HYBRID SYSTEM BASED DESIGN FOR THE COORDINATION AND CONTROL OF MULTIPLE AUTONOMOUS VEHICLES

By

Charles A. Clifton

Thesis

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

in

Electrical Engineering

August, 2005

Nashville, Tennessee

Approved:

Professor Takkuen John Koo

Professor George Cook

In loving memory of Jonesey, Trudy, Buffy, Clemmy, Dharma, Kimbo and Gandalf

ACKNOWLEDGEMENTS

This work would not have been possible without the continuous help, encouragement and support of my advisor, Dr. T. John Koo. In particular, I would like to express my gratitude for allowing me to join his research team, and for his insightful advice in the areas of research, career development and life in general. Under his guidance, I have widened my understanding in many different areas which have been pivotal in writing this thesis.

Moreover, I would like to acknowledge the members of the Embedded Systems Computing Lab: Abhishek Dubey, Hang Su, Jie Chen and Xianbin Wu, for all the help and feedback they have given me. In addition, I would like to thank Michael Quottrup for his advice on Uppaal, and his design of the timed automata upon which part of my design is based. I would also like to recognize Lewis Saettel for helping to acquire the components that made this project possible.

Most importantly, I would like to thank my wife, Channing, for her unwavering emotional support and love, and my two dogs Roszá and Divina who provide much needed distractions.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Throughout human history, man's fascination with flight has been evident in literature, science and innovation. In Greek mythology, the skilled artificer, Daedalus, upon imprisonment on the island of Crete, set to work to fabricate wings for himself and his young son Icarus. By attaching feathers with wax and thread, he gave the wing a gradual curvature like the wing of a bird. After completing his work and upon waving his wings, the artist found himself buoyed aloft, supported by the beaten air. Equipping his son in a similar fashion, Daedalus cautioned Icarus not to fly too high as the heat of the sun would melt the wax, nor too low as the sea would wet the wings making them too heavy to fly. The father and son then flew away passing Samos, Delos and Lebynthos. Elated by his newfound ability, the boy began to soar upward as if to reach heaven. The Sun's heat melted the wax which was binding the feathers, and Icarus fell fatally into the sea as his father cried and bitterly lamented his own arts [12].

Perhaps the desire to soar is rooted in man's inherent need for freedom, to cast off the shackles of gravity and ground-based travel. In the late 1400s, Leonardo da Vinci made significant advancements in the study of flight. His contribution included over 100 drawings that illustrated his theories on flight, and although he would never see one of his machines in operation, his ideas and theories were revolutionary at the time. In fact, the modern day helicopter bears a striking resemblance to one of Leonardo's flying machines [13]. It would take several hundred more years for Leonardo's dream of manned flight to become a reality.

Although they are perhaps the most famous, the Wright brothers were not the first to achieve flight. In fact, Sir George Cayley, the inventor of the science of aerodynamics, built a successful passenger-carrying glider in 1853. But Orville and Wilbur Wright did build the first practical, *controllable* and self-powered airplanes. Their first successful test flights were in 1903, and by 1904 their Flyer III was capable of fully-controllable stable flight for substantial periods [14].

Motivation

The Wright brothers' successful demonstrations helped aircraft gain acceptance in the scientific and civilian communities as a viable option for transportation, recreation and eventually warfare. Since then, advancement in the technology has been rapid and the diversity of vehicles developed has become broad. In addition to airplanes, helicopters were also being constructed at the beginning of the 20th century. In 1907, the brothers Louis and Jacques Bréguet, inspired by the work of French scientist Charles Richet, built their first human-carrying helicopter, the Bréguet-Richet *Gyroplane No. 1*, a quad-rotor, consisting of four light, fabric covered biplane-type wings per rotor. Diagonally opposite pairs rotated in different directions, thereby canceling the torque produced by air drag on the rotating blades, and power was generated by a 40 h.p., 8-cylinder Antionette internal combustion engine. The pilot had no means of control and stability was found to be very poor. However, the vehicle was reported to have briefly lifted off the ground as high as 1.5 m (5 ft). Although detractors point out that the helicopter would have been operating in the ground effect, it proved for the first time that powered rotors had the ability for vertical lift of pilot and machine [19]. Nowadays, helicopters are indispensable for operations where a runway is unavailable, or where sustained hover is required. This is evidenced by their wide use in search and rescue, emergency medical and military transport, as well as media and law enforcement surveillance just to name a few.

