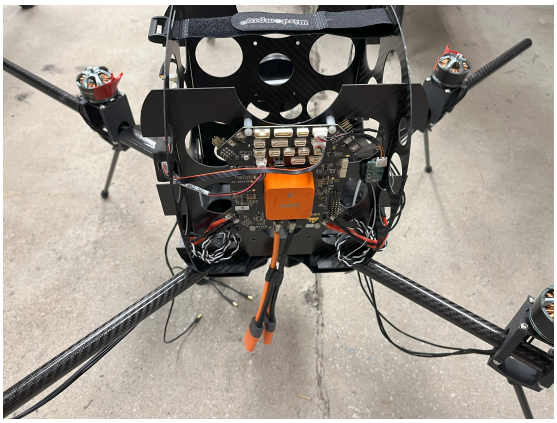
Appendix B

Detailed Prototype Aircraft Assembly

In this appendix, the assembly process for the prototype tailsitter is presented. The entire design and assembly process was conducted by the author, with guidance from the supervisors and funding from the NRC. There was a focus in the design process for every part to have the ability to be removable, so if anything was broken during testing, it could be replaced. The process for assembly was to first build the centre structure, then mount everything required for a multirotor, including all electronic components, as shown in Figure B.1. Additionally, the CAD models for just the carbon fibre payload bay and centre hub are shown in Figure B.2. These parts were assembled using four bolts, which also mount the diagonal carbon fibre tubes, along with epoxy, where the carbon fibre plates touch to reduce vibrations. The smaller holes were all used for mounting, either with bolts or alignment pins in the 3D printed parts, while the larger holes were used for weight reduction.

With the base structure and the electronics mounted, everything required for the first-time setup of ArduCopter could be configured, as defined here [13]. The setup included orienting the flight controller and GPS, calibrating the compass, accelerometer, transmitter, and ESCs, motor wiring, and carrier board output signals. The list of parameters modified to program the flight controller is given in Appendix C.

The wings were assembled next, which required gluing the carbon fibre ribs to the carbon fibre spars. Then the balsa leading and trailing edges were mounted onto the ribs, followed by sanding them to continue the shape of the airfoil ribs. The CAD model for the side view of the 1 mm thick rib is shown in Figure B.3. On the left is where the leading edge spar goes, with the trailing edge on the right. The second hole from the left is where the carbon fibre spar goes through each rib. The completed assembly of a short wing segment is shown in Figure B.4(a), while all of the built wings are shown in Figure B.4(b). The wings were then wrapped with Hanger 9 Transparent Ultracoat polyester covering to create the wing surface [67].



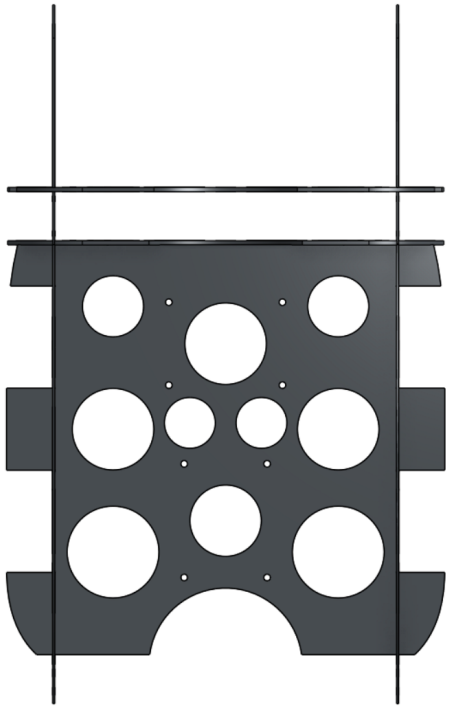
(a) Zoomed in view of flight controller electronics and the bottom of the centre hub.

(b) Zoomed out view including motors and diagonal carbon fibre tubes.

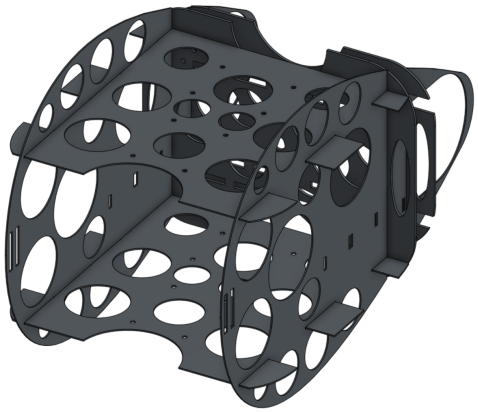
(c) Side of payload bay showing two ESCs and radio mounting.

(d) Top of payload bay showing the GPS position and mounting.

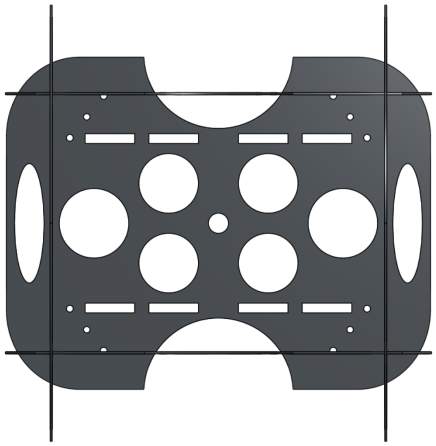
Figure B.1: Photos of the multirotor configuration of the tailsitter, including motors, ESCs, flight controller, and other sensors.



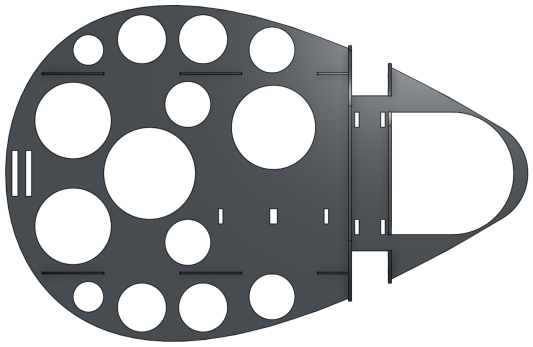
(a) Top view.



(b) Isometric view.



(c) Front view.



(d) Side view.

Figure B.2: Orthographic views of the carbon fibre plates making up the centre hub and payload bay.

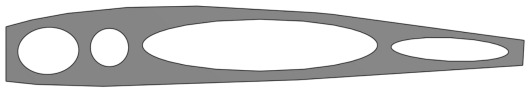
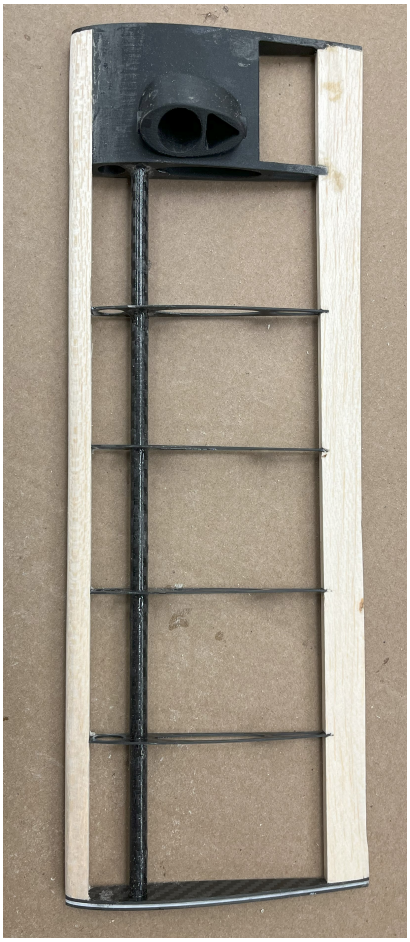


Figure B.3: CAD geometry of 1 mm thick airfoil rib for prototype wings.

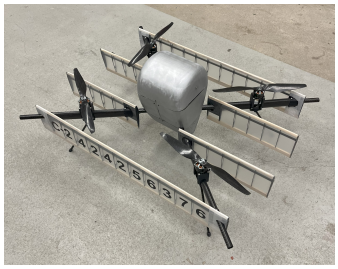


(a) Zoomed-in photo of a single short wing.

(b) Completed assembly of the 6 prototype wings.

Figure B.4: Photos of the wings of prototype aircraft, including carbon fibre ribs and spars, balsa leading and trailing edges, and 3D printing mounts.

(a) Front view showing wings and canopy.



(b) Angled view showing wings and canopy.

(c) Wing mounting on carbon fibre diagonal tube.

Figure B.5: Photos of fully assembled prototype aircraft.

With the wings completed, they can be mounted to the diagonal tubes and secured with bolts. Additionally, the propellers and canopy were also mounted at this time. The canopy segments were made by 3D printed negative molds, and using them to shape 3 layers of kevlar and resin. They were then spray-painted a silver colour to make the aircraft more visible and make the outer surface smoother. The aircraft at this stage is the final version used in flight, except for the payload and battery within the canopy, and is shown in Figure B.5. Figure B.5(c) specifically shows the mounting of the wing's 3D printed part to the diagonal tube.

Appendix C

Parameters

This Appendix contains the list of parameters changed for the test flights and simulation of the prototype aircraft for the custom version of ArduCopter. First, the parameters used for the simulated model are given in Table C.1, and the real aircraft parameters are listed in Table C.2. None of the COMPASS and INS parameters that are configured through the accelerometer and compass calibrations are listed.

Table C.1: ArduCopter parameters for simulated prototype aircraft.

Parameter	Original Value	New Value	Parameter	Original Value	New Value
AHRS_EKF_TYPE	3	10	MOT_SPIN_MAX	0.95	1
ANGLE_MAX	3000	8999	MOT_THST_EXPO	0.65	0.71
ARMING_CHECK	1	1045982	MOT_THST_HOVER	0.39	0.2307449
ATC_ACCEL_P_MAX	110000	78000	MOT_YAW_HEADROOM	200	300
ATC_ACCEL_R_MAX	110000	78000	PSC_ACCZ_I	1	0.44
ATC_ACCEL_Y_MAX	27000	22500	PSC_ACCZ_P	0.5	0.22
ATC_RAT_PIT_FLTD	20	15	RC2_REVERSED	0	1
ATC_RAT_PIT_FLTT	20	0	RC5_MAX	2000	1900
ATC_RAT_RLL_FLTD	20	15	RC5_MIN	1000	1100
ATC_RAT_RLL_FLTT	20	0	RC6_MAX	2000	1900
ATC_RAT_YAW_D	0	0.0001	RC6_MIN	1000	1100
ATC_RAT_YAW_FLTD	20	0	RC6_OPTION	0	109
ATC_RAT_YAW_FLTE	2.5	0	RC7_MAX	2000	1900
ATC_RAT_YAW_FLTT	20	0	RC7_MIN	1000	1100
ATC_RAT_YAW_I	0.02	0.018	RC7_OPTION	7	0
ATC_RAT_YAW_P	0.3	0.18	RC8_MAX	2000	1900
ATC_SLEW_YAW	6000	3000	RC8_MIN	1000	1100
AUTO_OPTIONS	0	1	RPM1_TYPE	0	10
AUTOTUNE_MIN_D	0.001	0.0005	SERVO1_MAX	1900	2200
BATT_ARM_VOLT	0	22.1	SERVO1_MIN	1100	1000
BATT_CAPACITY	3300	10000	SERVO2_MAX	1900	2200
BATT_CRT_VOLT	0	21	SERVO2_MIN	1100	1000
BATT_LOW_VOLT	10.5	21.6	SERVO3_MAX	1900	2200
CC_TYPE	0	3	SERVO3_MIN	1100	1000
COMPASS_DEC	0	0.2233572	SERVO4_MAX	1900	2200
FENCE_RADIUS	150	300	SERVO4_MIN	1100	1000
FLTMODE1	7	0	SERVO5_MAX	1900	2000
FLTMODE2	9	0	SERVO5_MIN	1100	1000
FLTMODE3	6	0	SERVO6_MAX	1900	2000
FLTMODE4	3	0	SERVO6_MIN	1100	1000
FLTMODE5	5	2	SIM_BARO_RND	0	0.2
FLTMODE6	0	5	SIM_PLD_LAT	-35.36326	0
FRAME_TYPE	0	1	SIM_PLD_LON	149.1652	0
LOIT_ACC_MAX	500	400	SR0_ADSB	0	4
LOIT_BRK_ACCEL	250	200	SR0_RAW_CTRL	0	4
LOIT_SPEED	1250	3000	STAT_BOOTCNT	1	535
MOT_BAT_VOLT_MAX	12.8	25.2	WP_YAW_BEHAVIOR	2	1
MOT_BAT_VOLT_MIN	9.6	19.8	WPNAV_SPEED	1000	3000

Table C.2: ArduCopter parameters for real prototype aircraft.

