5-2024

Development, Modeling, Identification, and Control of Tilt-Rotor eVTOL Aircraft

Clayton T. Spencer
Utah State University, clayton.spencer@usu.edu

DEVELOPMENT, MODELING, IDENTIFICATION, AND CONTROL OF TILT-ROTOR EVTOL AIRCRAFT

by

Clayton T. Spencer

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Aerospace Engineering

Approved:

Tianyi He, Ph.D. Major Professor	Douglas Hunsaker, Ph.D. Committee Member
Thomas Fronk, Ph.D. Committee Member	D. Richard Cutler, Ph.D. Vice Provost of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2024

ABSTRACT

Development, Modeling, Identification, and Control of Tilt-Rotor eVTOL Aircraft

by

Clayton T. Spencer, Master of Science

Utah State University, 2024

Major Professor: Tianyi He, Ph.D.
Department: Mechanical and Aerospace Engineering

This thesis includes the development, modeling, identification, and control of an electric-Vertical-Take-Off-and-Landing (eVTOL) aircraft with tiltable rotors. The front two rotors have tilting capability for transition flight from vertical-take-off to forward-level flight. In the design and analysis of an eVTOL aircraft platform, we study and characterize the layout and system architecture, select system components such as electric motors, batteries, and controllers, and integrate these components into the overall aircraft system. After that, a Six-Degree-of-Freedom (6-DoF) dynamical model is derived as a nonlinear equation and is further reduced to longitudinal motion. Afterwards, a Least-Squares-regression (LSR) method that can handle parameter constraints is developed to identify unknown system parameters using the real flight data. Lastly, this study performs an optimization of the Proportional Integral Derivative (PID) gains based on the identified model parameters using MATLAB.

(144 pages)

PUBLIC ABSTRACT

Development, Modeling, Identification, and Control of Tilt-Rotor eVTOL Aircraft

Clayton T. Spencer

This thesis includes the development, modeling, identification, and control of an electric-Vertical-Take-Off-and-Landing (eVTOL) aircraft with tiltable rotors. The front two rotors have tilting capability for transition flight from vertical-take-off to forward-level flight. This work details the development of an eVTOL aircraft and the selection of sub components such as electric motors, batteries, and controllers. After the aircraft build, mathematical model is derived to describe the motion of the aircraft. Unknown parameters in the mathematical model are identified using a Least-Squares-regression (LSR) method that can handle parameter constraints. This is done using real flight data collected from the aircraft. Lastly, this study performs an optimization of the Proportional Integral Derivative (PID) gains based on the identified model parameters using MATLAB.

To my family, past, present, and future.

ACKNOWLEDGMENTS

Before all, I would like to thank the Lord for helping me obtain and accomplish all that was necessary to complete this research. He really does care for all his creations.

I am very thankful to have been able to study under Dr. Tianyi He. His patience and mentorship enabled me to achieve more than what I thought was possible.

My gratitude also goes out to Dr. Douglas Hunsaker, Dr. Thomas Fronk, and Dr. Todd Moon. Their instruction will always be a valuable asset to me. They believed in me even when I didn't believe in myself.

Forever will I be grateful to my dear wife Amy for supporting me. Her help test flying and troubleshooting was essential to this work. I would not have completed this research without her.

Clayton T. Spencer

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACRONYMS

BEC	battery eliminator circuit
BFCS	body-fixed coordinate system
CG	center of gravity
COTS	commercial off the shelf
DEP	distributive electric propulsion
DoF	degree of freedom
EKF	extended kalman filter
EOM	equations of motion
ESC	electronic speed controller
eVTOL	electric vertical takeoff and landing
GUI	graphical user interface
IMU	inertial measurement unit
ICS	inertial coordinate system
LPV-MPC	linear parameter-varying model predictive control
LSR	least squares regression
LTV	linear time-varying
MFE	makflyeasy
NN	neural network
OLSR	ordinary least-squares regression
PID	proportional integral derivative
PPM	pulse position modulation
PWM	pulse width modulation
RPM	rotations per minute
STOL	short take-off and landing
UAV	unmanned aerial vehicle
USUUA1	utah state university unmanned aircraft1

CHAPTER 1

INTRODUCTION

The concept of eVTOL aircraft emerged many years ago and has received attentions again in recent years in both civilian and military contexts. The great advances in Distributed Electric Propulsion (DEP) has overcome several technical challenges of propulsion systems and enabled a new system-level design of eVTOL aircraft. DEP systems use multiple electric motors to power an aerospace vehicle. These motors are distributed across the airframe. The electric motors are typically smaller and lighter than traditional jet or piston engines, and they can be placed in locations that improve efficiency, reduce noise, augment maneuverability, and increase safety. The eVTOL aircraft can be manned or unmanned, and this thesis focuses on Unmanned Aerial Vehicles (UAVs). The proposed research seeks to gain the understanding about the aerodynamics and propulsor dynamics during transition flight through the collection of experimental data. Flight data is then used to perform system identification to determine unknown system parameters and provide a dynamic model for control design.

1.1 Examples of VTOL Aircraft

There are many types of VTOL or Short-Take-Off and Landing (STOL) aircraft today. VTOL aircraft are commonly broken up into a few categories such as tail-sitters [5], copters [6], and launchers or STOL UAVs [7]. There are other sub-categories, however, the previously mentioned categories cover the vast majority of VTOL aircraft. Some aircraft are a combination of the various categories which combine traits of the aggregate categories and avoid problems associated with each of the categories. These make up the hybrid VTOL category of vehicles. These aircraft can employ various types of propulsion systems and configurations, however, this study focuses on a subcategory of copters called hybrids. An

example of a hybrid VTOL is a tilt-rotor or tilt-wing aircraft as studied by the following sources [5, 8, 9]. This research uses a hybrid electric VTOL with tilting rotors and fixed wings.

1.2 Characteristics of hybrid VTOL Vehicles

Hybrid VTOL aircraft combine the benefits of a multi-copter, such as a quad-copter, and a fixed-wing airplane. Multi-copters offer advantages such as not requiring an airstrip, strong hovering capabilities, high maneuverability, and low sensitivity to wind. In contrast, fixed-wing aircraft have better fuel efficiency, longer flight duration, higher speeds, and a greater payload capacity. Tilt-rotor eVTOL aircraft have the ability of taking off vertically and then transition from vertical take-off to fixed-wing forward flight. The transition mechanism is the key feature that distinguishes different variants of hybrid eVTOL aircraft. These aircraft variants include: 1) fixed push/lift rotors; 2) tilt-rotors; 3) tilt-wing; and 4) tail-sitter.

Fixed push/lift rotor aircraft usually have one or more dedicated motors responsible for providing thrust during fixed-wing mode. These motors are then deactivated when the aircraft transitions to VTOL mode. Tilt-wing aircraft are distinguished by the ability to tilt the entire main lifting surfaces during flight. By tilting the wings of the aircraft, the vehicle can transition to fixed-wing mode while minimizing the number of necessary motors required for either flight mode. Tail-sitter aircraft do not have rotating motors or wings but execute VTOL operations by orienting the aircraft such that the thrust of the fixed-wing motors allows the vehicle to hover or vertically take off.

Tilt-rotor aircraft usually have two or more motors that can tilt forward or backward to transition from VTOL mode to fixed-wing mode. It is common to have one or more motors deactivate when the aircraft transitions to fixed-wing mode. This work studies a hybrid UAV with a tilt-rotor and fixed-wing configuration during the transition phase of flight. The aim is to enhance understanding of the aircraft system and dynamics throughout this intricate flight phase.

1.3 Components of eVTOL Vehicles

There are several key components of electric VTOL aircraft that enable different functionalities. (1) Electric motors. They convert electrical energy from the battery into rotational energy that drives the propellers or rotors. (2) Batteries. Electric VTOL aircraft rely on batteries to store and supply the electrical energy needed to power the electric motors. They are usually lithium-ion batteries and are placed strategically in the aircraft to optimize weight distribution and Center of Gravity (CG). (3) Flight control system. The flight control system is essential and is responsible for controlling the aircraft's movement and stability during flight. The flight control system includes control hardware and software that work together to maintain the aircraft's position and orientation or follow a designated flight trajectory. (4) Propellers or rotors. Propellers or rotors are the blades that convert the rotational energy from the electric motors into thrust. In eVTOL aircraft, they are often mounted on tilting arms or swiveling rotors that allow the aircraft to transition from vertical takeoff to horizontal flight. (5) Avionics. Avionics are the electronic systems that control and monitor the aircraft's performance. They include instruments, sensors, communication equipment, and navigation systems that provide the human or software pilot with real-time information about the aircraft's status and surroundings. (6) Airframe. The airframe is the physical structure of the aircraft, including the wings, fuselage, and landing gear. In eVTOL aircraft, the airframe is designed to be lightweight and aerodynamically efficient to maximize efficiency and range.

1.4 Brief Literature Review

The test vehicle in this work, called Utah State University Unmanned Aircraft1 (USUUA1), has a 2+2 tilt-rotor configuration. This means that the aircraft has four motors in total. The two front-mounted motors can tilt forward 90 degrees to transition the aircraft to fixed-wing mode. The back two motors do not tilt but deactivate when the aircraft successfully transitions from VTOL mode to fixed-wing mode. An example of a tilt-rotor aircraft is the MV-22B Osprey [10]. Motivation for this study is to understand how tilt-rotors and tilt mechanisms affect the state of the aircraft in transition flight. Characterizing system

parameters in transition flight can increase the understanding of how tilt-rotors and tilt mechanisms affect the aerodynamics of the vehicle.

While tilt-rotor aircraft are not a new concept, there is still much to be learned about them. This proposal seeks to identify model parameters using LSR which is a common method to estimate unknown system parameters [11]. The transition phase of flight for hybrid VTOL vehicles is complex not only due to the aerodynamic effects of tilting rotors but also the gyroscopic effects of the tilting action itself.

To reduce the complexity of this study, only the longitudinal aircraft dynamics are considered. Although this method is an approximation, separating the longitudinal and lateral aircraft dynamics simplifies the vehicle model and still gives insight about the system [12]. This approximation is useful as long as the maneuvers the test vehicle is subjected to do not significantly excite the lateral dynamics of the vehicle. These maneuvers are discussed later in this work.

1.4.1 Dynamic Model Development

An essential component of this work is the model of the aircraft. The mathematical model of an eVTOL aircraft plays a critical role in control system design and vehicle performance predictions. However, modeling the tilt-rotor eVTOL aircraft presents several challenges, such as the complex aerodynamics involved in transitioning from vertical to forward flight, aerodynamic interactions between fixed-wings and propellers, and the complex dynamics in transition phase [3]. One common approach of the modeling of eVTOL aircraft is to combine rigid-body dynamics and aerodynamics, where the aircraft is modeled as a set of interconnected rigid bodies subject to aerodynamic forces and moments.

Aero-propulsive modeling is a term used to describe the mathematical modeling of the aerodynamics and propulsor dynamics of aircraft [13]. Much work has been done to accurately model many different UAVs over various flight envelopes. It is common to begin modeling hybrid VTOL UAVs by using a standard set of assumptions for a rigid body aircraft [14].

1.4.2 System Identification of eVTOL UAV's

Many strategies for system identification are available. The three main categories of system identification or parameter estimation are white box, grey box, and black box methods. White box methods involve developing physics models and mathematical equations that accurately describe the motion of the vehicle. This involves developing a mathematical model of the vehicle dynamics that descries what inputs relate to what outputs of the system. Examples of white box modeling can be found in [14–16]. White box methods require accurate knowledge of the model structure and have the advantage of well-understanding and accurate predictions of input-output relationships. However, white box methods struggle when the physical system becomes highly complex and coupled. This is often the case for hybrid VTOL aircraft.

Black box system identification strategies usually involve data-driven modeling. This means that data is used to train a machine learning algorithm, such as a neural network, in order to develop a model for the aircraft. This method usually requires a significant amount of data and computing power to be able to properly train the machine learning algorithm. If the training data is of high quality and there is enough of it, black box methods can develop complicated models that describe the motion of the aircraft accurately. The performances of the black-box models are determined by data quality, and they are often valid for the operating conditions covered by the data. It is not uncommon for these algorithms to struggle when compared to validation data. This leaves some undesired uncertainty and lacks physical trustworthiness in the model. An example of a black box method is discussed in [17].

Grey box strategies are a combination of white box and black box methods. Combining the explicit physical knowledge of the system from white box methods and the learning capability of black box methods, grey box methods can use a mathematical description of the vehicle and data to identify the unknown parameters of the model. This approach leverages physical knowledge of the system and data. Examples of grey box methods can be found in [13, 18–20]. This work uses a grey box method to estimate parameters of a

dynamic model of a hybrid VTOL UAV. The approach used in this research for parameter estimation is an LSR method. A similar algorithm can be found in [21].

1.4.3 UAV Control Strategies

There are many control strategies available for all types of UAVs. Control strategies can be broken down into two main categories which are model-based and model-free methods. Model-based control algorithms rely on a mathematical model to design aircraft control. [16] developed a unique Linear Parameter-Varying Model Predictive Control (LPV-MPC) algorithm. Such a control method has also been successfully applied to the modeling and control of autonomous vehicles [22–24]. An hierarchical MPC controller was developed in [25]. These methods often rely on the dynamic model to predict vehicle behaviors in the prediction horizon. Often, they rely on the dynamic model to formulate and solve optimization problems to determine the control inputs.

Model-free algorithms are advantageous in the aspect that the complicated aircraft dynamics do not need to be modeled in these control algorithms. Besides, the model-free control provides an adaptive method for various operating conditions across different types of VTOL aircraft. However, these methods often require extensive tuning and testing to result in a good controller. [26] used a sliding mode control as well as an iterative learning control. An improved version of a Proportional Integral Derivative (PID) controller was presented in [27]. This research will use a PID controller developed by Ardupilot for flight data collection [3].

Recent works [28] on combining model-based and model-free control are attractive, particularly for the VTOL aircraft under complex uncertainty, but need more theoretical development for the stability and convergence analysis.

CHAPTER 2
RESEARCH OBJECTIVES FOR USUUA1 AND TECHNICAL APPROACH

2.1 Research Objectives and Milestones

Success of this research work is measured using three main objectives. (1) Develop and evaluate the performance of aircraft components and the aircraft system as a whole. (2) Identify aircraft parameters in dynamic model from real flight data. (3) Develop a PID controller for the pitch angle and optimize the PID gains to achieve specific performances. Individual tasks for each objective are listed below with milestones called out.

1. Develop and evaluate the performance of components and aircraft system.
   (a) Procure airframe and subsystem components.
   (b) Build Wings and fuselage using procured materials and tools.
   (c) Assemble wings and fuselage and create wiring system and diagram. Milestone: Develop a working tilt-rotor UAV.
   (d) Test subsystems on the "bench" in a laboratory environment and test aircraft hover capability.
   (e) Test the aircraft transition capability from quadcopter to fixed-wing flight. Milestone: Complete a forward and backward transition in flight autonomously.
2. Identify aircraft parameters in dynamic model from real flight data.
   (a) Develop the dynamic model and equations of motion for the aircraft and an LSR model to estimate parameters in the dynamic model.
   (b) Collect Data via flying the aircraft.
   (c) Write code to collect and post-process the flight data. Milestone: Code can unify flight data, separate transition data, and compute LSR with data.

(d) Validate the model with validation data. Milestone: model validated within 25 percent accuracy of validation data.

3. Develop a PID controller for the pitch angle of the vehicle.

   (a) Using MATLAB Simulink, create a control model to control the identified model of USUUA1.

   (b) Tune the PID gains of the identified model using the MATLAB PID tuner.

2.2 Aircraft Development and Component Selection

Most of the aircraft build and test for USUUA1 was completed in the fall of 2022. Bench testing and debugging was done in the spring of 2023. Flight testing began in the summer of 2023 and ended in the fall of 2023. After successful build and test of the USUUA1 aircraft, data must be collected from various flights. To identify the necessary parameters of the dynamic model the system must be perturbed using the tilt mechanism. To do this, various parameters in the flight control software are changed for each data collection flight. This method perturbs the system enough to identify the unknown system parameters developed in the modeling section of this document.

Data collection can be restricted by weather, time, and hardware constraints. These restrictions make data collection relatively difficult. Part of the collected data must be reserved for model validation purposes. Table 2.1 shows the data collection flight plan.

Table 2.1: Data collection plan using USUUA1.

Q_TLT_MAX (deg)	Q_TLT_RATE_DN (deg/s)	
	12	14
35	Test 1.1	Test 2.1
45	Test 1.2	Test 2.2
55	Test 1.3	Test 2.3
65	Test 1.4	

The two main parameters changed to perturb the system are Q_TLT_MAX and $Q\_TLT\_RATE\_DN$. These adjust the max tilt angle before transition completion and the max downward tilt rate respectively. Flight data is collected from the various sensors onboard USUUA1 and the data is logged by the Pixhawk flight controller. The data can be viewed and plotted in the Mission Planner software or in Matlab and is used to identify unknown parameters in the dynamic model of the aircraft.

2.3 Data Extraction and Unification

Mission Planner uses a built-in tool to convert the flight log data to a .mat file [3]. From the .mat file, the data can be processed and structured for use in system identification. The various sensors onboard USUUA1 have different sampling frequencies which poses a problem for data processing. Because all the data corresponds to different timestamps, it is hard to index and process the data with common Matlab functions. To get around this issue, code was developed to unify the lengths and timescales of the data vectors containing the flight log data.

The function *simple_data_manipulation* found in Appendix B takes in flight data, and outputs is the same data from the transition period of the flights. The driver code in Appendix B runs the data post-processing. An *lsr* function computes the LSR of the flight data and uses it to estimate unknown parameters in the system. All related functions and code for data post-processing are given in Appendix B. An example of work that employs a similar data manipulation technique is given in [29].

Verification of the data processing code functionality is done by comparing data before and after data processing as part of the post-processing algorithm. An example of this is shown in Figs. 2.1 and 2.2.

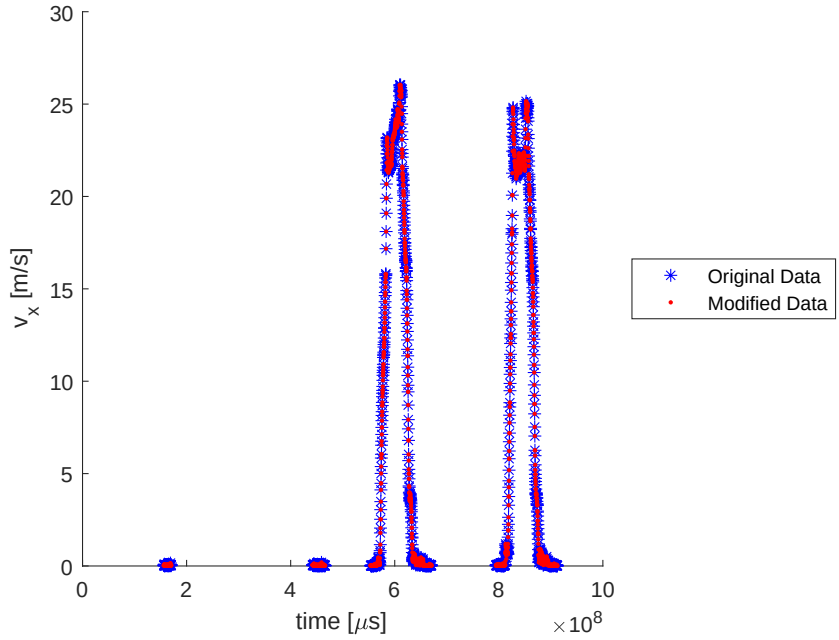


Fig. 2.1: Flight data before and after data unification during post-processing.

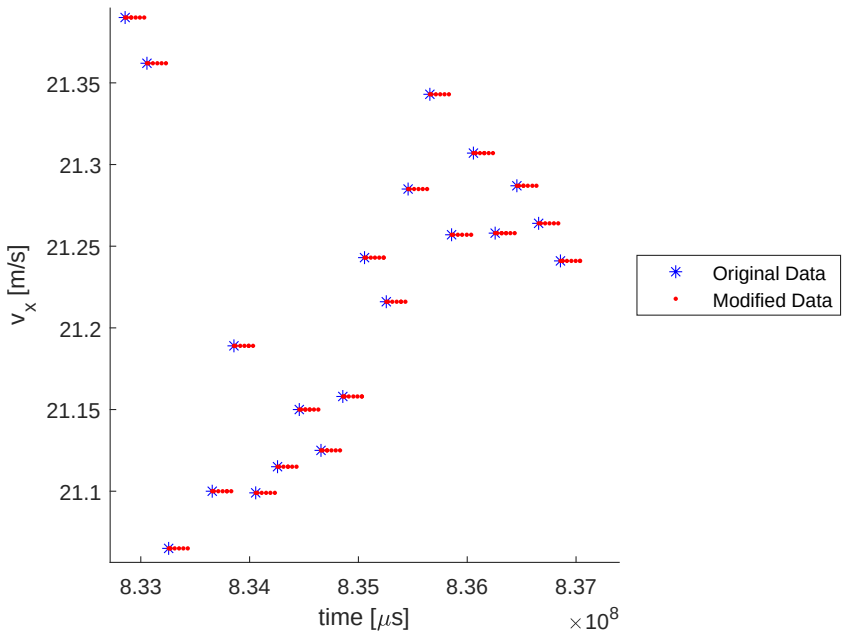


Fig. 2.2: Zoomed-in data before and after data unification during.

By observing Figs. 2.1 and 2.2, it can be determined that the data vector contains all the original data as well as the interpolated data from the post-processing described above. Subjecting all the collected data to this process unifies the data regardless of the sampling frequencies of the various sensors used for data collection.

2.4 Dynamic Model and EOM Derivation

Modeling USUUA1 entails deriving the equations of motion that govern the dynamics of the aircraft. This was done by following similar derivations from other works [12, 14]. Because this work uses a grey box method for parameter identification, a dynamic model must first be derived before the identification process can take place. The derivation for the dynamic model is given in Chapter 4.

2.5 Control Algorithm for USUUA1

Ardupilot employs an inner-outer loop PID configuration to control aircraft that run the Ardupilot software [3]. This work is not focused on changing Ardupilot's control algorithm. However, a model is developed in MATLAB Simulink to create a PID controller for the pitch angle of the vehicle. The gain values are tuned in this work using MATLAB's PID Tuner. The gains are optimized to achieve the desired settling time.

CHAPTER 3
DEVELOPMENT OF AIRCRAFT PLATFORM AND COMPONENT CHARACTERIZATION

This section documents the development and system-level integration of USUUA1. Development or optimization of individual UAV components is outside the scope of this work. It is desired that the airframe and other components of USUUA1 be Commercial Off The Shelf (COTS) parts which can easily be procured. To reduce the cost and complexity of the research, the selected components range from hobby-grade to mid-grade, commercially available components. This allows for rapid prototyping and repeatability. The overall system design is presented in this section by first selecting a 2+2 tilt-rotor fixed-wing hybrid VTOL airframe. Next, the flight control software platform is selected to control the aircraft and its subsystems and provide autonomous flight capability.

Full trade studies for the airframe and individual components of USUA1 are outside the scope of this work. The candidate components are not discussed in this work for brevity. The selection criteria for the aircraft components are:

- Cost of each piece-part
- Commercial availability of components or subsystems
- Compatibility with other subsystems
- Size/Weight of individual components
- Modularity of integration into the overall system
- Connection Reliability and hardware durability

The selected airframe is the MakeFlyEasy (MFE) Freeman 2100 shown in Fig. 3.1. This airframe comes as a kit and includes many supporting components such as motors,

Fig. 3.1: USUUA1, the experimental aircraft platform.

propellers, servos, and some wires. This greatly reduces development time and effort. In addition, the MFE Freeman 2100 is supported by setup documentation. The MFE Freeman 2100 is designed to work with the Ardupilot software for flight control and has documentation supporting software installation and configuration. Ardupilot was selected as the flight control software because it's free, open-source, and supported by a community of developers, and has comprehensive documentation [30].

3.1 System Layout

The main subsystems of the USUUA1 are power, communications, flight control, propulsion, and actuation. The power subsystem provides power to the electronic components of USUUA1 at the correct voltage. It is made up of the main flight battery, power module, Power distribution board, and battery eliminator circuit. USUUA1 is a radio-controlled (RC) UAV. To facilitate this, the communications subsystem uses a ground station, telemetry radios, receiver, Pulse Position Modulation (PPM) converter, transmitter, and antennas. Flight control is possible through the use of a flight control board, Inertial Measurement Unit (IMU), gyroscope, airspeed sensor (optional), compass, buzzer, and magnetometer. Some of these components are integrated into one unit such as the GPS module which has a compass and a GPS receiver. Propulsion components include electronic speed controllers (ESCs), motors, and propellers. The actuation subsystem is comprised of digital servo motors that actuate control surfaces and mechanisms.

The USUUA1 system is connected via wiring and each component is located in the aircraft to be modular for future improvements to the system. Fig. A.1 shows the wiring diagram for the system as a whole. The wiring diagram details the electronic layout of the aircraft as well as the system/subsystem layout of the aircraft. Each of USUUA1's wings houses two different ESCs to control the front and back motors located on either end of the carbon fiber motor pods. The wings also house one 3.6 kg and one 25 kg digital servo each responsible for actuating the ailerons and tilting mechanism respectively.

3.2 Component Selection

The most important component besides the airframe is the flight control board. The flight control board controls the other subsystems and relays system information back to the ground station via the communication subsystem. The Pixhawk 2.4.8 was chosen as the flight controller for USUUA1. The Pixhawk is compatible with the Ardupilot firmware and has enough output ports to support the entire system without any split connections like y-connectors. The Pixhawk is also capable of logging flight data onto an SD card for post-flight data analysis which is essential to this research. The Pixhawk is shown in Fig. 3.2.

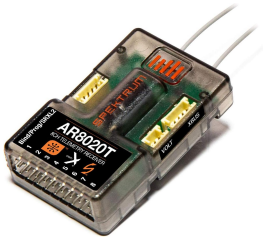
Fig. 3.2: Pixhawk 2.4.8. Image from [1].

Supporting components for the flight controller (Pixhawk) are the PPM converter, buzzer (optional), and GPS module. The IMU and the gyroscope are internal to Pixhawk itself. The PPM encoder converts the pilot commands through the radio transmitter into a

PPM signal which is sent to the flight controller. The IMU measures acceleration in the x, y, and z axes of the unit. The gyroscope measures rotation rates about the x, y, and z axes of the unit. The Pixhawk also has an internal compass to determine the orientation of the unit. An airspeed sensor can also be used with the Pixhawk, however, it is not required. This is because the airspeed can be approximated using the IMU and GPS units.

Input commands are given by the USUUA1 pilot through a radio transmitter to a radio receiver onboard the aircraft. The signal is then passed to the PPM encoder and then to the flight controller. The selected transmitter is the Spectrum NX8 in Fig. 3.3a and the selected receiver is the Spektrum AR8020T shown in Fig. 3.3b.

(a) Spektrum NX8 transmitter. Image from [31].



(b) Spektrum AR8020T receiver. Image from [32].

(c) USUUA1 flight battery. Image from [33].

These RC components are relatively reliable and provide a modular platform to link control inputs from the transmitter to the flight controller. Telemetry radios and antennas can also be included with the Pixhawk as was the case for the components selected for USUUA1. The Mission Planner software downloaded onto any computer can be used as the Ground Control Station (GCS) for USUUA1. The GCS communicates with the Pixhawk via the telemetry radios. One telemetry radio is connected to the GCS and the other is connected to a telemetry port on the Pixhawk.

Propulsion devices were selected from components available through MFE. The front and rear motors are brushless 500 kV and 450 kV motors respectively. ESCs were also selected from MFE. The front and rear ESCs are 40 A and 65 A ESCs respectively. the front motors are slightly more powerful to be able to provide all the propulsion during fixed-

wing flight. The digital servos were also selected from MFE as they seamlessly integrate with the airframe and the MFE-supplied BEC.

Power for the entire system is provided by one 6-cell battery. The selected battery is a Tattu, 16000 mAh, 22.2 V battery shown in Fig. 3.3c. This battery was chosen for its relatively high discharge rating, medium capacity, and high cell count. The combination of these parameters allows the battery to power the entire aircraft and provides enough power for the aircraft to transition from fixed-wing to VTOL aircraft. MFE designed the Freeman 2100 airframe to use a 22000 mAh battery to extend aircraft flight time to 90 minutes. This research doesn't require 90 minutes of flight time, therefore, USUUA1 can use a smaller battery. As indicated by Fig. A.1, the battery was selected to power the actuators, propulsion components, communication components, and flight control components in parallel. The Holybro PM02 power module was selected to support the 6-cell flight battery and to provide power to the flight controller.

3.3 Aircraft Assembly

Assembly of USUUA1 follows a similar process as outlined by MFE [34] in the Freeman 2100 documentation. First, the wing-to-fuselage connector was wired using standard soldering techniques. The wires were then run out to the carbon fiber motor pod on each wing. The motors and ESCs were then installed and wires were connected to the wing-to-fuselage wires. The 3.6 kg and 25 kg servos were installed and connected to the wing-to-fuselage wires. The aileron stiffeners were also installed using foam glue.

Assembly of the fuselage replicated the assembly of the fuselage shown in the MFE documentation [34]. Next, the tail servos were installed in the aft section of the fuselage and wired up to the main section of the aircraft. The wing and tail brackets were then installed using foam glue. All hardware is included in the Freeman 2100 kit. After the airframe components had all been installed and assembled, the electronic components were installed.

The Pixhawk was located just aft and under the main fuselage carbon stiffener. This was done to locate the Pixhawk as near to the CG of the aircraft as possible. This allows

the readings from the Pixhawk internal sensors to be reliable for the aircraft as a whole. The flight controller was installed first onto a vibration dampener. The dampener was then secured to a 3D-printed locator board shown in Appendix A.2. After locating the flight controller the power system was installed.

The battery was installed into the forward section of the aircraft where space allowed. and a wire harness was created using solder and shrink wrap to connect the battery to the actuator, propulsion, and flight control subsystems. One of the leads from the battery harness connects to the power module which connects to the flight controller and the power distribution board. The other lead connects to the BEC to power the servos. Installation in this way allows the power module to regulate power to the flight controller to avoid overpowering the flight control board. This also allows the BEC to regulate power to the servos to avoid overpowering the actuator system. The battery was secured using a 3D printed locator board shown in Appendix A.2.

Another crucial component to locate is the GPS module. Ardupilot [35] documents how to locate the GPS module-compass combination. Because of magnetic interference, the GPS module was put on top of a short mast in the forward section of the aircraft. The remaining components were located in the midsection of the fuselage for easy access and to keep the CG of the aircraft near the quarter-chord of the root of the main wing to increase stability. To secure the various components during flight, another locator board was designed and 3D-printed as shown in Appendix A.2. The components were then attached to the locating boards with a combination of foam glue, double-sided foam tape, and fiber-reinforced tape. Fig. 3.4 shows the location of the GPS module relative to the power components of the aircraft. The location of the GPS module shown in Fig. 3.4 reduces the magnetic interference of the compass to acceptable levels.

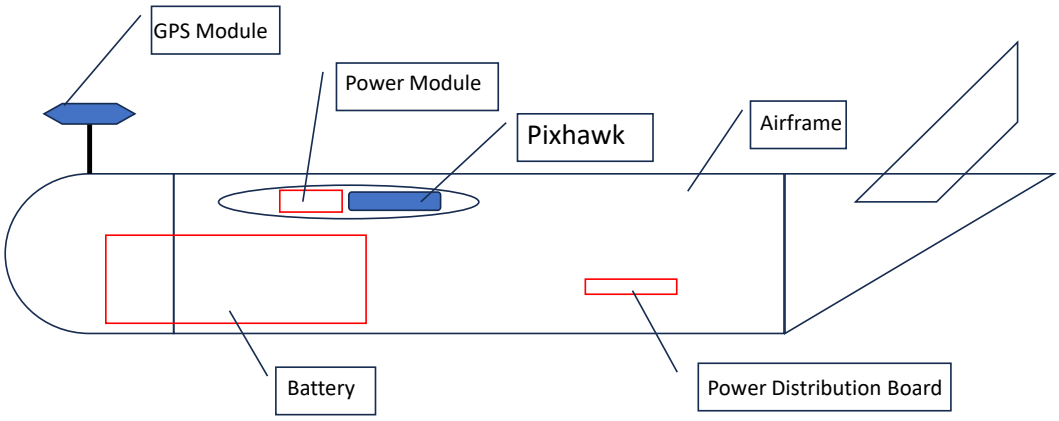


Fig. 3.4: Simplified side view of the GPS module location.