In recent years, the use of unmanned aerial vehicles (UAVs) has gained considerable attention for applications in which manned operation is considered dangerous or infeasible. The current generation of military UAVs has been in development for defense applications since the late 1980s [7]. The Department of Defense (DoD) currently possesses five major UAVs: the Air Force's Predator and Global Hawk, the Navy and Marine Corps' Pioneer, and the Army's Hunter and Shadow. Military operations in Afghanistan and Iraq in 2002 and 2003 have gained UAVs considerable acceptance in their ability to observe, track, target and even attack enemy forces. During the 1990s, the DoD invested more than $3 billion in UAV development, procurement and operations. That number

is expected to reach $10 billion by 2010. In addition, Section 220 of the DoD Authorization Act for FY01 specifies that a goal for the armed services is for one-third of the operational deep-strike aircraft fleet to be unmanned by 2010 [7]. Often referred to as remotely piloted vehicles (RPVs), drones, or robot planes, UAVs are defined by the DoD as

> a powered air vehicle that does not carry a human operator; can be land-, air-, or ship-launched; uses aerodynamic forces to provide lift; can be autonomous or remotely piloted; can be expendable or recoverable and can carry a lethal or non-lethal payload [7].

UAVs offer many advantages over manned aircraft. They eliminate risk to the pilot, are lightweight, hard to detect, cheaper to procure and they can exhibit a longer operational presence. They don't require life-support systems for pilots and can perform missions in which extremely small aircraft are required. Radio-controlled pilotless aircraft were used widely as targets for anti-aircraft gunnery training during WWII [8]. In addition to fixed-wing aircraft, pilotless helicopters are also now widely available. Helicopter UAVs (HUAVs) have distinct advantages over their fixed-wing cousins, due to their versatility in maneuverability, ability to hover for prolonged periods of time and take off and land vertically. Most commercial and military helicopters consist of a single rotor (with a tail rotor to oppose induced moment). Although exceptionally practical in some environments, use of a single-rotor helicopter indoors or in confined spaces can be dangerous due to control difficulties and exposed rotor blades. Another option is the use of four-rotor helicopters, whose blades can be smaller and can approach an obstacle with less danger of striking a rotor. Unlike the early Richet-Bréguet-Richet *Gyroplane No. 1*, today's four-rotor helicopters are fully controlled as HUAVs and piloted machines.

A major challenge in the current generation of UAV solutions is the ability to guarantee mission completion involving multiple autonomous vehicles. Indeed, the DoD has acknowledged the need for cooperative UAV flight. Although conflicts between manned and unmanned systems in Kosovo were resolved by segregating the airspace, the ultimate goal is to integrate UAVs with existing manned aircraft for increased effectiveness. Unlike the Predator which requires a team of people including a remote pilot and sensor crew to

operate, the next generation of UAVs will be required to position themselves when and where commanded for optimum use, whether maintaining close formations with other aircraft or keeping station in wide spread constellations [26]. Therefore, a method is needed that will guarantee multiple-vehicle mission execution while ensuring that safety constraints are met. Because this formulation will act as an interface between the human and machine world, the language involved must be understandable to both humans and computers.

Hybrid System Based Design

In this thesis a model-based system approach is taken to solve the multi-vehicle coordination and control problem, relying on a hybrid model and the concept of a bisimulation relation and system composability. Based on the high level mission control objectives and the low level vehicle specifics, we develop the system with a "meet-in-the-middle" approach, developing the system inward from top and bottom as shown in Figure 1.

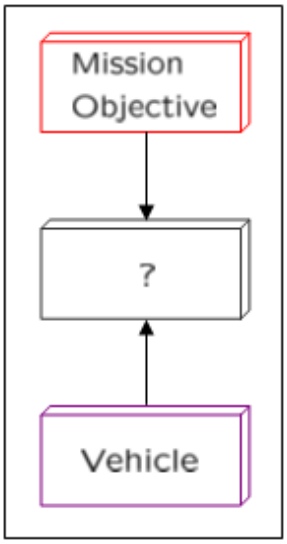


Figure 1: System design approach.

We can verify the design at each level and conclude with a high-level hybrid automaton (HA) model that is bisimular to the physical system and can be verified against specific constraints using verification tools that are commonly available.

A method for modeling the behavior of real-time systems has been developed in [2] using *timed automata*. A timed automaton (TA) is a finite state machine that has been extended with the use of clock variables. Several tools have recently been developed to design and analyze timed automata. One such tool is UppAal, developed jointly by Uppsala University and Aalborg University. UppAal is a tool designed to verify systems that can be modeled as networks of timed automata extended with integer variables, structured data types and channel synchronization [3]. Originally released in 1995, UppAal can be used to verify certain model specifications, for example, liveness and reachability, using computational tree logic (CTL). CTL is a formal specification language that can describe temporal attributes of a given system. The user can specify requirements such as "Vehicle A will never hit Vehicle B", or "Vehicle C will eventually reach destination X." It can be shown that the real-time system can be modeled using a multi-modal hybrid system [16]. By capturing the hybrid behaviors of the real-time multi-modal system, and provided that certain guarantees can be made about the real-time temporal properties, we shall see that the timed automaton can bisimulate the real-time system. Based on the CTL specifications, motion sequences generated by the UppAal model checker can be used as high-level commands for the real-time system.