Parameter	Original Value	New Value	Parameter	Original Value	New Value
ADSB_TYPE	0	1	FS_OPTIONS	16	0
AHRS_ORIENTATION	0	29	FS_THR_ENABLE	1	0
AHRS_TRIM_X	0	-0.00394275	FS_VIBE_ENABLE	1	0
AHRS_TRIM_Y	0	-0.003639916	LOIT_SPEED	1250	2000
ANGLE_MAX	3000	8500	MOT_BAT_VOLT_MAX	12.8	25.2
ARMING_CHECK	1	1048030	MOT_BAT_VOLT_MIN	9.6	19.8
ATC_ACCEL_P_MAX	110000	78000	MOT_SAFE_DISARM	0	1
ATC_ACCEL_R_MAX	110000	78000	MOT_SPIN_ARM	0.1	0.07
ATC_ACCEL_Y_MAX	27000	22500	MOT_SPIN_MIN	0.15	0.1
ATC_RAT_PIT_FLTD	20	15	MOT_THST_EXPO	0.65	0.71
ATC_RAT_PIT_FLTT	20	0	MOT_THST_HOVER	0.39	0.3267
ATC_RAT_RLL_FLTD	20	15	NTF_LED_TYPES	123079	231
ATC_RAT_RLL_FLTT	20	0	RC_PROTOCOLS	1	256
ATC_RAT_YAW_FLTD	20	0	RC1_MAX	2000	1892
ATC_RAT_YAW_FLTE	2.5	0	RC1_MIN	1000	1102
ATC_RAT_YAW_FLTT	20	0	RC2_MAX	2000	1897
ATC_RAT_YAW_I	0.02	0.018	RC2_MIN	1000	1102
ATC_RAT_YAW_P	0.3	0.18	RC3_MAX	2000	1895
AUTOTUNE_AGGR	0.1	0.075	RC3_MIN	1000	1102
AUTOTUNE_MIN_D	0.001	0.0005	RC3_TRIM	1500	1112
BARO1_DEVID	65540	721442	RC4_MAX	2000	1897
BARO2_DEVID	65796	721674	RC4_MIN	1000	1102
BATT_AMP_OFFSET	0	0.45	RC5_MAX	2000	1897
BATT_AMP_PERVLT	17	50	RC5_MIN	1000	1102
BATT_ARM_VOLT	0	21.9	RC5_TRIM	1500	1102
BATT_CAPACITY	3300	10000	RC6_MAX	2000	1897
BATT_CRT_VOLT	0	21	RC6_MIN	1000	1102
BATT_CURR_PIN	12	15	RC6_OPTION	0	109
BATT_FS_CRT_ACT	0	1	RC6_TRIM	1500	1102
BATT_FS_LOW_ACT	0	2	RC7_MAX	2000	1900
BATT_LOW_VOLT	10.5	21.6	RC7_MIN	1000	1100
BATT_VOLT_MULT	10.1	15.3	RC7_OPTION	7	0
BATT_VOLT_PIN	13	14	RC8_MAX	2000	1900
BRD_OPTIONS	0	1	RC8_MIN	1000	1100
BRD_SAFETY_MASK	16368	16383	RELAY1_FUNCTION	1	0
BRD_SAFETYOPTION	3	0	RSSI_TYPE	0	3
CAN_P1_DRIVER	0	1	RTL_ALT	1500	40000
CC_TYPE	0	3	SERIAL1_BAUD	57	115
COMPASS_ORIENT	0	25	SERIAL1_OPTIONS	0	4
COMPASS_ORIENT2	0	24	SERIAL1_PROTOCOL	2	23
COMPASS_PRIO1_ID	97539	590114	SERIAL4_PROTOCOL	5	-1
COMPASS_PRIO2_ID	131874	97539	SERIAL5_PROTOCOL	-1	1
COMPASS_PRIO3_ID	263178	0	SERIAL6_BAUD	57	115200
COMPASS_SCALE	1	0	SERIAL6_PROTOCOL	-1	22
COMPASS_SCALE2	1	0	SERVO1_FUNCTION	33	0
COMPASS_SCALE3	1	0	SERVO2_FUNCTION	34	33
COMPASS_USE3	1	0	SERVO2_MAX	1900	2000
DISARM_DELAY	10	0	SERVO2_MIN	1100	1000
EK3_IMU_MASK	3	7	SERVO3_FUNCTION	35	0
EK3_PRIMARY	0	1	SERVO4_FUNCTION	36	35
FENCE_ACTION	1	3	SERVO4_MAX	1900	2000
FLTMODE1	7	0	SERVO4_MIN	1100	1000
FLTMODE2	9	0	SERVO7_FUNCTION	0	34
FLTMODE3	6	2	SERVO7_MAX	1900	2000
FLTMODE4	3	2	SERVO7_MIN	1100	1000
FLTMODE5	5	2	SERVO8_FUNCTION	0	36
FLTMODE6	0	5	SERVO8_MAX	1900	2000
FRAME_TYPE	0	1	SERVO8_MIN	1100	1000
FS_DR_ENABLE	2	0	WPNAV_SPEED	1000	2000

Appendix D

Detailed Software in the Loop Simulation Configuration

The purpose of this appendix is to give the complete process for how the simulation environment was configured. While many of the components are described from various sources online, there little to no examples that go from start to end in detail. Moreover, many of the examples utilize Linux for their simulation, introducing more barriers for researchers to trying to start using SITL. The guide presented in this appendix will reduce the barrier of entry for researchers in the future, allowing for the safe and accurate testing of complex vehicles and controllers that utilize the ArduPilot infrastructure.

D.1 RealFlight Simulation Configuration

A RealFlight 9.5 simulation requires three main elements to function correctly. The first is a visual model of the aircraft, which is displayed in the physics environment so the aircraft's motion and overall size can be observed. A physics model is also required, which uses the visual model to determine the scale of the components. In addition, all the interactions between the aircraft and the simulated world are calculated based on the physics model. Finally, the simulated electronics need to be configured to send the desired control signals to each rotor. A recent post on the ArduPilot forums detailing this procedure for a quadplane is given here [150].

D.1.1 Visual Model

The entire prototype aircraft CAD was developed in OnShape, then the major aerodynamic sections were imported into 3DS Max 2020 as an ACIS file to create the visual model for simulation [60]. 3DS Max was used instead of OnShape as software is required that can assign a TGA colour file to the geometry, along with the linking of the components, telling RealFlight how each section of the vehicle eventually connects to the fuselage. The visual

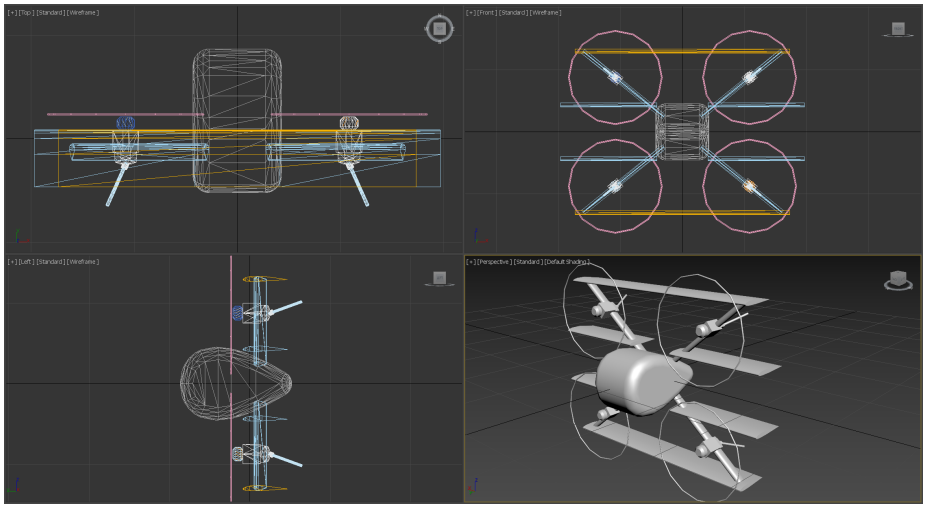


Figure D.1: Geometry and orientation used for simulation visualization.

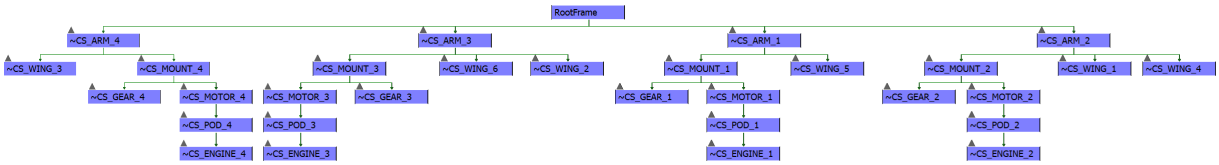


Figure D.2: 3ds Max 2020 link tree for simulated model.

model for the aircraft in 3DS Max 2020 is shown in Figure D.1. It should be noted that the vehicle is intentionally oriented as shown to align with what RealFlight 9.5 and ArduCopter expect for the SITL simulation.

At this stage, the parts need to be named using a specific convention where everything requires the prefix "˜CS_", except for the fuselage, which is called "RootFrame." Additionally, the propellers, which are shown as the rings in Figure reffig:A3dsMax, must be called an "ENGINE." The full list of components used for the simulation and their corresponding names is listed in Table D.1, and the linking tree that describes how the components connect is shown in Figure D.2. With these settings configured, the model must be exported as an .FBX file to be imported into RealFlight.

D.1.2 Physical Model

The .FBX file of the aircraft was then imported into RealFlight by navigating to Simulation > Import > 3D Model (FBX, KEX). The vehicle tab was used to confirm that the total mass, wing loading, and stall speed are close to the expected values throughout the setup procedure, shown in Figure D.3.

The physics of the simulated vehicle is all configured in the Physics tab. First, the air-

Table D.1: Component descriptions used in 3DS Max 2020 model of the prototype tailsitter and assigned names used in RealFlight 9.5.

Component Description	**3ds Max Name**
Full Canopy	RootFrame
Diagonal arm to motor 1	˜CS_ARM_1
Diagonal arm to motor 2	˜CS_ARM_2
Diagonal arm to motor 3	˜CS_ARM_3
Diagonal arm to motor 4	˜CS_ARM_4
Top large wing	CS_WING_1
Bottom large wing	˜CS_WING_2
Top left small wing	˜CS_WING_3
Top right small wing	˜CS_WING_4
Bottom left small wing	˜CS_WING_5
Bottom right small wing	˜CS_WING_6
Motor and landing gear mount 1	˜CS_MOUNT_1
Motor and landing gear mount 2	˜CS_MOUNT_2
Motor and landing gear mount 3	˜CS_MOUNT_3
Motor and landing gear mount 4	˜CS_MOUNT_4
Landing gear under motor 1	˜CS_GEAR_1
Landing gear under motor 2	˜CS_GEAR_2
Landing gear under motor 3	˜CS_GEAR_3
Landing gear under motor 4	˜CS_GEAR_4
Visual for motor 1	˜CS_MOTOR_1
Visual for motor 2	˜CS_MOTOR_2
Visual for motor 3	˜CS_MOTOR_3
Visual for motor 4	˜CS_MOTOR_4
Physics free frame for aligning motor 1 thrust	˜CS_POD_1
Physics free frame for aligning motor 2 thrust	˜CS_POD_2
Physics free frame for aligning motor 3 thrust	˜CS_POD_3
Physics free frame for aligning motor 4 thrust	˜CS_POD_4
Motor and propeller 1 size and position	˜CS_ENGINE_1
Motor and propeller 1 size and position	˜CS_ENGINE_2
Motor and propeller 1 size and position	˜CS_ENGINE_3
Motor and propeller 1 size and position	˜CS_ENGINE_4

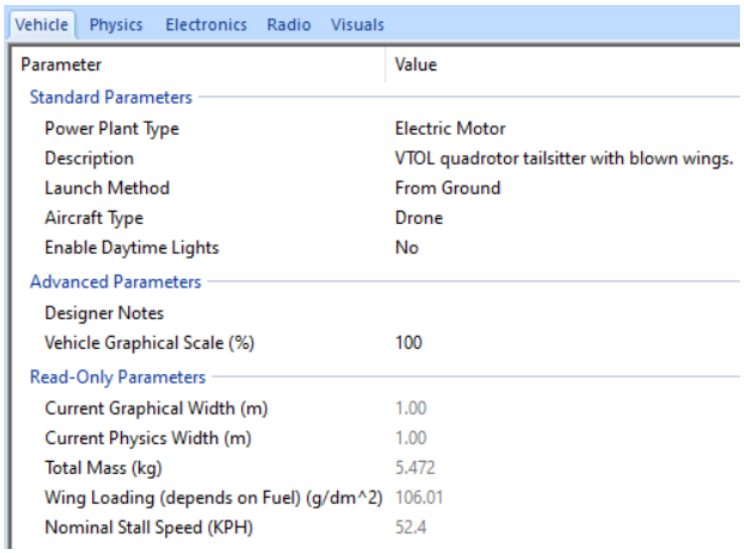


Figure D.3: RealFlight 9.5 vehicle settings for simulated tailsitter.

frame component allows for the adjustment of the CG, which should be set to 0 mm in all directions if the model is accurate. Next, the battery can be configured using the built-in cell types and then selecting the size and number of cells in series and parallel. For the simulation, the cell used was a 5000 mAh LiPo, with the total battery having 2 parallel sets of 6 cells in series. The expected weight of this configuration is 1.361 kg, closely approximating the 1.375 kg of the real battery [36]. The battery configuration is shown in Figure D.4.