3.4 Bench Testing and Component Characterization

This section follows closely to the Ardupilot Quadplane documentation and the MFE Freeman 2100 setup documentation [4,30]. Downloading Mission Planner was the first step to setting up USUUA1. Mission Planner can be downloaded from the Ardupilot website [36]. Calibration of the IMU, gyroscope, compass, and transmitter were completed by following the Ardupilot setup process.

3.4.1 ESC Calibration and Verification

Calibrating the ESCs was conducted by completing the following procedure:

1. Remove the motor propellers.
2. Turn on the transmitter.
3. Disconnect power to the ESCs on the aircraft if not already done.
4. Connect power from the battery to the flight controller.
5. Change to calibration mode using the parameter Q_ESC_CAL = 1.
6. Change aircraft mode to QSTABILIZE if not already done.

7. Arm the aircraft.
8. Move the throttle stick on the transmitter to maximum.
9. Re-connect power to the ESCs.
10. Wait for the beeping sound from the ESCs indicating they have registered maximum throttle .
11. Lower throttle stick on the transmitter to minimum.
12. Wait for the beeping sound from the ESCs indicating they have registered minimum throttle.

Once ESC calibration was complete, the calibration was verified by putting the aircraft into QSTABILIZE mode, arming the aircraft, and raising/lowering the throttle quickly. If the ESCs all begin to spin at approximately the same throttle input and increase speed by raising the throttle, then the calibration is successful and a motor test can be conducted. If the calibration is not successful, repeat the ESC calibration process.

3.4.2 Motor and Servo Spin Direction Test

Using the Graphical User Interface (GUI) from Mission Planner, the motor test was conducted for USUUA1 to verify the orientation and spin direction of the motors. Fig. 3.5 shows the motor layout (1-4) and the motor test sequence (A-D).

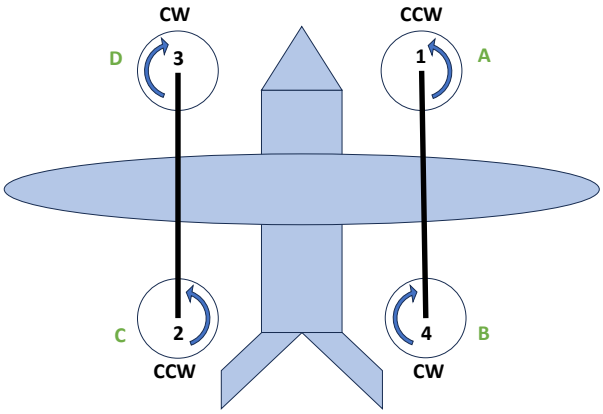


Fig. 3.5: Motor layout and motor test sequence.

After the motor test was conducted, the control surface deflection test was completed. The control surface connections were first adjusted to be neutrally at zero degrees of deflection. Next, the aircraft was put into manual mode and the pilot gave roll, pitch, and yaw inputs with the transmitter. control surface deflections were reversed as needed to make the pilot inputs match the actual control surface deflections. Any slight variance in the neutral point was adjusted electronically with the Ardupilot GUI and a straight edge.

3.4.3 Tilt Mechanism test and Motor Speed Characterization

Testing the tilting mechanism of USUUA1 was done to ensure the aircraft would transition to fixed-wing flight when commanded by the autopilot. This was accomplished by changing the flight mode between a quadcopter mode, such as QSTABILIZE, and FBWA. When changed to QSTABILIZE mode, and with all the necessary tilt-rotor parameters enabled as detailed by Ardupilot [37], the tilt-rotors remained upright at zero degrees of deflection. When changed to FBWA, the tilt-rotors rotated down to approximately 90 degrees of deflection. The min and max values were adjusted in the Mission Planner GUI to allow the tilt-rotors to be at 0 degrees in QSTABILIZE mode and 90 degrees in FBWA mode. This is an iterative process of selecting a trim signal value in the GUI and verifying the angle a protractor.

Once the Testing of the tilt mechanism was complete, the motors were characterized using a PWM input signal. The output motor speed for each motor was measured using an external, laser, tachometer. The output was plotted against the input and the resulting plot is shown in Fig. 3.6. Motors 2 and 4 show a linear relationship with the PWM signal which is desirable. Motors 1 and 3, which are the front tilting motors, show a nonlinear relationship with the PWM signal. While not ideal, the mapping shown in Fig. 3.6 models the output of motors 1 and 3 to an acceptable degree.

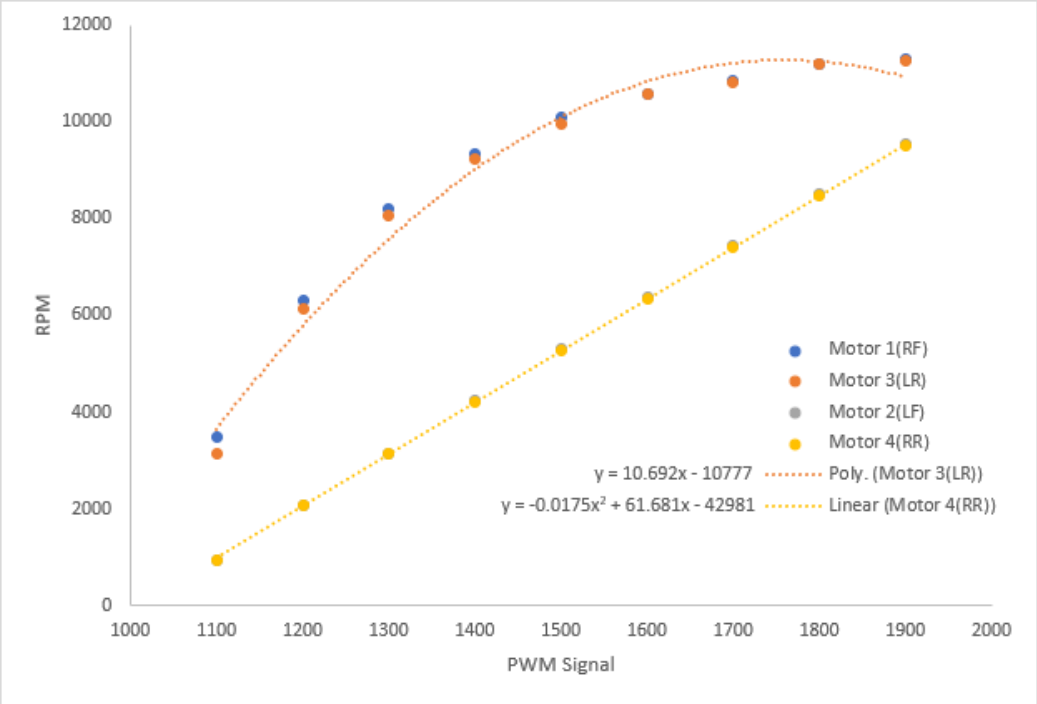


Fig. 3.6: Mapping of input PWM signal to output motor RPM.

Characterization of the tilting mechanism was carried out in a similar way as the motor characterization. The input was the PWM signal and the output was the tilt angle measured with a protractor. The resulting plot of the tilt mechanism input to output is shown in Fig. 3.7. From the results, the tilting motors reveal good linearity and accurate angular position in the mapping of the PWM signal to the tilting angle. This result is consistent with the truth that the tilting servo motors are closed-loop controlled.

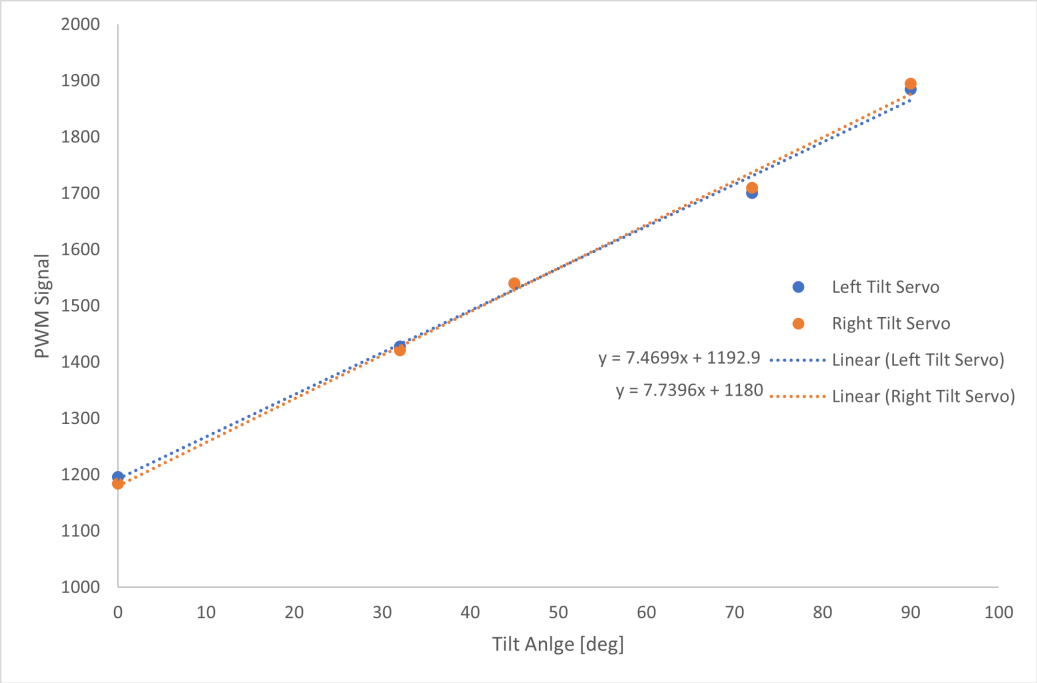


Fig. 3.7: Mapping of input PWM signal to output tilt angle.

3.4.4 Pre-Flight Checks

Before flying the aircraft, a pre-flight check was conducted before each flight. This helps reduce issues with reliability and repeatability between multiple flights. Table A.1 shows the pre-flight procedure to complete before each flight with USUUA1. Common problems that arise during the pre-flight procedure can usually be resolved by re-calibrating either the IMU or the compass on board the aircraft.

Failure to properly execute the pre-flight procedure can result in a failed flight or even crashing the aircraft. Following Table A.1 helps verify the sensors onboard the aircraft are outputting correct and usable data. Flights should not be conducted if any calibration or verification fails.

CHAPTER 4
DYNAMIC MODELING OF USUUA1

In this chapter, the dynamic model of VTOL aircraft in the transition flight is derived as a 6-DoF model and further reduced to the longitudinal motion. After that, the aerodynamic coefficients will be identified using the real flight data.

4.1 Governing Equations of Motion in 6-DoF

In this thesis, the modeling of the VTOL aircraft makes the following assumptions:

1. Aircraft can be approximated as a symmetric, rigid-body.
2. Neglect flow changes and flow interactions over the wing due to tilt angle change.
3. Wind disturbances are not considered.

Deriving the governing equations of motion follows a similar approach in [14, 16]. Reference frames are defined as follows: $I = [\mathbf{O}; \mathbf{x}^I, \mathbf{y}^I, \mathbf{z}^I]$ is an inertial frame with origin $\mathbf{O}$, and $B = [\mathbf{G}; \mathbf{x}^B, \mathbf{y}^B, \mathbf{z}^B]$ is the body-fixed frame with the aircraft center of mass $\mathbf{G}$. In the body-fixed frame, $\mathbf{x}^B$ points forward, $\mathbf{y}^B$ points to the right wing, and $\mathbf{z}^B$ points downward by right-hand-rule convention. Fig 4.1 shows the coordinate system of the aircraft.

The dynamic equations of motion for a generic aircraft with external forces and torques, using standard rigid-body assumptions, are shown in (4.1). The dynamic model is established in the aircraft body-fixed frame, and it is denoted by the superscript B.

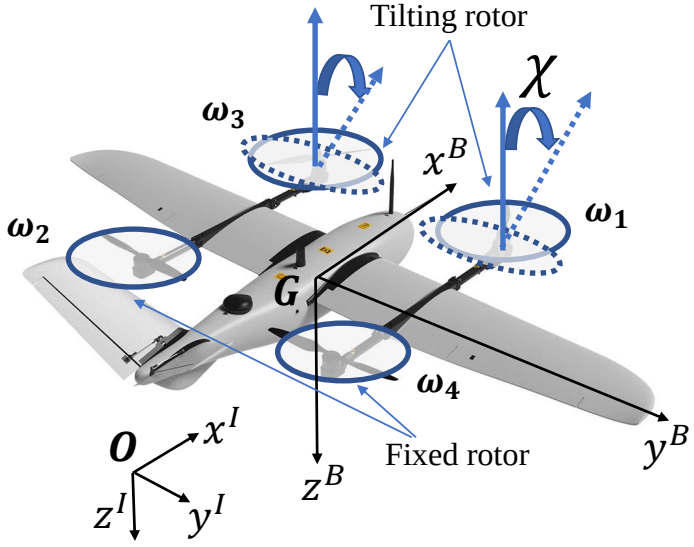


Fig. 4.1: Coordinate system of USUUA1.

$$
\begin{aligned}
\dot{\mathbf{p}}^B &= \mathbf{v}^B - \omega \times \mathbf{p}^B \\
\dot{\mathbf{v}}^B &= -\omega \times \mathbf{v}^B + g\mathbf{R}_I^B \mathbf{z}^I + \frac{1}{m}(\mathbf{F}_T^B + \mathbf{F}_a^B) \\
\dot{\mathbf{q}}_I^B &= \frac{1}{2}\begin{bmatrix} 0_{1\times 1} & -\omega^T \\ \omega & -\omega_\times \end{bmatrix} \mathbf{q}_I^B \\
\mathbf{N}^B\dot{\omega} &= -\omega_\times \mathbf{N}^B\omega + \mathbf{\Gamma}_T^B + \mathbf{\Gamma}_a^B
\end{aligned}
\tag{4.1}
$$

From (4.1), $\mathbf{q}_I^B = [\mathbf{q}_0, \mathbf{q}_v^T]^T$ is the attitude quaternion of the aircraft body-fixed frame relative to the inertial frame and $\omega = [\omega_x, \omega_y, \omega_z]^T$ is the angular rate vector of the aircraft. The vector $\mathbf{p}^B = [\mathbf{p}_x^B, \mathbf{p}_y^B, \mathbf{p}_z^B]^T$ is the position of the aircraft and $\mathbf{v}^B = [\mathbf{v}_x^B, \mathbf{v}_y^B, \mathbf{v}_z^B]^T$ is the velocity vector of the aircraft. The mass of the aircraft is m, g is the acceleration due to gravity, and $\mathbf{z}^I$ is the direction of gravity in the inertial frame. The forces due to the thrusting elements and aerodynamic elements of the aircraft are $\mathbf{F}_T^B$ and $\mathbf{F}_a^B$ respectively. The torques due to the thrusting elements and aerodynamic elements of the aircraft are $\mathbf{\Gamma}_T^B$ and $\mathbf{\Gamma}_a^B$ respectively. From (4.1), $\omega_\times$ is the skew-symmetric matrix related to the cross

product [14, 16]. In other words, $\mathbf{u} \times \mathbf{v} = \mathbf{u}_{\times}\mathbf{v}$ for any vector $\mathbf{u}, \mathbf{v}$. $\mathbf{N}^B$ is the inertia tensor of the aircraft, which is assumed to be diagonal, as shown in (4.2).

$$\mathbf{N}^B = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \tag{4.2}$$

The rotation matrix $\mathbf{R}_I^B$, which describes the rotation from the inertial frame to the body-fixed frame, is given by (4.3) where $\mathbf{I}_3$ is a 3×3 identity matrix. and $\mathbf{q}_{skew}$ is given in (4.4).

$$\mathbf{R}_I^B = \mathbf{I}_3 + 2q_0\mathbf{q}_{skew} + 2(\mathbf{q}_{skew})^2 \tag{4.3}$$

$$\mathbf{q}_{skew} = \begin{bmatrix} 0 & -q_z & q_y \\ q_z & 0 & q_x \\ -q_y & q_x & 0 \end{bmatrix} \tag{4.4}$$

The assumptions at the beginning of this section correspond to the aircraft in the transition flight regime where the aircraft converts to fixed-wing flight. Forces from the thrust elements, or motors, as well as forces due to the aerodynamics of the aircraft, are expressed as (4.5) and (4.6) respectively [14, 16].

$$\mathbf{F}_T^B = \sum_{i=1} \begin{bmatrix} \cos\chi_i & 0 & -\sin\chi_i \\ 0 & 1 & 0 \\ \sin\chi_i & 0 & \cos\chi_i \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ C_T\omega_i^2 \end{bmatrix} \tag{4.5}$$

$$\mathbf{F}_a^B = \frac{\rho}{2} S|\mathbf{v}^B|(C_L\mathbf{v}^{\perp B} - C_D\mathbf{v}^B) \tag{4.6}$$

In (4.5), χ_i is the tilt angle of the rotors, ω_i is the rotor speed, and $i \in \{1,2,3,4\}$. The tilting angle χ_i is defined positive in the counterclockwise. In (4.6), ρ is air density, S is the reference wing area, C_L is the aircraft coefficient of lift, C_D is the aircraft coefficient of

drag, and C_T is a quasi-coefficient of the thrust of the motors. Unlike C_L or C_D, C_T is not assumed to be non-dimensional but follows the definition in (4.5) to map the squared rotor speed to thrust force.

Torques from the thrusting elements and the aerodynamic surfaces are expressed as (4.7) and (4.8) respectively.

$$\mathbf{\Gamma}_T^B = \begin{bmatrix} \cos\chi & 0 & -\sin\chi \\ 0 & 1 & 0 \\ \sin\chi & 0 & \cos\chi \end{bmatrix} \begin{bmatrix} -C_T l & -C_T l & C_T l & C_T l \\ -C_T l & C_T l & C_T l & -C_T l \\ -C_Q & C_Q & -C_Q & C_Q \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} \tag{4.7}$$

$$\mathbf{\Gamma}_a^B = \begin{bmatrix} C_{F_a} & 0 & 0 \\ 0 & C_{F_{vL}} & 0 \\ 0 & 0 & C_{F_{vR}} \end{bmatrix} \begin{bmatrix} \delta_a \\ \delta_{vL} \\ \delta_{vR} \end{bmatrix} \tag{4.8}$$

In (4.7), l is the distance along the y-axis between the center of mass G and the center of the propellers. The Euler angle representation of the aircraft orientation is given as (4.9) where $\beta_I^B = [\phi, \theta, \psi]$ and $\mathbf{T}$ is the matrix shown in (4.10).

$$\dot{\beta}_I^B = \mathbf{T}\omega \tag{4.9}$$

$$\mathbf{T} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{bmatrix} \tag{4.10}$$

From (4.7)–(4.10), C_Q is the coefficient of torque of the propellers, C_F is the effectiveness of the control surfaces, $\delta_a, \delta_{vL}, \delta_{vR}$ are the deflections of the ailerons, left v-tail, and right v-tail respectively.

4.2 Dynamic Model of Longitudinal Motion

Deriving the model for USUUA1 follows with various simplifications to reduce model complexity, because this thesis focuses on the longitudinal motion, i.e. motion in the x-z plane. The following assumptions are further made in the longitudinal motion:

1. Both of the front motors spin at the same rate, so that the torque is balanced during the tilting of front motors. Similarly, the back motors also spin at the same rate. This means $\omega_1 = \omega_3$, and $\omega_2 = \omega_4$.

2. Transition is executed using only the tilting front rotors, and only longitudinal motion is activated in transition. During the transition, the lateral dynamics will not be excited. This means $\chi = \chi_1 = \chi_3$, and $\chi_2 = \chi_4 = 0$.

3. USUUA1 takes off vertically as a quad-copter and then begins transition when it reaches the planned altitude. The control surfaces are not functioning in the take-off phase and transition phase. Their contributions to aircraft motion are assumed as neglected due to the slow speed during takeoff and transition.

Solving (4.1) for $\dot{\omega}$ results in (4.11).

$$\dot{\omega} = (\mathbf{N}_B)^{-1}[-\omega_\times \mathbf{N}^B \omega + \mathbf{\Gamma}_T^B + \mathbf{\Gamma}_a^B] \tag{4.11}$$

Using the previously stated assumptions and expressions of forces and torques, it can be shown that the dynamic model of USUUA1 in longitudinal motion becomes (4.12).

$$\begin{aligned}
\dot{p}_x^B &= v_x^B - \omega_y p_z^B \\
\dot{p}_z^B &= v_z^B + \omega_y p_x^B \\
\dot{v}_x^B &= -\omega_y v_z^B - g\sin\theta + \frac{2C_T}{m}[\omega_1^2 \sin\chi] + \frac{\rho S}{2m}\sqrt{(v_x^B)^2 + (v_z^B)^2}(C_L v_z^B - C_D v_x^B) \\
\dot{v}_z^B &= \omega_y v_x^B + g\cos\theta - \frac{2C_T}{m}[\omega_1^2 \cos\chi + \omega_2^2] + \frac{\rho S}{2m}\sqrt{(v_x^B)^2 + (v_z^B)^2}(-C_L v_x^B - C_D v_z^B) \\
\dot{\theta} &= \omega_y \\
\dot{\omega}_y &= \frac{2C_T l(\omega_1^2 - \omega_2^2)}{I_{yy}^B}
\end{aligned} \tag{4.12}$$

It should be noted that I_{yy}^{B} is the moment of inertia about the y-axis of the aircraft in the body-fixed frame. Starting here, the body-frame superscripts will be dropped for simplicity. This work uses data collected from the open-loop system. Doing this, allows the modeling of the aircraft to be done without considering the controller dynamics. This helps simplify the dynamic model of the aircraft. The actuator dynamics are not included in this model because it is assumed that there are no control surface deflections during the transition phase. Additionally, it is assumed that the response speed of the rotors is high enough, compared to the response of the system, such that the motor dynamics do not need to be included in this model.

Based on the available onboard sensors and filtering algorithms [38], the following parameters in (4.12) are known: $\{p_x, p_z, v_x, v_z, \omega_y, g, \theta, m, \omega_1, \omega_2, \omega_y, l, \rho, S, \chi\}$. The unknown parameters are aerodynamic coefficients $\{C_T, C_L, C_D\}$. Different models can also be used for various parameters to increase model accuracy in future work. For example, C_L or C_D can be assumed as constant during the transition flight or a higher degree polynomial model could be used instead. However, increasing the order of the model does not always guarantee increased accuracy. Increasing the order of the model does, however, increase the complexity of modeling by adding additional unknown parameters to be identified.

4.3 Unknown System Parameter Identification By Quadratic Programming with Constraints

After the model is derived, the unknown parameters are identified. Experimental flight data from USUUA1 is to be used to drive a Least-Squares Regression (LSR) method to identify the unknown parameters of the model [11, 39].

This work develops a new LSR algorithm to identify the unknown model parameters compared to the existing work in [13]. At first, the details of LSR in [13] are given. Let the vector of parameters to estimate be η, and the regression equation to estimate the unknown parameters be shown in (4.13).

$$\tilde{\mathbf{y}} = \tilde{\mathbf{X}}\eta + \tilde{\nu} \tag{4.13}$$

Here, $\tilde{\mathbf{X}}$ is a matrix of regressors, $\tilde{\mathbf{y}}$ is the response vector, and $\tilde{\nu}$ is the measurement error which is considered to be zero-mean and uncorrelated. The matrix $\tilde{\mathbf{X}}$ is considered to be "error-free". Solving for the unknown parameter vector in (4.13) involves minimizing the cost function in (4.14) [13].

$$\min \mathbf{J}(\eta) = \frac{1}{2}(\tilde{\mathbf{y}} - \tilde{\mathbf{X}}\eta)^T(\tilde{\mathbf{y}} - \tilde{\mathbf{X}}\eta) \tag{4.14}$$

It should be noted that the minimization problem is a convex optimization problem without constraints. The analytical solution of the LSR optimization problem that can render the optimal estimation of unknown parameters η is given in (4.15)

$$\hat{\eta} = [(\tilde{\mathbf{X}}^T\tilde{\mathbf{X}})]^{-1}(\tilde{\mathbf{X}}^T\tilde{\mathbf{y}}) \tag{4.15}$$

The response vector calculated from the estimated parameters is expressed as (4.16).

$$\hat{\tilde{\mathbf{y}}} = \tilde{\mathbf{X}}\hat{\eta} \tag{4.16}$$

Due to the experimental nature of the data collection process in this work, the data are collected in discrete time [16, 40]. Therefore, a discrete-time model must be discretized from the continuous-time model using the Euler discretization formula shown in (4.17). Considering the nature of nonlinear systems, the sampling time selected must be small enough to ensure the responses of the discretized model match with that of the continuous-time model.

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \dot{\mathbf{x}}\Delta t \tag{4.17}$$

The LSR algorithm in [13] has an obvious drawback, that is, it cannot address the constraints of the physical variables. However, in this thesis, the unknown aerodynamic coefficients need to satisfy some physical constraints. For example, the thrust coefficient C_T is always positive. Next, the derivations of the new identification algorithm are presented.

The new identification algorithm can address the constraints on the model parameters. This helps constrain the model parameters to values which have physical meaning.

4.3.1 Parameter Estimation of Constant Values

We first assume that the unknown system parameters are constant values. Before applying (4.17) the dynamic equations of motion are reduced to those containing unknown parameters. The reduced-dimension version of (4.12) is given as (4.18)

$$\begin{aligned}
\dot{v}_x &= -\omega_y v_z - g\sin\theta + \frac{C_T}{m}[2\omega_1^2 \sin\chi] + \frac{\rho S}{2m}\sqrt{v_x^2 + v_z^2}(C_L v_z - C_D v_x) \\
\dot{v}_z &= \omega_y v_x + g\cos\theta - \frac{C_T}{m}[2\omega_1^2 \cos\chi + 2\omega_2^2] - \frac{\rho S}{2m}\sqrt{v_x^2 + v_z^2}(C_L v_x + C_D v_z) \\
\dot{\omega}_y &= \frac{2C_T l(\omega_1^2 - \omega_2^2)}{I_{yy}}
\end{aligned} \tag{4.18}$$

It should be noted here, (4.18) is still in the body-fixed frame but the superscripts are omitted for simplicity. Applying (4.17) to (4.18) and writing in the form of (4.13) yields (4.19), where $k \in \{1, 2, 3, ..., n\}$, is the index value of the current time step.

$$\begin{bmatrix} v_x \\ v_z \\ \omega_y \end{bmatrix}_{k+1} = \begin{bmatrix} v_x \\ v_z \\ \omega_y \end{bmatrix}_k + \begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{bmatrix}_k \begin{bmatrix} C_T \\ C_L \\ C_D \end{bmatrix}_k + \Delta t \begin{bmatrix} -\omega_y v_z - g\sin\theta \\ \omega_y v_x + g\cos\theta \\ 0 \end{bmatrix}_k \tag{4.19}$$

Here, the values for the elements of the regressor matrix $\mathbf{X}$ are given in (4.20)

$$\begin{aligned}
X_{11} &= \Delta t \frac{2\omega_1^2 \sin\chi}{m} & X_{12} &= \Delta t \frac{\rho S v_z \sqrt{v_x^2 + v_z^2}}{2m} & X_{13} &= \Delta t \frac{\rho S v_x \sqrt{v_x^2 + v_z^2}}{2m} \\
X_{21} &= \Delta t \frac{-2\omega_1^2 \cos\chi - 2\omega_2^2}{m} & X_{22} &= -\Delta t \frac{\rho S v_x \sqrt{v_x^2 + v_z^2}}{2m} & X_{23} &= -\Delta t \frac{\rho S v_z \sqrt{v_x^2 + v_z^2}}{2m} \\
X_{31} &= \Delta t \frac{2l(\omega_1^2 - \omega_2^2)}{I_{yy}} & X_{32} &= 0 & X_{33} &= 0
\end{aligned} \tag{4.20}$$

In (4.20), Δt is the discrete time step. Higher-order polynomial models for the unknown

parameters can be used in (4.19). This would lengthen the unknown vector η and widen the matrix of regressors $\mathbf{X}$. A polynomial model is derived in the next section of this chapter.

Solving (4.1) for η gives (4.21)

$$\begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{bmatrix}_k^{\dagger} \left\{ \begin{bmatrix} \dot{v}_x \\ \dot{v}_z \\ \dot{\omega} \end{bmatrix}_{k+1} - \begin{bmatrix} v_x \\ v_z \\ \omega \end{bmatrix}_k - \Delta t \begin{bmatrix} -\omega_y v_z - g\sin\theta \\ \omega_y v_x + g\cos\theta \\ 0 \end{bmatrix}_k \right\} = \begin{bmatrix} C_T \\ C_L \\ C_D \end{bmatrix}_k \tag{4.21}$$

Here the $\mathbf{X}^{\dagger}$ is the pseudo inverse of the regressor matrix. This is necessary because $\mathbf{X}$ is not guaranteed to be invertible. Stacking all of the data points up in (4.21) creates a regressor matrix with dimensions $3 \times 4n$, where n is the number of data points collected. This tall matrix is used to compute the LSR solution for η expressed in (4.21).

Using this method allows the unknown parameters to result in values which may not make physical sense due to the lack of constraints on the parameters. To constrain the parameters to retain physical meaning, this research poses a Quadratic Programming optimization problem in (4.22).

$$\begin{aligned} \min \quad & \frac{1}{2}\eta^T \mathbf{H}\eta + f^T\eta \\ \text{s.t.} \quad & \underline{\eta} \leq \eta \leq \overline{\eta} \end{aligned} \tag{4.22}$$

Where, $\underline{\eta}$, $\overline{\eta}$ are lower and upper bounds of η; $\mathbf{H}$ is the Hessian and is expressed as $\mathbf{H} = 2\mathbf{X}^T\mathbf{X}$, and $f = -2\mathbf{X}^T\mathbf{X}^{\dagger}$. This leads to a cost function which is equivalent to (4.14). The derivation follows as (4.23)

$$\begin{aligned} & (\mathbf{X}^{\dagger} - \mathbf{X}\eta)^T(\mathbf{X}^{\dagger} - \mathbf{X}\eta) \\ = \quad & ({\mathbf{X}^{\dagger}}^T - \eta^T\mathbf{X}^T)(\mathbf{X}^{\dagger} - \mathbf{X}\eta) \\ = \quad & {\mathbf{X}^{\dagger}}^T\mathbf{X}^{\dagger} + \eta^T\mathbf{X}^T\mathbf{X}\eta - \eta^T\mathbf{X}^T\mathbf{X}^{\dagger} - {\mathbf{X}^{\dagger}}^T\mathbf{X}\eta \end{aligned} \tag{4.23}$$

Because $\mathbf{X}^{\dagger^T}\mathbf{X}^{\dagger}$ does not depend on η, it does not affect the problem. The resulting quadratic programming problem becomes (4.24)

$$\begin{aligned} &\eta^T\mathbf{X}^T\mathbf{X}\eta - 2\mathbf{X}^{\dagger^T}\mathbf{X}\eta \\ &= \frac{1}{2}\eta^T(2\mathbf{X}^T\mathbf{X})\eta + (-2\mathbf{X}^T\mathbf{X}^{\dagger})^T\eta \end{aligned} \tag{4.24}$$

This optimization problem can be solved using the quadprog function in MATLAB's optimization toolbox [41]. This allows the resulting vector η to be constrained to more physically meaningful values. Implementation of this technique can be seen in the LSR code function found in Appendix B.