Object of Study

In the Embedded Computing Software Laboratory (ECSL) at Vanderbilt University, we have developed the Vanderbilt Embedded Computing Platform for Autonomous Vehicles (VECPAV), an end-to-end design platform for the rapid development and deployment of control and motion planning solutions for unmanned aerial vehicles. This platform greatly simplifies and speeds the design and test stages of our model-based approach and allows us to demonstrate our methodology.

The final system is composed of several subsystems, utilizing a variety of concepts adopted from linear and nonlinear systems theory, state estimation theory, and hybrid and discrete event systems theory. At the lowest level, our focus is directed to the continuous-time control of an RC quad-rotor helicopter. The inherent instability of an

aerial vehicle such as this provides a challenging control problem. We address this problem using linear state feedback, nonlinear back-stepping, and approximate linearization [17] methods.

At the top layer we are interested in specifying a mission using a formal language that can be understood by humans and computers alike, thereby providing an interface through which an operator can communicate with the system. We use (CTL) as this interface. These specifications can be verified using UppAal and a timed automata model of the system based on the design by Quottrup et al. [27]. The verification can generate high-level motion commands to drive the vehicles.

We bridge the gap between the top and bottom levels by designing a multi-modal command interface to translate high level motion commands into signals that the real-time controller can understand. This is shown in Figure 2. The combination of the command

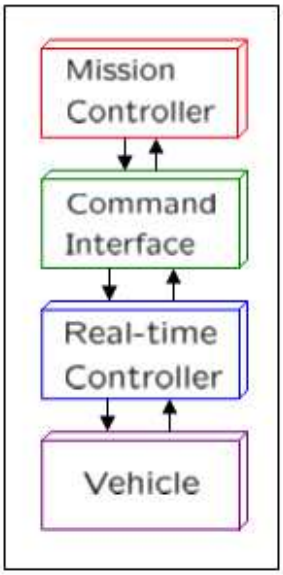


Figure 2: Hybrid system architecture.

interface and the real-time controller is then modeled using a hybrid automaton. It can be shown that there is a bisimulation relation between the timed automaton and the hybrid automaton. Thus, by using the commands generated by the TA verification as high level motion commands to the TA we can guarantee that the TA specifications will hold for the HA and our real-time hybrid system implementation.

Scope of the Thesis

In Chapter II, we present the techniques for developing the UAV model which is divided into two subsystems, a nonlinear continuous-time outer subsystem and a discrete-time linear inner subsystem. By assuming that the helicopter is a rigid body upon which forces and moments act, we derive a dynamical model based on the Newton-Euler equations. We obtain a model of the inner subsystem by performing system identification (SID). We obtain the SID input-output data by first flying the helicopter manually and recording the control and flight data. Later, we use the closed loop controller and excite the system by injecting a bandlimited signal onto each control input. Using the data gathered, we derive a 6^{th}-order, discrete-time linear model based on the SID Techniques from [20].

In addition, we show that we can provide full state feedback by using an Extended Kalman Filter (EKF) to do state estimation. The EKF serves two purposes: it provides filtered values of the measurement data which is corrupted with noise from the sensors, and it provides estimates of the system states that are not otherwise measurable. We show that the EKF can provide reasonable estimates of the system states, and explain how to construct the EKF from the discrete and continuous models to obtain estimates for the states of both inner and outer subsystems.

In Chapter III we present the controller design. Here we take advantage of the fact that the system is differentially flat. All states and outputs of differentially flat systems can be expressed as functions of the outputs and their derivatives [18]. The controller is divided into two subsystems, an outer nonlinear controller utilizing the differential flatness, and an inner controller designed using the model obtained from performing SID and the state estimates available from the EKF. We use the position and heading as outputs, which are available as measurements from the tracking system. However, taking their derivatives magnifies the noise. Instead we can use the state estimates from the EKF. We then use the SID model, along with the nonlinear outer model to design the controller in simulation using Simulink. Once a suitable controller is designed, it is tested in implementation and parameters are tuned accordingly. If the model is accurate,

the simulation reflects the actual implementation within a certain range and parameter tuning is minimal. The process can be repeated if the model is not representative of the actual system, which can happen if the dynamics are not sufficiently excited and since we do not model the system noise.