Next, components can be added relative to their parents by right-clicking on the parent and adding the correct component type. The parent and child relationships should match the linking previously configured in Figure D.2. For example, the fuselage was first added to the Airframe, which has the shape of the canopy but includes the mass of everything within the canopy to simplify the model. The component type for the fuselage is called "fuselage," and the only settings changed were the physical shape and mass. The shape for the physical model can be defined by selecting RootFrame (the canopy as shown in Table D.1), then right-clicking the component from the Physics tab, and selecting "fit to visual." The mass was set to 1.55 kg to include all parts making up the centre hub, canopy, and payload of the prototype aircraft, and the changes are shown in Figure D.5.

The wings were then added as a child component to the fuselage, following the same procedure. Each was assigned a visual frame, a mass of 0.18 kg for the large wings, while the small wings had a mass of 0.095 kg, matching the weight distribution from the CAD model. The wing root and tip were both set to the NACA 2412 airfoil with a chord length of 140 mm. The parameters used to define the physics for wing 1 are shown in Figure D.6, and the same process was repeated for the others.

Parameter	Value
Standard Parameters	
Location in Parent (m)	x=0, y=-.050, z=0
Number Cells in Series	6
Number Cells in Parallel	2
Cell Type	LiPoly 5000 mah
Dimensions (mm)	x=176, y=66, z=58
Advanced Parameters	
Speed Controller Resistance (Ohm)	.0030
Speed Controller Current Limit (Amp)	0
Read-Only Parameters	
Component Type	Battery
Current Mass of Self (g)	1361
Energy Remaining (%)	100
Total Voltage (v)	25.2
Total Capacity	10000

Figure D.4: RealFlight 9.5 battery settings for simulated tailsitter.

Parameter	Value
Standard Parameters	
Location in Parent (m)	x=-.001, y=.221, z=-0
Weight (g)	1550
Dimensions (mm)	x=218, y=353, z=231
Advanced Parameters	
Strength Multiplier (%)	100
Visual Frame	RootFrame

Figure D.5: RealFlight 9.5 fuselage settings for simulated tailsitter.

Parameter	Value
Standard Parameters	
Location in Parent (m)	x=-.440, y=-.200, z=.330
Weight (g)	180
Wing Length (mm)	880
Chord at Root (mm)	140
Chord at Tip (mm)	140
Airfoil at Tip	NACA 2412
Airfoil at Root	NACA 2412
Dihedral (deg)	0
Leading Edge Sweep (deg)	0
Incidence at Root (deg)	0
Washout at Tip (deg)	0
Advanced Parameters	
Strength Multiplier (%)	100
Visual Frame	~CS_WING_1

Figure D.6: RealFlight 9.5 wing settings for simulated tailsitter.

The next step taken was to add the four diagonal arms, which were also modelled as wings due to their airfoil shape. These each had a mass of 0.13 kg and a chord length of 40 mm. The only major change compared to the other wings is the dihedral set to account for the angle with the ground, with the full settings shown in Figure D.7.

The motor mounts were then defined as fuselage components under their corresponding diagonal arms. These parts were each assigned a mass of 0.06 kg, then fit to match their visuals defined in 3DS Max. The same process was repeated to make the motors children of the motor mounts, with a mass of 0.05 kg. Additionally, each mount had a skid landing gear component added to it to represent the carbon fibre landing gear. These each had a mass of 7 g, a length of 105 mm, and required rotations about the X and Y axes for proper alignment. The settings for Skid 1 for the landing gear under motor 1 is shown in Figure D.8.

Attached to the top of each motor fuselage component is a movable pod represented by a plane in 3ds Max. These pods allow for the adjustment of the angle and position of the thrust vector of the rotors in the simulation. They have no mass or aerodynamic properties. The parameters set for each of the pods are shown in Figure D.9.

The final components to configure are the engines, which are children of the respective pods and represent the motors and propellers. Each of the engines has a throttle servo assigned, which is defined in Section D.1.3. Additionally, the propeller diameter, pitch, and type are assigned here, which were all configured to match the APC 15x10 inch electric propellers used [90] on the prototype. A custom torque generator was used to match the performance presented by [91] and [112], with the parameters shown in Figure D.10. The

Parameter	Value
Standard Parameters	
Location in Parent (m)	x=.090, y=-.235, z=-.070
Weight (g)	130
Wing Length (mm)	410
Chord at Root (mm)	40
Chord at Tip (mm)	40
Airfoil at Tip	NACA 0024
Airfoil at Root	NACA 0024
Dihedral (deg)	-38.9
Leading Edge Sweep (deg)	0
Incidence at Root (deg)	0
Washout at Tip (deg)	0
Advanced Parameters	
Strength Multiplier (%)	100
Visual Frame	~CS_ARM_1

Figure D.7: RealFlight 9.5 diagonal arm settings for simulated tailsitter.

Parameter	Value
Standard Parameters	
Location in Parent (m)	x=.055, y=-.180, z=0
Weight (g)	7.0
Max Steering Angle (deg)	30
Rotation about Y (Gear Up) (deg)	0
Rotation about Y (Gear Down) (deg)	-90
Rotation about X (Gear Up) (deg)	0
Rotation about X (Gear Down) (deg)	120
Landing Gear Length (mm)	105
Retract Servo	<None>
Retract Servo Reverse	No
Steering Servo	<None>
Steering Servo Reverse	No
Swivels Freely	No
Sound Effect Volume (%)	100
Advanced Parameters	
Strength Multiplier (%)	100
Visual Frame	~CS_GEAR_1

Figure D.8: RealFlight 9.5 landing gear settings for simulated tailsitter.

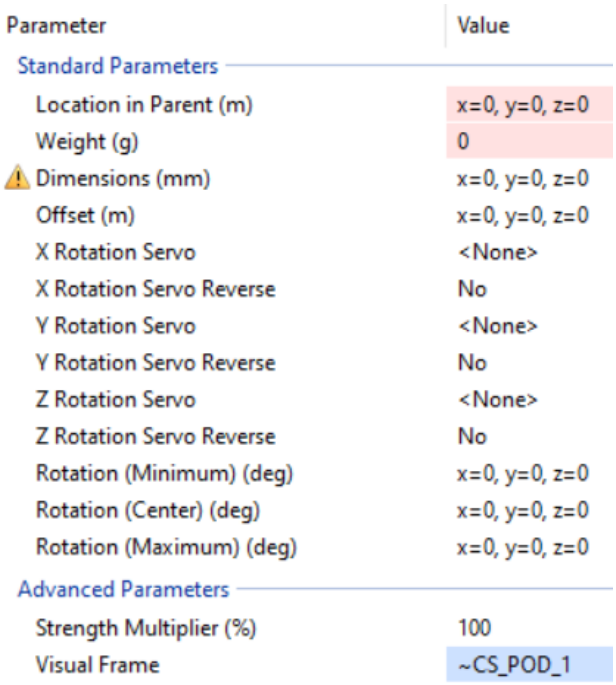


Figure D.9: RealFlight 9.5 movable pod settings for simulated tailsitter.

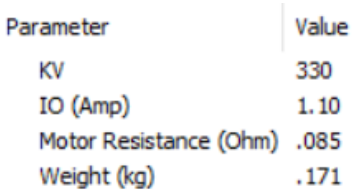


Figure D.10: RealFlight 9.5 torque generator settings for simulated tailsitter.

last parameter to change is the rotation direction. Motors 1 and 2 were set to clockwise, while 3 and 4 were set to counterclockwise, as expected by ArduCopter [119]. It should be noted that this is the opposite of the layouts presented by ArduCopter. The full list of parameters for the engines is shown in Figure D.11.

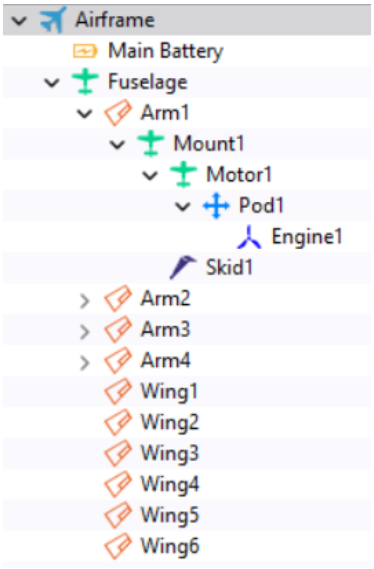
This completes the setup of the aircraft physics for the simulation. The list of component relationships is shown in Figure D.12(a), but the children of only arm 1 are fully shown. Additionally, the visualization of the configured physics within RealFlight 9.5 is shown in Figure D.12(b).

D.1.3 Simulated Electronics

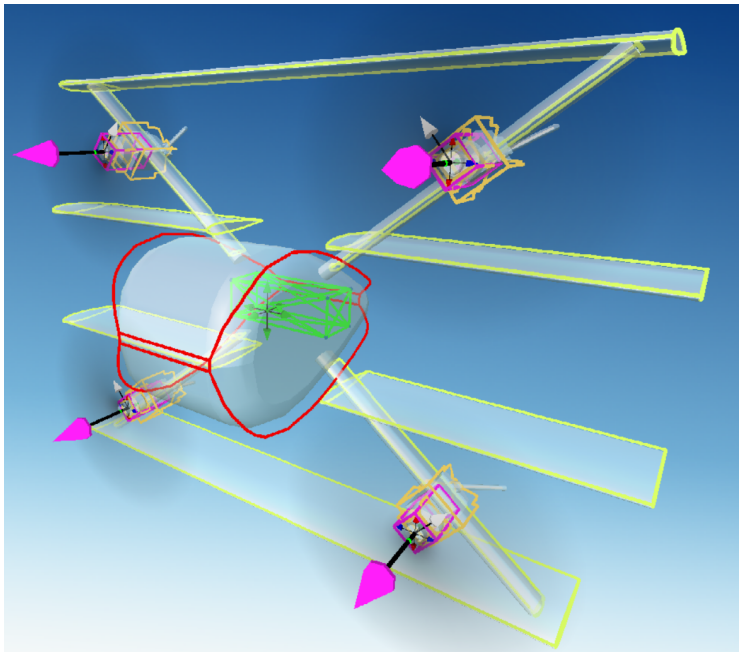
The simulated electronics for RealFlight 9.5 are configured in the Radio and Electronics tabs. For a quadrotor aircraft, only four servo signals are required, sending a PWM signal to each motor defined in the Electronics tab and shown in Figure D.13(a). The only change

Parameter	Value
Standard Parameters	
Location in Parent (m)	x=0, y=0, z=0
Throttle Servo	Motor 1
Throttle Servo Reverse	No
Sound Profile	<None>
Engine Sound Volume (%)	100
Down Thrust (deg)	0
Right Thrust (deg)	0
Prop Pitch (mm)	254
Prop Diameter (mm)	381
Number of Blades	2
Power (%)	100
Gear Ratio	1.00
Prop Spins Clockwise	Yes
Torque Generator	Tmotor MN4014 KV400
Propeller Type	APC Electric
Advanced Parameters	
Is Ducted Fan	No
Is Pusher Prop	No
Is Motor Reversible	No
Has Speed Control Brake	No
Prop Visual Scale (%)	100
Engine to Show	None
Visual Frame	~CS_ENGINE_1

Figure D.11: RealFlight 9.5 engine settings for simulated tailsitter.

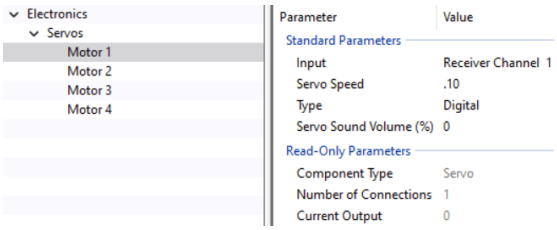


(a) List of physics components and parent/child relationships.

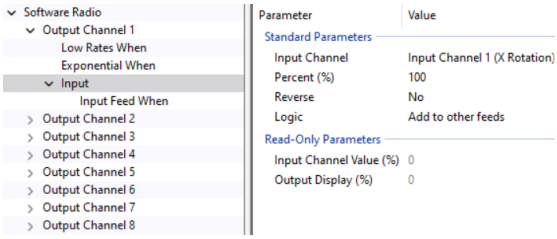


(b) Visualization of simulated physics.