4.3.2 Parameter Estimation of Polynomial Dependency

The derivation of the model assuming polynomial unknown parameters is similar to the constant case with some differences. For the polynomial case, (4.18) is the same except for the unknown parameters. The polynomial version of the unknown parameters is shown in (4.25), where the variable denoting the polynomial dependency is expressed as ν. This means that the unknown coefficients are polynomial functions of the variable ν. For example, ν could be assumed to be the absolute velocity of the vehicle which would mean that the unknown coefficients would be functions of the absolute velocity of the aircraft.

$$\begin{aligned} C_T &= a_1\nu^2 + a_2\nu + a_3 \\ C_L &= b_1\nu^2 + b_2\nu + b_3 \\ C_D &= c_1\nu^2 + c_2\nu + c_3 \end{aligned} \tag{4.25}$$

Here, a_i, b_i, and c_i, $i = 1, 2, 3$ are the unknown coefficients of the unknown parameters C_T, C_L, C_D. The derivation follows by modifying (4.19) as (4.26).

$$\begin{bmatrix} v_x \\ v_z \\ \omega \end{bmatrix}_{k+1} = \begin{bmatrix} v_x \\ v_z \\ \omega \end{bmatrix}_{k} + \Big[X_{3\times 9} \Big]_k \Big[\eta_{9\times 1} \Big]_k + \Delta t \begin{bmatrix} -\omega_y v_z - g\sin\theta \\ \omega_y v_x + g\cos\theta \\ 0 \end{bmatrix}_k \tag{4.26}$$

Here $X_{3\times 9}$ is the regressor matrix with dimension 3×9 and $\eta_{9\times 1}$ is the vector of unknowns with dimension 9×1. These are given as (4.27) and (4.28).

$$X_{3\times 9} = \begin{bmatrix} X_{11} & X_{12} & X_{13} & X_{14} & X_{15} & X_{16} & X_{17} & X_{18} & X_{19} \\ X_{21} & X_{22} & X_{23} & X_{24} & X_{25} & X_{26} & X_{27} & X_{28} & X_{29} \\ X_{31} & X_{32} & X_{33} & X_{34} & X_{35} & X_{36} & X_{37} & X_{38} & X_{39} \end{bmatrix} \tag{4.27}$$

$$\eta_{9\times 1} = [a_1, a_2, a_3, b_1, b_2, b_3, c_1, c_2, c_3]^T \tag{4.28}$$

The elements of the regressor matrix $\mathbf{X}_{3\times 9}$ are given in (4.29)

$$\begin{aligned}
&X_{11} = (\Delta t)\lambda\nu^2 && X_{12} = (\Delta t)\lambda\nu && X_{13} = (\Delta t)\lambda \\
&X_{14} = (\Delta t)\xi\nu^2 v_z && X_{15} = (\Delta t)\xi\nu v_z && X_{16} = (\Delta t)\xi v_z \\
&X_{17} = (\Delta t)\xi\nu^2 v_x && X_{18} = (\Delta t)\xi\nu v_x && X_{19} = (\Delta t)\xi v_x \\
&X_{21} = (\Delta t)\mu\nu^2 && X_{22} = (\Delta t)\mu\nu && X_{23} = (\Delta t)\mu \\
&X_{24} = (\Delta t)\xi\nu^2 v_x && X_{25} = (\Delta t)\xi\nu v_x && X_{26} = (\Delta t)\xi v_x \\
&X_{27} = (\Delta t)\xi\nu^2 v_z && X_{28} = (\Delta t)\xi\nu v_z && X_{29} = (\Delta t)\xi v_z \\
&X_{31} = (\Delta t)\zeta && X_{32} = (\Delta t)\zeta && X_{33} = (\Delta t)\zeta \\
&X_{34} = 0 && X_{35} = 0 && X_{36} = 0 \\
&X_{37} = 0 && X_{38} = 0 && X_{39} = 0
\end{aligned} \tag{4.29}$$

Here, $\{\lambda, \xi, \mu, \zeta, \nu\}$ are given as (4.30) respectively. It should be noted that the expression for ν is not restricted to that of absolute velocity. However, this work uses $\nu = V$, where V is absolute velocity.

$$\begin{aligned}
\lambda &= \frac{2\omega_1^2 \sin\chi}{m} \\
\xi &= \frac{S\rho}{2m}\nu \\
\mu &= \frac{2(\omega_1^2 \cos\chi + \omega_2^2)}{m} \\
\zeta &= \frac{2l(\omega_1^2 - \omega_2^2)}{I_{yy}} \\
\nu &= \sqrt{v_x^2 + v_z^2}
\end{aligned} \tag{4.30}$$

The rest of the unknown polynomial parameter derivation is similar to the constant unknown parameter case and follows from (4.21). Because the unknown coefficients in this derivation take polynomial form, the constraints in (4.22) need to be modified. It is considered that the parameter is separated in its range, with three values $\nu_{max}, \nu_{mid}, \nu_{min}$ which must be evaluated. The constraints are shown in (4.31).

The formulation of the constraints can be done by first expanding (4.25) to (4.31) and then rearranging to give the form shown in (4.32).

$$\begin{aligned}
C_T(\nu_{max}) &= a_1\nu_{max}^2 + a_2\nu_{max} + a_3 \leq \kappa_1 \\
C_T(\nu_{mid}) &= a_1\nu_{mid}^2 + a_2\nu_{mid} + a_3 \leq \kappa_2 \\
C_T(\nu_{min}) &= a_1\nu_{min}^2 + a_2\nu_{min} + a_3 \leq \kappa_3 \\
C_L(\nu_{max}) &= b_1\nu_{max}^2 + b_2\nu_{max} + b_3 \leq \kappa_4 \\
C_L(\nu_{mid}) &= b_1\nu_{mid}^2 + b_2\nu_{mid} + b_3 \leq \kappa_5 \\
C_L(\nu_{min}) &= b_1\nu_{min}^2 + b_2\nu_{min} + b_3 \leq \kappa_6 \\
C_D(\nu_{max}) &= c_1\nu_{max}^2 + c_2\nu_{max} + c_3 \leq \kappa_7 \\
C_D(\nu_{mid}) &= c_1\nu_{mid}^2 + c_2\nu_{mid} + c_3 \leq \kappa_8 \\
C_D(\nu_{min}) &= c_1\nu_{min}^2 + c_2\nu_{min} + c_3 \leq \kappa_9
\end{aligned} \tag{4.31}$$

$$\mathbf{A}\eta \leq \kappa \tag{4.32}$$

Here, $\mathbf{A}$ is a 9×9 matrix of maximum, middle, and minimum values of the polynomial variable which each unknown coefficient is a function of. The symbol κ is a 9×1 vector of bounding values. Expanding (4.32) yields (4.33), which can be directly implemented in quadratic programming with constraints in MATLAB.

$$\begin{aligned}
\mathbf{A} &= \begin{bmatrix} \mathbf{\Pi} & \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} \\ \mathbf{0}_{3\times 3} & \mathbf{\Pi} & \mathbf{0}_{3\times 3} \\ \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} & \mathbf{\Pi} \end{bmatrix} \\
\mathbf{\Pi} &= \begin{bmatrix} \nu_{max}^2 & \nu_{max} & 1 \\ \nu_{mid}^2 & \nu_{mid} & 1 \\ \nu_{min}^2 & \nu_{min} & 1 \end{bmatrix} \\
\eta &= [a_1, a_2, a_3, b_1, b_2, b_3, c_1, c_2, c_3]^T \\
\kappa &= [\kappa_1, \kappa_2, \kappa_3, \kappa_4, \kappa_5, \kappa_6, \kappa_7, \kappa_8, \kappa_9]^T
\end{aligned} \tag{4.33}$$

These constraints are used to constrain the quadratic programming optimization problem described previously. This way, the unknown parameters can be constrained to something which has physical meaning. Without the constraints, the LSR algorithm can cause the unknown coefficients to take values that have no physical meaning.

CHAPTER 5
AIRCRAFT CONTROL AND PID GAIN OPTIMIZATION

5.1 Vehicle Control Algorithm

Because USUUA1 begins the flight as a quadcopter and finishes the transition as a fixed-wing aircraft, it is necessary to understand the high-level control algorithm of the Ardupilot software. This is especially true for when the aircraft is in quadcopter configuration because quadcopters are inherently unstable. For this reason, this chapter focuses on quadcopter mode controllers and algorithms. USUUA1 runs the Ardupilot firmware/software for flight control. The flight control software uses nested PID loops to control the aircraft. Fig. 5.1 shows the control strategy of USUUA1 on a high level. This is Ardupilot's high-level control algorithm.

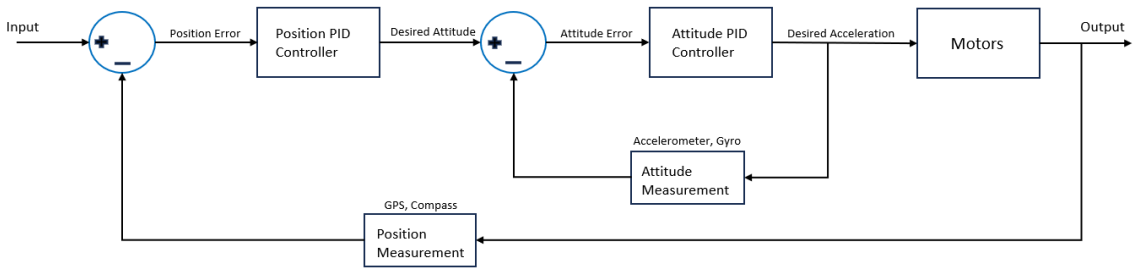


Fig. 5.1: High-level control algorithm for USUUA1.

Ardupilot's position and navigation controller uses nested PID loops to control the position and navigation of the aircraft [2]. Fig. 5.2 shows the Ardupilot position and navigation control loops.

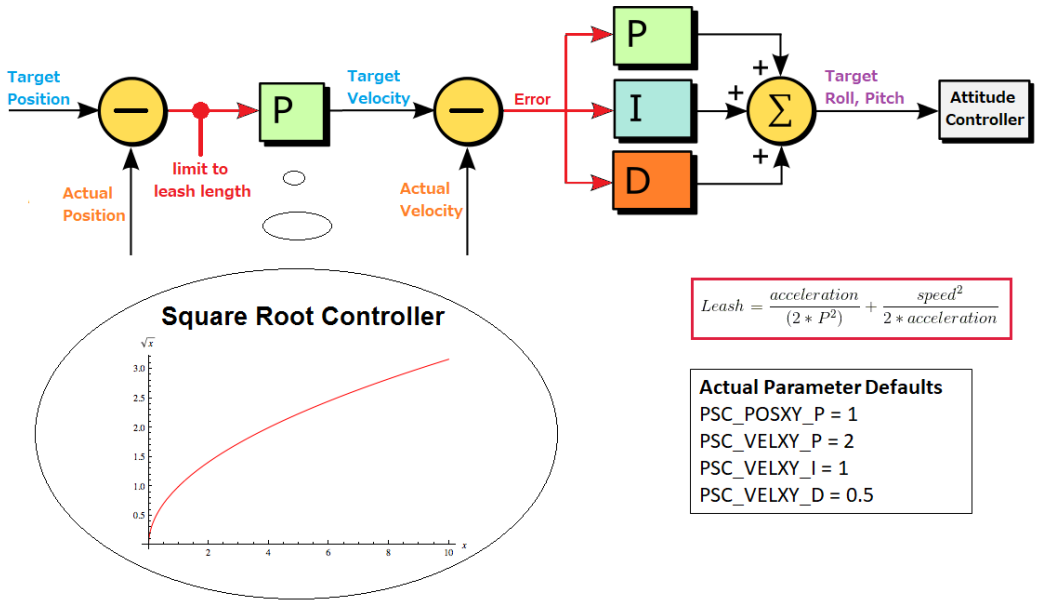


Fig. 5.2: Control loops for the position controller from Ardupilot [2].

Here, "leash" refers to a virtual "rubber-band-like" guide for the aircraft [2]. To visualize what is going on in this controller one can think of the vehicle attached to one end of a rubber band. The other end is constantly moving to where the desired position is and the vehicle follows as if it were attached via the rubber band. Position in the x and $y$axies are controlled by taking a position and passing it through a P controller to get a desired velocity. The desired velocity is then passed through a PID controller to get the desired acceleration. The desired lean angle is then calculated based on the desired acceleration value and the result is passed to the attitude controller.

The control algorithm for attitude also uses nested PID control loops [3]. A diagram of the attitude controller is shown in A.1. The attitude controller has an outer loop for controlling the position and an inner loop for controlling the velocities of the aircraft. Fig. 5.3 shows the Ardupilot attitude control loops.

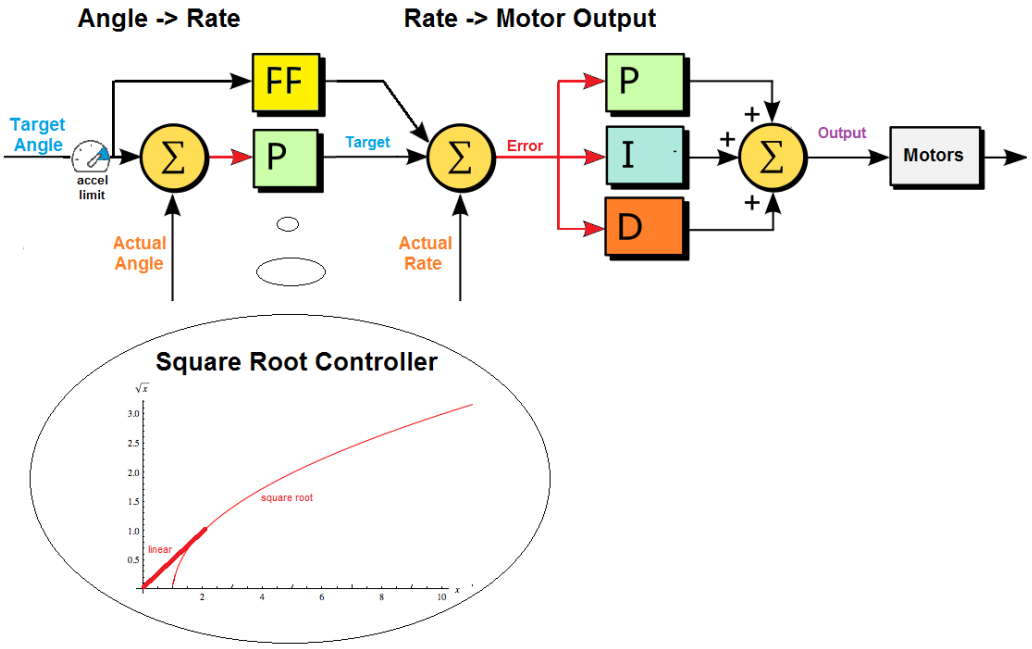


Fig. 5.3: Control loops for the attitude controller from Ardupilot [3].

Here, the feed-forward term, or FF term, is used to help smooth the response of the vehicle [3]. This allows the rate loop to be able to correct for the FF term separately from The output of the angle loop. The square root controller for the angle loop is essentially a proportional or, P, controller but it is an acceleration limiting approach to a P controller which means this control strategy takes into account the acceleration rate limits of the aircraft. Large errors can be handled with this strategy. This control algorithm is Ardupilot's low-level control. Above this is the position and navigation controller.

Altitude, or z axis, is controlled similarly to that of the x and y axes. The input z position error is passed through a P controller to convert the error to a desired velocity. The desired velocity is then passed through another P controller to convert the error to a desired acceleration. Desired acceleration is then passed through a PID controller to convert the acceleration to a throttle value. This throttle value is then sent to the attitude controller. Vehicle performance and stability can be tuned via the Mission Planner GUI.

5.2 Tuning PID Gain Values

User access to the controller parameters is provided in the Mission planner through the tuning GUI which is found under the config tab shown in Fig. 5.4.

Fig. 5.4: Mission Planner GUI for tuning parameters [3].

From here, the pilot can make changes to the PID gain values and tune the aircraft controllers. The manufacturer of the USUUA1 airframe [4] provides a parameter file with the PID gain values tuned for their aircraft. However, the vehicle corresponding to the provided tune could be significantly different than USUUA1. Therefore, the USUUA1 pilot conducted a manual tune on the quadcopter configuration of the aircraft. Values for the fixed-wing PID gains from the manufacture were found to be acceptable without further tuning.

Ardupilot has an autotune feature within the Mission Planner, however, it requires a basic, manual, tune for it to work properly. Even after conducting an autotune on an

aircraft, the system may need small changes to be made for a better tune. Because of this, the pilot of USUUA1 uses the simple manual tune for USUUA1. The tuning process is detailed in the Ardupilot documentation [42].

5.3 PID Gain Optimization

Rather than rely on an onboard auto-tune from Ardupilot or a manual tune for USUUA1, part of this work is to optimize the PID gain values for settling time. This can be accomplished using a PID tuner from MATLAB [43]. This PID controller with optimized gain values for settling time is intended to give insight into the control parameters. Applying the optimized PID gain values to the physical hardware is left as future work for this research.

To simplify the state space model for control purposes, this work aims to control the error between the predicted states of the aircraft and the measured states of the aircraft. This is done by identifying the deviations, or error states, in the dynamic model of the aircraft. This is done by deriving the error dynamics. The dynamic model of the error can be derived using (5.1).

$$\begin{aligned} \dot{\mathbf{x}} &= f(\mathbf{x}, \mathbf{u}) \\ \dot{\mathbf{x}}^* &= f(\mathbf{x}^*, \mathbf{u}^*) \\ \mathbf{x}_e &= \mathbf{x}_e - \mathbf{x}_e^* \\ \dot{\mathbf{x}}_e &= f(\mathbf{x}, \mathbf{u}) - f(\mathbf{x}^*, \mathbf{u}^*) \end{aligned} \tag{5.1}$$

Here, $\mathbf{x}$ is the state vector, $\mathbf{x}^*$, is the predicted state vector, $\mathbf{u}$ is the vector of inputs, and $\mathbf{u}^*$ is the predicted vector of inputs. It can be observed from (5.1) that the change in error is defined by the difference of the actual dynamics and the predicted dynamics. Using a 1^{st} order Taylor expansion approximation on the dynamics gives (5.2)

$$f(x_1, x_2, ...) \approx f(x_1^*, x_2^*, ...) + \frac{\partial f}{\partial x_1} e_{x_1} + \frac{\partial f}{\partial x_2} e_{x_2} + ... \tag{5.2}$$

Combining (5.2) with (5.1) gives (5.3).

$$\dot{\mathbf{x}}_e = \frac{\partial f}{\partial \mathbf{x}^*}\mathbf{x}_e + \frac{\partial f}{\partial \mathbf{u}^*}\mathbf{u}_e \tag{5.3}$$

Before applying (5.1), (5.2), and (5.3) to the equations of motion previously derived for USUUA1, some assumptions and simplifications are made:

1. small angle approximation i.e. $\cos\theta \approx 1$ and $\sin\theta \approx \theta$.
2. Tilt angle χ is a scheduled, non-perturbed value.
3. v_z can be approximated as constant.
4. v_x is 0 at the beginning of the transition flight.

Applying these and (5.3) to the equations of motion for USUUA1 results in (5.4).

$$\begin{aligned}
\dot{\mathbf{e}}_x &= -\mathbf{e}_w\mathbf{v}_z - g\mathbf{e}_\theta + \frac{2C_T}{m}[(\sin\chi)\mathbf{e}_{u1}] + (\frac{\partial f}{\partial \mathbf{v}_x}|_{vx=vx_0})\mathbf{e}_x + (\frac{\partial f}{\partial \mathbf{v}_z}|_{vz=vz_0})\mathbf{e}_z \\
\dot{\mathbf{e}}_z &= -\frac{2C_T}{m}[(\cos\chi)\mathbf{e}_{u1} + \mathbf{e}_{u2}] + (\frac{\partial f}{\partial \mathbf{v}_x}|_{vx=vx_0})\mathbf{e}_x + (\frac{\partial f}{\partial \mathbf{v}_z}|_{vz=vz_0})\mathbf{e}_z \\
\dot{\mathbf{e}}_\theta &= \mathbf{e}_\omega \\
\dot{\mathbf{e}}_\omega &= \frac{2C_T l}{m}(\mathbf{e}_{u1} - \mathbf{e}_{u2})
\end{aligned} \tag{5.4}$$

Further simplification can be made by considering the control for only the pitch angle of the aircraft. Because (5.4) is a linear approximation, the state space model can be derived as (5.5)

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{5.5}$$

Where $\{\dot{\mathbf{x}}, \mathbf{A}, \mathbf{B}, \mathbf{u}\}$ are shown in (5.6)

$$\begin{bmatrix} \dot{\theta}_e \\ \dot{\omega}_e \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \theta_e \\ \omega_e \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{2C_T l}{I_{yy}} \end{bmatrix} [\mathbf{u1} - \mathbf{u2}] \tag{5.6}$$

CHAPTER 6
RESULTS AND ANALYSIS

6.1 LSR Assuming Constant Parameter Estimation

The transition period data is collected by executing multiple aircraft transitions. The constraints for the unknown parameters are $\underline{\eta} = [0, -1, 0]^T$ and $\overline{\eta} = [1, 1, 1]^T$. Each transition corresponds to a different max tilt angle and max tilt speed as shown in Table 2.1. Constants required for calculations are given in Table 6.1.

Table 6.1: Constant values used in LSR calculations.

Constant	**Values**
$\Delta t\,[s]$	0.01
$g\,[m/s^2]$	9.81
$l\,[m]$	0.362
$m\,[kg]$	11
$\rho\,[kg/m^3]$	0.72
$S\,[m^2]$	0.725

Fig. 6.1 shows the transition data of the system outputs and Fig. 6.2 shows the system inputs.

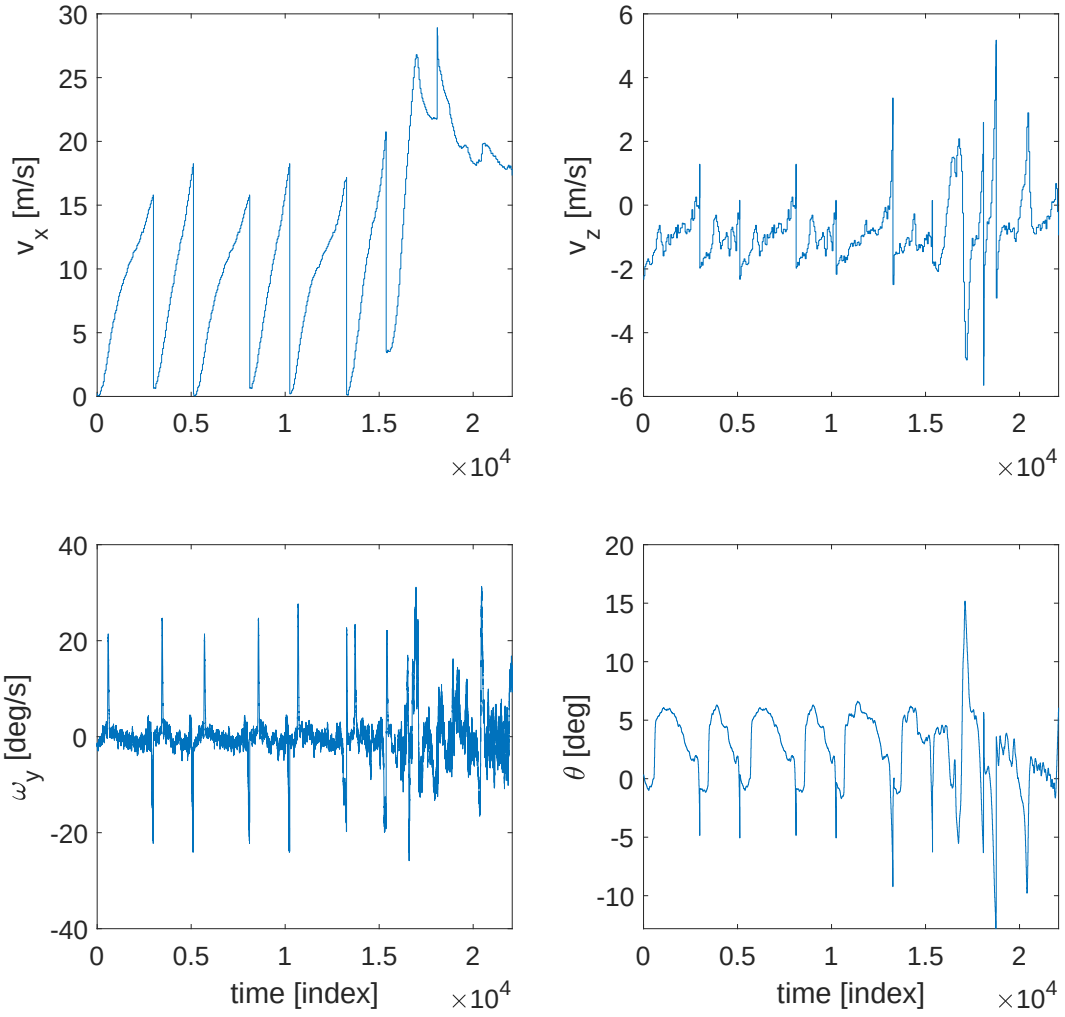


Fig. 6.1: Plots of states v_x, v_z, ω_y, and θ.

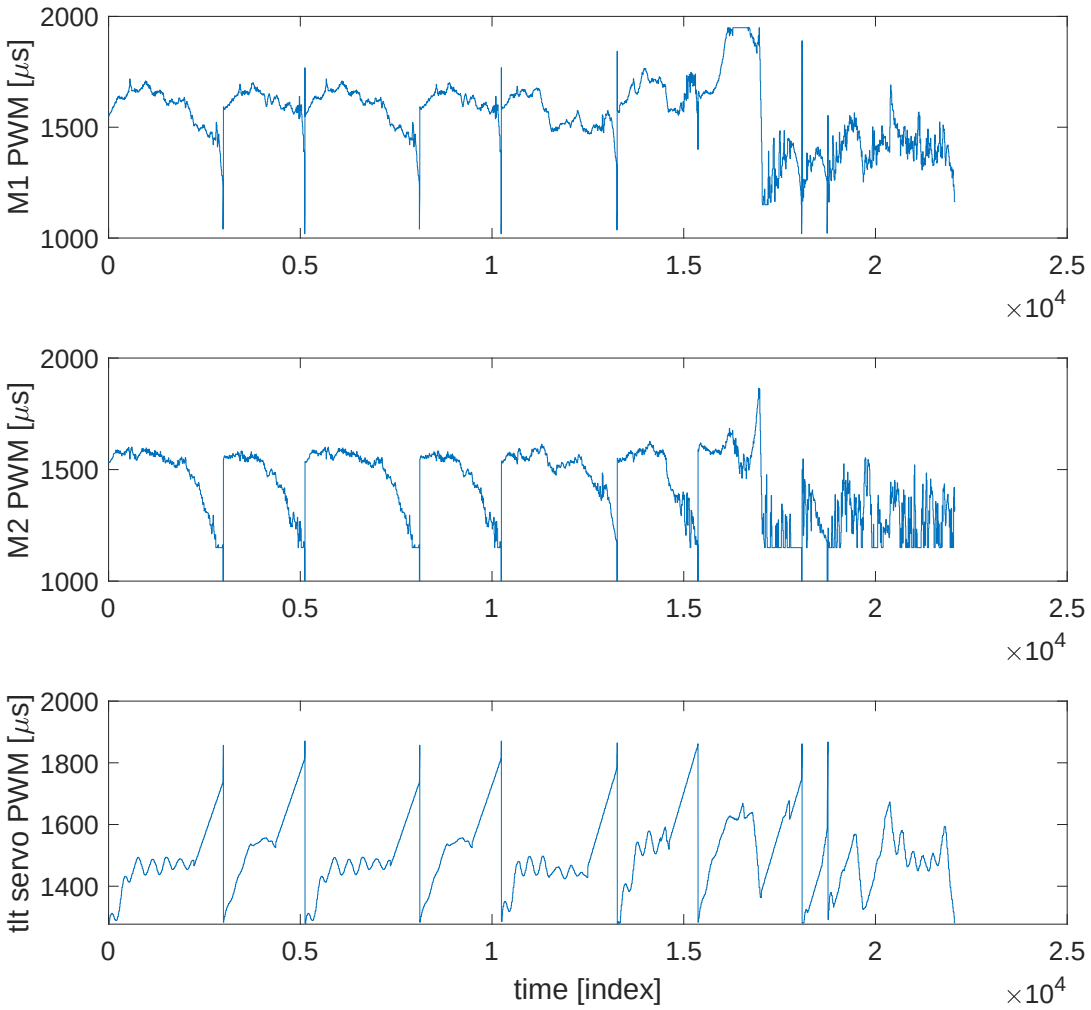


Fig. 6.2: Plots of inputs of rotor speeds and servos.

System inputs are M1 (motor 1 and 3 signals), M2 (motor 2 and 4 signals), and tlt servo PWM (tilt servo signals). System outputs are speed in the x direction v_x, speed in the z direction v_z, pitch rate ω_y, and pitch angle θ. From the flight data collected, v_x is estimated using ground speed. This is done because it's a relatively accurate way to estimate the airspeed of the vehicle, without an airspeed sensor. An airspeed sensor was not used because of reliability issues.

The label $[\mu s]$ is used to denote units of micro-seconds and the unit $[index]$ is used to denote the time index of the virtual time vector. These units describe the PWM signal and the virtual time vector respectively. The term index refers to the virtual time step of the data. For example, index 1 is an integer value used to denote the first time step. This is done for visualization purposes. The actual time stamp for each test flight is measured from the time the flight controller was powered on and is only used in this work during the post-processing algorithm to unify the data vectors.

68% of the collected data is used to estimate the unknown parameters using the LSR algorithm and the remaining 32% is used as validation data to compare the model to experimental data. This results in at least two transition periods used for validation. The resulting unknown parameters are given in Table 6.2.

Table 6.2: Unknown coefficient values assuming constant model.

Coefficients	**Values**
C_T	2.41×10^{-6}
C_L	0.72
C_D	0.23

The associated error, assuming constant coefficients, is 0.0047. This error comes from applying the Relative Root Mean Square Error (RRMSE) (6.1).

$$\begin{aligned} error &= |\mathbf{x_{k+1}} - \mathbf{x_k} - \mathbf{r_k} - \mathbf{X}\eta| \\ \mathbf{x_{k+1}} &= [v_{x_{k+1}}^T, v_{z_{k+1}}^T]^T \\ \mathbf{x_k} &= [v_{x_k}^T, v_{z_k}^T]^T \\ \mathbf{r_k} &= [-\Delta t(\omega_{y_k} v_{z_k} + gsin(\theta_k))] \\ RRMSE &= \sqrt{\frac{\frac{1}{n}\sum_{k=1}^{n}(error)^2}{\sum_{k=1}^{n}|\mathbf{x_{k+1}} - \mathbf{x_k} - \mathbf{r_k}|^2}} \end{aligned} \tag{6.1}$$

Here, the matrix $\mathbf{X}$, the vector η, and Δt are the matrix of regressors, the vector of identified, unknown, parameters, and the time step respectively. Implementation of the error calculation can be found in Appendix B. Using the validation data, the resulting states of the aircraft are predicted using the LSR model. Figs. 6.3 and 6.4 show the predicted model compared to the experimental data.

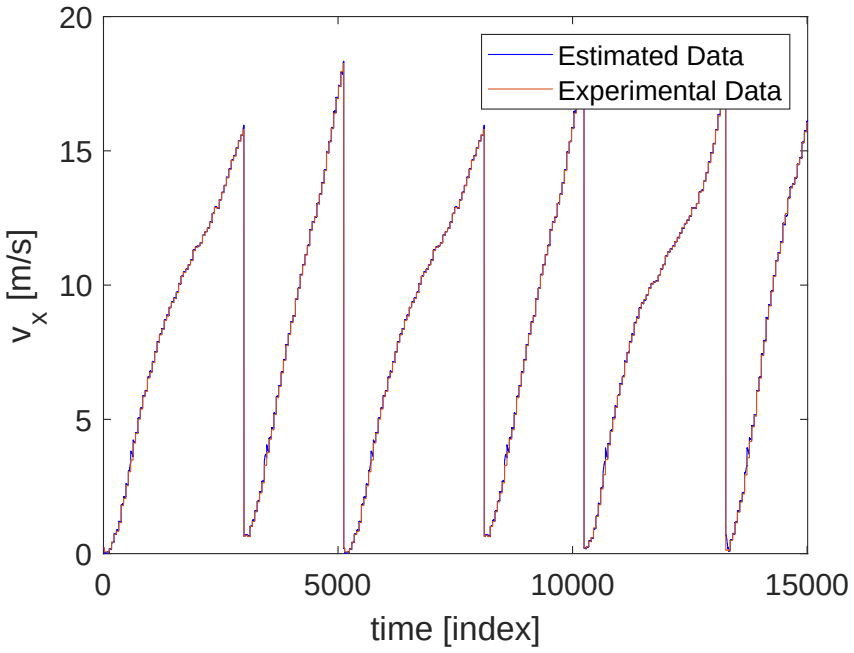


Fig. 6.3: Comparison of the velocity in the x direction assuming constant coefficients.