In Chapter IV, we focus on the hybrid system architecture. Using a multi-phase modeling approach consisting of the vehicle model, the real-time controller, a hybrid system based command interface and mission controller, we show that the total hybrid system can bisimulate the timed automata model created using the UppAal tool. In addition we can scale the system to include multiple vehicles. The system and scaling is shown in Figure 3.

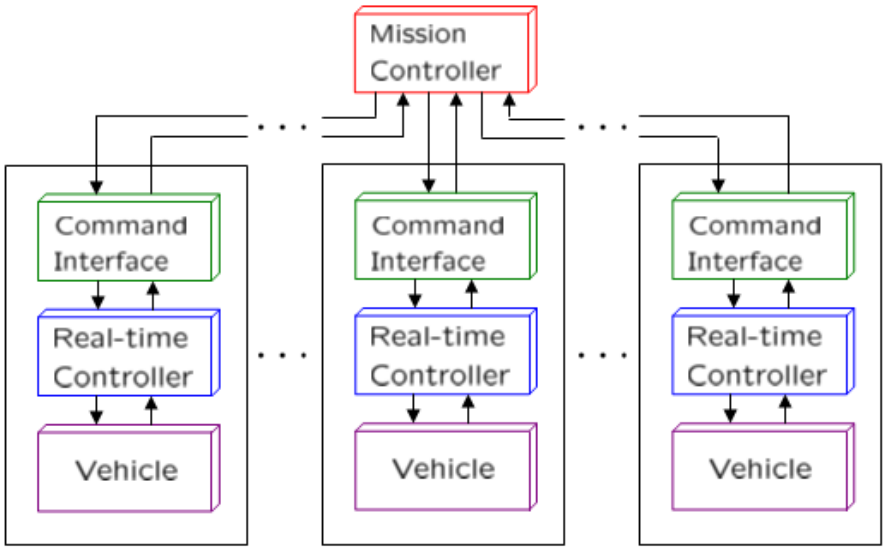


Figure 3: Hybrid system architecture for multiple vehicles.

By guaranteeing certain temporal properties of the real-time performance, we can create a series of [timed/event]-triggered commands to direct the robots to specific way-points, which have been verified against the specifications formulated using CTL for certain safety and reachability characteristics. We thereby coordinate the multi-robot movement while ensuring that certain constraints have not been violated.

In Chapter V, we give an overview of the integrated development platform (VEC-PAV) that has been designed in the ECSL at Vanderbilt. Although currently enabled

for two Draganflyers, the platform is designed in such a way that extending control to other types of vehicles is straightforward, and incorporating additional similar vehicles is simple. The platform contains two main components in addition to the vehicles and the model checker: a tracking system designed by Phoenix Technologies for highly accurate motion capture and real-time sensor data processing, and an off the shelf highly automated and integrated system for simulation, code generation, compilation, distribution and hard real-time processing developed by Opal-RT Technologies. The combination of these components greatly speeds the controller and system development and deployment phases by reducing the programming and compilation burden on the lab researchers, and eliminating the risks associated with translating code manually. In addition, the platform is scalable and flexible due to the vehicle-independent tracking system and the ability to simultaneously track a large number of vehicles.

CHAPTER II

SYSTEM MODEL AND STATE ESTIMATION

Helicopter Model

A UAV helicopter possesses certain abilities and advantages that make it highly desirable for both military and civilian applications in which other aircraft could not operate. Among these advantages are their versatility in maneuverability, ability for vertical takeoff and landing (VTOL), ability to hover for long periods of time and the fact they can operate in constrained locations like indoors or in urban environments. However, in order to take full advantage of these features, control and coordination systems should be in place that can carry out multi-agent, multi-objective missions and free the operator to concentrate on sensor information being relayed back. The design of these control and coordination systems can be highly challenging and complex. In order to reduce the complexity we can use compositional methods, which break the system into manageable subsystems. UAV flight management systems often contain controllers for switching between different modes of operation [16]. For example, there could be different control modes for hovering, trajectory following, and takeoff and landing. Combining the discrete and continuous elements of these systems can be a challenging aspect of the analysis and design. Fortunately, we can model the continuous and discrete components using a hybrid system, which is a system containing both discrete-time and continuous-time dynamics. Chapter IV will go focus on the hybrid system design and analysis. In this chapter we present the helicopter model and a method for estimating the states.