Figure D.12: RealFlight 9.5 completed physical model.



(a) Example of servo setup for motors.



(b) Example of radio configuration for channel 1.

Figure D.13: RealFlight 9.5 electronics configuration.

between servos is the motor receiving the signal. In the radio tab, the output channels are configured, pass information to Mission Planner if an RC transmitter is used. For each channel (8 was required for the transmitter used), the low rates and exponential rates were set to never occur, since this is processed by the flight controller. The input was set by connecting the radio to the desired output of Mission Planner based on how the servos were configured. The percentage of the signal was set to 100, ensuring that each channel is always active. These configuration parameters are shown for channel 1 in Figure D.13(b).

D.2 Software in the Loop Simulation Setup Procedure

ArduPilot is an open-source autopilot that has been developed to work for a variety of remotely controlled vehicles, including multirotors and fixed-wing aircraft [8]. It is the flight controller software that was used and modified in Chapter 4, but can also be simulated using SITL. The SITL feature enables the ability to thoroughly test the flight controller capabilities by running the same code as implemented on a real aircraft. In this section, the setup procedure for the SITL simulation used to test the autopilot and physics of the vehicle setup in Section 3.2 is presented. While the ability to test SITL just using Mission Planner and RealFlight 9.5 is possible, the presented method allows for an efficient workflow that includes modification of the ArduPilot code.

D.2.1 Requirements

The methods for modifying the ArduPilot code discussed and presented in the forums and tutorials are mainly accomplished using Linux. To utilize the same system in Windows 10 or 11, WSL is required [81]. WSL allows for the Windows functionality to be used while still being able to access Linux applications. A Linux distribution is also required, such as Ubuntu [68]. ArduPilot recommends using Eclipse, NotePad++, VSCode, or Atom to modify the code, as they have been utilized previously with the ArduPilot project [12]. A recommendation of the author is to use VSCode due to the ability to connect directly to WSL and utilize the Ubuntu terminal within the editor [80].

To verify the usage and performance of any modified code, SITL simulation should be utilized, which can be accomplished through Ubuntu by following the documentation to set up SITL on Windows [123]. However, a SITL simulation ran using a GCS such as Mission Planner allows the user to better represent the usage of the system if physically implemented [4]. This also enables the use of a visualization and physics-based simulation programs such as RealFlight [12, 95].

D.2.2 Windows Subsystem for Linux

The WSL can be set up in Windows 10 and 11 following the documentation provided by ArduPilot [14] and Microsoft [81]. The purpose of using a Linux subsystem is to allow the building of the code using Waf. The Waf framework is a Python-based build system that is used to compile software projects in efficient and structured processes. It also enables developers to build the ArduPilot code for specific uses, such as selecting the hardware to implement it on.

D.2.2.1 Installation

To set up a Windows Subsystem for Linux, the system must first be enabled within Windows through the following process:

```
Control Pannel > Programs > Turn Windows features on or off
```

Then select the checkbox beside Windows Subsystem for Linux, followed by clicking the OK button. Next, a Linux distribution such as Ubuntu needs to be installed from the Microsoft Store. At the time of writing, Ubuntu 22.04.1 LTS was used, however, different versions are not expected to impede usage. Once installed, the user can launch Ubuntu and then select a username and password. This information will be required whenever making changes to the system configuration or running sudo commands. When WSL is installed, it must be configured to the desired version. While WSL 2 is faster, it cannot upload code or connect to a physical autopilot, nor can it run a RealFlight simulation [14]. Therefore, if these utilities are required, WSL 1 should be used instead, as done for this project. The preferred version of WSL can be set using the following commands within Windows PowerShell.

```
wsl --set-version Ubuntu 1
wsl --set-version Ubuntu 2
```

D.2.2.2 Git on Ubuntu

To use Git and GitHub on Ubuntu, it must be installed first by using the following commands, allowing the user to conduct the typical operations used with Git on the files within WSL.

```
sudo apt-get update
sudo apt-get install git
sudo apt-get install gitk git-gui
```

D.2.2.3 Cloning ArduPilot

To clone the ArduPilot repository, first navigate to the desired file location in Ubuntu. The current directory is where the cloned files will be located. The entire project can then be cloned to the Linux system using the command:

```
git clone --recurse-submodules https://github.com/username/ardupilot
```

where username must be replaced with the user's GitHub username. When cloned, the files can all be accessed with Ubuntu or easily modified through VSCode, as discussed below. After cloning ArduPilot, the user can move into the new directory using

```
cd ArduPilot
```

Then, install a set of required packages with the command

```
Tools/environment_install/install-prereqs-ubuntu.sh -y
```

To make these changes permanent, the user must reload the path using

```
. ~/.profile
```

D.2.3 Visual Studio Code

The next process is to get VSCode (or another editor) installed and working with the WSL. VSCode will incorporate itself automatically with WSL on Windows computers [14]. To connect to the WSL, click on the blue button in the bottom left corner that says "Open a Remote Window" when hovered on. Then click on the "Connect to WSL" option in the pop-up window. Once connected, the user can navigate to the folder containing the forked and cloned ArduPilot repository. Here, the entire codebase can be seen, allowing the user to modify, add, or remove any files as needed.

D.2.4 Building the Code

Building the code allows the user to create the files required to implement the flight controller, either for SITL testing or on physical hardware. After a fresh cloning of the ArduPilot repository, the code should be built to ensure all files were copied correctly. A video explaining how to build the code in Linux is shown here [10], with the steps outlined as follows. The code can be built either using an Ubuntu terminal or the WSL terminal within VSCode. Start by changing directories into the main ArduPilot folder, such that the terminal has the prefix

```
username@device:~/ardupilot$
```

where username is the same as what was created when the Linux distribution was connected for the first time, and device is the name of the computer being used. Next, the user should determine if any changes have been made to the code that need to be committed to Git using

```
git status
```

If it returns with a message "Your branch is up to date with 'origin/master,'" then the user can continue. If the vehicle type or board type that the code will be built changes, the user must clean traces of past build processes. It can also be done at any point before building the code to ensure any changes made are applied. The following command is used to clean the build process:

```
./waf distclean
```

Now the desired board to be built on needs to be configured, which can either be applied for physical hardware or for SITL. The entire list of boards can be found by using the command

```
./waf list_boards
```

For example, a build process meant to be used for SITL can be configured using the command

```
./waf configure --board SITL.
```

Now the build process can be used to create the flight controller file for implementation or testing. The user must also define the vehicle type they wish to use, such as copter, plane, or any other vehicle supported by ArduPilot. In addition, if the user desires to use a custom flight controller as discussed later, they must add the suffix:

```
--enable-custom-controller
```

Therefore, a build file for a multirotor that will be using a custom controller can be built with

```
./waf copter --enable-custom-controller
```

The files created by this process can be found from the Windows File Explorer in the WSL directory given by:

```
Linux > Ubuntu > home > username > ardupilot > build > board > bin
```

where username is the name given for the Linux distribution, and board is the board selected in the configuration step.

D.2.5 MATLAB and Simulink Attitude Controller

The C++ code generation tool in MATLAB and Simulink can be utilized to create new flight controllers for ArduPilot, as described here [7]. The Simulink model must be configured as ArduPilot expects, using the following steps. First, in the settings, the solver must be set to discrete with fixed steps of 0.0025 seconds (400 Hz), representing the refresh rate of ArduPilot's main code. Also, in hardware implementation, the device vendor should be changed to ARM Compatible and the device type to ARM Cortex-M.

The custom controller was implemented using MATLAB and Simulink so that it could be configured with ArduCopter following the procedure described in Section 3.3.2. The fuzzy surfaces were constructed using the Fuzzy Logic Designer application [127] and parameters previously defined, which gave a .fis file for each map. Then each file was converted into a surface that could be used as inputs for a lookup table block using the gensurf function [128]. The proportional controller with variable gain based on pitch is shown in Figure D.14, and the fuzzy logic PID controller for yaw is shown in Figure D.15.

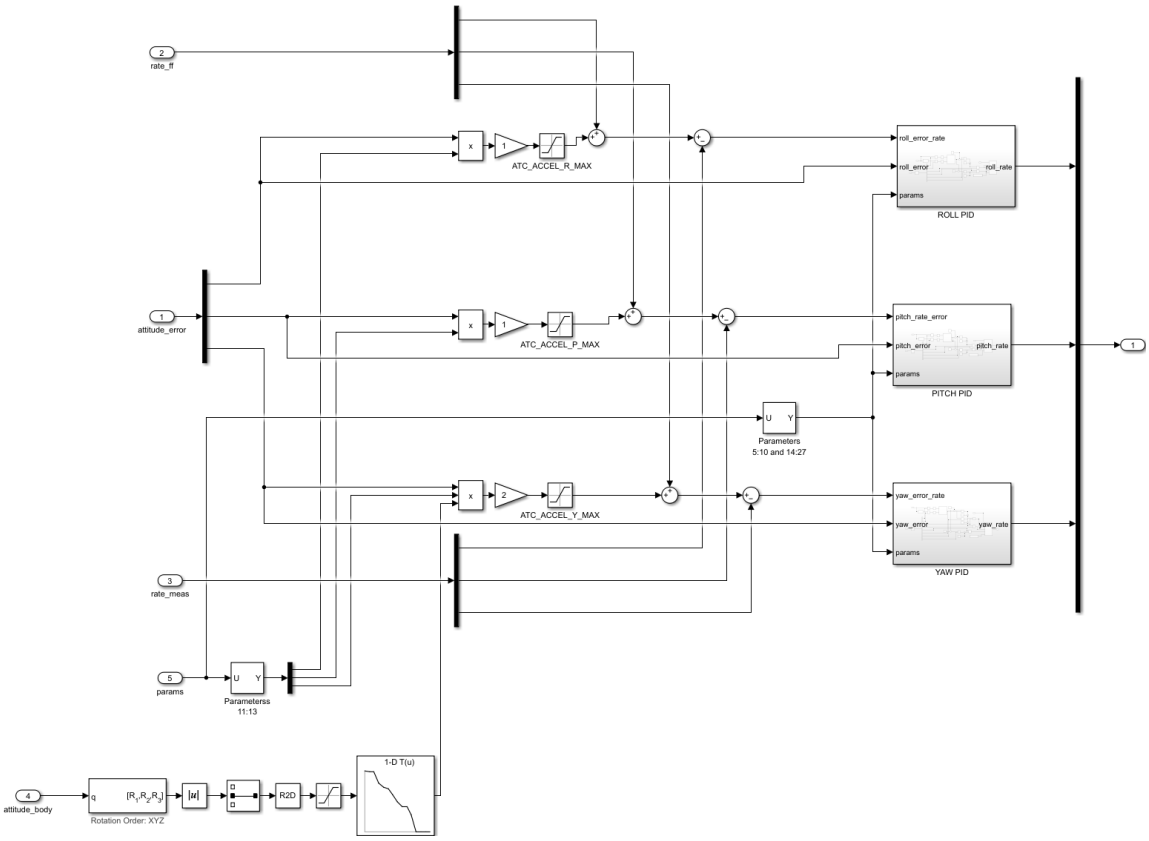


Figure D.14: Simulink variable proportional controller with fuzzy logic lookup table.

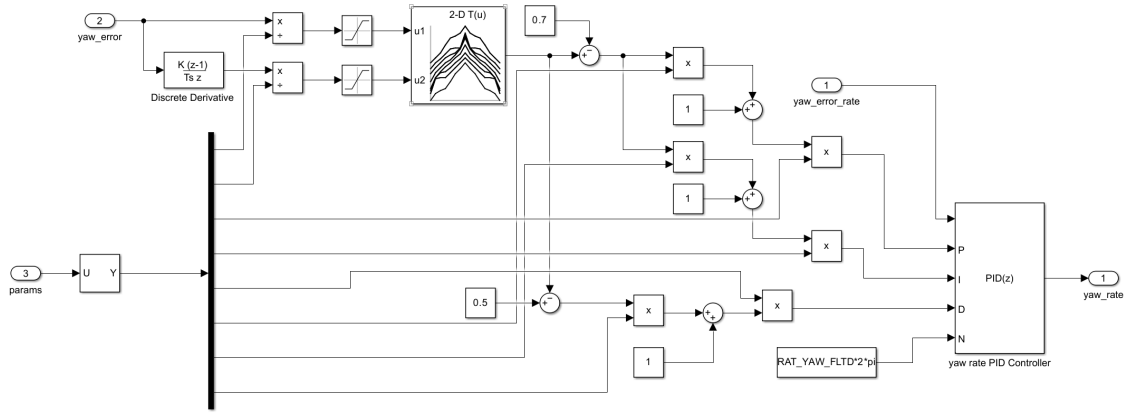


Figure D.15: Simulink fuzzy PID with fuzzy logic lookup table for yaw axis.