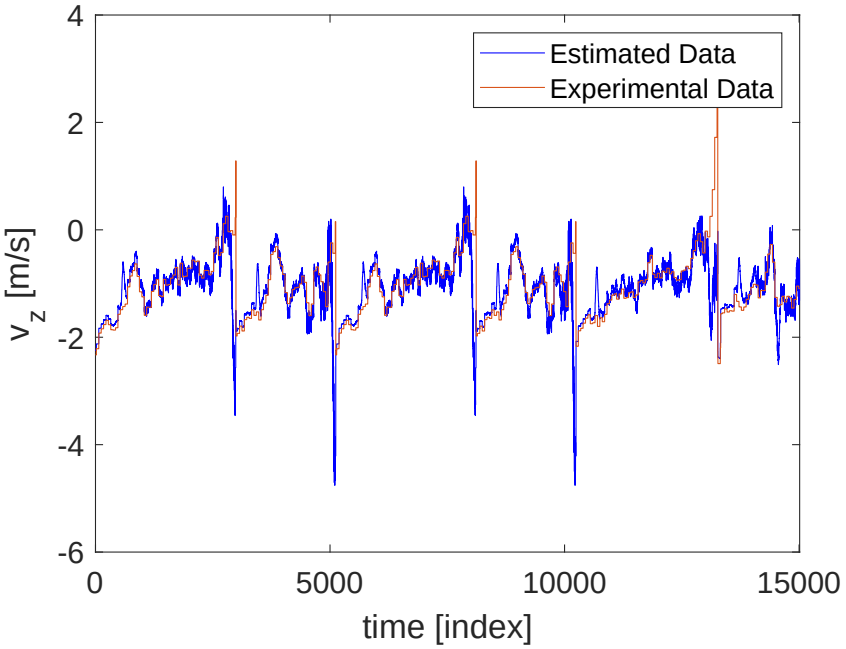


Fig. 6.4: Comparison of the velocity in the z direction assuming constant coefficients.

The plot for pitch rate, ω_y, is not included here. This is because it's a near perfect match with the experimental data using the methods in this work. This occurs because the equation for ω_y depends only on the motor speed inputs of the system and not on the states of the system. Constant values for the system are given in the code in Appendix B and also in Table 6.1.

Using validation data, the predicted model compared to the experimental model is shown in Figs. 6.5 and 6.6. This corresponds to the remaining 32% of the collected data.

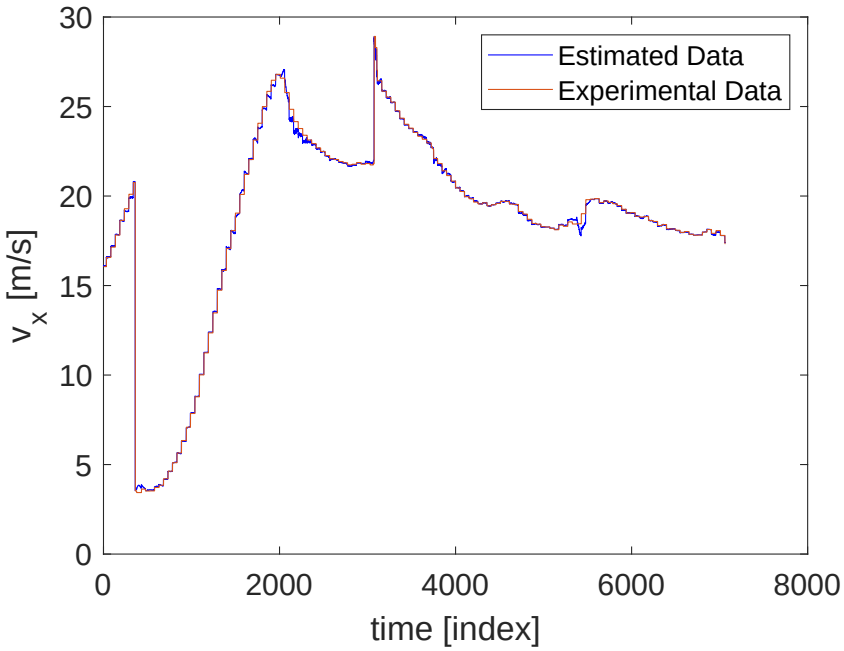


Fig. 6.5: Validation of the velocity in the x direction assuming constant coefficients.

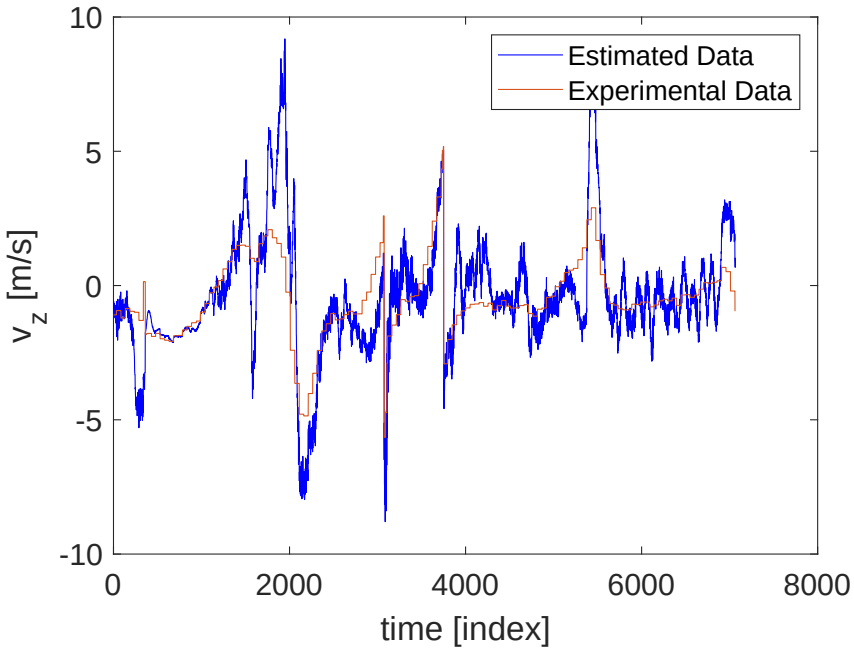


Fig. 6.6: Validation of the velocity in the z direction assuming constant coefficients.

Fig. 6.7 shows the resulting unknown coefficients for each test flight assuming the coefficients are constants.

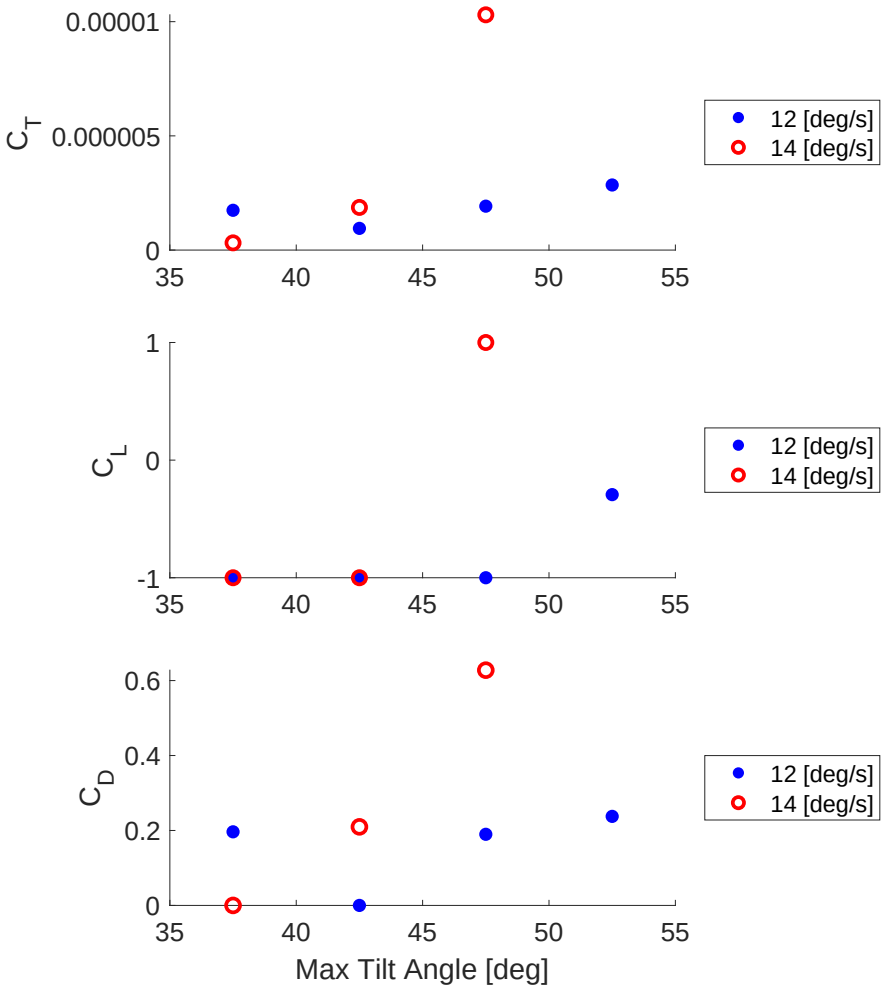


Fig. 6.7: C_T, C_L, C_D coefficients varying with tilt speeds.

These results correspond to data from Test 1.1 through 2.3. The transition data from these test flights are fed into the LSR algorithm shown in Appendix B to determine the unknown coefficients.

Fig. 6.7 shows results from assuming the unknown vector values to be constant. However, this assumption can be relaxed to allow the unknown parameters to take polynomial form. By relaxing the constant parameter assumption, the constant case can be compared to the polynomial case.

6.2 LSR Assuming Polynomial Parameter Estimation

This section shows data resulting after relaxing the constant coefficient assumption to allow the unknown coefficient model to take polynomial form. Specifically, the unknown coefficients are assumed to take a 2^{nd} order polynomial form. The unknown coefficients are assumed to be functions of absolute velocity. Constraints for the polynomial case have $\nu_{max} = 25$ m/s, $\nu_{mid} = 12.5$ m/s, $\nu_{min} = 0$ m/s, where ν is the absolute velocity of the aircraft. The bounding vector is $\kappa = [0, 0, 0, 1, 1, 1, 0, 0, 0]$. Figs. 6.8 and 6.9 show the estimated model compared to the experimental data. This corresponds to 68% of the collected data.

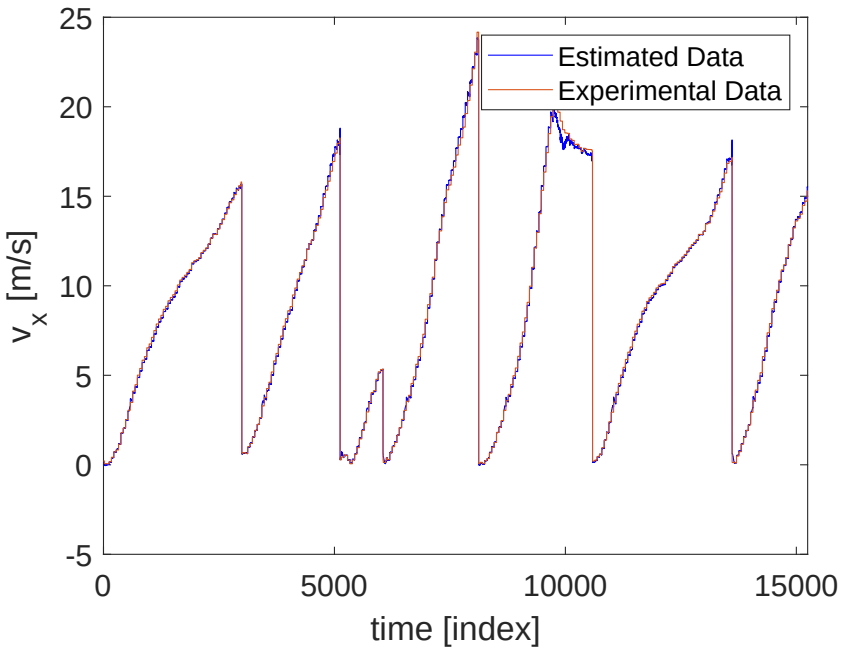


Fig. 6.8: Comparison of the velocity in the x direction assuming polynomial coefficients.

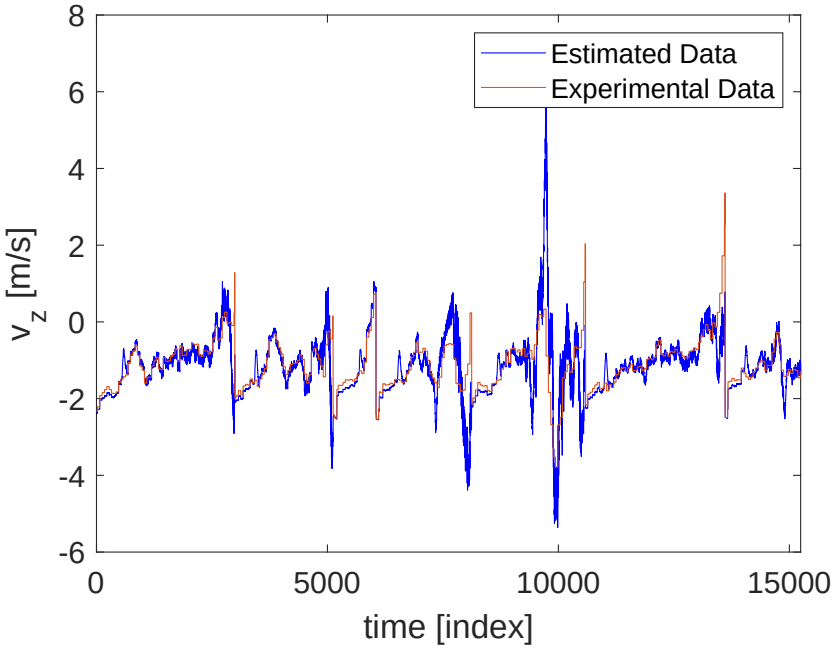


Fig. 6.9: Comparison of the velocity in the z direction assuming polynomial coefficients.

Figs. 6.10 and 6.11 show the validation between the estimated model and experimental data using the remaining 32% of the collected data. The resulting unknown coefficients are shown in Table 6.3.

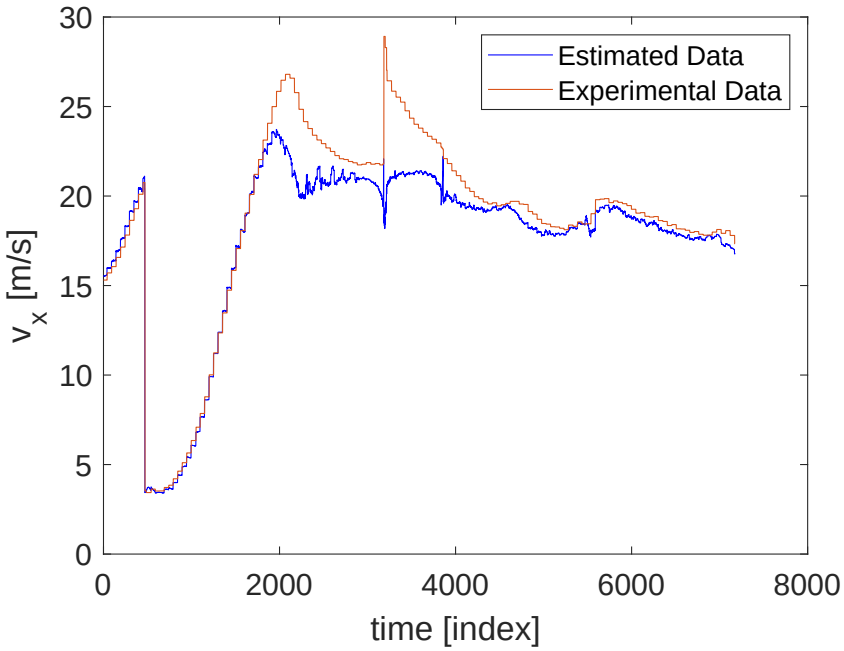


Fig. 6.10: Validation of the velocity in the x direction assuming polynomial coefficients.

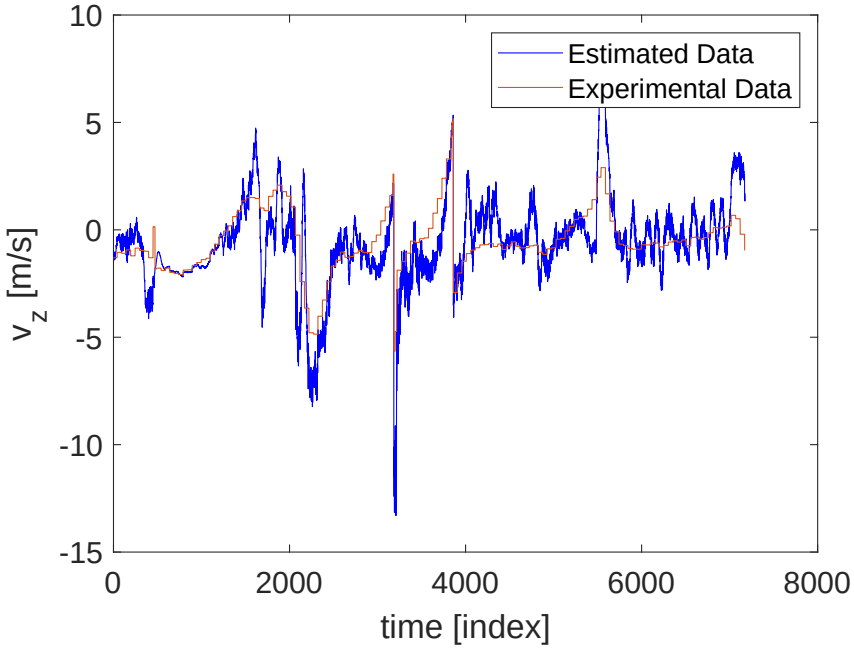


Fig. 6.11: Validation of the velocity in the z direction assuming polynomial coefficients.

Table 6.3: Unknown coefficient values assuming polynomial model.

Unknown Coefficients			
C_T	$a_1 = 5.56 \times 10^{-6}$	$a_2 = -8.07 \times 10^{-5}$	$a_3 = 2.07 \times 10^{-4}$
C_L	$b_1 = 0.17$	$b_2 = -6.48$	$b_3 = 53.01$
C_D	$c_1 = 0.049$	$c_2 = -0.50$	$c_3 = 4.47 \times 10^{-14}$

The polynomial model for the unknown coefficients has $RRMSE = 0.0059$. Figs. 6.12–6.14 show the plots for the unknown coefficients of the system assuming a polynomial model. The state plots and model comparison plots for each individual test flight, both for the constant coefficient and polynomial coefficient models, can be found in Appendices A.3 and A.4 respectively.

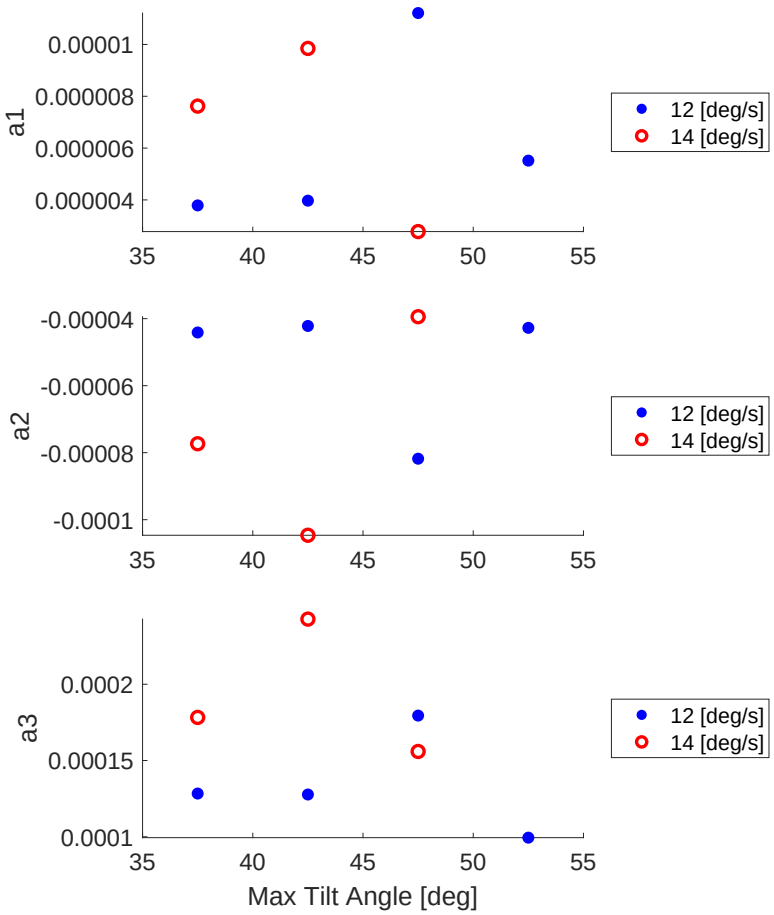


Fig. 6.12: C_T coefficients varying with tilt speed.

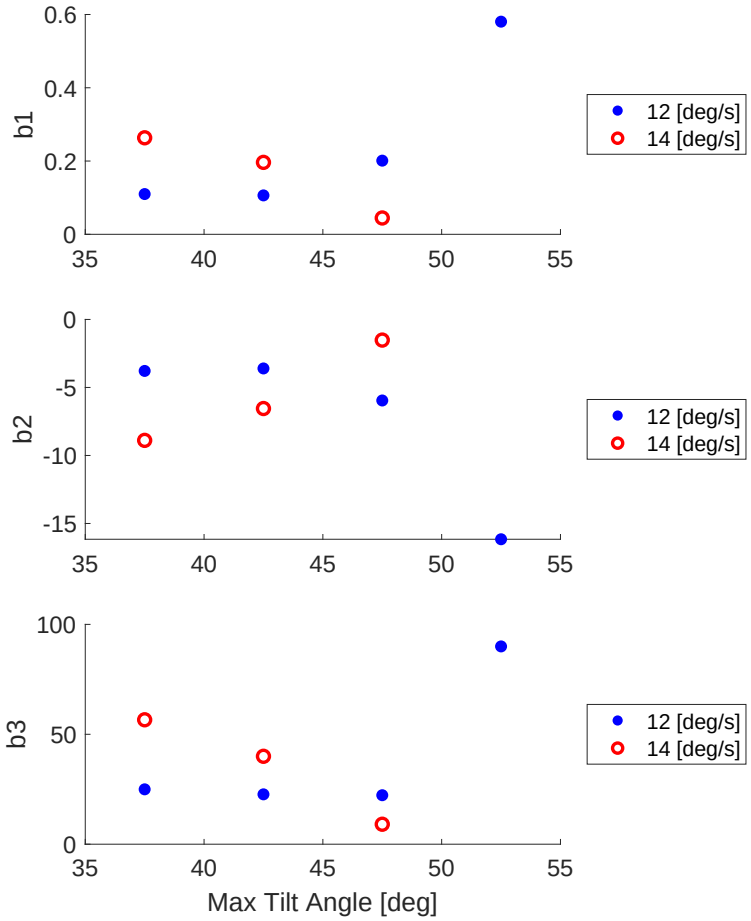


Fig. 6.13: C_L coefficients varying with tilt speed.

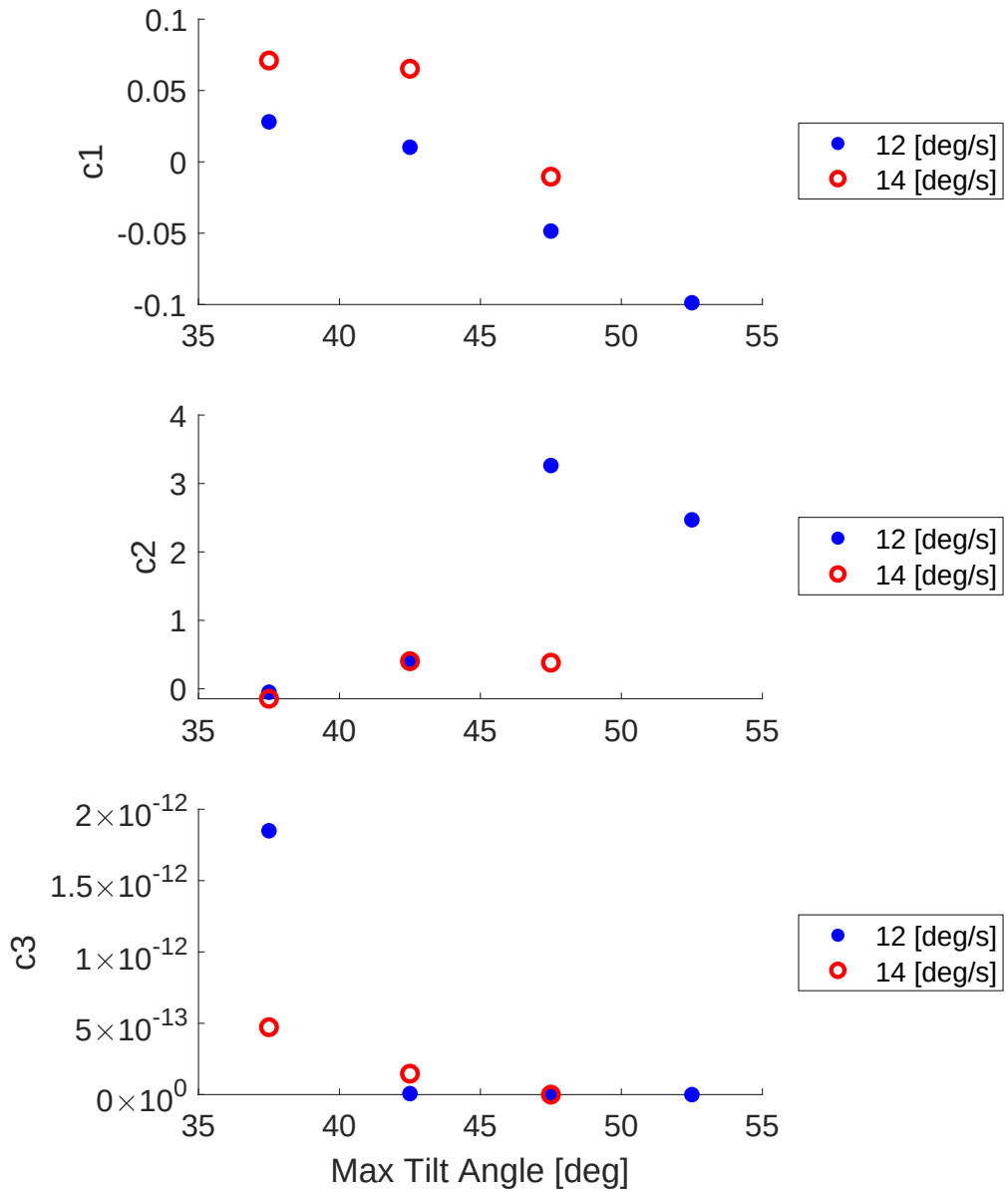


Fig. 6.14: C_D coefficients varying with tilt speed.

6.3 PID Gain Value Optimization Assuming Constant Model

Using the MATLAB PID Tuner, the gain values are optimized to reduce the settling time of the system. The resulting PID gain values for the system at the beginning of transition, with control matrix $B = [0, 4.36 \times 10^{-6}]^T$, are shown in Table 6.4 where P, I, D, and N are the proportional, integral, derivative, and notch filter gains respectively.

Table 6.4: Constant model PID values at the beginning of transition.

PID Gains		**System Performance**	
P	4136448.19	Rise Time [s]	0.15
I	1512150.32	Settling Time [s]	1.23
D	1803307.62	Overshoot [%]	14.3
N	1142.83	Peak	1.14

The controlled signal for the pitch angle is the difference between the front and back rotor PWM signals squared. Nominal PWM signals are between 1000 (armed) and 2000 (max) μs. When squared this difference becomes order 10^6 which is the approximate order of the PID gain values at the beginning of transition. This is because, at the beginning of transition, the aircraft needs to have a large difference between the front and back rotor speeds. The front rotors must spin much faster than the back rotors in order to keep the vehicle level and execute the transition. To compensate for these large gain values, a knockdown factor on the order of 10^{-6} could be added to the controller to account for this and result in more "nominal" gain values. The response of the system at the beginning of transition is shown in Fig. 6.15.

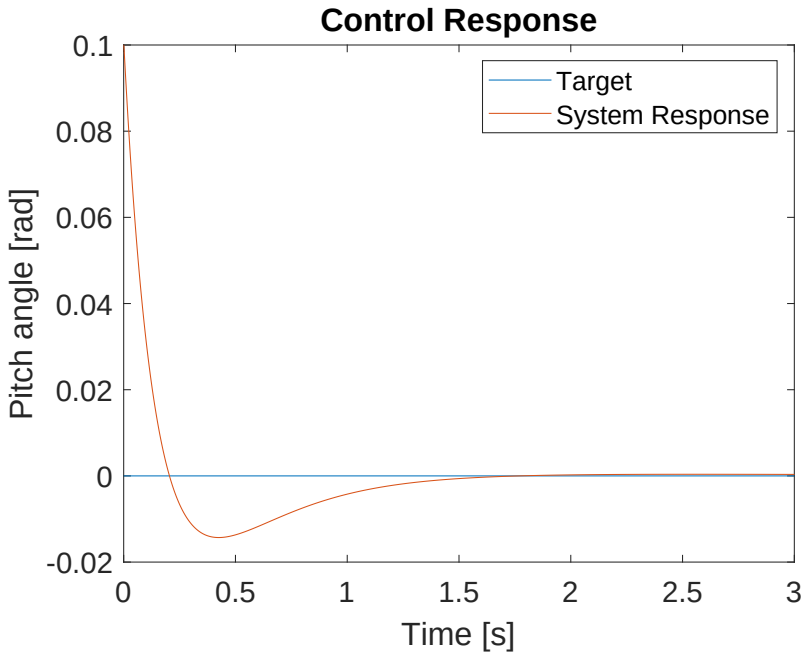


Fig. 6.15: PID control response of pitch angle at the beginning of transition.

The resulting PID gain values for the system at the end of transition, with input matrix $B = [0, 20.34]^T$, are shown in Table 6.5. The response of the system at the end of transition is shown in Fig. 6.16.

Table 6.5: Constant model PID values at the end of transition.