Control of a helicopter is accomplished by producing forces and moments to generate accelerations and thereby change the position, velocity and orientation of the vehicle. In order to sustain a hover, a helicopter must balance these forces that act to move it from its equilibrium. The most commonly known helicopter contains two rotors, an overhead rotor to provide thrust and thereby lift the helicopter off the ground, and a tail rotor to counterbalance the induced moment caused by air drag on the main rotor. Typical

helicopters can adjust the tilt of the main rotor to produce rolling and pitching action, and increase or decrease the tail rotor to affect the yaw. In addition, the pilot can adjust the vertical thrust produced by the main rotor. In contrast to the single-rotor helicopter, a quad-rotor like the Draganflyer IV, balances the induced moment by rotating two rotors clockwise and the other two rotors counterclockwise as shown in Figure 4.

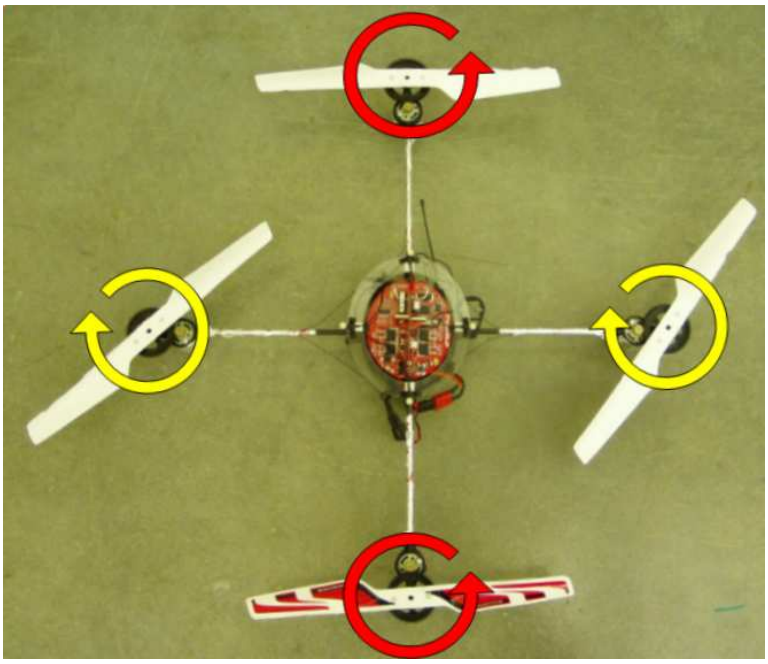


Figure 4: Draganflyer IV rotation of rotors.

The Draganflyer maneuvers by varying the lift force and induced moment of each of its four rotors. Because there is no control over blade attack angle or pitch angle, the Draganflyer can only change individual rotor lift forces and moments by varying the rotor speed as shown in Figure 5. Here the total thrust $T \in \mathbb{R}$, and torque $\tau^b \in \mathbb{R}^3$ is the sum of the forces f_1^b, f_2^b, f_3^b and f_4^b and moments τ_1^b, τ_2^b, τ_3^b and τ_4^b, respectively.

To make flying the vehicle easier, Draganfly Innovations Inc. has incorporated three gyroscopes and various onboard electronics. The electronics serve several purposes: to process the radio commands, help stabilize the roll, pitch and yaw rates, and map the commands on the four radio channels (ailerons, elevator, throttle, rudder) to appropriate rotor speeds. In order to model the quad-rotor helicopter, we divide the vehicle into

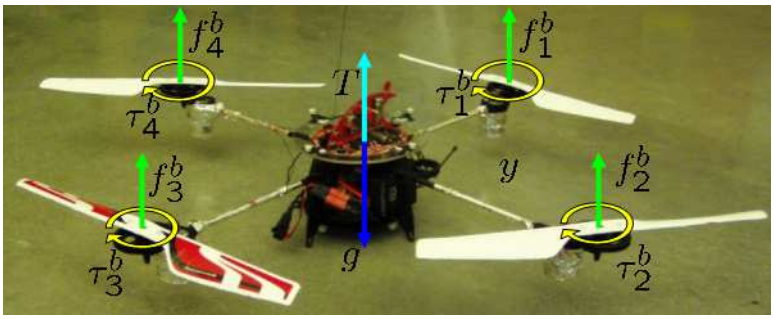


Figure 5: Individual forces generated by the Draganflyer.

three subsystems, the *onboard electronics*, *force and moment generation*, and *rigid body dynamics*. The combination of these subsystems, along with their respective inputs and outputs is shown in Figure 6. In this paper we regard the helicopter as a rigid body in

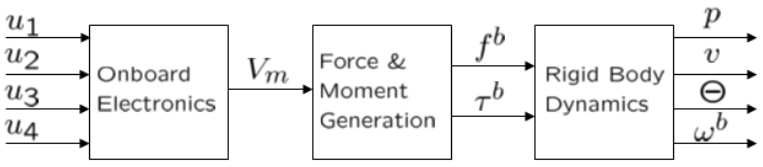


Figure 6: Block diagram of helicopter system.

order to apply the Newton-Euler equations of motion for such a system. In addition, because we do not have direct access to the motor speeds, as this is handled by the onboard electronics, we have chosen to combine the *onboard electronics* and *force and moment generation* systems and obtain an approximate model of this section of the helicopter system using system identification techniques. We will rely on this derived model throughout the design process for simulation and state estimation purposes.