D.2.5.1 Block Settings

The input and output blocks must have their data type set to single. Additionally, it is recommended to set the output data type of all blocks throughout to single. This setting is found in

```
Block Parameters > Signal Attributes > Data type: single
```

It should be noted that only discrete time blocks can be used. In the case of PID controllers, the PID block can be changed to discrete time by opening the block parameters and selecting discrete time under the time domain section.

D.2.5.2 Code Generation

To generate the C++ code from the Simulink file, the Embedded Coder app must be installed from the Simulink Add-Ons [126]. Then, in the Model Settings, navigate to Code Generation to make the following changes. First, the System Target File must be set to ert.tlc with a description of Embedded Encoder, found by selecting the "Browse" button on the right in the Target Selection box. In the same area, the Language should be set to C++, while C++11 (ISO) is selected for the Language standard. In the Build process section, select the check boxes for both Generate code only and Package code and artifacts. Additionally, MinGW64 | gmake (64-bit Windows) must be selected from the drop-down options for Toolchain, and Faster Builds was selected for the Build configuration. Now, the user should navigate to the Optimization section within Code Generation. The Default parameter behaviour must be set to Tunable to allow the variables to change throughout the use of the flight controller (i.e., the attitude error).

If the Block and Model Settings are both configured, the last step to generate the code is to configure the code mappings. These settings can be found by clicking on the Embedded Coder button under the Apps ribbon, then in the C++ CODE ribbon that opens, click on Code Interface, then Code Mappings. This will open a section at the bottom of Simulink called Code Mappings—Component Interface. Here, the Data Visibility for Inports, Outports, and Model parameters need to be changed to public from the drop-down box. Also, the Member Access Method on the right must be changed to None for all Model Element Categories.

The code generated will utilize three functions: Initialize, Terminate, and Periodic or step. The latter is what is used each time the flight controller is called by ArduPilot. Its settings can be modified by navigating to the Functions tab in the Code Mappings – Component Interface, then selecting the void step function in the Method Preview column. In the box that pops up, the C++ Type Qualify must be changed to Pointer for all options. After this, press the Validate button to ensure that all settings have been configured correctly for code generation. Note that the validation will fail if there are parameters that are used in the

Simulink model that have not been assigned values in the current workspace. Therefore, if there are parameters that are used from the ground control station configuration (i.e., PID tuning parameters), these must be assigned an initial value in the workspace. These values do not need to match what was set in the ground control station. Alternatively, parameters can be set as inputs into the Simulink controller, read from ArduCopter's parameter list, which is the method used in Chapter 4.

The next step is to generate the code by selecting the blue button under the C++ CODE ribbon. This code will be in a zip folder found in the same directory as the Simulink model. The code generated can be used as a new flight controller within ArduPilot.

D.2.5.3 Adding custom controller to ArduPilot

The process for adding a custom attitude flight controller to ArduCopter has already been explained here [7]. However, this section aims to combine all the information required to accomplish this process into a single document while also improving the workflow organization.

1. The first step is to copy and paste the files AC_CustomControl_Empty.cpp and AC_CustomControl_Empty.h from the same Linux directory

   ```
   Ardupilot > libraries > AC_CustomControl
   ```

   This can be easily achieved by navigating the explorer within VSCode and modifying files like how Windows File Explorer is used. These files essentially contain the code required to use the Simulink code generated within ArduPilot. However, some modification to both the file names and contents is still required. First, the file names need to be changed, which will remain consistent throughout.

2. In the posted example, the endings of the file names are changed to

   - AC_CustomControl_XYZ.cpp
   - AC_CustomControl_XYZ.h.

3. Next, in the files just created and renamed, the verbiage must be updated so that all instances of AC_CustomControl_Empty match the file names previously defined. In addition, the word Empty can be replaced by just the suffix to the above files.

4. The number of custom controller types also must be increased from 2 to 3 by increasing the number of the following line in the file AC_CustomControl.h. This allows for an additional controller to be added without needing to change any of the existing custom controllers.

```
#define CUSTOMCONTROL_MAX_TYPES 3
```

5. In the same file, the section with the commented title "add custom controller here" must have the new controller added. This is another parameter that will be called by different files, so the name must be used again. The code once the XYZ controller is added looks like

```
// add custom controller here
enum class CustomControlType : uint8_t {
CONT_NONE            = 0,
CONT_EMPTY           = 1,
CONT_PID             = 2,
CONT_XYZ             = 3,
}; // controller that should be used
```

6. In the file AC_CustomControl.cpp, the line

```
#include "AC_CustomControl_XYZ.h"
```

needs to be added below the other controllers, where the above name is the same as previously defined in step 2. This section of the code would appear as follows when complete.

```
#include "AC_CustomControl_Backend.h"
#include "AC_CustomControl_Empty.h"
#include "AC_CustomControl_PID.h"
#include "AC_CustomControl_XYZ.h"
```

7. In the same file, the custom controller code file needs to be added as a new backend parameter. This section of code, using the file names discussed herein, should be as follows. This section of code tells ArduPilot which controller to use, assigning the XYZ controller to number 3. Ensure to increment all numbers within these lines of code, as shown.

```
// parameters for empty controller
AP_SUBGROUPVARPTR(_backend, "1_", 6,
    AC_CustomControl, _backend_var_info[0]),
// parameters for PID controller
AP_SUBGROUPVARPTR(_backend, "2_", 7,
    AC_CustomControl, _backend_var_info[1]),
```

```
// parameters for XYZ controller
AP_SUBGROUPVARPTR(_backend, "3_", 8,
    AC_CustomControl, _backend_var_info[2]),
AP_GROUPEND
```

8. Using the same file, copy and paste, then modify the section of code that enables custom controllers to allow the user to select the controller type. After modification, the code should look as follows. Note that this section needs to include names defined in steps 1 and 5.

```
switch (CustomControlType(_controller_type))
{
    case CustomControlType::CONT_NONE:
        break;
    case CustomControlType::CONT_EMPTY:
        // This is template backend. Don't initialize it.
        // _backend = new AC_CustomControl_Empty(
                *this, _ahrs, _att_control, _motors, _dt);
        // _backend_var_info[get_type()] =
                AC_CustomControl_Empty::var_info;
        break;
    case CustomControlType::CONT_PID:
        _backend = new AC_CustomControl_PID(
                *this, _ahrs, _att_control, _motors, _dt);
        _backend_var_info[get_type()] =
                AC_CustomControl_PID::var_info;
        break;
    case CustomControlType::CONT_XYZ:
        _backend = new AC_CustomControl_XYZ(
                *this, _ahrs, _att_control, _motors, _dt);
        _backend_var_info[get_type()] =
                AC_CustomControl_XYZ::var_info;
        break;
    default:
        return;
}
```

9. The updated code should then be compiled within the VSCode terminal from the ardupilot directory using the command:

```
./waf configure --board sitl --debug copter
```

If the compilation runs without error, a simple SITL simulation could be used to verify that the new controller is usable and configured correctly. The following command can be used to start a basic ArduCopter SITL simulation from the ardupilot directory.

```
cd Tools/autotest/
./sim_vehicle.py -v ArduCopter
```

D.2.6 Connecting Mission Planner and RealFlight to SITL

The benefits of using a SITL with a ground control station, such as Mission Planner, and a physics model and visualizer, such as RealFlight, are twofold. First, the use of a ground control station in the simulation allows the user to learn how to set up physical hardware, since the user interface and techniques are the same. This includes setup procedures, configuring the ArduPilot settings, and reading the heads-up display. Secondly, the ability to visualize how the aircraft is flying and then compare it with the heads-up display will allow the user to learn how they are related. Additionally, physics-based simulations like RealFlight can be used to simulate very specific aircraft, giving the ability to test the devices that will be implemented, creating an accurate simulation.

D.2.6.1 RealFlight 9.5

The flight simulation and visualization software used was RealFlight 9.5 [95], but other programs are also capable. RealFlight 9.5 was selected due to its accurate simulation of model aircraft and the ability to customize the vehicles if desired. To use RealFlight 9.5 as the physics simulation for SITL with Mission Planner, a few settings must be changed as outlined in [15]. First, the RealFlight link must be enabled, which can be found by navigating through

```
Simulation > Settings > Physics > RealFlight Link Enabled > Yes
```

Next, the simulation must be configured to run even when using other programs such as Mission Planner. This is accomplished by setting Pause Sim When in Background and Pause Sim When in Menu both to No, which are found in the same physics tab as above.

D.2.6.2 Mission Planner

To include Mission Planner in the SITL simulation, the application must be open before starting it. Once the simulation has started, the parameters for the RPAS can be programmed so the aircraft flies as desired.

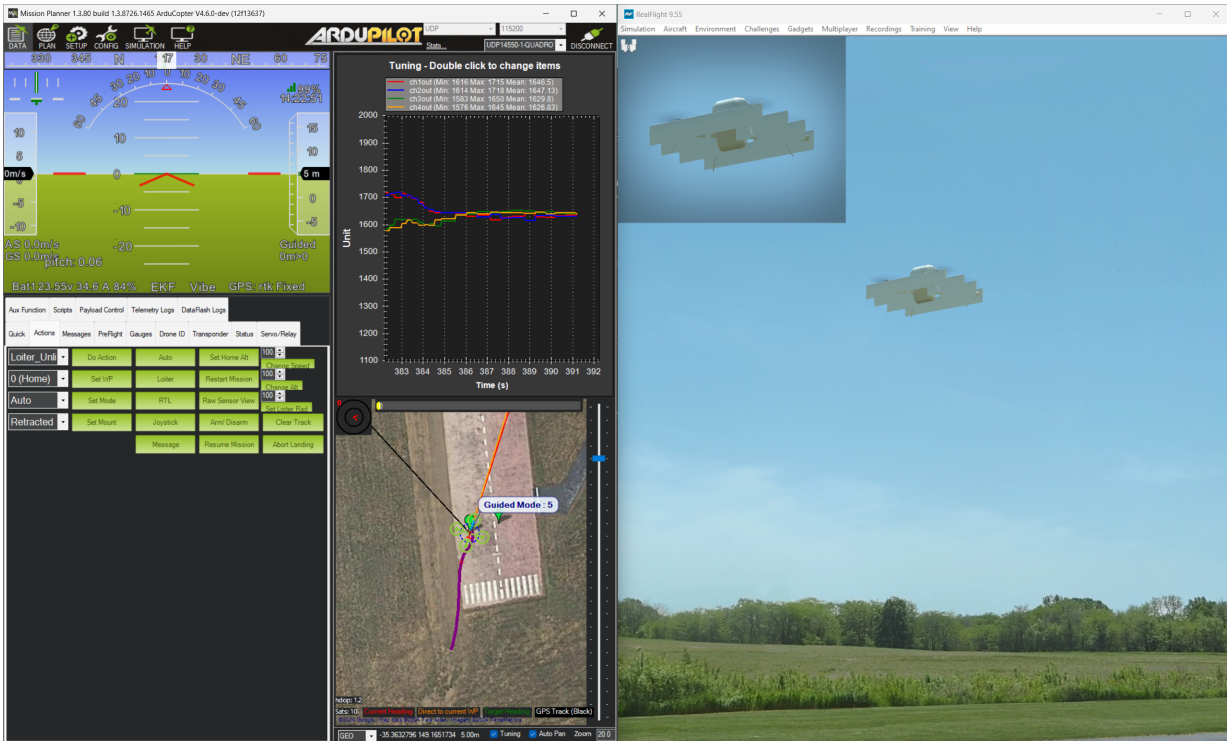


Figure D.16: SITL simulation with the Mission Planner app on the left and RealFlight 9.5 on the right.

D.2.6.3 Running SITL

The last step in the process is to run the SITL simulation from the Linux terminal of choice, with preference for the VSCode built-in terminals. For this action, the user requires their IPv4 Address, which can be found using the following command in Windows PowerShell.

```
ipconfig
```

The SITL simulation that is simultaneously connected to RealFlight 9.5 and Mission Planner can be started from the folder /ardupilot/Tools/autotest using the command:

```
./sim_vehicle.py -v TYPE - quad output add IPv4 -f flightaxis:IPv4
```

where IP is replaced by the IPv4 Address, and TYPE is replaced by the version of ArduPilot to be used, such as ArduCopter. Once running, commands or inputs to the simulation can either be given through the Linux terminal or through Mission Planner. A screenshot showing the RealFlight 9.5 and Mission Planner SITL simulation of the prototype tailsitter is shown in Figure D.16.