PID Gains		**System Performance**	
P	1.11	Rise Time [s]	0.15
I	0.404	Settling Time [s]	1.23
D	0.48	Overshoot [%]	14.3
N	1142.83	Peak	1.14

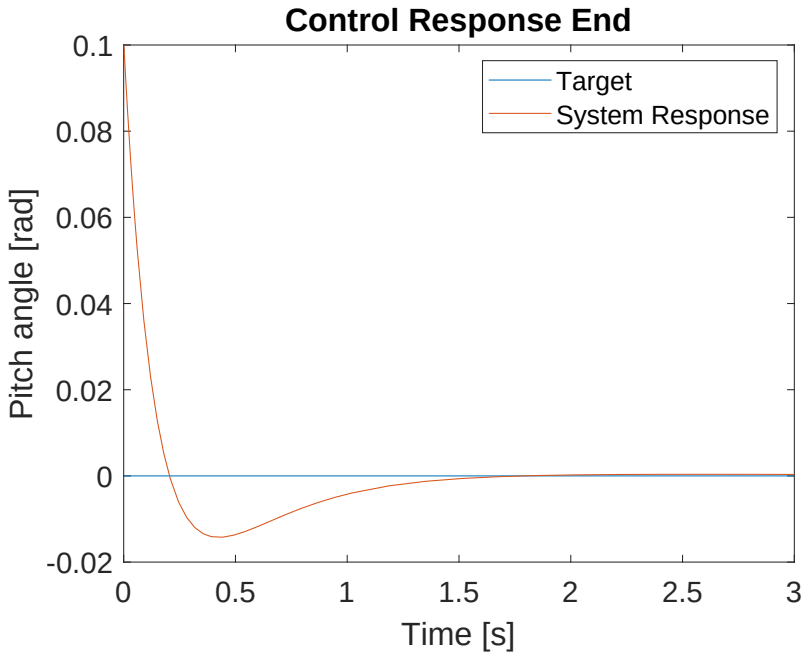


Fig. 6.16: PID control response of pitch angle at the end of transition.

6.4 PID Gain Value Optimization Assuming Polynomial Model

The optimized gain values for the polynomial model at the beginning of transition, with input matrix $B = [0, 8.79 \times 10^{-5}]$, are shown in Table 6.6. The response of the system at the beginning of transition is shown in Fig. 6.17.

Table 6.6: Polynomial model PID values at the beginning of transition.

PID Gains		**System Performance**	
P	255987.84	Rise Time [s]	0.15
I	93580.79	Settling Time [s]	1.23
D	111599.32	Overshoot [%]	14.3
N	1142.83	Peak	1.14

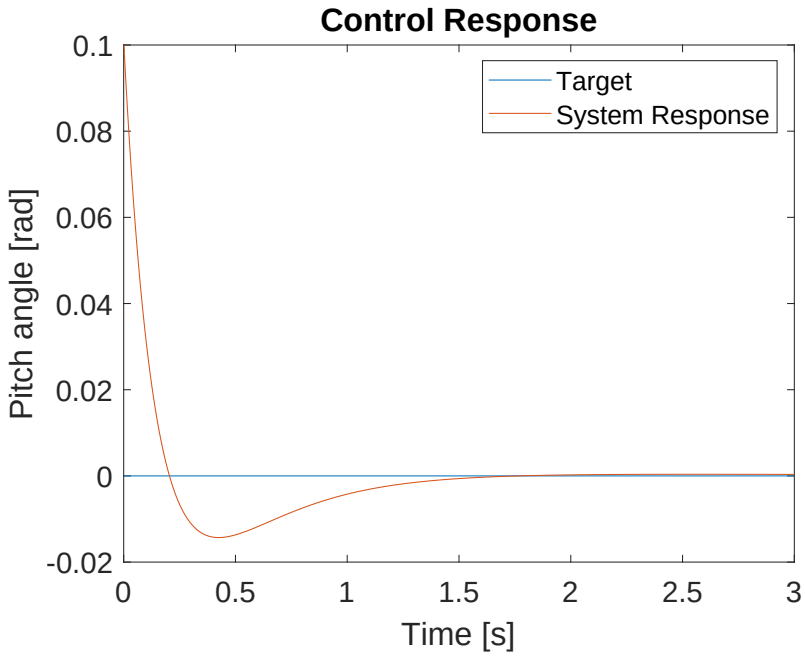


Fig. 6.17: PID control response of pitch angle at the beginning of transition.

The PID gain values at the end of transition, with input matrix $B = [0, 5.32 \times 10^{-4}]$, are shown in Table 6.7. The response of the system at the end of transition is shown in Fig. 6.18.

Table 6.7: Polynomial model PID values at the end of transition.

PID Gains		**System Performance**	
P	42321.13	Rise Time [s]	0.15
I	15467.93	Settling Time [s]	1.23
D	18446.21	Overshoot [%]	14.3
N	1142.83	Peak	1.14

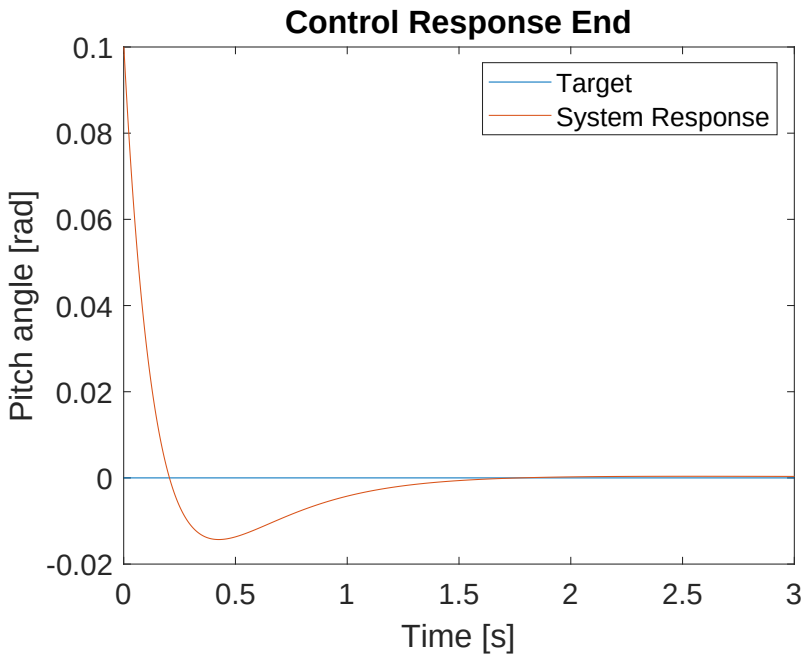


Fig. 6.18: PID control response of pitch angle at the end of transition.

CHAPTER 7
LESSONS LEARNED, CONCLUSIONS, AND FUTURE WORK

7.0.1 Lessons Learned From Aircraft Build

Some challenges with the build of the aircraft were part adhesion, wire sizing, electronic connectors, and component location. The USUUA1 airframe is mainly comprised of foam, plastic, and carbon fiber. Plastic interfaces are used to secure the foam pieces of the aircraft. a foam-safe glue is used to adhere the plastic interfaces to the foam parts. Simply applying foam glue to the parts and then pressing them together does not create a strong enough bond to connect the parts. To increase the bond strength, light sanding or scoring should be applied to the plastic and foam parts before the glue is applied. This improves the adhesion of the glue with glued parts.

Another challenge is to size the wiring between all of the electronic components correctly. If the wiring gauge is too small then the wire can become hot enough to melt the shielding during flight and melt surrounding components. To avoid this, the wires should be sized according to the manufacturer's specifications. A good rule of thumb for COTS parts is the connecting wire should be the same gauge as the wiring that comes with each part. While this may not be optimal, it can reduce the risk of melting electronic components and connectors.

Because of the many electronic components within USUUA1, there are many electrical connections. In many cases, it is best to use correctly soldered and insulated connections between components. However, due to the experimental nature of the hardware, it was necessary to connect and disconnect various components multiple times. Quick connectors, such as bullet connectors, helped solve this modularity issue with USUUA1.

7.0.2 Lessons Learned From Aircraft Test

When testing the aircraft and its subsystems it is best to start small. This means that testing should be conducted on each subsystem individually before testing the system as a whole. This helps troubleshoot problems with each subsystem. Many tests do not require the propellers or wings to be installed such as testing GPS and compass data. In these instances when certain components aren't required for a given test, it is best to remove or disable them until the test is completed. After, the components can be reinstalled or re-enabled to test subsystem integration.

7.1 Conclusions from Results

This section gives the conclusions that can be drawn from this research. These conclusions are based on the data and methods detailed in this work.

7.1.1 Data Processing Capability

Flight data collected from the aircraft can contain data for more than one flight. For this reason, various toggles were included in the data post-processing code found in Appendix B. These toggles can be used to analyze individual flight data. The code can extract the transition data from ".mat" files containing flight data. Because the various sensors onboard the aircraft can have different sampling rates, the code is also able to unify the vectors containing all of the data to the same size. After data post-processing, the code also calculates the LSR of the data and estimates the unknown parameters. This is done with no interaction from the user except to run the code and ensure all the flight data files are found in the working directory.

At the end of the driver code, there is also a section to develop a PID controller to control the pitch angle of the aircraft using Simulink. This functionality requires some interaction from the user in the current state of the code. The user must run the PID tuner to optimize the PID gain values. Future improvements to the code could help automate the PID control development process. Results from the LSR algorithm are displayed on the console as well as the calculated error.

7.1.2 LSR Results Analysis

Observing the comparison and validation plots of the states of the aircraft shows that both the constant and polynomial versions of the LSR match the experimental data relatively closely. Interestingly, the constant unknown parameter case appears to match better than the polynomial case. This suggests that the constant model is closer to the true dynamics of the vehicle than the polynomial case.

For both the constant and polynomial cases, the comparison for v_z does not match the experimental data as well as v_x. This could be due to the difference in magnitude between the two. It could also be attributed to measurement noise being greater for the velocity in the z-direction. Higher-quality sensors could improve accuracy and reduce errors due to measurement noises.

Estimation of the unknown parameters for the constant and polynomial cases resulted in typical values for C_L and C_D. However, C_T resulted in a value much smaller than anticipated. This may be due to the difference between the unknown parameter and typical values given in manufacturer data sheets for the coefficient of thrust. Dimensional analyses of the equations of motion reveal that the unknown parameter is not dimensionless and must have a length and mass unit associated with it. In contrast, coefficients of thrust are often reported by manufacturers as dimensionless like the coefficient of lift or drag. These coefficients of thrust often correspond to the propellers alone and not the propulsion system as a whole. The same discrepancy exists with the polynomial case but is not as clear due to the 2^{nd} order polynomial model. More investigation into the unknown parameter C_T is needed to understand the dimensional nature of the coefficient.

Identified parameter trends for the constant model, appear to have a similar pattern between the 12 deg/s and the 14 deg/s data. The trends for the 14 deg/s case seem to follow the 12 deg/s but shifted left. More data is needed to verify if there is any correlation between these trends. The trends for the identified polynomial model do not appear to have any distinguishable trend. This could be due to the actual dynamic model being more similar to the constant model than the polynomial model. Another explanation could be

due to neglecting changes to the airflow over the wing from the prop-wash of the tilt rotors during the transition. Adding these effects to the dynamic model could help reveal the effect this has on the aircraft during transition.

Based on the results, and the collected flight data in this work, it can be concluded that the unknown parameters in the dynamic model of USUUA1 are more closely modeled as constant values. These results are true for low-speed transitions with relatively slow tilting speeds as shown in this research.

7.1.3 PID Control and System Response Analysis

Based on the results, the PID gain values are much smaller at the end of the transition period than at the beginning. This is because the aircraft has transitioned to fixed-wing mode and no longer needs to control the aircraft's attitude with the difference between the front and back rotor speeds. After the transition, the aircraft attitude is controlled using the control surface deflections, which are assumed to be zero during the transition. This can be seen by observing the values for the input matrix at the beginning and end of the transition. The input matrix values are lower at the beginning of the transition and higher at the end. This requires higher PID gain values at the beginning of the transition than at the end. Creating a linear interpolation between the beginning and ending PID gain values could produce a linear gain schedule for the pitch angle controller. More development is needed to develop such a gain schedule.

7.2 Future Work and Refinements

There are many areas for potential improvement of this work. More research is needed in the future to fully understand the dynamics of the transition flight period of eVTOL aircraft. This future work includes the following:

1. Upgrade hardware and electronics to professional-grade components with redundant sensors.

2. Improve manual tune of PID gain values for data collection and include dynamic notch filter and the other filter tuning to reduce the amount of noise in the system.

3. Include lateral dynamics and flow change effects due to tilt-able rotors to the dynamic model for the aircraft.

4. Amplify data collection plan to include flight data for Q_TLT_MAX up to 70 deg and Q_TLT_RATE_DN up to 20 deg/s .

5. Compare the results of LSR results to that of a different identification algorithm .

6. Develop a gain scheduling control algorithm based on flight data for all states of the aircraft.

REFERENCES

[1] FLYROBO. (2019) Pixhawk 2.4.8 px4 32 bit flight controller with safety switch and buzzer for drone high quality. [Online]. Available: https://doc.makeflyeasy.com/#/zh-cn/freeman

[2] ArduPilot Dev Team. (2023) Quadcopter position control. [Online]. Available: https://ardupilot.org/dev/docs/code-overview-copter-poscontrol-and-navigation.html

[3] ——. (2023) Quadplanes. [Online]. Available: https://ardupilot.org/dev/docs/apmcopter-programming-attitude-control-2.html

[4] MakeFlyEasy Dev Team. (2023) Quadplanes. [Online]. Available: https://doc.makeflyeasy.com/#/zh-cn/freeman

[5] S. S. Husain, L. T. Rasheed, R. A. Mahmod, E. K. Hamza, N. M. Noaman, and A. J. Humaidi, "Design of robust controller for tail-sitter vtol aircraft," in *2023 IEEE 8th International Conference on Engineering Technologies and Applied Sciences (ICETAS)*, 2023, pp. 1–6.

[6] P. Kemao, D. Miaobo, M. C. Ben, C. Guowei, L. K. Yew, and H. Lee Tong, "Design and implementation of a fully autonomous flight control system for a uav helicopter," in *2007 Chinese Control Conference*, 2007, pp. 662–667.

[7] S. Ma and W. Wang, "The longitudinal modeling of a new concept v/stol uav in transition flight by using the method of system identification," in *2018 10th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)*, vol. 01, 2018, pp. 32–35.

[8] J. Belák and M. Hromčík, "Unified attitude control strategy for tilt-rotor vtol aircraft," in *2023 American Control Conference (ACC)*, 2023, pp. 2830–2835.

[9] P. Pradeep and P. Wei, "Energy optimal speed profile for arrival of tandem tilt-wing evtol aircraft with rta constraint," in *2018 IEEE CSAA Guidance, Navigation and Control Conference (CGNCC)*, 2018, pp. 1–6.

[10] T. E. Strand and J. Ennis, "V-22 osprey unprepared surface short takeoff and landing evaluation," in *2010 IEEE Aerospace Conference*, 2010, pp. 1–9.

[11] T. K. Moon and W. C. Stirling, *Mathematical Methods and Algorithms for Signal Processing*. Upper Saddle River, NJ: Prentice Hall, 2000, ch. 7, pp. 381–388.

[12] W. F. Phillips, *Mathematical Methods and Algorithms for Signal Processing*. Hoboken, N.J: Wiley, 2010, ch. 7,11, pp. 715–753,1037.

[13] B. M. Simmons, "System identification for evtol aircraft using simulated flight data," in *AIAA SciTech 2022 Forum*, 2022, p. 2409.

[14] G. J. Ducard and M. Allenspach, "Review of designs and flight control techniques of hybrid and convertible vtol uavs," *Aerospace Science and Technology*, vol. 118, p. 107035, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1270963821005459

[15] K. M. Kleinhesselink, "Stability and control modeling of tiltrotor aircraft," Ph.D. dissertation, 2007, copyright - Database copyright ProQuest LLC; ProQuest does not claim copyright in the individual underlying works; Última actualización - 2023-02-24. [Online]. Available: https://login.dist.lib.usu.edu/login?url=https://www.proquest.com/dissertations-theses/stability-control-modeling-tiltrotor-aircraft/docview/304857185/se-2

[16] S. Burton, T. He, and W. Su, "Trajectory planning and lpv model predictive control of tilt-rotor vtol aircraft." *2023 American Control Conference (ACC), American Control Conference (ACC), 2023*, pp. 2836 – 2841, 2023. [Online]. Available: https://dist.lib.usu.edu/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.10155861&site=eds-live

[17] V. R. Puttige and S. G. Anavatti, "Comparison of real-time online and offline neural network models for a uav," in *2007 International Joint Conference on Neural Networks*, 2007, pp. 412–417.

[18] Q. Li, "Grey-box system identification of a quadrotor unmanned aerial vehicle," Ph.D. dissertation, Citeseer, 2014.

[19] A. Rasheed, "Grey box identification approach for longitudinal and lateral dynamics of uav," in *2017 International Conference on Open Source Systems and Technologies (ICOSST)*, 2017, pp. 10–14.

[20] M. A. Jamil, M. Ahsan, M. J. Ahsan, and M. A. Choudhry, "Time domain system identification of longitudinal dynamics of a uav: A grey box approach," in *2015 International Conference on Emerging Technologies (ICET)*, 2015, pp. 1–6.

[21] J. H. N. Daniel Mårtensson, "Mataveid," https://github.com/DanielMartensson/Mataveid, 2020.

[22] W. Wei, G. Yin, and T. He, "Physics-informed data-based lpv modeling and validations of lateral vehicle dynamics," *IEEE Transactions on Intelligent Vehicles*, vol. 9, no. 1, pp. 2459–2468, 2024.

[23] W. Wei, W. Zhuang, Q. Gao, G. Yin, and T. He, "Online modeling of lateral vehicle dynamics via recursive integrated physics-data-based method," *IEEE Transactions on Intelligent Vehicles*, pp. 1–9, 2023.

[24] R. Song, Z. Ye, L. Wang, T. He, and L. Zhang, "Autonomous wheel loader trajectory tracking control using lpv-mpc," in *2022 American Control Conference (ACC)*. IEEE, 2022, pp. 2063–2069.

[25] C. T. Aksland and A. G. Alleyne, "Hierarchical model-based predictive controller for a hybrid uav powertrain," *Control Engineering Practice*, vol. 115, p.

104883, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S096706612100160X

[26] C. An, S. Jia, J. Zhou, and C. Wang, "Fast model-free learning for controlling a quadrotor uav with designed error trajectory," *IEEE Access*, vol. 10, pp. 79 669–79 680, 2022.

[27] Z. Chekakta, M. Zerikat, Y. Bouzid, and M. Abderrahim, "Model-free control applied for position control of quadrotor using ros," in *2019 6th International Conference on Control, Decision and Information Technologies (CoDIT)*, 2019, pp. 1260–1265.

[28] S. Qu, T. He, and W. Su, "Mixed model predictive control and data-driven control of a tiltrotor evtol aircraft," in *AIAA SCITECH 2024 Forum*, 2024, p. 0517.

[29] B. C. Multon, "3d-printed morphing wings for controlling yaw on flying-wing aircraft," Master's thesis, Utah State University, Logan, UT, 2021.

[30] ArduPilot Dev Team. (2023) Copter attitude control. [Online]. Available: https://ardupilot.org/plane/docs/quadplane-support.html

[31] Spektrum. (2023) Nx8 8-channel dsmx transmitter only. [Online]. Available: https://www.spektrumrc.com/product/nx8-8-channel-dsmx-transmitter-only/SPMR8200.html

[32] ——. (2023) Ar8020t dsmx 8-channel telemetry receiver. [Online]. Available: https://images.search.yahoo.com/search/images;_ylt=Awr9zjf5qMNl5FgDmxNXNyoA;_ylu=Y29sbwNncTEEcG9zAzEEdnRpZAMEc2VjA3BpdnM-?p=Spektrum+AR8020T&fr2=piv-web&type=C210US662D20161212&fr=mcafee#id=0&iurl=https%3A%2F%2Fimages-na.ssl-images-amazon.com%2Fimages%2FI%2F71BY114FUSL._AC_SL1500_.jpg&action=click

[33] Flex RC. (2023) Tattu 16000mah 30c 22.2v 6s lipo battery pack with xt90-s plug. [Online]. Available: https://flexrc.com/product/tattu-16000mah-30c-22-2v-6s-lipo-battery-pack-with-xt90-s-plug/

[34] MakeFlyEasy Dev Team. (2023) Quadplanes. [Online]. Available: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/http://fw.makeflyeasy.com/Freeman/Document%20%20Freeman%20Installation%20Notes.pdf

[35] ArduPilot Dev Team. (2023) Magnetic interference. [Online]. Available: https://ardupilot.org/plane/docs/common-magnetic-interference.html#magnetic-interference

[36] ——. (2023) Choosing a ground station. [Online]. Available: https://ardupilot.org/plane/docs/common-choosing-a-ground-station.html#mission-planner

[37] ——. (2023) Quadplane setup tips. [Online]. Available: https://ardupilot.org/plane/docs/quadplane-tips.html#tilt-rotor-setup

[38] ——. (2023) Quadcopter position control. [Online]. Available: https://ardupilot.org/plane/docs/new-roll-and-pitch-tuning.html#filtering

[39] S. S. Husain, L. T. Rasheed, R. A. Mahmod, E. K. Hamza, N. M. Noaman, and A. J. Humaidi, "Design of robust controller for tail-sitter vtol aircraft," in *2023 IEEE 8th International Conference on Engineering Technologies and Applied Sciences (ICETAS)*, 2023, pp. 1–6.

[40] C. Kaya and J. Martínez, "Euler discretization and inexact restoration for optimal control." *Journal of Optimization Theory and Applications*, vol. 134, no. 2, pp. 191–206 – 206, 2007. [Online]. Available: https://dist.lib.usu.edu/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edselc&AN=edselc.2-52.0-35148872918&site=eds-live

[41] Quadprog. MathWorks. Accessed on 2024-01-30. [Online]. Available: https://www.mathworks.com/help/optim/ug/quadprog.html?searchHighlight=quadprog&s_tid=srchtitle_support_results_1_quadprog

[42] ArduPilot Dev Team. (2023) Quadcopter position control. [Online]. Available: https://ardupilot.org/plane/docs/common-tuning.html

[43] Matlab documentation. MathWorks. Accessed on 2024-01-30. [Online]. Available: https://www.mathworks.com/help/control/ref/pidtuner-app.html

APPENDICES

APPENDIX A

SUPPORTING INFORMATION FOR USUUA1

A.1 Supporting USUUA1 Information

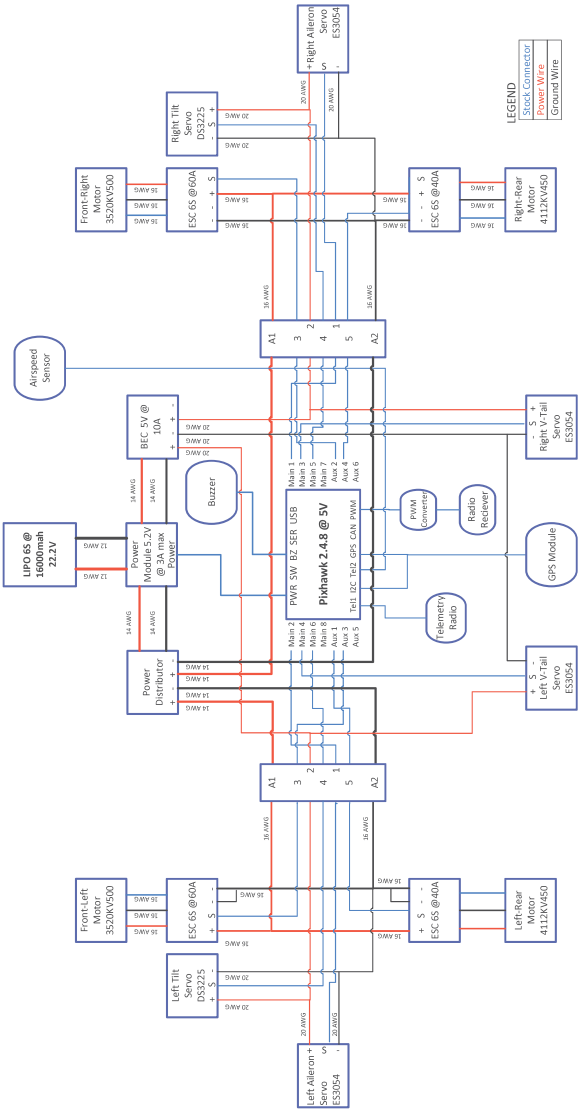


Fig. A.1: USUUA1 system wiring diagram modified from [4].

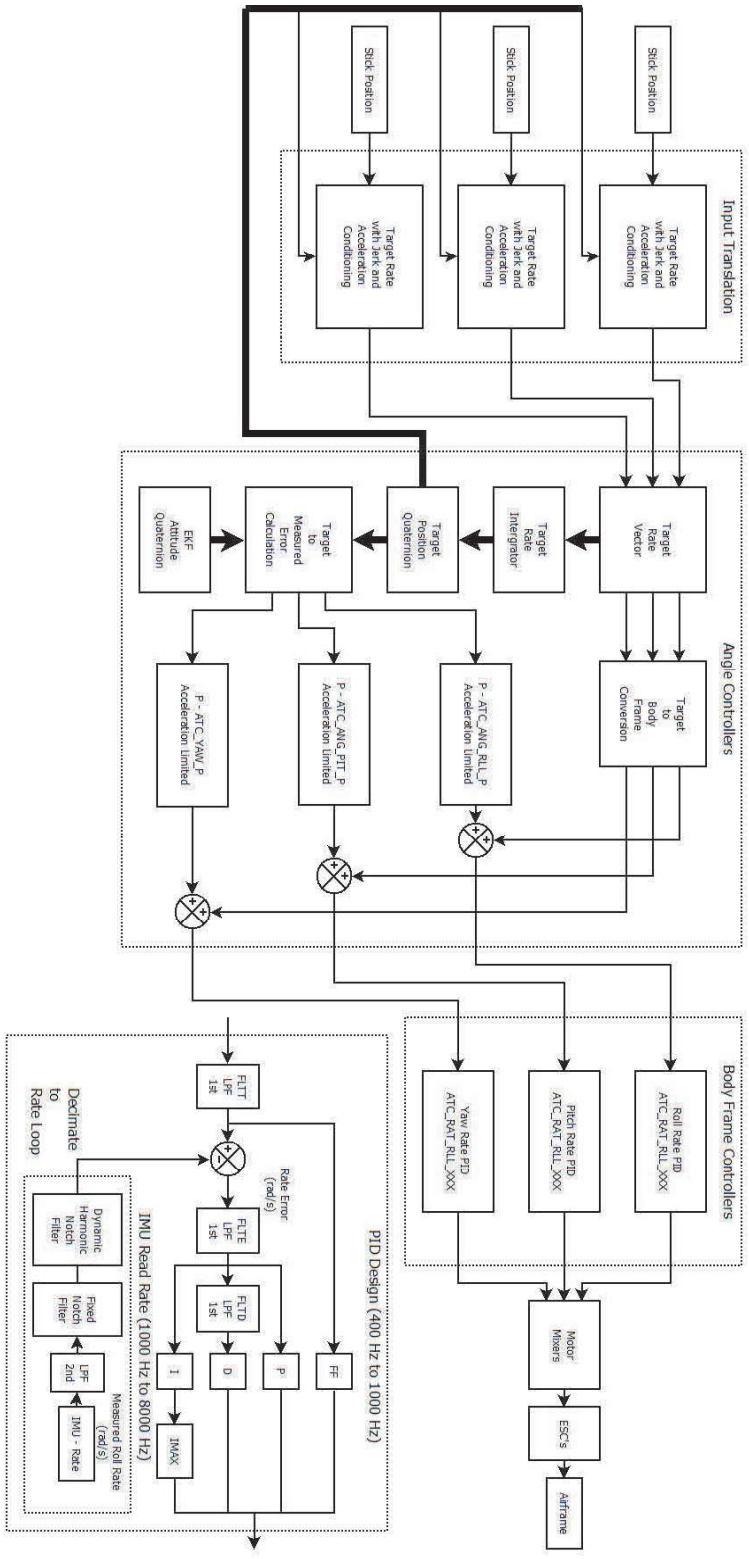


Fig. A.2: Control diagram for USUUA1 taken from [3].

Table A.1: Pre-flight checklist for USUUA1.

Pre-flight Checklist	
#	**Description**
1	Connect telemetry radio to ground station and open Mission Planner.
2	Connect power to the fuselage of the aircraft and wait till the startup process is completed.
3	Press the connect button in Mission Planner to connect to the vehicle via telemetry radio.
4	Wait until the Pixhawk achieves a 3D GPS fix indicated by the HUD, audio tone, and green blinking LED.
5	Begin the compass calibration in Mission Planner. Rotate at least 360 degrees on all 6 sides. Repeat the calibration if compass errors occur.
6	Verify the IMU measurements by opening the raw data window in Mission Planner and rotating the vehicle approximately ± 30 degrees for the pitch and roll axes. Rotate to North, East, South, and West directions in the yaw axis.
7	Disconnect power to the aircraft and connect the wings, tail pieces, and tail servos.
8	Repeat Steps 2-4.
9	Ensure props will not contact the ground when rotated down then change the flight mode to manual or a fly-by-wire mode.
10	Using the radio transmitter, verify the control surface directions. Change directions if needed in Mission Planner.
11	Change the flight mode back to a qloiter or similar quadcopter mode.
12	verify the flight mission in Mission Planner if applicable.
13	Arm the aircraft. After arming, if flying an autonomous mission, switch to auto mode and the aircraft will begin flight.

A.2 Component Locator Boards

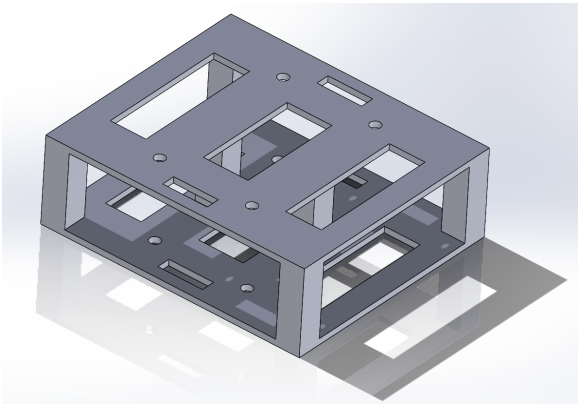


Fig. A.3: flight control locator board that secures the Pixhawk flight controller.

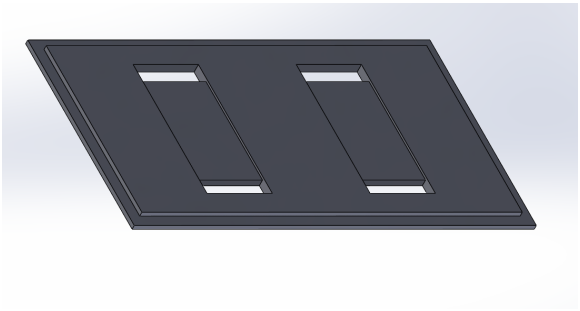


Fig. A.4: Locator board to secure the battery to the fuselage of USUUA1.

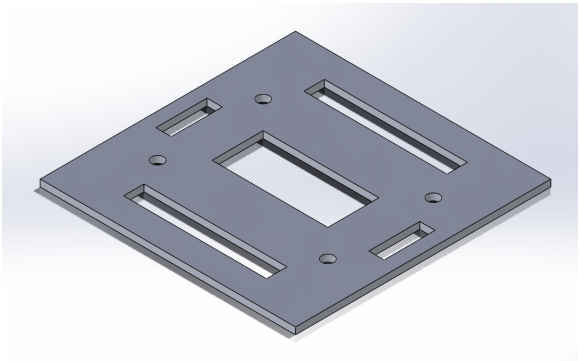


Fig. A.5: Locator board that secures the power distribution board and the BEC.

A.3 Flight Data and LSR Results Assuming Constant Coefficients

This section shows state plots and model comparisons for flight tests using the constant unknown coefficient LSR model. Figs. A.6–A.9 show the state outputs, state input, v_x comparison, and v_z comparison plots.