Rigid Body Dynamics

The helicopter is a highly nonlinear dynamical system. Consider the system depicted in Figure 7. We can express the motion of the helicopter using the Newton-Euler equations for a rigid body. Rigid motion preserves the distances between points and the angles between vectors. Although the Draganflyer frame will flex and vibrate in operation,

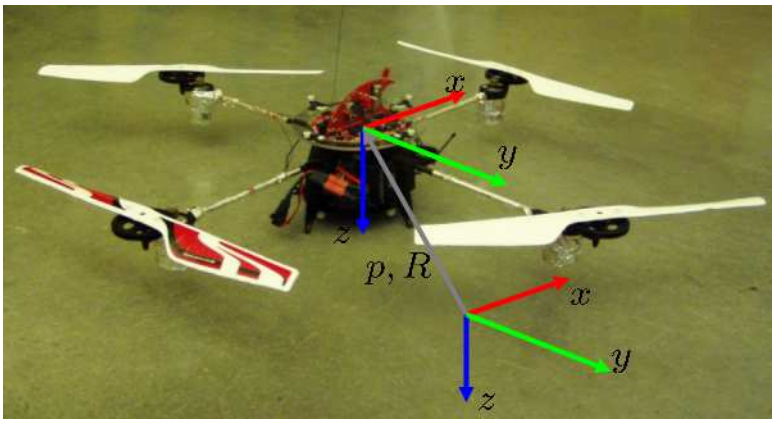


Figure 7: Helicopter coordinate system.

attempting to measure and model the dynamics and effects of this perturbation would be extremely difficult, and the overall effect would probably be negligible. Therefore, the rigid body assumption is reasonable. In addition, because the helicopter will be moving at slow speeds, the air drag acting on the frame during translational and rotational motion is small and will therefore be neglected in the modeling.

Newton equations

As in [25] let $p \in \mathbb{R}^3$ be the position and $R \in SO(3)$ a rotation matrix associated with a coordinate frame attached to the center of mass of a rigid body, relative to an inertial frame. The notation SO abbreviates *special orthogonal* as given in [25]. We let $f \in \mathbb{R}^3$ be the force applied at the center of mass m, with coordinates of f specified with respect to the inertial frame. Since the mass is constant, we can write the equations for the translational motion based on Newton's Law as

$$f = m\ddot{p} \tag{1}$$

which is independent of the angular motion because we have used the center of mass to represent the position.

Euler equations

The equations for the angular motion are derived by equating the applied torque to a change in the angular momentum of the system. The angular momentum is given by $\mathcal{I}'\omega^s$ where $\omega^s \in \mathbb{R}^3$ is the spatial angular velocity and

$$\mathcal{I}' = R\mathcal{I}R^T$$

is the instantaneous inertial tensor. The equations for the angular motion can then be found from

$$\tau = \frac{d}{dt}(R\mathcal{I}R^T\omega^s)$$

where $\tau \in \mathbb{R}^3$ is the torque specified relative to the inertial frame. Carrying out the differentiation yields

$$\begin{aligned} \tau &= \dot{R}\mathcal{I}R^T\omega^s + R\mathcal{I}\dot{R}^T\omega^s + R\mathcal{I}R^T\dot{\omega}^s \\ &= \dot{R}R^T\mathcal{I}'\omega^s + \mathcal{I}'R\dot{R}^T\omega^s + \mathcal{I}'\omega^s \\ &= \omega^s\mathcal{I}' \times \omega^s - \mathcal{I}'\omega^s \times \omega^s + \mathcal{I}'\omega^s \end{aligned}$$

where we have used the that $RR^T = I$, $\dot{R}R^T + R\dot{R}^T = 0$, where I is an identity matrix, and $\omega^s = \dot{R}R^T$. The term $\mathcal{I}'\omega^s \times \omega^s$ in the above equation is zero, giving the dynamic equation as