Appendix E

Detailed Control Literature Review

E.1 Multirotor Review

For multirotor UASs, many off-the-shelf solutions exist for their control. One prominent example is the open-source ArduCopter suite produced by ArduPilot [6]. The benefit of this type of flight controller is that it can be adapted based on the specific needs of each aircraft. For example, the number of motors, their orientation, sensor usage, speed and attitude limits, and more can all be activities and utilized depending on the parameters set in a GCS such as Mission Planner or QGroundControl [4, 41]. However, since this needs to work on a vast range of aircraft, it uses a generic P plus PID attitude control method to send PWM signals to the motors for their actuation [3]. Similar off the shelf flight controllers include the Dronecode PX4 Pixhawk system, KISS by Flyduino, Betaflight, Cleanflight, and Arduino [2, 23, 34, 42, 46].

E.1.1 Agile Multirotor Control and Modelling

Many multirotor controllers linearize the system around hover for the aircraft, making it easier to apply model based techniques. However, since the desired controller will only have a single flight mode and experience large changes in the pitch axis, these methods cannot be applied. Therefore, only agile or highly manoeuvrable controllers for multirotors are investigated.

Huang et al. investigated the aerodynamics and control of quadcopters during aggressive manoeuvres rather than just when in hover [59]. They argue that many research groups, including many in the previous work examined in this document, only use control dynamics that have been simplified for hover control, neglecting the dynamics of higher-speed flight. The first aerodynamic effect discussed is blade flapping, which impacts attitude control. The effect tilts the effective rotor plane, which will create a moment force about the aircraft's centre of gravity. The second effect discussed is the thrust variation experienced in

forward flight that can reduce the upward thrust produced and therefore impact altitude control. These dynamics were then added to the PID controller for the UAS investigation. After performing a stall turn test, it was seen that the addition of these dynamics results in a controller that performs much better at higher flight speeds and angles.

Using basic Newtonian relationships, Tahir et al. developed a state space model of a quadrotor to help understand the dynamics [116]. First, a 3 DOF model was made, which examined the attitude (angles and angular accelerations) of the aircraft, dependent only on the thrust from each of the four motors as the input. For the 6 DOF model, both the attitude, and position and velocity in space were modelled as the states. In this case, the four inputs were changed to simplify the model. These inputs were the total force in the z-direction of the quadcopter, the pitch torque, the yaw torque, and the roll torque. Tahir et al. then furthered their work by developing an optimal control system for their previous quadcopter that utilized the 6 DOF state space model [115]. The aircraft was stabilized to desired positions and attitudes using a LQR, which is a form of feedback regulation. The authors observed that by changing the weights in the Q and R matrices, the quadcopter can become more or less agile, resulting in a decrease or increase in endurance, respectively. Next, a linear - quadratic - Gaussian (LQG) controller was implemented that utilizes a Kalman filter to reduce noise because the use of a LQR controller in a noisy environment is impractical.

Tengis and Batmunkh also investigated creating a state space representation of quadcopters based on the equations that describe their dynamics [124]. The state space model was developed using kinematics, dynamics from Newton-Euler and Euler-Lagrange equations, and the dynamics of the inertial frame from the Coriolis equation. The model was controlled using full state feedback and was simulated in MATLAB. The feedback gain matrix was defined using pole placement such that the settling time was around 2 seconds, and the overshoot was limited to 4 percent. Using MATLAB, the closed-loop control method was simulated, showing that the simplified state-space model had realistic dynamics.

Srinivasan has also investigated the modelling and control of a quadcopter, with an effort to measure specific parameters of the aircraft [107]. The authors related the battery voltage and PWM signal of the ESC to the rotational speed of the rotor, measured the thrust and torque at different rotational velocities, and measured the moments of inertia. Using this data, the rigid body dynamics of the aircraft were represented with a state space model. The model uses total thrust and torque in the yaw, pitch, and roll directions as inputs. The quadcopter was then controlled using a feedback system and a cascade scheme, where the angular rate is controlled by the inner loop, and the angle is controlled by the outer loop. The position was controlled by creating a linear state space model for the system and using feedback with the LQR optimization method.

The work presented by Abbasi et al. also improved upon the use of PID controllers for multirotor attitude control with the addition of fuzzy logic to adapt the gains [1]. The

authors used MATLAB to develop their FLC to help the aircraft reach the hover state more quickly while remaining stable. The controller achieved this by using larger proportional and integral gains when the measured error was large, but reducing these gains when the error was small. The opposite was applied to the derivative gains to help add damping to the system when it was stable.

E.1.2 Quaternion modelling and control

A basic mission requirement for a tailsitting aircraft is to rotate 90 degrees around the pitch axis to transition from VTOL to forward flight, which can result in gimbal lock if using the standard Euler angle representation. One solution to this problem is to represent the aircraft attitude using Quaternions [82, 86].

Fresk and Nikolakopoulos have created a flight controller for quadcopter attitude that is based fully on Quaternions [47]. The authors start by explaining quaternion math as it relates to quadcopter control. Quaternions are represented with a scalar, q_0, and three vectors, q_1, q_2, and q_3, that are commonly arranged in the following two ways:

$$\mathbf{q} = q_0 + q_1 i + q_2 j + q_3 k \tag{E.1}$$

$$\mathbf{q} = [q_0 \ q_1 \ q_2 \ q_3]^T \tag{E.2}$$

The product of two quaternions by using the Kronecker product is explained, noting that this is not commutative like rotations. A method to represent Euler angles as quaternions, and vice versa, is presented to help represent the dynamics of a quadcopter system that utilizes quaternions. The modelling process using quaternions is then discussed. It should be noted that this is a relatively simple model that does not consider some of the dynamic effects previously mentioned, like blade flapping and thrust variation. The model was derived using the Euler-Newton equations for the dynamics of a rigid body, and then using the right-hand quaternion derivative to get the final representation. For the controller, a P^2 feedback loop was used that did not convert between quaternions and Euler angles. After testing, the controller showed its ability to manoeuvre the aircraft through a flip, which could cause gimbal lock for a standard method.

Zhao et al. used quaternions for trajectory and yaw control of a quadcopter which utilized a command filtered backstepping technique [146]. This report points out how the linearization of multirotors at hover results in good control near this point, but worse performance as the vehicle gets further from this orientation. The description of a unit quaternion is presented, noting that any attitude can be described by two quaternions, q and $-q$. The values that represent the quaternion are the same, but the rotating direction about the rotating axis changes. The authors also describe a method to express the angular velocity at a body frame with respect to an inertial frame. For the controller, the authors first describe the tracking error dynamics of the position and velocity control separately, then the corresponding Lyapunov functions and their time derivatives. The vehicle attitude

and angular velocity control are then defined to be used in the backstepping controller. When the yaw is also being considered in tracking, the analytical derivation is more difficult. The backstepping controller with a command filter was then simulated to test its trajectory controllability. It was found that this implementation was capable of following the desired trajectory well.

Cariño et al. also designed a quaternion-based control scheme that focused on the position, or global control of a quadrotor [29]. The kinematic quaternion model was also derived from Newton-Euler equations. The attitude and position control algorithms are then explained, showing that they are both linear and exponentially stable. The system was then simulated in Python, which was also used to create a LQR controller. The simulation results show that the model will exponentially converge to a given orientation. In addition, tests were conducted to show that the aircraft can be controlled to converge from an initial position to a different desired position. These control strategies were then implemented on a quadcopter and tested in an outdoor environment. Even when experiencing perturbations such as wind, the controller was still able to control the attitude and position to the desired references.

E.2 Tailsitter Control Review

This section investigates the relevant control strategies implemented for tailsitting vehicles, separated into three primary categories. The first being controllers that can be implemented for a wide range of tailsitting vehicles, while the second focuses on aircraft with control surfaces. The third section discusses the methodologies that are specifically applied to copter tailsitters, which is the same aircraft type as the LaBWing. Therefore, the solutions presented in this section are used to compare the strengths and weaknesses, along with illustrating where current literature gaps can be filled.

E.2.1 Standard Tailsitter Modelling and Control

Bapst et al. presented a flying wing tailsitter with two rotors that used a single controller for all flight modes instead of blending between different conditions or using gain scaling [20]. The vehicle dynamics are expressed, noting that they are only first-order approximations and do not fully represent the complex aerodynamic effects of the aircraft. The rotational moment is also shown, along with the time derivative of the quaternion, the quaternion rate matrix, and the rotation matrix from the body to the Earth frame. The authors also presented an estimation for the airflow after being propelled by the rotors, which is a method obtained from [72]. Using this method, the airspeed after the rotor was then mapped based on the airspeed coming into the propeller and the input power. For the flight controller, a cascaded approach was used in which the position and velocity errors are used to calculate a desired acceleration that is combined with a desired yaw angle to define a new desired attitude and thrust. This is a common approach for multirotors and

was expanded upon to add the forward flight dynamics of a tailsitter, such as lift and drag. Using quaternion multiplication, the desired angle for the yaw and acceleration axis can be computed, where the product is the desired attitude. The control theory was then simulated and applied to a test vehicle, with a 25-degree angular error and a 0.5 m position error.

Verling et al. discussed the challenges experienced when designing a convertible tailsitter and its controller [135]. The vehicle is a fixed-wing style aircraft with 2 control surfaces and two rotors positioned in front of the wing. The airflow of the propellers is used to ensure enough airflow over the wings occurs when in the hover orientation to maintain the authority from the elevons (combined elevator and aileron). The avionics platform used on this aircraft was the Pixhawk autopilot, including its EKF for attitude estimation. To model the aircraft, the authors conducted wind tunnel tests to estimate the aerodynamics. Data was collected for the forces and moments created by the propeller and wing interaction when changing the angle of attack, elevon deflection, motor speed, and inlet air speed. Using this data, the aircraft can be modelled aerodynamically and then represented as a rigid body. To avoid singularities, the model was developed using a coordinate-free representation, known as the SO(3). The authors designed an error function for the controller based on monotonic error, the ability to work in all orientations, and independence of heading. The deflection of the elevons and RPM of the rotors was calculated using a state-based control allocation that accounted for the states used for the wind tunnel tests. The controller was then implemented on the Pixhawk system for outdoor testing. The implemented controller was able to track the transition between modes with a maximum error of 25 degrees.

Kuang et al. have created a model and flight controller specific for a unique style of tailsitter which utilizes thrust vectoring, a stand for VTOL, and controllable, high-angle of attack, forward landings [61]. The authors note that quaternion feedback was a common method to solve the singularities experienced by tailsitters using Euler angles, but these can have problems with large heading errors. To fix this, Euler angles that switched between horizontal and vertical representations for the different flight modes were used, noting they are intuitive to understand and computationally light. The aircraft used for this design is a similar shape to a fighter jet, but has two vectoring rotors at the end of both wings. The model of the aircraft is then described, which incorporates the weight, vectored thrust, and aerodynamics of the vehicle. The flight controller for this design avoids singularity through switching between horizontal and vertical Euler angle representations based on flight mode. To avoid oscillation when switching, a hysteresis method is used, so the switching points have an overlap. The attitude controller utilizes a linear or constant acceleration approximation to limit values like targets for pitch and roll, reducing the loads on servos, getting faster responses, and preventing overshooting. In addition, the angular velocity is calculated to be used in a feed-forward loop before calculating the target angles. For the altitude control, the authors decided to have a slower response for throttle when in forward flight to avoid oscillation, but need faster responses in the VTOL orientation to adequately stabilize the inverted pendulum of the aircraft. The overall structure of the controller is a feedback loop with P and PID loops. The controller was tested on a phys-

ical aircraft, and results show that there was roll oscillation due to rapid angle of attack changes, giving an angular error of 10 degrees, and consistently more lift was produced than expected, resulting in an altitude error of up to 3 m.

Chiappinelli and Nahon made a flight controller for an X-VERT VTOL from Horizon Hobby which is a flying wing with 2 rotors and 2 control surfaces [32]. The controller was capable of flying the aircraft using a single flight mode. The authors developed a quaternion-based controller which does not have a preferred operating condition and can perform the large attitude changes needed for tailsitters. The flight controller was inspired by an aerobatic fixed-wing aircraft controller, shown here [27]. The authors tested this aircraft and controller in Simulink using a 200 Hz Real Time Sync block. The equations of motion are also described for the simulation, along with the Dryden turbulence model. The thrusters were modelled with an ESC and motor that creates a rotational speed based on the throttle request and current battery voltage. Based on this and the current air velocity, the thrust and torque of the propeller are computed. The advance ratio is also calculated, which is used to find the thrust and power coefficients based on data from the University of Illinois Urbana-Champaign (UIUC) database [25]. The wings were modelled as flat plate segments, each with its airspeed based on thrusters, and therefore have different lift, drag, and pitching moments. The airspeed for the wing segments behind the rotors was calculated using momentum theory. A simplified model was also needed for the flight controller that also represents the flight dynamics. It predicts the moment forces on the body and forward thrust based on the motor thrust and elevon deflections. The controller used was a cascaded PID regulator with separate sections for the position, attitude, trust, and the mixer. The system and controller were simulated in Simulink, resulting in attitude errors over 20 degrees and position errors of 5 m when in windy environments.