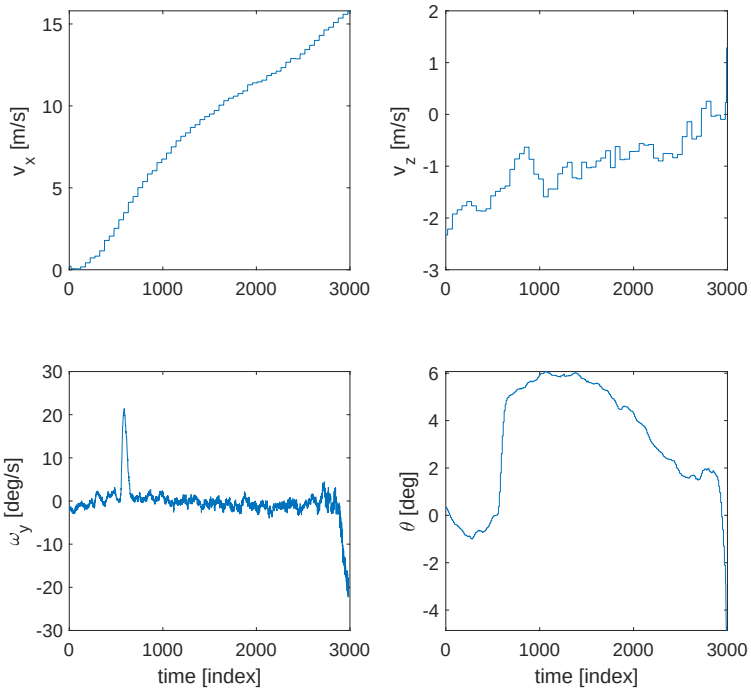


Fig. A.6: Test1.1 state plots.

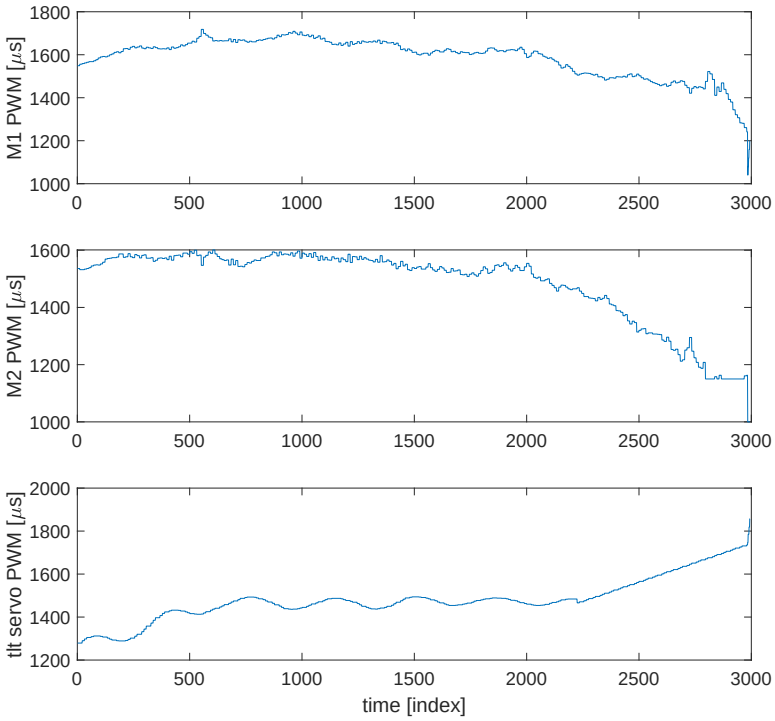


Fig. A.7: Test1.1 input plots.

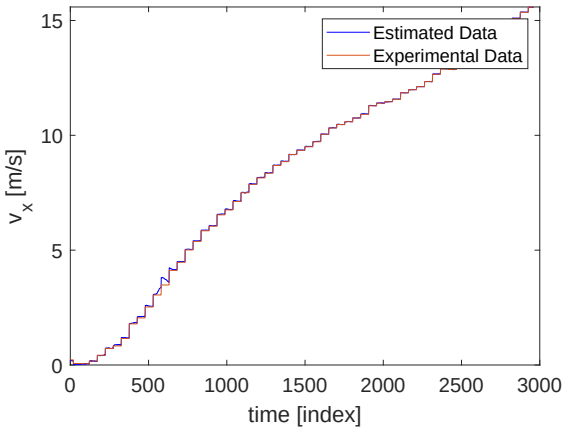


Fig. A.8: Test1.1 comparison of the velocity in the x direction assuming constant coefficients.

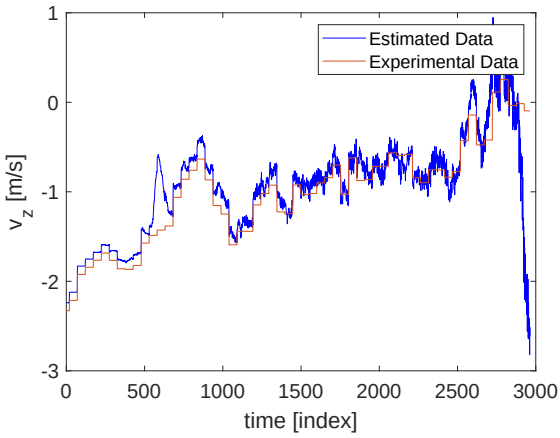


Fig. A.9: Test1.1 comparison of the velocity in the z direction assuming constant coefficients.

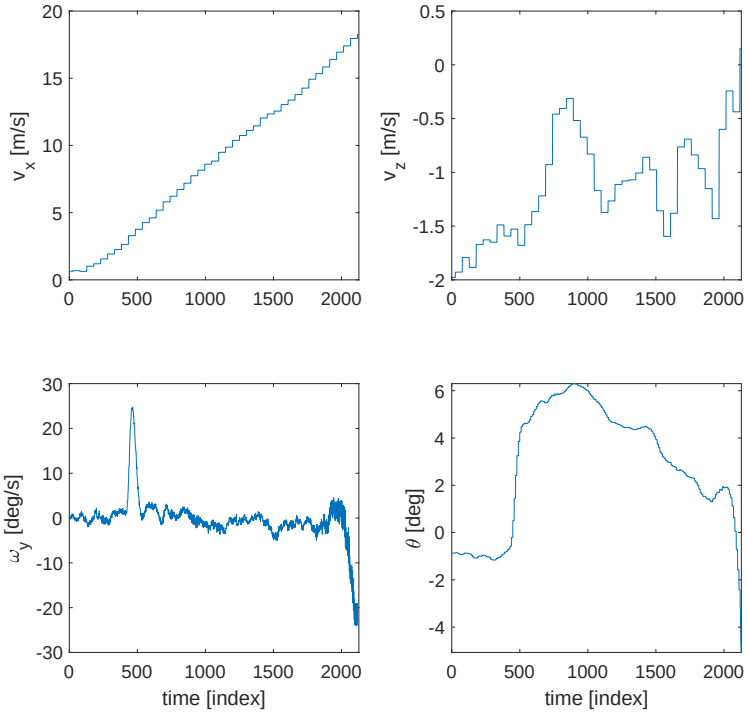


Fig. A.10: Test1.2 state plots.

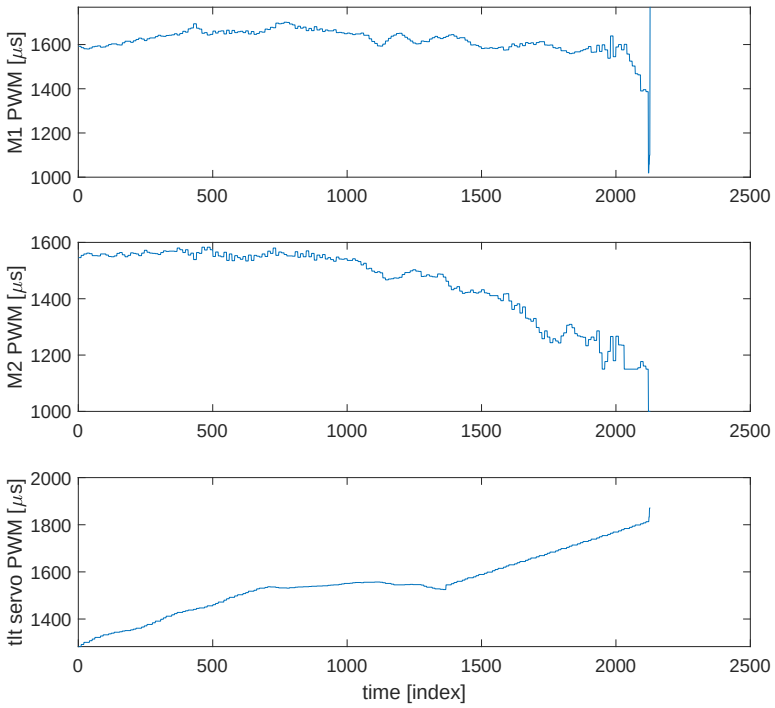


Fig. A.11: Test1.1 state input plots.

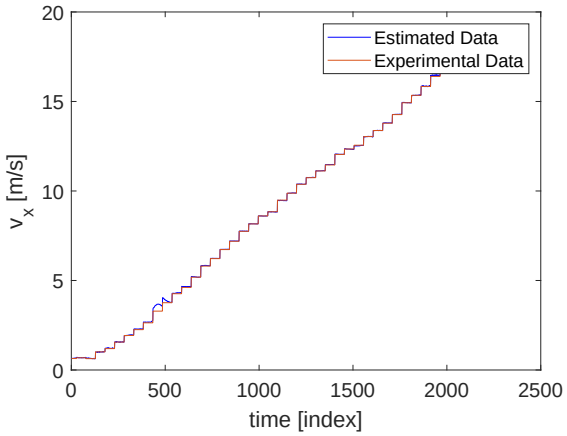


Fig. A.12: Test1.2 comparison of the velocity in the x direction assuming constant coefficients.

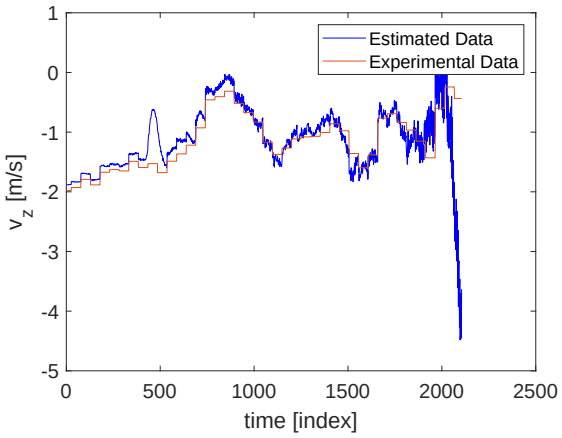


Fig. A.13: Test1.2 comparison of the velocity in the z direction assuming constant coefficients.

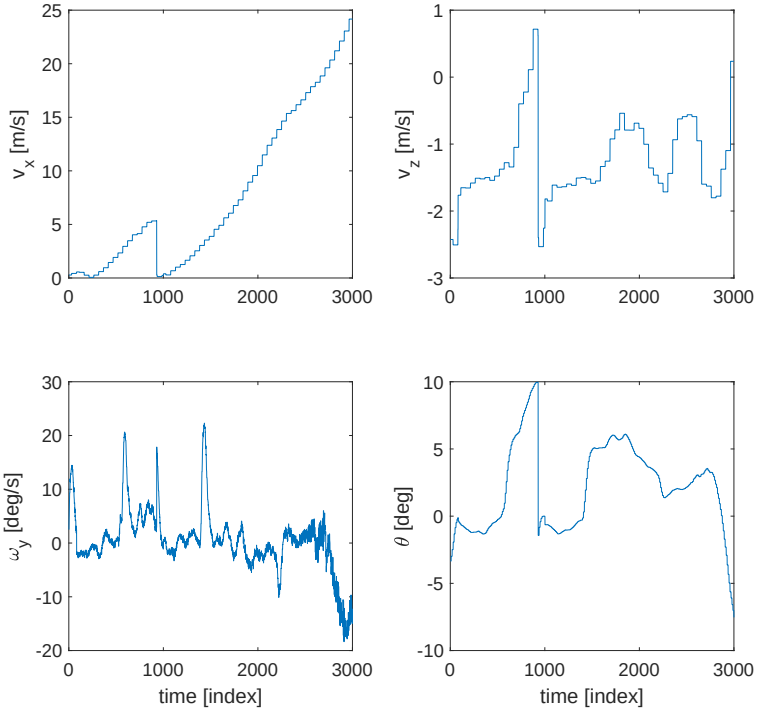


Fig. A.14: Test1.3 state plots.

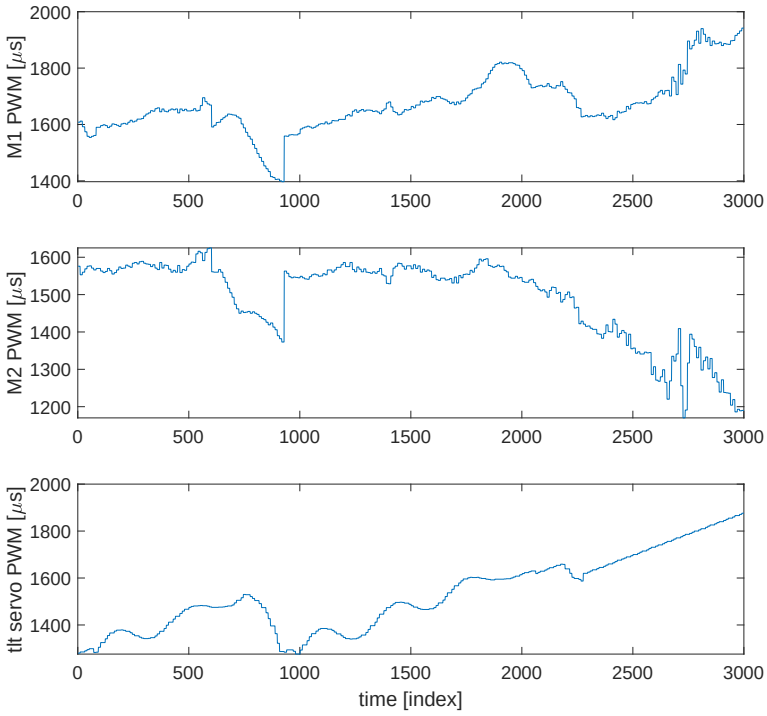


Fig. A.15: Test1.3 input plots.

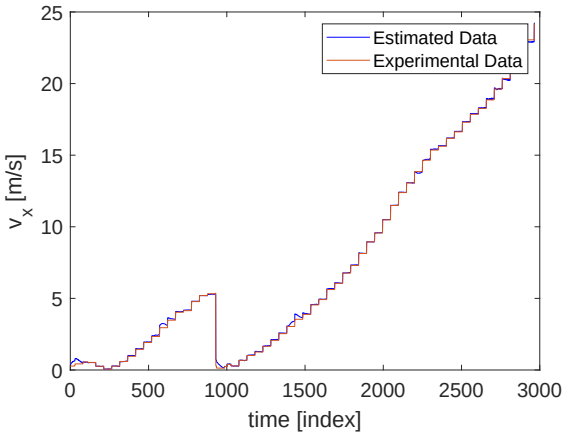


Fig. A.16: Test1.3 comparison of the velocity in the x direction assuming constant coefficients.

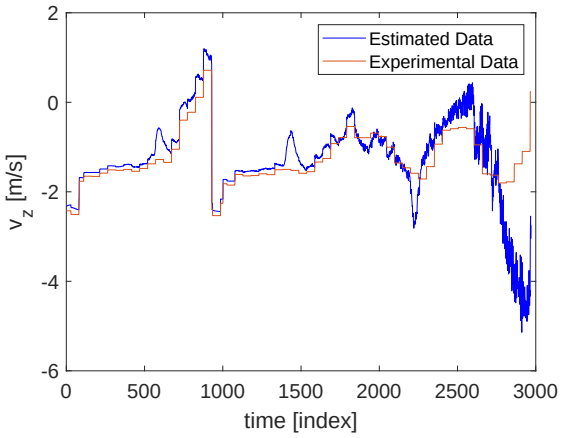


Fig. A.17: Test1.3 comparison of the velocity in the z direction assuming constant coefficients.

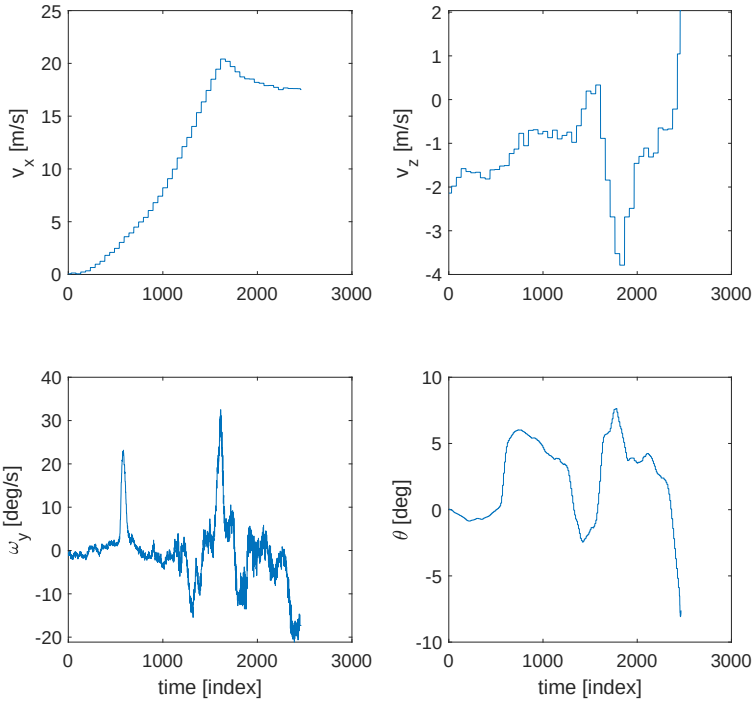


Fig. A.18: Test1.4 state plots.

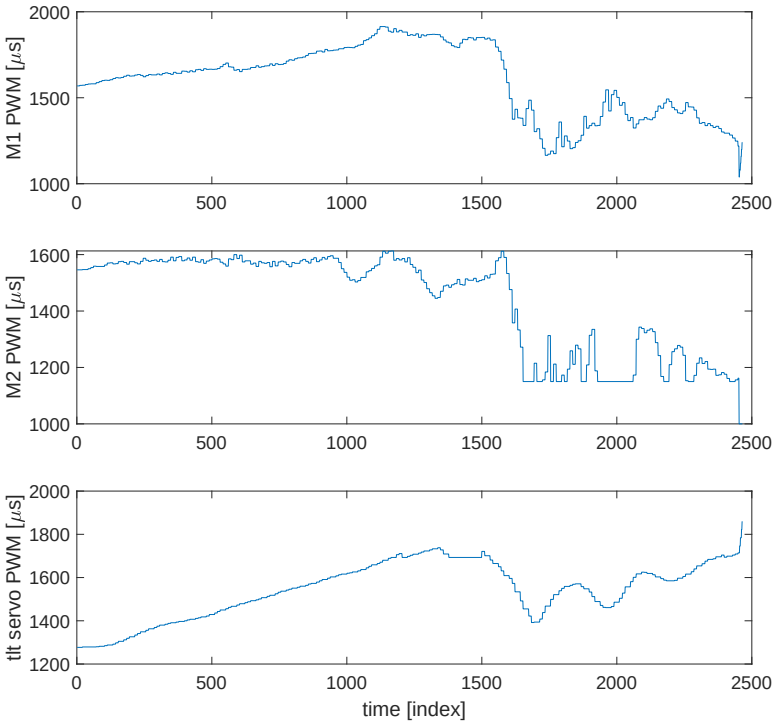


Fig. A.19: Test1.4 input plots.

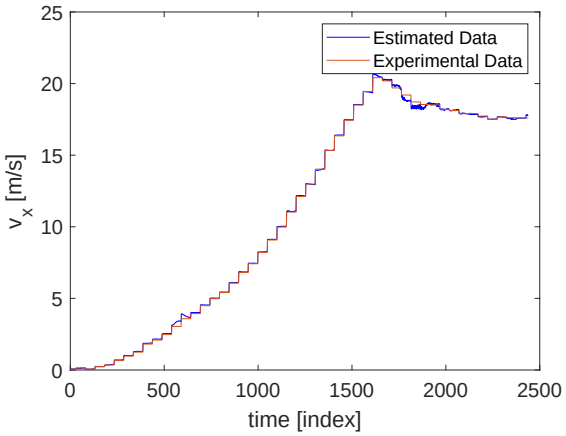


Fig. A.20: Test1.4 comparison of the velocity in the x direction assuming constant coefficients.

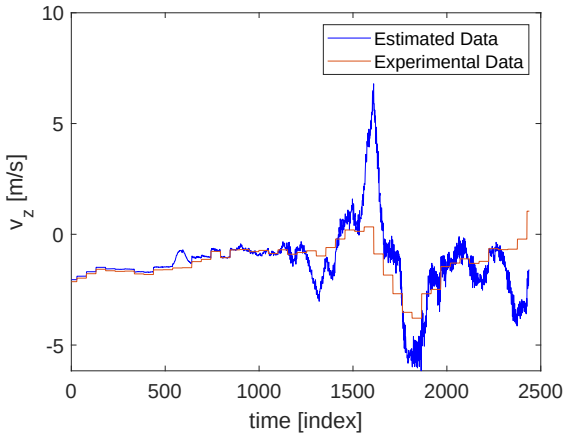


Fig. A.21: Test1.4 comparison of the velocity in the z direction assuming constant coefficients.

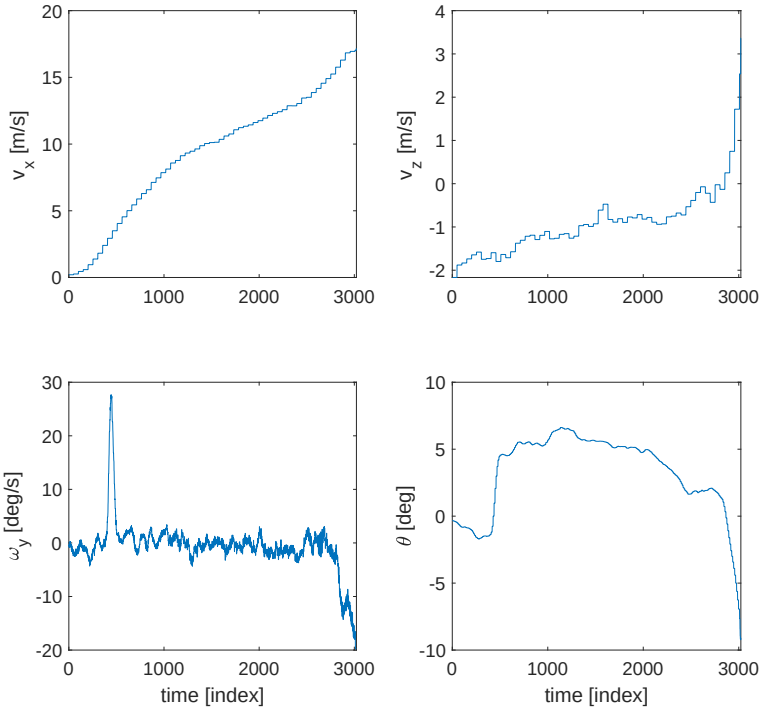


Fig. A.22: Test2.1 state plots.

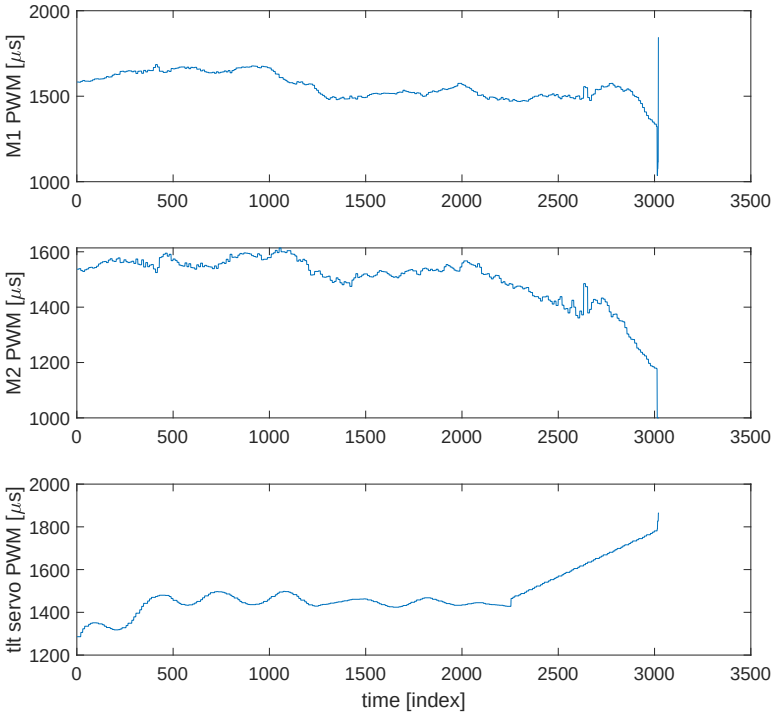


Fig. A.23: Test2.1 input plots.

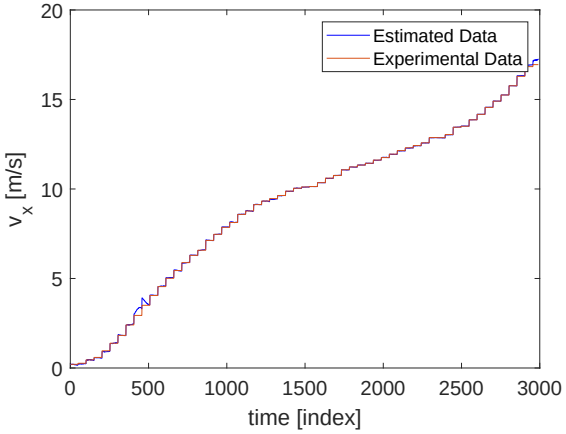


Fig. A.24: Test2.1 comparison of the velocity in the x direction assuming constant coefficients.

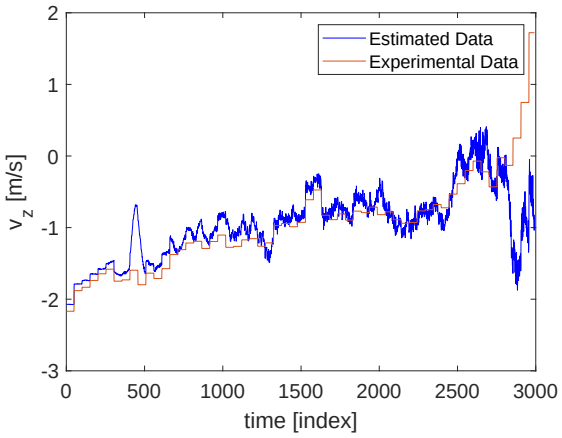


Fig. A.25: Test2.1 comparison of the velocity in the z direction assuming constant coefficients.

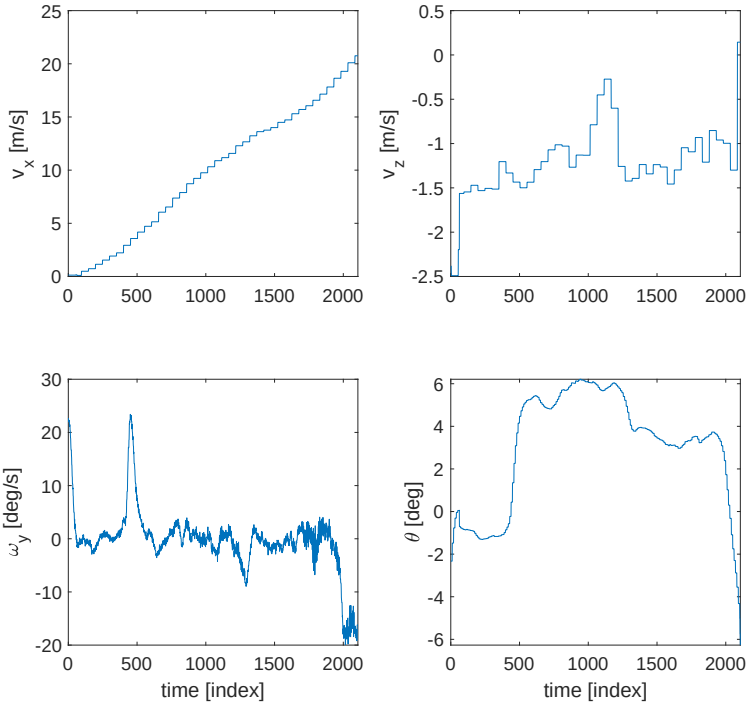


Fig. A.26: Test2.2 state plots.

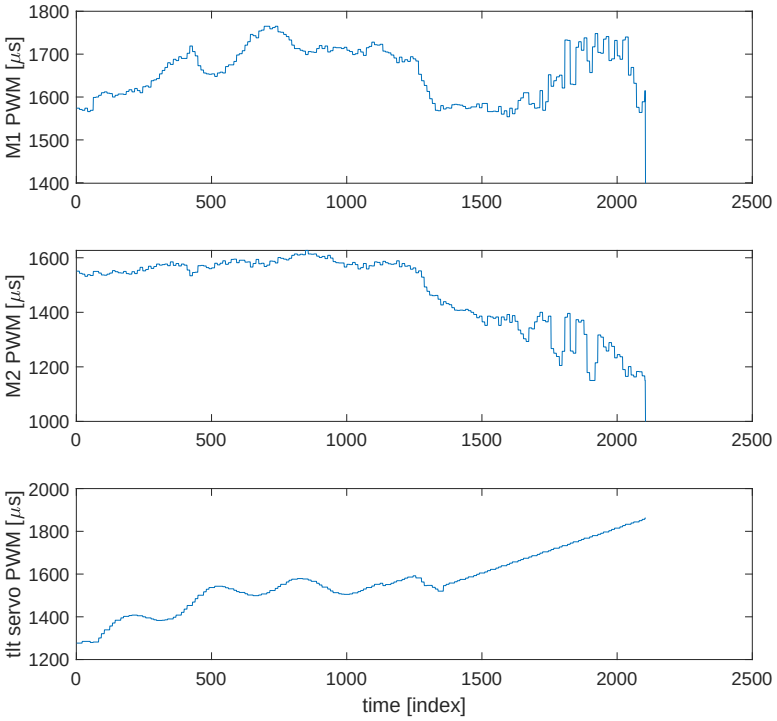


Fig. A.27: Test2.2 input plots.

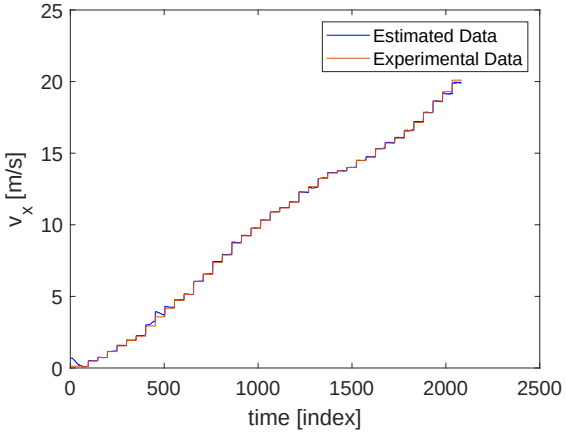


Fig. A.28: Test2.2 comparison of the velocity in the x direction assuming constant coefficients.

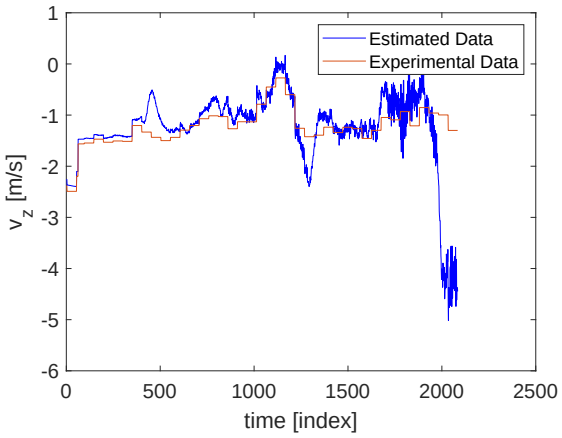


Fig. A.29: Test2.2 comparison of the velocity in the z direction assuming constant coefficients.