$$\omega^s\mathcal{I}' \times \omega^s + \mathcal{I}'\omega^s = \tau \tag{2}$$

which is called *Euler's equation*. Rewriting (1) and (2) in terms of the body coordinates yields the *Newton-Euler equation* given by

$$\begin{bmatrix} mI & 0 \\ 0 & \mathcal{I} \end{bmatrix} \begin{bmatrix} \dot{v}^b \\ \dot{\omega}^b \end{bmatrix} + \begin{bmatrix} \omega^b \times mv^b \\ \omega^b \times \mathcal{I}\omega^b \end{bmatrix} = \begin{bmatrix} f^b \\ \tau^b \end{bmatrix} \tag{3}$$

where $I \in \mathbb{R}^{3\times3}$ is an identity matrix, $v^b \in \mathbb{R}^3$ is the body velocity vector, $\omega^b \in \mathbb{R}^3$ is the body angular velocity vector and $\mathcal{I} \in \mathbb{R}^{3\times3}$ is the inertia tensor in body coordinates. The

position $p \in \mathbb{R}^3$ and velocity $v = \dot{p} \in \mathbb{R}^3$ of the center of mass are specified relative to the spatial frame in *North-East-Down* (x-y-z) configuration, where $x = [1\ 0\ 0]^T$, $y = [0\ 1\ 0]^T$ and $z = [0\ 0\ 1]^T$. The sum of forces and torques applied to the helicopter result from the combination of forces and moments produced by all the rotors. It is important to point out that there is significant cross coupling between forces and moments produced by the four rotors. For example, changes in the rolling (pitching) moments will have a direct effect on the lateral (longitudinal) acceleration. An in-depth analysis of the cross-couplings and inertias associated with the Draganflyer has been presented in [23]. The complexity required to model the individual rotor forces and moments is outside the scope of this paper, therefore we will model the helicopter as a lumped system and assume that all four rotor shafts are perfectly aligned with the (vertical) z-axis. We define the thrust, $T \in \mathbb{R}$, to be the combined lift force of the four rotors. Therefore, the body force $f^b = [0\ 0\ T]^T$, expressed in the body coordinates, contains no elements in x or y directions. In addition, the Inertia tensor $\mathcal{I}$ will be assumed to be an identity matrix. As shown in [17], for several choices of output variables exact input-output linearization fails to linearize the whole state space and results in having unstable zero dynamics. We will adopt the method given in [17] in which the weak couplings between forces and moments are neglected and an approximated model is used.

Let $R \in SO(3)$ be a rotation matrix that gives the orientation of the body coordinate frame with respect to the spatial coordinate frames. We next derive an expression for R parameterized by ZYX Euler angles $\Theta = [\phi\ \theta\ \psi]^T \in \mathbb{S}^3$ rotated about the x, y, z axes respectively. As shown in [25], if we rotate about a given axis $\alpha = [\alpha_1\ \alpha_2\ \alpha_3]^T \in \mathbb{R}^3$ at unit velocity for t units of time, then the net rotation is given by $R(\alpha, t) = e^{\hat{\alpha}t}$. Where $\hat{\alpha} \in so(3)$ is a skew-symmetric matrix given by

$$\hat{\alpha} = \begin{bmatrix} 0 & -\alpha_3 & \alpha_2 \\ \alpha_3 & 0 & -\alpha_1 \\ -\alpha_2 & \alpha_1 & 0 \end{bmatrix}. \tag{4}$$

We define the following elementary rotations about the x-, y-, z-axes:

$$R_x(\phi) := e^{\hat{x}\phi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\phi & -s\phi \\ 0 & s\phi & c\phi \end{bmatrix}, \tag{5}$$

$$R_y(\theta) := e^{\hat{y}\theta} = \begin{bmatrix} c\theta & 0 & s\theta \\ 0 & 1 & 0 \\ -s\theta & 0 & c\theta \end{bmatrix}, \tag{6}$$

$$R_z(\psi) := e^{\hat{z}\psi} = \begin{bmatrix} c\psi & -s\psi & 0 \\ s\psi & c\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{7}$$

where $s\theta$ and $c\theta$ are short for $\sin(\theta)$ and $\cos(\theta)$ respectively. Using these definitions we can derive the rotation matrix as

$$\begin{aligned} R(\Theta) &= e^{\hat{z}\psi}e^{\hat{y}\theta}e^{\hat{x}\phi} \\ &= \begin{bmatrix} c\theta c\psi & s\phi s\theta c\psi - c\phi s\psi & c\phi s\theta c\psi + s\phi s\psi \\ c\theta s\psi & s\phi s\theta s\psi + c\phi c\psi & c\phi s\theta s\psi - s\phi c\psi \\ -s\theta & s\phi c\theta & c\phi c\theta \end{bmatrix}. \end{aligned} \tag{8}$$