Zhaoying Li et al. developed a non-linear controller using dynamic inversion and state feedback to ensure stability [65]. The aircraft in this work is a design with 4 rotors that are not aligned horizontally when in the hover orientation. Each rotor is connected to a wing segment or part of a V-tail with a control surface that is not level to the ground when in forward flight. The mathematical model for this aircraft describes the velocity, the acceleration from the rotor thrust, and the angular velocity from their torque. The forces from the rotors, wings and disturbances are all summed and rotated, to be expressed using roll, pitch, and yaw directions. Then the desired angles are converted into quaternions. In addition, the total torque is calculated from the rotors, control surfaces, and disturbances, to be used to find the moments of the aircraft. Using these calculations, a NDI controller was implemented. The controller finds the tracking error of the angular velocity using quaternions. Then, using time derivatives and the dynamic inversion technique, the rotor thrust is calculated. A linear error representation of the system using state space is also deduced from this technique to account for all disturbances and uncertainties. Using eigenstructure assignment, the disturbances are regulated with state feedback. Using this linear model of the errors, a transfer function can be extracted and compensated for in a robust controller. A simulation for the aircraft and controller was then conducted, which showed a

good ability to manage disturbances and transition the aircraft from hover to forward flight.

Liu et al. presented an H_∞ controller for a tailsitter with an 'X' arranged quadrotor attached through 4 wings with control surfaces to a large fuselage that contains a rotor at the tip and a main wing with two control surfaces [66]. The authors describe the dynamic model of the aircraft, explaining how the angle of attack and relationships for acceleration and angular acceleration were obtained. The forces and moments experienced by the aircraft in the model include the thrust from the rotors, coaxial counter-rotating propellers, aerodynamic forces on the wings and fuselage, and disturbances. The attitude for the model was described using quaternions to avoid the singularity seen with Euler angles. In addition, the model also accounted for the dynamic parametric uncertainties of the vehicle by adding a parameter uncertainty matrix to the attitude model. Using quaternion tracking error and attitude tracking errors, the attitude error for the tailsitter was derived. With the aircraft model equations, a description for the required torques and forces was shown that was used to calculate the required rotor speeds and aileron deflections. The adaptive controller designed contained two main sections. The first is the H_∞ state feedback controller for tracking performance and the second is an adaptive controller to limit the impact brought on by disturbances. The entire closed-loop system contained separate controllers for roll, pitch, and yaw. Each of these controllers had independent H_∞ and adaptive controllers. The authors note that the complexity of the controller designed is comparable to a PID controller, making it very practical to use in real-life situations. The controller was proven to be robust through solving a lemma, and showed that the attitude tracking error is bounded. The system was simulated to present the robustness by including external disturbances that could be corrected for. The controller implemented had a single flight mode and only used a one coordinate system.

Tal and Karaman developed a global controller for agile trajectory tracking of a flying wing tailsitter with 2 rotors and 2 control surfaces [117]. The flight controller utilizes an incremental control action, so it does not need a globally accurate dynamics model. Instead, the controller uses the flight model to predict the changes in linear and angular acceleration. The reference frame, Newton-Euler equations of motion, forces, and moments are all described to create the flight model, utilizing quaternions for attitude representation. A differential flatness model of the flight dynamics was then created to obtain the linear and angular acceleration control algorithm from the attitude and control inputs. In addition, the trajectory controller uses the angular velocity expression derived to obtain a feedforward signal that depends on the jerk and yaw rates. The trajectory tracking controller designed used a PD controller for the position, velocity, attitude, and angular rates, while an INDI is used to control the acceleration command based on incremental updates of the attitude and total thrust. Although not all aerodynamic properties of the aircraft are modelled, the controller shows a good ability to track position, limiting the position and angular errors to under 0.3 m and 15 degrees respectively, through a variety of tests in an ideal environment.

The work presented by Zhang et al. involved a tailsitter with a single wing, two rotors, and two control surfaces [144]. The authors used a dual fuzzy logic PID controller to control the aircraft with a single flight mode, attempting to reduce oscillations and improve performance when compared to a PID controller. One of the fuzzy controllers was used to set the universal gains based on airspeed and attitude error. The second was used to independently tune the proportional, integral, and derivative gains based on the error and error derivative. The results presented show that the oscillation experienced in forward flight with the PID controller can be reduced by applying fuzzy logic to adjust the gains. Additionally, the use of fuzzy logic has made the aircraft more responsive without introducing large overshoot. The controller was tested in both simulated and real environments.

E.2.2 Quadcopter tailsitter modelling and control

The Quadshot is a tailsitter with 4 rotors surrounding and a single wing, presented by Sinha et al [101]. The control for this aircraft is accomplished through differential thrust in the rotors plus control surfaces along the wing. A separate controller was implemented for hover, forward flight, and an acrobatic flight mode where the pilot's input on a transmitter are used to determine the desired angular rates. To eliminate the singularity problem with Euler angles in hover mode, a unique attitude representation was used that combined quaternions for heading and Euler angles for roll and pitch. Each of the controllers used the same PID control strategy, but had different gains. The transition was completed by utilizing the forward flight mode with a linear change in the desired pitch angle. A transition test showed a pitch error of up to 35 degrees, with maximum errors of 10 degrees for roll and yaw.

The Quadshot was also worked on by Flores et al. who utlized LSF for control [45]. The controller is desirable as it combines all flight modes into one such that there is no switching, and considers the limitations of the actuators. However, this controller was only implemented on the transition process in 2D simulations. The controller uses time-scale separation to track the linear and angular acceleration dynamics separately, where the desired aircraft pitch is the primary result. It should be noted that the closed loop system with this controller is asymptotically stable using the Lyapunov technique. Simulation results using MATLAB shows that the controller is able to manoeuvre the aircraft as desired and account for initial condition errors. The maximum errors in simulation are small, appearing to be 4 degrees in the pitch axis and a 0.01 m/s velocity error in any direction.

The VertiKUL UAS by Hochstenbach et al. was one of the first tailsitters that used only rotors and no control surfaces for all thrust and control in all flight modes [58]. It was designed for use in parcel delivery using a numerical method to optimize the range for up to a 1 kg payload. The control for the VertiKUL was implemented using ArduPilot as described in Section 4.1 and 4.2, using the generalized P+PID controllers [3]. The aircraft was tested in all flight modes and was shown to have only reactive control due to the use

of PID controllers, which resulted in significant altitude and pitch errors.

Wang et al. aimed to develop a fully autonomous copter tailsitter with a single wing which had twice the efficiency of a similar quadcopter [136]. The authors describe the sizing process for the wings, fuselage, and propulsion system. The vibrations of the rotors, aerodynamic disturbances, and their impact on the wings are described and modelled. Using linear and angular momentum theorems, the authors described the linear and angular velocity and accelerations which are used to find the moments and forces of the aircraft. The aerodynamics for the aircraft are then described and investigated using aerodynamics analysis, simulation, and wind tunnel tests. Similarly, the thrust and torque of the motors were tested and plotted based on the advance ratio, airspeed, and PWM signal. A PID controller was implemented for both the inner and outer loops for position and attitude control, respectively. The simulation results show there were significant errors, oscillations, and delays between the desired and measured positions and attitudes. The authors note that the aircraft can fly with half the current at almost 1.5 times the speed for almost 1.5 times longer than a similarly sized quadcopter.

Zhang et al. developed a flight controller for a quadrotor tailsitting UAS with no control surfaces [145]. Rotor performance at high ground speed was modelled since forward flight is an important aspect of tailsitters, and dynamic thrust reduces the force produced. The attitude was represented using quaternions, which utilize 4 variables instead of the typical 3 with Euler Angles to avoid singularity problems and gimbal lock [86]. A model was created to represent the kinematics, dynamics, aerodynamics, and actuator dynamics of the vehicle. The attitude was controlled using a PD controller, while the altitude was controlled with a PID controller. However, it exhibited a throttle drop at the beginning of the transition, which resulted in an altitude loss.

An alternative X-frame quadrotor tailsitter with a single wing was developed by Lyu et al [75]. There are two elevons in addition to the four rotors to increase the moment in the pitch axis. The aircraft had three separate control modes being hover, forward flight, and the transition, and the avionics were based on the Pixhawk system [42]. The altitude through the transition was regulated using a PID controller with feedforward from the aerodynamic forces of the wings. The attitude was controlled with a cascaded structure with a PID controller for the angular velocity, and a P controller for the angle. A "Quaternion linear" method was used for the attitude error decomposition to avoid singularities while achieving linear error rates. A mixer was utilized to compute the required thrust for each motor and deflection of the two elevons based on the desired thrust and moments. The controller was tested on the aircraft and was able to track the desired attitude and altitude with errors of 20 degrees and 1.5 meters, respectively. In addition, oscillations were observed with and without the use of elevons.

The same aircraft was also investigated by Zhou et al., who developed a unified control method so only one controller was required for all flight modes [148]. After describing

very similar flight dynamics, the authors then conducted wind tunnel tests to determine the lift, drag, and side force coefficients of the flying wing based on angle of attack and slideslip angle. The control structure in this literature was similar to that of quadcopters since the motor actuation is the same. A dual-loop approach was used, with the inner loop for attitude control and the outer loop for position control. The desired attitude was tracked using a P controller, and the angular error was corrected using a PID controller. The desired trajectory was tracked using the calculated position error, followed by a PID controller with feedforward to calculate the desired acceleration. Since tailsitters have significant aerodynamic forces in forward flight that should be considered, the altitude and thrust were not solved analytically. Instead, an objective function was solved as an optimization problem, along with the use of anti-windup for the PID integrator. The controller was then tested in a simulation to show that it can be used for all three flight modes without major attitude or position errors. The controller was then tested on the aircraft, however, the position loop could not be implemented due to the inability to run the controller calculation within the refresh rate. The attitude controllers showed a good ability to track throughout the flight regime, but had overshoot and significant errors up to 15 degrees along with delays.

A quadrotor based tailsitter with a single flying wing body was designed by Lyu et al [73]. The aircraft is an example of adding a small angle to the motor mounts to increase the roll moments in hover. The authors describe the kinematics of the aircraft, with the attitude represented using Z-X-Y Euler angles. Assuming a rigid body and using the Newton-Euler equation, the transitional and rotational dynamics were described. The thrust and torque of the rotors were described using the method initially presented by Brandt and Selig [24], which was adopted and expanded here [25] to show and quantify test results of many UAS propellers. The thrust and torque coefficients used in these equations were obtained using a wind tunnel test with the rotor mounted to a force sensor. The aerodynamic forces and moments were also described using the coefficients that were all obtained through wind tunnel tests. The controller the authors designed for this aircraft made use of an attitude and a transition controller. The attitude controller was a cascaded design that was comprised of a P controller for the angles, and a PID controller for the angular velocity. The transition controller was used to attempt to maintain a constant altitude throughout the transition for accurate and stable flight. A mixer was then used to calculate the thrust required for each rotor. Using the model developed, a simulation was produced to test the controller. The results show that the aircraft can transition to forward flight without much error, but the transition back to hover causes a 2 m drop in altitude. Tests were also conducted on a real vehicle with similar results, however, errors up to 5 degrees were seen in the roll axis, along with worse altitude control. The flight envelope of the aircraft was then described, showing the power required and resulting speed throughout the angles of attack required for a tailsitter.

Using a velocity DOB, Lyu et al. furthered the work on their previous controller [74]. The new controller aims to estimate and compensate for any wind disturbances by utilizing

C optimal control theory. Note that this was only being applied to the aircraft in hover mode. The DOB uses the output of the plant and subtracts the control signal to estimate the disturbance of the system. This estimation was then filtered through a low pass filter (LPF) and can then compensate for the disturbance. The authors showed that this system can be described using a simple transfer function. The authors added disturbance terms to each direction of the aerodynamic models, which were then linearized such that a frequency domain model could be created. Using frequency response data in the simulated model, the authors were able to obtain a fitted transfer function for the DOB. However, this transfer function was a non-minimum phase system because of an unstable zero. To solve this problem, an H_∞ based DOB controller was designed, which aims to minimize the H_∞ norm of the weighted transfer function to find an optimized feedback controller. The controller was then implemented on an aircraft, and compared to a DOB designed without H_∞. The results show that the controller improves the aircraft position accuracy by 2.5 to 4.5 times, depending on wind conditions, reducing the impact of wind disturbance on the system in hover.