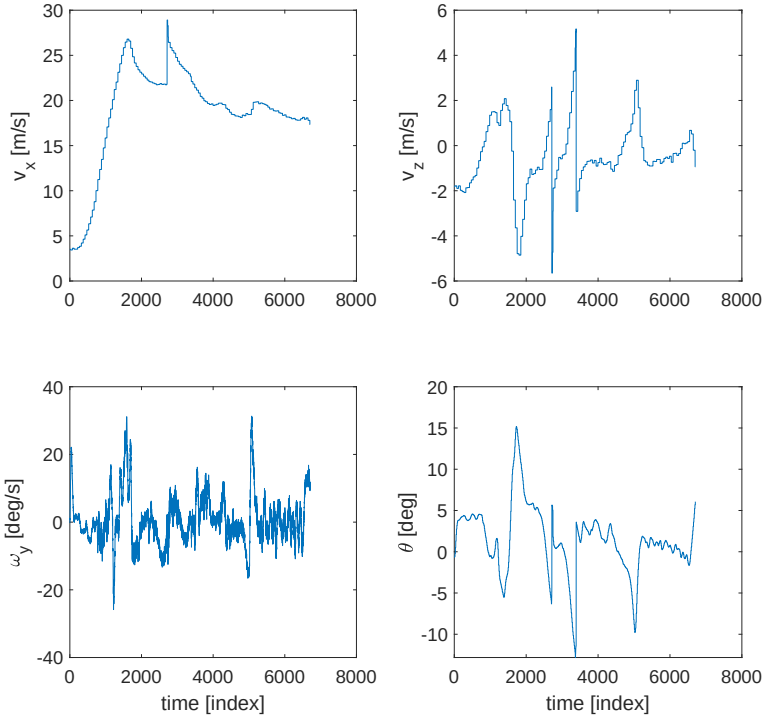


Fig. A.30: Test2.3 state plots.

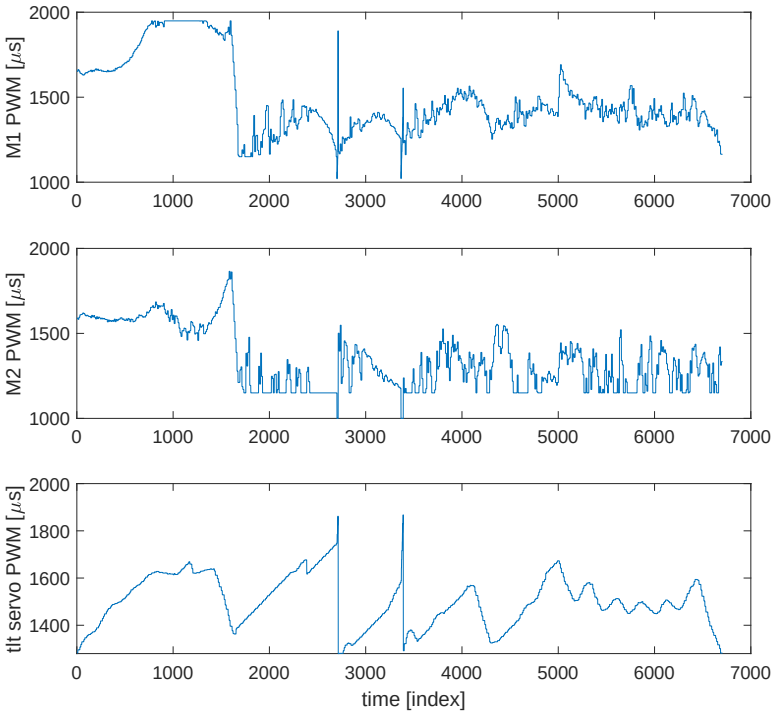


Fig. A.31: Test2.3 input plots.

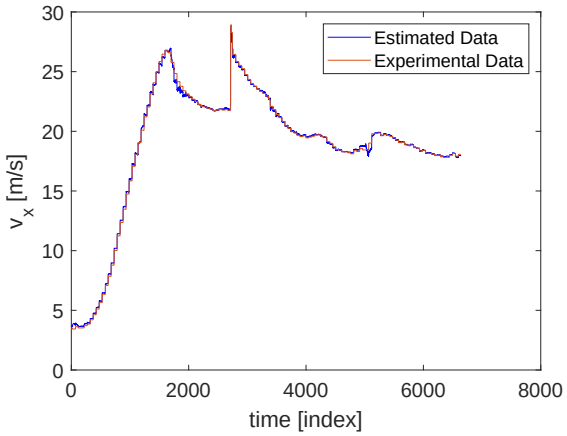


Fig. A.32: Test2.3 comparison of the velocity in the x direction assuming constant coefficients.

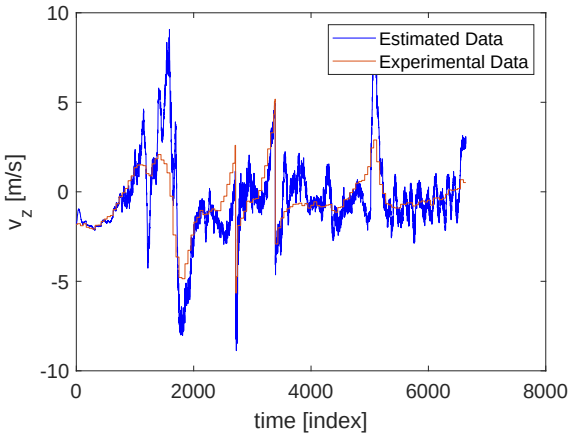


Fig. A.33: Test2.3 comparison of the velocity in the z direction assuming constant coefficients.

A.4 Flight Data and LSR Results Assuming Polynomial Coefficients

This section shows model comparisons for flight tests using the polynomial unknown coefficient LSR model. The state plots are the same as the constant unknown coefficient case previously shown. This is because they correspond to the same data.

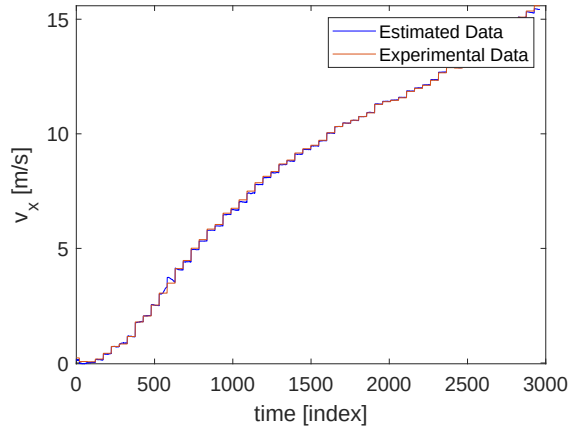


Fig. A.34: Test1.1 comparison of the velocity in the x direction assuming polynomial coefficients.

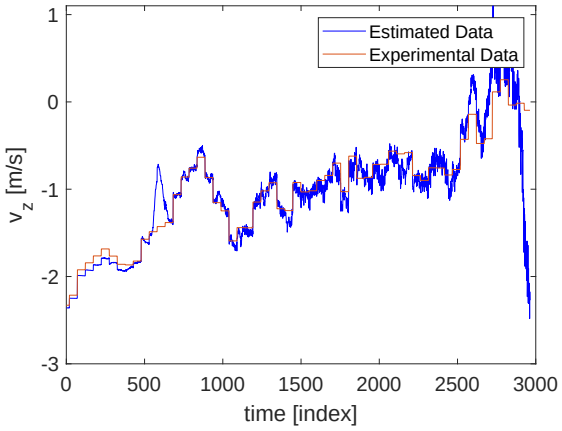


Fig. A.35: Test1.1 comparison of the velocity in the z direction assuming polynomial coefficients.

Test 1.2 plots:

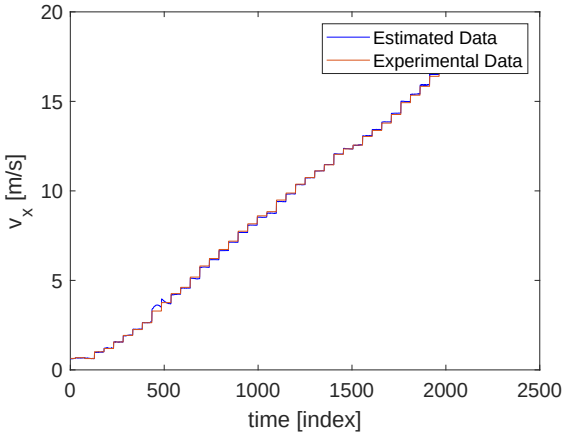


Fig. A.36: Test1.2 comparison of the velocity in the x direction assuming polynomial coefficients.

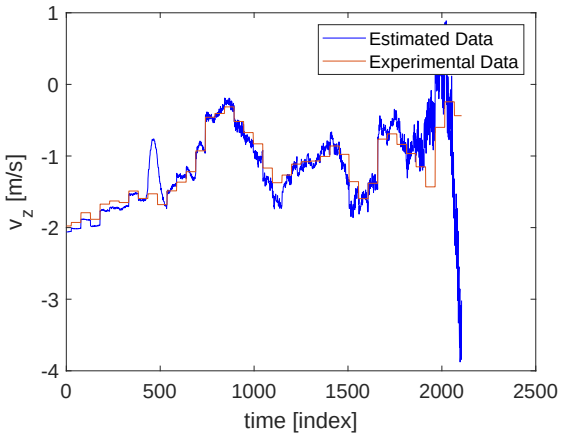


Fig. A.37: Test1.2 comparison of the velocity in the z direction assuming polynomial coefficients.

Test 1.3 plots:

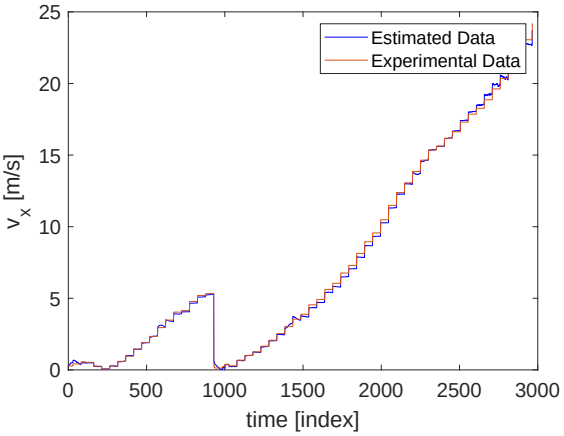


Fig. A.38: Test1.3 comparison of the velocity in the x direction assuming polynomial coefficients.

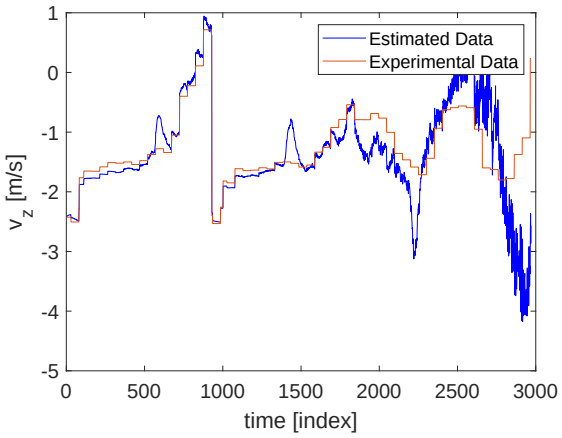


Fig. A.39: Test1.3 comparison of the velocity in the z direction assuming polynomial coefficients.

Test 1.4 plots:

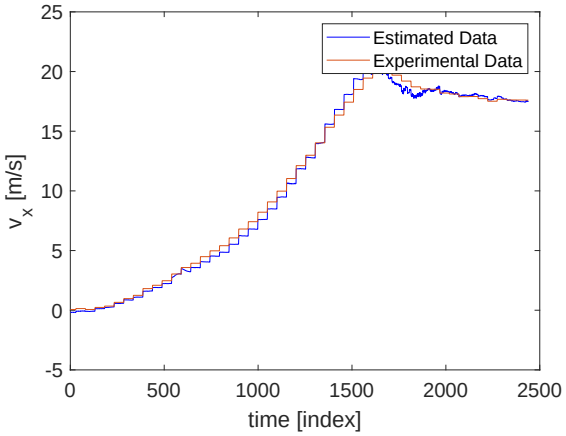


Fig. A.40: Test1.4 comparison of the velocity in the x direction assuming polynomial coefficients.

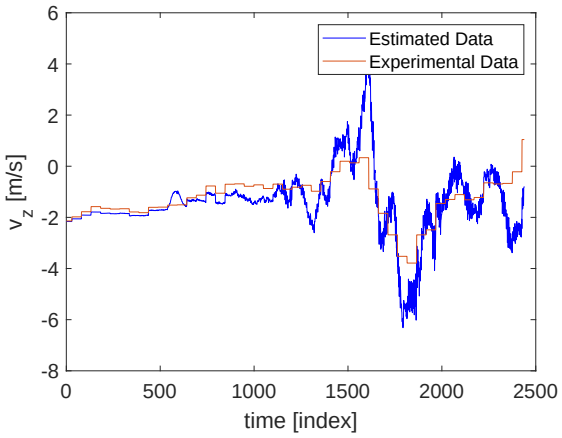


Fig. A.41: Test1.4 comparison of the velocity in the z direction assuming polynomial coefficients.

Test 2.1 plots:

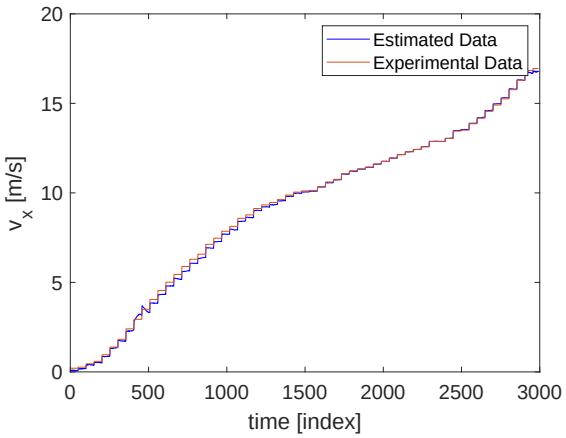


Fig. A.42: Test2.1 comparison of the velocity in the x direction assuming polynomial coefficients.

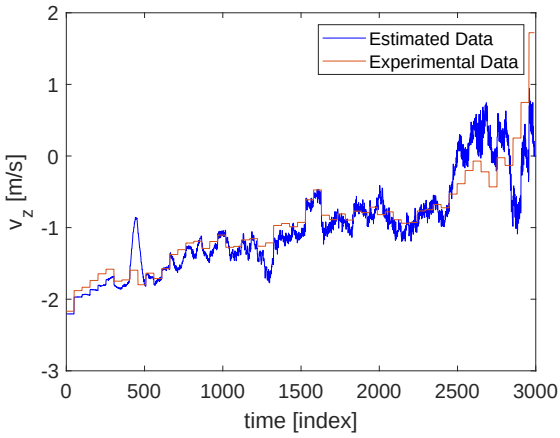


Fig. A.43: Test2.1 comparison of the velocity in the z direction assuming polynomial coefficients.

Test 2.2 plots:

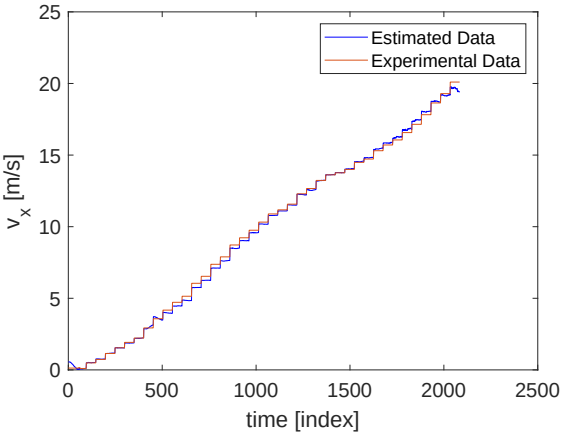


Fig. A.44: Test2.2 comparison of the velocity in the x direction assuming polynomial coefficients.

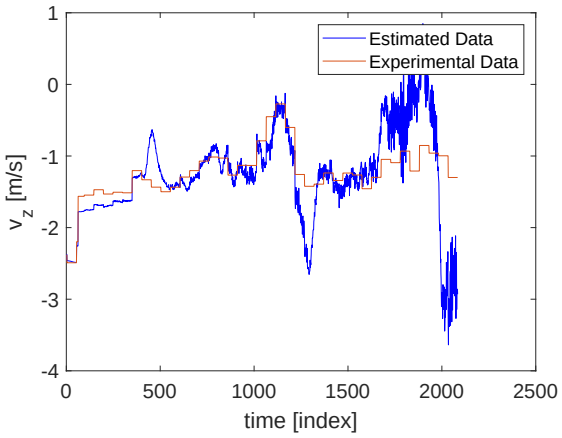


Fig. A.45: Test2.2 comparison of the velocity in the z direction assuming polynomial coefficients.

Test 2.3 plots:

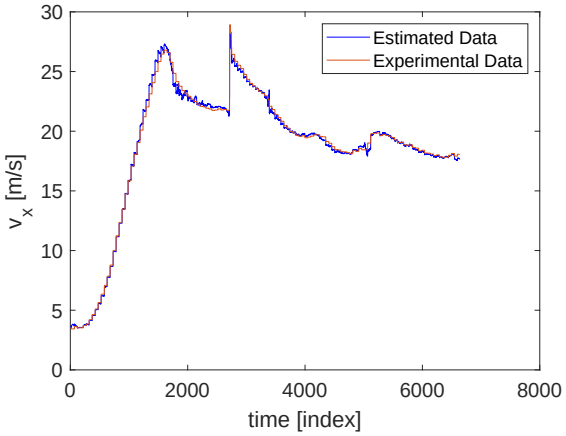


Fig. A.46: Test2.3 comparison of the velocity in the x direction assuming polynomial coefficients.

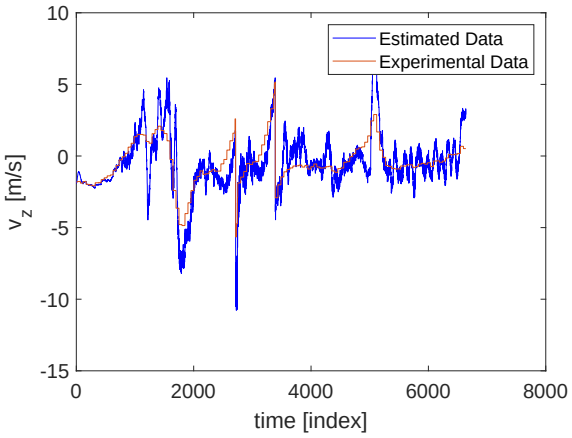


Fig. A.47: Test2.3 comparison of the velocity in the z direction assuming polynomial coefficients.

APPENDIX B

SUPPORTING MATLAB CODE FUNCTIONS AND ALGORITHMS

B.1 Code For Data Post-Processing Assuming Constant Unknown Coefficients

Listing B.1: Driver code for data processing, LSR , and PID control

```
% Clayton Spencer
% This script is driver code to manipulate and process experimental flight
% data in support of a Masters Thesis

close all; clear all; clc

% Add the names of all the .mat files that need to be read in
fltdat = ["Test1_1_1_2.mat", "Test1_1_1_2.mat", "Test2_1_2_2.mat", "Test2_3.
    mat"];

% Toggle the below lines to analyse each individual flight
% fltdat = ["Test1_1_1_2.mat"];
% fltdat = ["Test1_3_1_4.mat"];
% fltdat = ["Test2_1_2_2.mat"];
% fltdat = ["Test2_3.mat"];
% fltdat = ["Test3_1.mat"];


% create the storage vectors for all the necessary parameters
u = []; w = []; q = []; theta = []; m1 = []; m2 = []; tlt = [];
time = cell(1, length(fltdat)); % time array for each flight time vector
timeL = [];

% Loop through all the flight logs to collect all of the flight data
% current data structure:
% data = [u,w,q,theta,m1,m2,tlt]
```

```
26 for i = 1:length(fltdat)
27     [tempt, tempd] = simple_data_manipulation(fltdat(i));
28
29     % concatenate all the data into their own column vectors
30     u = [u;tempd(:,1)]; w = [w;tempd(:,2)]; % velocities
31     q = [q;tempd(:,3)]; % rotation rate about y axis wy
32     theta = [theta;tempd(:,4)]; % Euler angles
33     m1 = [m1;tempd(:,5)]; m2 = [m2;tempd(:,6)]; tlt = [tlt;tempd(:,7)]; %
           control signals
34
35     % Store the time vectors from each test flight
36     timeL = [timeL;tempt]; % long time vector of all the time stamps
           concatenated
37     time{i} = {tempt}; % Cell array to contain columns of different lengths
38 end % i loop
39
40 % add all of the desired data into one vector
41 data = [u,w,q,theta,m1,m2,tlt];
42
43 %% Virtualize the time stamps
44 % Because each flight during each test has different timestamps that
45 % relate to the Epoch 01/01/1970, data cannot be plotted in time as is. To
46 % get around this, a "virtual timestamp" is created. Doing this will lose
47 % the actual reference to "real" time of the data. However, the data can be
48 % plotted in one large sequence to demonstrate the continuous nature of the
49 % data as it is fed into System Identification algorithms. The time vector
50 % will then be a sequence of integers corresponding to each new data point.
51
52 [virtime,dt] = virtualize(timeL); % create the virtual time vector
53 dt = 0.01; % [s]
54 % Use the virtual time vector for plotting the concatonated flight data
55 % Use the virtual time vector for plotting the concatonated flight data
56 % State plots
57 figure
58 subplot(2,2,1)
```

```
59 plot(virtime,data(:,1))
60 % title('Ground Speed','FontSize',16)
61 % xlabel('time [index]','FontSize',16)
62 ylabel('Vx [m/s]','FontSize',16)
63 set(gca,'FontSize',14);
64
65 subplot(2,2,2)
66 plot(virtime,data(:,2))
67 % title('Verticle Speed','FontSize',16)
68 % xlabel('time [index]','FontSize',16)
69 ylabel('Vz [m/s]','FontSize',16)
70 set(gca,'FontSize',14);
71
72 subplot(2,2,3)
73 plot(virtime,data(:,3))
74 % title('Pitch Rate','FontSize',16)
75 xlabel('time [index]','FontSize',16)
76 ylabel('q [deg/s]','FontSize',16)
77 set(gca,'FontSize',14);
78
79 subplot(2,2,4)
80 plot(virtime,data(:,4))
81 % title('Elevation Angle','FontSize',16)
82 xlabel('time [index]','FontSize',16)
83 ylabel('\theta [deg]','FontSize',16)
84 set(gca,'FontSize',14);
85
86 % Input signal plots
87 figure
88 subplot(3,1,1)
89 plot(virtime,data(:,5))
90 % title('Motor 1 and 3','FontSize',16)
91 % xlabel('time [index]','FontSize',16)
92 ylabel('M1 PWM [\mu/s]','FontSize',16)
93 set(gca,'FontSize',14);
```

```
94
95 subplot(3,1,2)
96 plot(virtime,data(:,6))
97 % title('Motor 2 and 4','FontSize',16)
98 % xlabel('time [index]','FontSize',16)
99 ylabel('M2 PWM [\mu/s]','FontSize',16)
100 set(gca,'FontSize',14);
101
102 subplot(3,1,3)
103 plot(virtime,data(:,7))
104 % title('Tilt Servo L/R','FontSize',16)
105 xlabel('time [index]','FontSize',16)
106 ylabel('tlt servo PWM [\mu/s]','FontSize',16)
107 set(gca,'FontSize',14);
108
109 %% LSR
110 % This section computes the Least Squares Regression for the data.
111 [eta,output] = lsr_TH(data,virtime,dt); % computes the Least Squares
       Regression to estimate eta
112
113 % Output the results
114 fprintf('C_T: %f\n ', eta(1))
115 fprintf('C_L: %f\n ', eta(2))
116 fprintf('C_D: %f\n ', eta(3))
117
118 %% PID Tuning
119 % Define the input and output of the system
120 n = round(length(output(:,1))*0.68);
121 vx = output(1:n,1); vxval = output(n:end,1);
122 vz = output(1:n,2); vzval = output(n:end,2);
123 wy = output(1:n,3); wyval = output(n:end,3);
124 theta = output(1:n,4); thetaval = output(n:end,4);
125 m1 = output(1:n,5); m1val = output(n:end,5);
126 m2 = output(1:n,6); m2val = output(n:end,6);
127 tlt = output(1:n,7); tltval = output(n:end,7);
```

```
128
129 % Create a State Space model for trim point 0
130 vx0 = 0; vz0 = vz(1); theta0 = 0; wy0 = 0; tlt0 = tlt(1); % initial
      conditions
131 rho = 1.215; % density of air [kg/m^3]
132 S = 0.725; % Characteristic area [m^2]
133 m = 11; % aircraft mass [kg]
134 g = 9.81; % acceleration due to gravity [m/s^2]
135 l = 0.362; % Characteristic length [m]
136 h = rho*S/(2*m);
137 Iyy = 0.4;
138
139 % Create State Matrices
140 A0 = [0, 1;
141       0, 0];
142
143 B0 = [0;
144       2*eta(1)*l/Iyy];
145 C0 = [1, 0];
146 D0 = [0];
147
148 % Convert to transfer functions
149 [n0_1,d0_1] = ss2tf(A0,B0,C0,D0); % numerator and denominator corresponding
      to the first input
150 G0_1 = tf(n0_1,d0_1) % transfer function corresponding to the first input
151
152 figure
153 load PID_output.mat
154 plot(ans.Time,ans.Data(:,1));
155 hold on
156 plot(ans.Time,ans.Data(:,2));
157 title('Control Response','FontSize',16);
158 xlabel('Time [s]','FontSize',16);
159 xlim([0,3]);
160 ylabel('Pitch angle [rad]','FontSize',16);
```

```
161 legend('Target','System Response');
162 set(gca,'FontSize',14);
163 hold off
164
165 n = 2900; % data point number at end of first transition
166 vx1 = vx(n); vz1 = vz0; theta1 = theta(n); wy1 = wy(n); tlt1 = tlt(n); %
      initial conditions
167
168 % Calculate C_T
169 th = output(n,1); % initial velocity
170 C_T1 = eta(1)*th^2 + eta(2)*th + eta(3);
171
172 % Create State Matrices
173 A1 = [0, 1;
174       0, 0];
175
176 B1 = [0;
177       2*C_T1*l/Iyy];
178 C1 = [1, 0];
179 D1 = [0];
180
181 % Convert to transfer functions
182 [n1_1,d1_1] = ss2tf(A1,B1,C1,D1,1); % numerator and denominator
      corresponding to the first input
183 G1_1 = tf(n1_1,d1_1) % transfer function corresponding to the first input
184
185 figure
186 load PID_output_end.mat
187 plot(ans.Time,ans.Data(:,1));
188 hold on
189 plot(ans.Time,ans.Data(:,2));
190 title('Control Response End','FontSize',16);
191 xlabel('Time [s]','FontSize',16);
192 xlim([0,3]);
193 ylabel('Pitch angle [rad]','FontSize',16);
```

```
194 legend('Target','System Response');
195 set(gca,'FontSize',14);
196 hold off
197
198
199 disp('Program: Simple Driver Finished')
```

Listing B.2: Function which post-processes flight data

```
1  function [time,data_o] = simple_data_manipulation(file_name)
2
3  % Clayton Spencer
4  % Flight Data collection and manipulation:
5  % The goal of this script is to parse data from a .mat file that contains
6  % all the flight data from the VTOL project flights and put them into a
7  % usable form: data = [vx,vz,wy,theta,m1,m2,tlt]
8
9  % Time the program
10 tic;
11
12 %% Read in the flight data.
13 load (file_name)
14
15 % Extract the data into sections
16 % Euler angles
17 time_euler = ATT(:,2); % time vector of the euler angle variables
18 theta = ATT(:,6); % elevation angle [deg]
19
20 % velocites
21 time_vel = GPS_0(:,2);
22 u = GPS_0(:,12); % ground speed vx [m/s]
23 w = GPS_0(:,14); % vertical speed vz [m/s]
24
25 % Rate data
26 time_rate = RATE(:,2); % time vector of the rate variables
27 q = RATE(:,7); % pitch rate [deg/s]
```

```
28
29 % Control data
30 time_control = RCOU(:,2); % time vector of the control variables
31 m1 = RCOU(:,11); % pwm signal for motor1 and assumed motor 3
32 m2 = RCOU(:,12); % pwm signal for motor2 and assumed motor 4
33 tlt = RCOU(:,7); % pwm signal for the left tilt-servo and assumed right tilt
       servo
34
35 % plot an example of one of the pieces of data for later comparison
36 m2_copy = m2;
37 u_copy = u;
38 figure
39 hold on
40 plot(time_vel,u_copy,'b*')
41 title('Velocity Data Before Processing')
42 xlabel('time [\mu s]')
43 ylabel('vx [m/s]')
44
45 %% Create the master time vector
46
47 % add each time vector to the end of one master time vector
48 time_master = [time_euler;time_vel;time_rate;time_control];
49 time = unique(time_master); % this command sorts the vector
50 % from smallest to greatest and deletes repeat values.
51
52 %% Adjust data vectors
53 % overwrite the velocity data vectors to adjust
54 [u,dtu] = get_common_data(time,time_vel,u);
55 [w,dtw] = get_common_data(time,time_vel,w);
56
57 % overwrite the rate data vectors to be the adjusted vectors
58 [q,dtq] = get_common_data(time,time_rate,q);
59
60 % overwrite the euler angle data vectors to be the adjusted vectors
61 [theta,dttheta] = get_common_data(time,time_euler,theta);
```

```
62
63 % overwrite the control data vectors to be the adjusted vectors
64 [m1,dtm1] = get_common_data(time,time_control,m1);
65 [m2,dtm2] = get_common_data(time,time_control,m2);
66 [tlt,dttlt] = get_common_data(time,time_control,tlt);
67
68 data = [u,w,q,theta,m1,m2,tlt];
69 [rows,columns] = size(data); % get the number of columns of the data vector
70
71 plot(time,u,'r.')
72 title('Velocity Data After Processing')
73 xlabel('time [\mu s]')
74 ylabel('vx [m/s]')
75 hold off
76
77 %% Single Flight Analysis
78 % Split the data into individual flights to determine the effects of tilt
79 % rate and max tilt angle if needed.
80
81 mid  = round(length(u)/2); % toggle the next two lines with the 2 below it
82 % data = data(1:mid,:); % First flight data
83 % time = time(1:mid,:); % First flight time
84 % data = data(mid:end,:); % Second flight
85 % time = time(mid:end,:); % First flight time
86
87 %% Start Point
88 % if there is unwanted data at the beginning of the file, cut it out
89 % Use the minimum value for the tlt servo signal as a reference
90 ref = 1200;
91 iter = 0;
92 for i=1:length(time)
93     if tlt(i) > ref
94         iter = iter + 1; % increment the iteration counter
95     else %
96         % The signal for the tilt rotor has reached an acceptable start
```

```
97          % point
98          break % exit the loop
99      end % > ref
100 end % i
101 iter = iter + 1; % increment one more time
102
103 % adjust the time vector
104 time = time(iter:end); % This cuts off the undesirable portion of the
        beginning of the vector
105
106 % adjust the data vectors as well
107 dummy = [];
108 for i=1:columns
109     temp_data = data(:,i);
110     temp_data = temp_data(iter:end);
111     dummy = [dummy,temp_data]; % cut off the undesireable beginning of the
            data
112 end % i
113 data = dummy; % overwrite the data vectors
114
115 % check to make sure the time vector and the data vectors are the same size
116 s = length(time) - length(data(:,1))
117
118 %% Extract transition period data
119
120 % Find the transition period using the tilt servo time signal as reference
121 transtime = transition(time,data(:,end)); % output is the ref time vector
122
123 % Using the ref time vector, cut and keep only transition period data from
124 % the provided flight data.
125 dummy = [];
126 for i=1:columns
127     % add data corresponding to transition to the dummy data vector
128     dummy = [dummy,cut(transtime,time,data(:,i))];
129 end % end i loop
```

```
130
131 % update the time vector and the data vector for output
132 time = transtime;
133 data_o = dummy;
134
135 % Stop the timer
136 elapsedTime = toc;
137
138 % Display the elapsed time
139 disp(['Elapsed time for each flight test: ' num2str(elapsedTime) ' seconds'
      ]);
140
141 end % data_manipulation
```