The same aircraft was also worked on by Zhou et al. but in this case, a controller in the frequency domain was implemented [147]. The model for this work uses the same theory as previously described for this aircraft in [73] and [74]. However, the motor acceleration and gyroscopic effects were not considered since they have relatively small values relative to the rest of the dynamics. While the aircraft has a separate controller for the inner and outer loops, only the inner loop was discussed in this report, corresponding to the attitude. A P controller in the outer loop was used for calculating the desired attitude based on the attitude error. Quaternions were used in this case to remove the singularity problem that would cause local instability when the logarithmic function becomes singular. A sweepsine input was added to the controller to get data used for creating a full spectrum model. Plotting the frequency responses shows that two main sections describe the aircraft. The first section at lower frequencies represents the main aircraft dynamics, while the higher frequencies represents the dynamic modes of the aircraft. The spectrum model was then used to create transfer functions to represent the aircraft dynamics. The controller was then designed using a loop shaping technique which utilized integral, lead, and rolloff transfer functions. The aircraft was then tested using this controller in hover mode, as well as throughout the transition in an outdoor test. The controller showed a good overall ability to track the desired attitude. However, since the controller only has a feedback component for correction, it was only responsive shown by the delay between the desired and actual attitudes.

Li et al. discussed the modelling and control of a plus-oriented copter tailsitter which uses a MPC to manage position [64]. MPC attempts to take the dynamics of the aircraft to predict performance based on the system states. The aircraft for this work was a flying wing with 2 control surfaces that was attached to a quadcopter to achieve the forward flight and VTOL capabilities of tailsitters. The aerodynamic model for this aircraft considered the rotorwash, which impacted sections of the wing. This was incorporated into

the model by calculated the forces and moments on each section of the wing, which were divided based on airflow. To determine the performance of the aircraft at large angles of attack (greater than 20 degrees), a closed-wall wind tunnel experiment was conducted on a scaled wing segment. The results were then compiled in a lookup table for the controller to use. The thrust of the rotor was also tested experimentally and plotted such that the thrust and torque could be predicted based on the throttle. For the MPC developed in this work, only the position was managed, while the attitude and actuator mixer are controlled by a Pixhawk [42]. The prediction model assumes the form of a discrete-time state space model to be used for prediction and control. The disturbances that can be measured are used for prediction through feed-forward signals, allowing for faster responses. This model considers crosswind as a disturbance and the force it will cause when the vehicle is in hover. Unmeasured disturbances can also be used to reject unknown disturbances if steady-state errors are observed. Assuming that the aircraft has little deviation when in hover flight, the state space model was linearized. For an MPC, an objective function is required that sums the squares of the weighted state errors and control input changes. A hardware-in-the-loop simulation was conducted to prove the controller's operation. The simulation shows good ability to track position as desired when the vehicle is limited to 30-degree rotations in roll and pitch. The controller was also tested on a physical vehicle and showed a good ability to reject disturbances to the system, such as crosswind.

The modelling and control of another quadrotor tailsitter with no control surfaces but large mounting angles for the rotors was developed by Xu et al [141]. The authors note the benefits of using only rotors for control rather than a combination of rotors and control surfaces, commenting on weight and bandwidth savings, along with the removal of nonlinear relationships. The body frame for this work is represented using Z-X-Y Euler angles, but quaternions are used in the attitude controller. Newton's equation of motion, along with the aircraft mass and inertial matrix, was used to create a rotational and vertical dynamic model. In addition, the aerodynamic and rotor forces are described based on lift, drag, thrust, and the aircraft's orientation. The controller for this aircraft was split into an attitude controller and an altitude controller that had a feedforward component from the attitude. The authors designed the angular velocity portion of the attitude controller using loop shaping. This is accomplished by adding a sweep input to the system, then measuring the output afterward. A spectral analysis is then used to find the gain and phase delay of the system. Using these results, a fitted model is used to represent the system. A notch filter was also implemented to decrease the impact of modes caused by motor vibrations. In this case, the loop shaping controller was designed using a PID compensator. For the altitude controller, first, the aerodynamics of the lift and drag were described. The controller used is a feedforward-feedback parallel structure. The authors then tested the notch filter in a hover test, showing its ability to stabilize the system. Transition tests were then conducted on the aircraft, showing an overall ability to track the desired attitudes, but a 2 m altitude error was observed.

The same airframe was also worked on by Xu and Zhang, who created a feedback ILC

to regulate the position error for the Pugachev's Cobra manoeuvre [142]. The authors assumed that the aircraft was a rigid body and showed the aircraft's transnational aerodynamics using the acceleration caused by gravity, the rotors, and the aerodynamic forces. In addition, the specific acceleration was measurable using the onboard Pixhawk system [42]. Next, the authors explain how the accelerations for the Z and Y directions are obtained, noting that the dynamics are linear to the control actions and are not dependent on complex model parameters. The authors then describe the controller structure in detail, which consisted of altitude, lateral, attitude, and X-acceleration controllers. The altitude controller uses feedforward compensation from the input and PD feedback stabilization for the error. The lateral controller was comprised of a P controller segment. The attitude and X-acceleration controllers are described in the previous work on this aircraft, discussed above [141]. The authors note that the previous controller was unable to track the altitude very well when subject to disturbances, and therefore decided to implement the iterative learning strategy. To add the ILC, first a lifted domain model was created which adds the learning correction to the attitude controller. The continuous disturbance of the system, which represents the unmodelled aircraft dynamics, is then estimated in an iteration domain Kalman filter. The controller was then tested on the aircraft by performing the Pugachev's Cobra manoeuvre. The aircraft was able to track the desired pitch well, though with noticeable delay since the controller is primarily feedback-based. In addition, the lateral error through the manoeuvre reaches almost 0.5 m. However, the controller does maintains a constant altitude throughout the manoeuvre after 4 to 5 iterations, limiting the error to 0.23 m.

Reddinger et al. created a flight controller for a quadrotor biplane tailsitter, which was also very similar to the structure of the LaBWing [97]. A scheduled LDI controller was used, which required the linearization of the vehicle dynamics, represented by a state space model. A wind frame representation of the state space model was used, which allowed for control of the aircraft during the transition. However, the model was not developed for hover or forward flight, and the planned transition path had a large altitude variation, differing from the stable flight desired for the LaBWing.

Differential flatness was utilized by McIntosh et al. to reduce the computational load required for trajectory planning for a QBiTs transition [77]. The purpose of the trajectory planner was to find a path from an initial to a final condition based on the dynamics of the aircraft and any constraints. The differential flatness comes from assuming that the out-of-plane (not in the direction of transition) variables are constant. The model of the aircraft was defined by previous work, shown here [109], which was based on a NDI approach. However, for computational efficiency, a simplified model was utilized for the trajectory planning, accounting only for the longitudinal motion. It was assumed that the other motions would be stabilized by the inner (attitude) control loop. The simplified model was then converted to a linear representation, being a differentially flat model. For this model, the inertial position was set as flat outputs, and the state variables were then related to these values. To track trajectory, two cascaded NDI control architectures were

utilized. The first one was for the hover mode, and the second for forward flight. These two controllers were always running and were selected based on the pitch angle of the aircraft. In addition, a blender was used to combine the control inputs of both controllers as the transition proceeded. The controller was first tested on transitions from hover to forward flight. The tests show that the aircraft could track vertical position well, but this involved a desired 27-foot increase in altitude, and could not maintain the desired horizontal speed. The attitude tracking for this controller showed some significant errors in the pitch angle.

A quadcopter-based tailsitter with an annular outer wing, along with its aerodynamics and control was designed and analyzed by Gill and D'Andrea [49]. To model the aircraft, the dynamic equations were shown, based on the vehicle rotational velocities, propellers, gravity, airspeed, moments created by motors, inertia tensors, and motor relationships. The forces and moments describing the unique annular wing were also presented, which were calculated using the corresponding coefficients. These coefficients were measured through a wind tunnel test and were parameterized using piece-wise functions. The modelling of the rotors for this aircraft was discussed in detail in their previous work, shown here [50]. The controller used a cascaded scheme with many components. The first was a position controller that used a PID loop to implement the desired acceleration of the frame. The orientation of this acceleration frame was controlled with an outer control allocation block. This block was used to find the required attitude and total thrust of all the rotors. Feedforward body rates were given to the attitude controller using the same approach as the outer control allocation block, but used the reference acceleration instead of the desired acceleration. The attitude controller, which considered the tilt and twist errors, used a PD controller. The inner control allocation found the required rotational speeds of the rotors based on the commanded values for thrust and torque found in the previous sections. To solve this problem, an iterative algorithm was used, and the results were shown to converge very quickly. This calculation was initialized using what the results would be assuming hover conditions. The motor controller utilizes a proportional – integral (PI) controller for feedback and feedforward inputs to calculate the required voltage for each motor. The controller was implemented on the aircraft and tested in a straight line manoeuvre and a circular flying manoeuvre, both in outdoor environments with wind gusts up to 5 m/s. The authors note that the aircraft showed a good ability to track the desired position. There was some altitude and horizontal error observed through the transitions of approximately 1-2 m. For the circular test, the position and speed were tracked very well, with small discrepancies seen in altitude throughout the test.

An alternative method to an iterative learning feedforward controller was presented by Raf et al [93]. This controller was for a QBiT which attempts to perfect the transition while only using a nominal aerodynamic model. The authors note that the modelling errors were compensated for better with each trial, and that the controller was computationally simple, so it can be used on real aircraft with ease. For the aerodynamic model, the lift was the sum of the wing sections which do and do not have an interaction with the propeller downwash. The drag for this model was also calculated in the same way. The lift and

drag coefficients for these calculations were based on simple models rather than complex wind tunnel measurements. However, wind tunnel tests were used to parameterize the propeller performance as the thrust produced reduces significantly as the inlet airspeed increases. The data obtained from these tests was arranged using a neural network, which was later used by the iterative learning algorithm for the feedforward thrust demanded. For the iterative learning controller, the feedforward trajectories for the pitch and thrust were given fourth-order polynomials with scalars to be determined. A terminal error cost function was then defined to optimize the fourth-order polynomials for the transition. The polynomials were then updated in each iteration of the test to minimize the terminal error, and therefore improve the transition. A previously designed lateral controller using PD feedback logic was used to maintain the altitude through feedforward data for the pitch and thrust. The authors also presented a robust controller for attitude, which was used to track the desired pitch, yaw, and roll angles. Z-X-Y Euler angles were used to prevent singularities by moving the singularity from the pitch to the roll axis. A cascaded approach was used with the angular rate being in the inner loop and the attitude being on the outer loop. The controllers were then implemented on the aircraft to test the performance in transition tests. Multiple tests were conducted so the iterative learning controller could improve, and satisfactory results were obtained after 5 trials. The transition resulted in an altitude error of up to 0.5 m, which is a respectable result considering the simplicity of the model.

A unique quadcopter tailsitter with an X-wing arrangement that is angled 45 degrees off of level ground was investigated by Xin et al [140]. Three separate controllers were used to account for all three flight modes being hover, forward flight, and the transition. Dual Euler angles were used for attitude representation, where one set was aligned with hover and the second aligned with forward flight. Halfway through the transition, the attitude representation would swap to the appropriate frame to avoid Euler angle singularities. The aircraft dynamics were modelled using the common Newton-Euler equations for aircraft. However, the aerodynamic forces and moments were obtained using a set of equations defined in [22] and coefficients solved using CFD. To control the aircraft, NDI was utilized, which attempts to linearize the nonlinear dynamics of the aircraft. The controller focused on regulating the angles and angular rates, and was both simulated and implemented on a test aircraft. The resulting altitude errors were up to 10 m, and angular errors reached 35 degrees.

Dalwadi et al. investigated three different types of trajectory tracking (attitude and position) flight control schemes for QBiTs [35]. These methods included BSC, ITSMC, and a hybrid of both methods. The authors presented the mathematical model of the aircraft, focusing on the derivation of linear and angular accelerations, along with the aerodynamic moments and forces. The authors then described the design of the hybrid controller for hover mode. In this mode, the thrust should counter the aircraft's weight, while aerodynamic forces and moments are near zero, and the angular velocity is also close to zero. The attitude controller was a BSC, while the position controller was a ITSMC. For the

ITSMC, the sliding surface and reaching law were first defined. The desired pitch and roll angles were then expressed based on the attitude error, and within limits used to avoid singularities. These angles were fed into the BSC for attitude control. The authors defined the control laws for all three axes and noted that the system is asymptotically stable. The transition mode controller was then described, noting that it has essentially the same dynamics as in hover, but with the addition of the aerodynamic forces and moments. For the forward flight mode, the dynamics are similar to fixed-wing aircraft, with the variables defined for the QBiT before being transformed to the forward flight axis. The same types of controllers are used as explained for the hover modes, and they are defined using similar techniques. The controller was tested in a simulation using MATLAB and Simulink to investigate the trajectory tracking ability. The test involved takeoff, transition to forward flight, forward flight with varying trajectories and speeds, transition to hover, and a landing. The tests showed that the hybrid controller performs much better than just a BSC or ITSMC. However, in all tests, there was still significant error and overshoot observed between the desired and measured attitude and position, though the desired inputs were rapidly changed.