Listing B.3: Function which unifies all the flight data vectors

```
1  function [new_data,dt] = get_common_data(time_master,time_reference,data)
2  % This function takes a master time vector, a reference time vector, and a
3  % data vector and creates a new data vector of the same size as the master
4  % time vector. The latest data value is re-used when the time indeces of
5  % the master time vector and the reference time vector do not match.
6
7  % The master time vector contains all the values of a pool of time vectors.
8  % each reference time value is found in the master time vector.
9  % This results in the data vector having repeated values until reaching a
10 % reference time index that matches the master time vector. Then, the data
11 % vector value is updated in the new data vector.
12
13 % Find the average time step in the time vector
14 t1 = time_reference(1:end-1); % time vector without the last element
15 t2 = time_reference(2:end); % time vector without the first element
16 t1m2 = abs(t1 - t2); % absolute value of the difference between the values
17 tavg = mean(t1m2); % find the average time step value in [\mus]
18 dt = tavg*10^-6; % convert from \mus to [s]
19
20 new_data = zeros(length(time_master),1); % initialize the new data vector
```

```
21 new_data(1) = data(1); % set first data value
22 curr_ref_time = time_reference(2); % initialize the current reference time
       value
23 curr_data_index = 2; % initial data vector index
24 for i=2:length(time_master)-1 % Loop through all the values of the master
       time vector
25     % check if the reference data and time index need to be updated
26     if time_master(i) == curr_ref_time
27         new_data(i) = data(curr_data_index); % add a value to the new data
               vector
28         curr_data_index = curr_data_index + 1; % increment data index
29
30         if curr_data_index < length(data)
31             curr_ref_time = time_reference(curr_data_index);
32         else
33             curr_ref_time = time_reference(curr_data_index-1);
34         end
35
36     else
37         new_data(i) = data(curr_data_index - 1); % add a value to the new
               data vector
38         continue % move on to the next iteration
39     end
40
41 end % end for loop (i)
42 end % end get_common_data function
```

Listing B.4: Function which extracts the transition period flight data

```
1 function [tout,dout] = transition(time,data)
2 % This function finds the transition periods within test flight data. It
3 % then outputs a reference time vector that corressponds to data during
4 % transition periods only. This function assumes the data does not begin
5 % obove the threshold.
6
7 % set transition pwm signal thresholds using the tilt servo signal as
```

```
% reference.
% below the threshold, the vehicle is in quadcopter mode
% above the threshold, the vehicle is in fixed wing mode
% within the threshold, the vehicle is transitioning from quad to fixed
% wing which is the time period of desired data.
min = 1275; % min pwm signal where transition begins
max = 1880; % max pwm signal where transition completes

% storage arrays
tout = [];
dout = [];
fwd = false; % flag, true if vehicle is transitioning forward and false if
    transitioning backwards
for i=2:length(time) % loop through every data point

    if data(i-1) < min && data(i) > min % detect if flag needs to be changed
        fwd = true;
    elseif data(i-1) > max && data(i) < max % detects backwards transition
        fwd = false;
    else % transition hasn't changed
        % continue with the current falg value
    end % end flag changing condition

    if fwd % if in forward transition
        if data(i) < max  && data(i) > min % if transition hasn't completed
            tout = [tout;time(i)]; % add time index during transition
        else % vehicle has transitioned completely
            % This case should never be reached
        end % end min<data<max

    else % not in forward transition
        % do nothing
    end % end data(i-1)....

end % end i loop
```

```
42
43 end % end transition
```

Listing B.5: Function which cuts away non-transition period flight data

```
1 function dout = cut(reftime,time,data)
2 % reftime is a reference (correct) time vector
3 % time is a vector of timestamps associated with the data vector
4 % data is the data associated with the time vector
5
6 % This function keeps only the data points where the timestamp is also in
7 % the reftime. The returned value is a vector which now coincides with the
8 % reference time vector
9 dout = [];
10 for i=1:length(time)
11     if ismember(time(i),reftime) % the current timestamp is in reftime
12         dout = [dout;data(i)]; % add the data to the output vector
13     else % the current timestamp isn't in the reftime
14         % do not add the data at this timestamp to the output vector
15     end % ismember
16 end % end i loop
17 end % cut
```

Listing B.6: Function which creates a virtual time vector for visualization purposes

```
1 function [virtime,dt] = virtualize(time)
2 % This function finds the average timestep of a time vector. This function
3 % also takes in a vector of time stamps and outputs a vector of
4 % "virtual" timestamps. The virtual timestamps are an integer sequence from
5 % 1 to the number of timestamps in the original time vector
6
7 t1 = time(1:end-1); % time vector without the last element
8 t2 = time(2:end); % time vector without the first element
9 t1m2 = abs(t1 - t2); % absolute value of the difference between the values
10 tavg = mean(t1m2); % find the average time step value in [\mus]
11 dt = tavg*10^-6; % convert from \mus to [s]
12
```

```
13 virtime = zeros(length(time),1); % pre-allocate the virtual time vector
14 for i=1:length(time)
15     virtime(i) = i; % use the loop iteration as the counter and the virtual
           value
16 end % i loop
17
18 end % virtualize
```

Listing B.7: Code which computes the LSR for estimating the constant unknown parameters

```
1 function [eta1,output] = lsr_TH(data,time,dt)
2 % This function computes the Least Squares Regression for flight data. The
3 % result is the estimated unknown parameters of the dynamic model of the
4 % aircraft
5
6 % Parse the data out into individual signals for readability's sake. The
7 % expected order of the data is: data = [vx,vz,wy,theta,m1,m2,tlt];
8 vx = data(:,1); % aircraft velocity in the x direction [m/s]
9 vz = data(:,2); % aircraft velocity in the z direction [m/s]
10 wy = data(:,3); % angular velocity about the y-axis [rad/s]
11 theta = data(:,4); % elevation angle [deg]
12 m1_pwm = data(:,5); % PWM signal for motor 1 and 3 [\mus]
13 m2_pwm = data(:,6); % PWM signal for motor 2 and 4 [\mus]
14 tlt_pwm = data(:,7); % PWM signal for tilt servos left and right [\mus]
15
16 % The PWM signals need to be converted into angular velocites. This is done
17 % using experimental mappings
18 % initialize vectors for the motors and the tilt servo
19 m1rad = zeros(length(m1_pwm),1);
20 m2rad = zeros(length(m2_pwm),1);
21 tltdeg = zeros(length(tlt_pwm),1);
22 tltrad = zeros(length(tlt_pwm),1);
23 thetarad = zeros(length(theta),1);
24
25 % Loop through the motor and tilt servo signals to convert the signals from
26 % their raw form to standard units
```

```
27 m1 = zeros(length(m1_pwm),1);
28 m2 = zeros(length(m2_pwm),1);
29 for i=1:length(m1_pwm)
30     % Conversions for calculation and plotting
31     m1(i) = (-0.0175*m1_pwm(i)^2 + 61.681*m1_pwm(i) - 42981); % [RPM]
32     m2(i) = (10.692*m2_pwm(i) - 10777); % [RPM]
33     tltrad(i) = deg2rad((tlt_pwm(i) - 1180)/7.7396); % [rad]
34     m1rad(i) = (2*pi/60)*(m1(i)); % [rad/s]
35     m2rad(i) = (2*pi/60)*(m2(i)); % [rad/s]
36     tltdeg(i) = rad2deg(tltrad(i)); % [deg]
37     thetarad(i) = deg2rad(theta(i)); % [rad]
38 end % i loop
39
40 % Output the converted data
41 output = [vx,vz,wy,thetarad,m1rad,m2rad,tltrad];
42
43 % % initialize necessary values
44 rho = 1.215; % density of air [kg/m^3]
45 S = 0.725; % Characteristic area [m^2]
46 X = zeros(length(data(1)),4);
47 m = 11; % aircraft mass [kg]
48 g = 9.81; % acceleration due to gravity [m/s^2]
49 L = 0.362; % Characteristic length [m]
50 X = []; % matrix of regressors
51 xkp1 = []; % state values at x_k+1
52 xk = []; % state values at x_k
53 rem = []; % remaining values
54 for i=1:length(data)-1
55     % Calculate the regressors
56     x11 = dt*(2*m1rad(i)^2*sin(tltrad(i)));
57     x12 = dt*(rho*S*vz(i)*sqrt(vx(i)^2 + vz(i)^2))/(2*m);
58     x13 = -dt*(rho*S*vx(i)*sqrt(vx(i)^2 + vz(i)^2))/(2*m);
59     x21 = -dt*(2*m1rad(i)^2*cos(tltrad(i)) + 2*m2rad(i)^2)/m;
60     x22 = -dt*(rho*S*vx(i)*sqrt(vx(i)^2 + vz(i)^2))/(2*m);
61     x23 = -dt*(rho*S*vz(i)*sqrt(vx(i)^2 + vz(i)^2))/(2*m);
```

```
62
63      Xtemp = [x11,x12,x13;
64               x21,x22,x23];
65      X = [X;Xtemp]; % Stack up the regressor matrix
66
67      % Calculate x_k+1
68      xk1_1 = vx(i+1);
69      xk1_2 = vz(i+1);
70      xkp1 = [xkp1;xk1_1;xk1_2]; % Stack up the vector truth value
71
72      % Calculate x_k
73      xk1 = vx(i);
74      xk2 = vz(i);
75      xk = [xk;xk1;xk2]; % Stack up the vector
76
77      % Calculate he remaining values
78      rem1 = dt*(-wy(i)*vz(i) - g*sin(thetarad(i)));
79      rem2 = dt*(wy(i)*vx(i) + g*cos(thetarad(i)));
80      rem = [rem;rem1;rem2]; % Stack up the vector
81
82 end % i loop
83
84 % Check the condition number of the matrix
85 condition = cond(X);
86
87 % Estimate the unknown parameters [C_T,C_L,C_D]
88 % Use the first 68% of the data for LSR and the last 32% for validation
89 n = round(length(xkp1) * 0.68); % index for the first 68% of the data
90
91 % Use optimization to introduce upper and lower bounds to the unknowns
92 % eta1 = pinv(X)*(xkp1 - xk - rem);
93 eta1 = quadprog(2*((X)')*X,-2*(X)'*(xkp1 - xk - rem)
       ,[],[],[],[],[0;-1;0],[1;1;1]);
94
95 % Check regression error
```

```
96 err = sqrt((norm((xkp1 - xk - rem) - X*eta1))^2/(length(xkp1)*(norm(xkp1 -
       xk - rem))^2))
97 singval = svd(X);
98
99 % Compare model and experiment
100 xkp1m = xk(1:n) + X(1:n,:)*eta1 + rem(1:n); % regular data
101 xkp1m_val = xk(n+1:end) + X(n+1:end,:)*eta1 + rem(n+1:end); % validation
       data
102
103 % seperate the states for comparison with vx,vz
104 vxm = []; % modeled vx
105 vzm = []; % modeled vz
106 for i=1:2:length(xkp1m)-1
107     vzm = [vzm;xkp1m(i+1)]; % This may need to be vzm depending on n
108     vxm = [vxm;xkp1m(i)]; % This may need to be vxm depending on n
109 end % i loop
110 t = linspace(1,length(vxm),length(vxm))';
111
112 % seperate the states for validation with vx,vz
113 vxmval = []; % modeled vx
114 vzmval = []; % modeled vz
115 for i=1:2:length(xkp1m_val)-1
116     vxmval = [vxmval;xkp1m_val(i+1)]; % This may need to be vzm depending on
           n
117     vzmval = [vzmval;xkp1m_val(i)]; % This may need to be vxm depending on n
118 end % i loop
119 tval = linspace(1,length(vxmval),length(vxmval))';
120
121 % Plot comparisons  for regular data
122 figure
123 plot(t,vxm,'b')
124 hold on
125 plot(time(1:length(t)),vx(1:length(t)))
126 title('Vx Comparison','FontSize',16)
127 ylabel('vx [m/s]','FontSize',16)
```

```
128 xlabel('time [index]','FontSize',16)
129 legend('Estimated Data', 'Experimental Data')
130 set(gca,'FontSize',14);
131 hold off
132
133 figure
134 plot(t,vzm,'b')
135 hold on
136 plot(time(1:length(t)),vz(1:length(t)))
137 title('Vz Comparison','FontSize',16)
138 ylabel('vz [m/s]','FontSize',16)
139 xlabel('time [index]','FontSize',16)
140 legend('Estimated Data', 'Experimental Data')
141 set(gca,'FontSize',14);
142 hold off
143
144 % Plot comparisons  for validation data
145 figure
146 plot(tval,vxmval,'b')
147 hold on
148 plot(tval,vx(length(t):end-abs(length(vx(length(t):end)) - length(tval))))
149 title('Vx Validation','FontSize',16)
150 ylabel('vx [m/s]','FontSize',16)
151 xlabel('time [index]','FontSize',16)
152 legend('Estimated Data', 'Experimental Data')
153 set(gca,'FontSize',14);
154 hold off
155
156 figure
157 plot(tval,vzmval,'b')
158 hold on
159 plot(tval,vz(length(t):end-abs(length(vz(length(t):end)) - length(tval))))
160 title('Vz Validation','FontSize',16)
161 ylabel('vz [m/s]','FontSize',16)
162 xlabel('time [index]','FontSize',16)
```

```
163 legend('Estimated Data', 'Experimental Data')
164 set(gca,'FontSize',14);
165 hold off
166
167 end % lsr
```

B.2 Code For Data Manipulation Assuming Polynomial Unknown Coefficients

The driver code and the LSR code is slightly different for the polynomial unknown case. The following code has been modified from the code for the constant unknown parameter case.

Listing B.8: Driver code for data processing, LSR , and PID control

```
1 % Clayton Spencer
2 % This script is driver code to manipulate and process experimental flight
3 % data in support of a Masters Thesis
4
5 close all; clear all; clc
6
7 % Add the names of all the .mat files that need to be read in
8 fltdat = ["Test1_1_1_2.mat", "Test1_3_1_4", "Test2_1_2_2", "Test2_3.mat",];
9
10 % Test individual flight files using the below toggles
11 % fltdat = ["Test1_1_1_2.mat"];
12 % fltdat = ["Test1_3_1_4.mat"];
13 % fltdat = ["Test2_1_2_2.mat"];
14 % fltdat = ["Test2_3.mat"];
15 % fltdat = ["Test3_1.mat"];
16
17 % create the storage vectors for all the necessary parameters
18 u = []; w = []; q = []; theta = []; m1 = []; m2 = []; tlt = [];
19 time = cell(1, length(fltdat)); % time array for each flight time vector
20 timeL = [];
21
22 % Loop through all the flight logs to collect all of the flight data
```

```
% current data structure: data = [u,w,q,theta,m1,m2,tlt]
for i = 1:length(fltdat)
    [tempt, tempd] = simple_data_manipulation(fltdat(i));

    % concatenate all the data into their own column vectors
    u = [u;tempd(:,1)]; w = [w;tempd(:,2)]; % velocities
    q = [q;tempd(:,3)]; % rotation rate about y axis wy
    theta = [theta;tempd(:,4)]; % Euler angles
    m1 = [m1;tempd(:,5)]; m2 = [m2;tempd(:,6)]; tlt = [tlt;tempd(:,7)]; %
        control signals

    % Store the time vectors from each test flight
    timeL = [timeL;tempt]; % long time vector of all the time stamps
        concatenated
    time{i} = {tempt}; % Cell array to contain columns of different lengths
end % i loop

% add all of the desired data into one vector
data = [u,w,q,theta,m1,m2,tlt];

%% Virtualize the time stamps
% Because each flight during each test has different timestamps that
% relate to the Epoch 01/01/1970, data cannot be plotted in time as is. To
% get around this, a "virtual timestamp" is created. Doing this will lose
% the actual reference to "real" time of the data. However, the data can be
% plotted in one large sequence to demonstrate the continuous nature of the
% data as it is fed into System Identification algorithm. The time vector
% will then be a sequence of integers corresponding to each new data point.

virtime = virtualize(timeL); % create the virtual time vector

% Use the virtual time vector for plotting the concatonated flight data
% State plots
figure
subplot(2,2,1)
```

```
56 plot(virtime,data(:,1))
57 % title('Ground Speed','FontSize',16)
58 % xlabel('time [index]','FontSize',16)
59 ylabel('Vx [m/s]','FontSize',16)
60 set(gca,'FontSize',14);
61
62 subplot(2,2,2)
63 plot(virtime,data(:,2))
64 % title('Verticle Speed','FontSize',16)
65 % xlabel('time [index]','FontSize',16)
66 ylabel('Vz [m/s]','FontSize',16)
67 set(gca,'FontSize',14);
68
69 subplot(2,2,3)
70 plot(virtime,data(:,3))
71 % title('Pitch Rate','FontSize',16)
72 xlabel('time [index]','FontSize',16)
73 ylabel('q [deg/s]','FontSize',16)
74 set(gca,'FontSize',14);
75
76 subplot(2,2,4)
77 plot(virtime,data(:,4))
78 % title('Elevation Angle','FontSize',16)
79 xlabel('time [index]','FontSize',16)
80 ylabel('\theta [deg]','FontSize',16)
81 set(gca,'FontSize',14);
82
83 % Input signal plots
84 figure
85 subplot(3,1,1)
86 plot(virtime,data(:,5))
87 % title('Motor 1 and 3','FontSize',16)
88 % xlabel('time [index]','FontSize',16)
89 ylabel('M1 PWM [\mu/s]','FontSize',16)
90 set(gca,'FontSize',14);
```

```
91
92 subplot(3,1,2)
93 plot(virtime,data(:,6))
94 % title('Motor 2 and 4','FontSize',16)
95 % xlabel('time [index]','FontSize',16)
96 ylabel('M2 PWM [\mu/s]','FontSize',16)
97 set(gca,'FontSize',14);
98
99 subplot(3,1,3)
100 plot(virtime,data(:,7))
101 % title('Tilt Servo L/R','FontSize',16)
102 xlabel('time [index]','FontSize',16)
103 ylabel('tlt servo PWM [\mu/s]','FontSize',16)
104 set(gca,'FontSize',14);
105
106 %% LSR
107 % This section computes the Least Squares Regression for the data.
108 [eta,output] = lsr(data,virtime); % computes the Least Squares Regression to
       estimate eta
109
110 % Output the results
111 fprintf('a1: %f\n ', eta(1))
112 fprintf('a2: %f\n ', eta(2))
113 fprintf('a3: %f\n ', eta(3))
114 fprintf('b1: %f\n ', eta(4))
115 fprintf('b2: %f\n ', eta(5))
116 fprintf('b3: %f\n ', eta(6))
117 fprintf('c1: %f\n ', eta(7))
118 fprintf('c2: %f\n ', eta(8))
119 fprintf('c3: %f\n ', eta(9))
120
121 %% PID Tuning
122 % This section creates a linearized state space model and transfer function
123 % for the system. This allows the use of the PID tuner within simulink.
124 % Define the input and output of the system
```

```
125 n = round(length(output(:,1))*0.68);
126 vx = output(1:n,1); vxval = output(n:end,1);
127 vz = output(1:n,2); vzval = output(n:end,2);
128 wy = output(1:n,3); wyval = output(n:end,3);
129 theta = output(1:n,4); thetaval = output(n:end,4);
130 m1 = output(1:n,5); m1val = output(n:end,5);
131 m2 = output(1:n,6); m2val = output(n:end,6);
132 tlt = output(1:n,7); tltval = output(n:end,7);
133
134 % Create a State Space model for trim point 0
135 vx0 = 0; vz0 = vz(1); theta0 = 0; wy0 = 0; tlt0 = tlt(1); % initial
      conditions
136 rho = 1.215; % density of air [kg/m^3]
137 S = 0.725; % Characteristic area [m^2]
138 m = 11; % aircraft mass [kg]
139 g = 9.81; % acceleration due to gravity [m/s^2]
140 l = 0.362; % Characteristic length [m]
141 h = rho*S/(2*m);
142 Iyy = 0.4;
143
144 % Calculate C_T
145 th = sqrt(output(1,1)^2 + output(1,2)^2); % initial pitch angle
146 C_T = eta(1)*th^2 + eta(2)*th + eta(3);
147
148 % Create State Matrices
149 A0 = [0, 1;
150       0, 0];
151
152 B0 = [0;
153       2*C_T*l/Iyy];
154 C0 = [1, 0];
155 D0 = [0];
156
157 % Convert to transfer functions
```

```
158 [n0_1,d0_1] = ss2tf(A0,B0,C0,D0); % numerator and denominator corresponding
      to the first input
159
160 G0_1 = tf(n0_1,d0_1) % transfer function corresponding to the first input
161
162 figure
163 load PID_output.mat
164 plot(ans.Time,ans.Data(:,1));
165 hold on
166 plot(ans.Time,ans.Data(:,2));
167 title('Control Response','FontSize',16);
168 xlabel('Time [s]','FontSize',16);
169 xlim([0,3]);
170 ylabel('Pitch angle [rad]','FontSize',16);
171 legend('Target','System Response');
172 set(gca,'FontSize',14);
173 hold off
174
175 % Define state at end of transition
176 n = 2900; % data point number at end of first transition
177 vx1 = vx(n); vz1 = vz0; theta1 = theta(n); wy1 = wy(n); tlt1 = tlt(n); %
      initial conditions
178
179 % Calculate C_T
180 th = sqrt(output(n,1)^2 + output(1,2)^2); % initial velocity
181 C_T1 = eta(1)*th^2 + eta(2)*th + eta(3);
182
183 % Create State Matrices
184 A1 = [0, 1;
185       0, 0];
186
187 B1 = [0;
188       2*C_T1*l/Iyy];
189 C1 = [1, 0];
190 D1 = [0];
```

```
191
192 % Convert to transfer functions
193 [n1_1,d1_1] = ss2tf(A1,B1,C1,D1,1); % numerator and denominator
       corresponding to the first input
194
195 G1_1 = tf(n1_1,d1_1) % transfer function corresponding to the first input
196
197 figure
198 load PID_output_end.mat
199 plot(ans.Time,ans.Data(:,1));
200 hold on
201 plot(ans.Time,ans.Data(:,2));
202 title('Control Response End','FontSize',16);
203 xlabel('Time [s]','FontSize',16);
204 xlim([0,3]);
205 ylabel('Pitch angle [rad]','FontSize',16);
206 legend('Target','System Response');
207 set(gca,'FontSize',14);
208 hold off
209
210
211 disp('Program: Simple Driver Finished');
```

Listing B.9: Code which computes the LSR for estimating the polynomial unknown parameters

```
1 function [eta1,output] = lsr(data,time)
2 % This function computes the Least Squares Regression for flight data. The
3 % result is the estimated unknown parameters of the dynamic model of the
4 % aircraft
5
6 % Parse the data out into individual signals for readability's sake. The
7 % expected order of the data is: data = [vx,vz,wy,theta,m1,m2,tlt];
8 vx = data(:,1); % aircraft velocity in the x direction [m/s]
9 vz = data(:,2); % aircraft velocity in the z direction [m/s]
10 wy = data(:,3); % angular velocity about the y-axis [rad/s]
```

```
theta = data(:,4); % elevation angle [deg]
m1_pwm = data(:,5); % PWM signal for motor 1 and 3 [\mus]
m2_pwm = data(:,6); % PWM signal for motor 2 and 4 [\mus]
tlt_pwm = data(:,7); % PWM signal for tilt servos left and right [\mus]

% initialize vectors for the motors and the tilt servo
m1rad = zeros(length(m1_pwm),1);
m2rad = zeros(length(m2_pwm),1);
tltdeg = zeros(length(tlt_pwm),1);
tltrad = zeros(length(tlt_pwm),1);
thetarad = zeros(length(theta),1);

% The PWM signals need to be converted into angular velocites. This is done
% using experimental mappings.
% Loop through the motor and tilt servo signals to convert the signals from
% their raw form to standard units
m1 = zeros(length(m1_pwm),1);
m2 = zeros(length(m2_pwm),1);
for i=1:length(m1_pwm)
    % Conversions for calculation and plotting
    m1(i) = (-0.0175*m1_pwm(i)^2 + 61.681*m1_pwm(i) - 42981); % [RPM]
    m2(i) = (10.692*m2_pwm(i) - 10777); % [RPM]
    tltrad(i) = deg2rad((tlt_pwm(i) - 1180)/7.7396); % [rad]
    m1rad(i) = (2*pi/60)*(m1(i)); % [rad/s]
    m2rad(i) = (2*pi/60)*(m2(i)); % [rad/s]
    tltdeg(i) = rad2deg(tltrad(i)); % [deg]
    thetarad(i) = deg2rad(theta(i)); % [rad]
end % i loop

% Output the converted data
output = [vx,vz,wy,thetarad,m1rad,m2rad,tltrad];

% % initialize constant values and storage vectors
dt = 0.01; % time step [s]
rho = 1.215; % density of air [kg/m^3]
```

```
S = 0.725; % Characteristic area [m^2]
X = zeros(length(data(1)),4); % regressor matrix
m = 11; % aircraft mass [kg]
g = 9.81; % acceleration due to gravity [m/s^2]
L = 0.362; % Characteristic length [m]
X = []; % matrix of regressors
xkp1 = []; % state values at x_k+1
xk = []; % state values at x_k
rem = []; % remaining values
for i=1:length(data)-1
    % Calculate reused values
    lam = (2*sin(tltrad(i))*m1rad(i)^2)/m;
    mu = 2*(m1rad(i)^2*cos(tltrad(i)) + m2rad(i)^2)/m;
    xci = (rho*S/(2*m))*sqrt(vx(i)^2 + vz(i)^2);
    V = sqrt(vx(i)^2 + vz(i)^2);

    % Calculate the regressors for the first diff eq
    x11 = dt*lam*V^2; % order 2
    x12 = dt*lam*V; % order 1
    x13 = dt*lam; % order 0
    x14 = dt*xci*V^2*vz(i); % order 2
    x15 = dt*xci*V*vz(i); % order 1
    x16 = dt*xci*vz(i); % order 0
    x17 = -dt*xci*V^2*vx(i); % order 2
    x18 = -dt*xci*V*vx(i); % order 1
    x19 = -dt*xci*vx(i); % order 0

    % Calculate the regressors for the second diff eq
    x21 = -dt*mu*V^2; % order 2
    x22 = -dt*mu*V; % order 1
    x23 = -dt*mu; % order 0
    x24 = -dt*xci*V^2*vx(i); % order 2
    x25 = -dt*xci*V*vx(i); % order 1
    x26 = -dt*xci*vx(i); % order 0
    x27 = -dt*xci*V^2*vz(i); % order 2
```

```
81      x28 = -dt*xci*V*vz(i); % order 1
82      x29 = -dt*xci*vz(i); % order 0
83
84      % Organize into the regression matrix
85      Xtemp = [x11,x12,x13,x14,x15,x16,x17,x18,x19;
86               x21,x22,x23,x24,x25,x26,x27,x28,x29];
87
88      % Stack up the regressor matrix
89      X = [X;Xtemp];
90
91      % Calculate x_k+1
92      xk1_1 = vx(i+1);
93      xk1_2 = vz(i+1);
94      xkp1 = [xkp1;xk1_1;xk1_2]; % Stack up the vector truth value
95
96      % Calculate x_k
97      xk1 = vx(i);
98      xk2 = vz(i);
99      xk = [xk;xk1;xk2]; % Stack up the vector
100
101     % Calculate he remaining values
102     rem1 = -dt*(wy(i)*vz(i) + g*sin(thetarad(i)));
103     rem2 = dt*(wy(i)*vx(i) + g*cos(thetarad(i)));
104     rem = [rem;rem1;rem2]; % Stack up the vector
105
106 end % i loop
107
108 % Check the condition number of the matrix
109 condition = cond(X);
110
111 % Estimate the unknown parameters [C_T,C_L,C_D]
112 % Use the first 68% of the data for LSR and the last 32% for validation
113 n = round(length(xkp1) * 0.68); % index for the first 68% of the data
114
115 % Use optimization to introduce upper and lower bounds to the unknowns
```

```
116 Xqp = X(1:n,:);
117 xkp1qp = xkp1(1:n);
118 xkqp = xk(1:n);
119 remqp = rem(1:n);
120 thMID = 12.5; % min V value
121 thmax = 25; % max V value
122 A = [-thMID^2,-thMID,-1,0,0,0,0,0,0; % A matrix from Constraints for Ax =< b
123      -thmax^2,-thmax,-1,0,0,0,0,0,0;
124      0,0,-1,0,0,0,0,0,0;
125      0,0,0,-thMID^2,-thMID,-1,0,0,0;
126      0,0,0,-thmax^2,-thmax,-1,0,0,0;
127      0,0,0,0,0,-1,0,0,0;
128      0,0,0,0,0,0,-thMID^2,-thMID,-1;
129      0,0,0,0,0,0,-thmax^2,-thmax,-1;
130      0,0,0,0,0,0,0,0,-1;];
131 b = [0;0;0;1;1;1;0;0;0]; % b vector from Constraints for Ax =< b
132 eta1 = quadprog(2*((Xqp)')*Xqp,-2*(Xqp)'*(xkp1qp - xkqp - remqp),A,b
      ,[],[],[],[]);
133
134 % Check regression rrms error
135 % err = norm((xkp1 - xk - rem) - X*eta1)/norm(length(xkp1))
136 err = sqrt((norm((xkp1 - xk - rem) - X*eta1))^2/(length(xkp1)*(norm(xkp1 -
      xk - rem))^2))
137 singval = svd(X);
138
139 % Compare model and experiment
140 xkp1m = xkqp + Xqp*eta1 + remqp; % regular data
141 xkp1m_val = xk(n+1:end) + X(n+1:end,:)*eta1 + rem(n+1:end); % validation
      data
142
143 % seperate the states for comparison with vx,vz
144 vxm = []; % modeled vx
145 vzm = []; % modeled vz
146 for i=1:2:length(xkp1m)-1
147     vzm = [vzm;xkp1m(i+1)]; % This may need to be vzm depending on n
```

```
148     vxm = [vxm;xkp1m(i)]; % This may need to be vxm depending on n
149 end % i loop
150 t = linspace(1,length(vxm),length(vxm))';
151
152 % seperate the states for validation with vx,vz
153 vxmval = []; % modeled vx
154 vzmval = []; % modeled vz
155 for i=1:2:length(xkp1m_val)-1
156     vxmval = [vxmval;xkp1m_val(i+1)]; % This may need to be vzm depending on
          n
157     vzmval = [vzmval;xkp1m_val(i)]; % This may need to be vxm depending on n
158 end % i loop
159 tval = linspace(1,length(vxmval),length(vxmval))';
160
161 % Plot comparisons  for regular data
162 figure
163 plot(t,vxm,'b')
164 hold on
165 plot(time(1:length(t)),vx(1:length(t)))
166 title('Vx Comparison','FontSize',16)
167 ylabel('vx [m/s]','FontSize',16)
168 xlabel('time [index]','FontSize',16)
169 legend('Estimated Data', 'Experimental Data')
170 set(gca,'FontSize',14);
171 hold off
172
173 figure
174 plot(t,vzm,'b')
175 hold on
176 plot(time(1:length(t)),vz(1:length(t)))
177 title('Vz Comparison','FontSize',16)
178 ylabel('vz [m/s]','FontSize',16)
179 xlabel('time [index]','FontSize',16)
180 legend('Estimated Data', 'Experimental Data')
181 set(gca,'FontSize',14);
```

```
182 hold off
183
184 % Plot comparisons  for validation data
185 figure
186 plot(tval,vxmval,'b')
187 hold on
188 plot(tval,vx(length(t):end-abs(length(vx(length(t):end)) - length(tval))))
189 title('Vx Validation','FontSize',16)
190 ylabel('vx [m/s]','FontSize',16)
191 xlabel('time [index]','FontSize',16)
192 legend('Estimated Data', 'Experimental Data')
193 set(gca,'FontSize',14);
194 hold off
195
196 figure
197 plot(tval,vzmval,'b')
198 hold on
199 plot(tval,vz(length(t):end-abs(length(vz(length(t):end)) - length(tval))))
200 title('Vz Validation','FontSize',16)
201 ylabel('vz [m/s]','FontSize',16)
202 xlabel('time [index]','FontSize',16)
203 legend('Estimated Data', 'Experimental Data')
204 set(gca,'FontSize',14);
205 hold off
206
207 end % lsr
```