We obtain an expression for $\dot{\Theta}$ by direct differentiation of the rotation matrix R. As shown in [6] we see that

$$\dot{R}R^T = \begin{bmatrix} 0 & -\omega_z^b & \omega_y^b \\ \omega_z^b & 0 & -\omega_x^b \\ -\omega_y^b & \omega_x^b & 0 \end{bmatrix}, \tag{9}$$

where

$$\omega^b = \begin{bmatrix} \omega_x^b \\ \omega_y^b \\ \omega_z^b \end{bmatrix}. \tag{10}$$

From (9) we can extract the following independent equations

$$\begin{aligned} \omega_x^b &= \dot{r}_{31}r_{21} + \dot{r}_{32}r_{22} + \dot{r}_{33}r_{23} \\ \omega_y^b &= \dot{r}_{11}r_{31} + \dot{r}_{12}r_{32} + \dot{r}_{13}r_{33} \\ \omega_z^b &= \dot{r}_{21}r_{11} + \dot{r}_{22}r_{12} + \dot{r}_{23}r_{13} \end{aligned} \tag{11}$$

where $\dot{r}_{ij}$ and r_{ij} are the elements of the i^{th} row and j^{th} column of $\dot{R}$ and R, respectively and $i, j \in \{1, 2, 3\}$. It can be shown, as in [6], that the relation between $\dot{\Theta}$ and ω^b can be expressed as

$$\omega^b = E(\Theta)\dot{\Theta} \tag{12}$$

where $E(\Theta) \in R^{3\times 3}$ is a Jacobian given as

$$E(\Theta) = \begin{bmatrix} 1 & 0 & -s\theta \\ 0 & c\phi & c\theta s\phi \\ 0 & -s\phi & c\theta c\phi \end{bmatrix}. \tag{13}$$

Letting $\Psi(\Theta) = E^{-1}(\Theta)$ we get the following dynamic equation for the Euler angles:

$$\dot{\Theta} = \Psi(\Theta)\omega^b \tag{14}$$

$$= \begin{bmatrix} 1 & s\phi t\theta & c\phi t\theta \\ 0 & c\phi & -s\phi \\ 0 & s\phi/c\theta & c\phi/c\theta \end{bmatrix} \omega^b, \tag{15}$$

where $t\theta$ is an abbreviation for $\tan(\theta)$. The ZYX Euler angle parameterization of R contains singularities at $\theta = \pm\pi/2$. During normal operation, the trajectory of the helicopter should not pass through these singularities. However, if it happens that the

helicopter is forced to travel through the singularities, we can easily switch to another angle set convention for parameterization of the rotation matrix. Since $v = R(\Theta)v^p$, we can write the equations of motion as

$$\dot{p} = v, \tag{16}$$

$$\dot{v} = \frac{1}{m}R(\Theta)f^b, \tag{17}$$

$$\dot{\Theta} = \Psi(\Theta)\omega^b, \tag{18}$$

$$\dot{\omega}^b = I^{-1}(\tau^b - \omega^b \times I\omega^b). \tag{19}$$

As mentioned above, we combine the components of the helicopter into a lumped model in order to simplify the modelling. The onboard controller computes the necessary V_m for regulating the body angular velocities, ω^b, and thrust, T, according to the input signals, $\omega_d^b \in \mathbb{R}^3$ and $T_d \in \mathbb{R}$ provided by the radio transmitter.

System Composition

We divide the system into an outer and inner subsystem as shown in Figure 8.

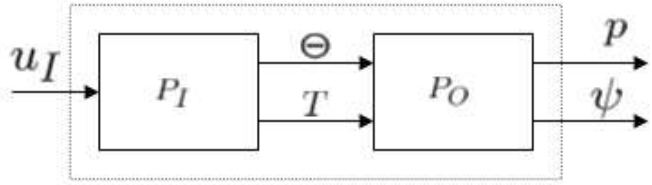


Figure 8: Inner and outer subsystems.

Outer Subsystem

The outer system is defined as

$$\Sigma_O : \begin{cases} y_O = \begin{bmatrix} p \\ \psi \end{bmatrix} \\ \dot{x}_O = f_O(x_O, y_I) \end{cases}, \tag{20}$$

where $y_O \in \mathbb{R}^3 \times \mathbb{S}$ is the outer output vector and $x_O = [p^T \ v^T]^T \in \mathbb{R}^6$ is the outer state vector. The vector field of the outer dynamics is defined by

$$f_O(x_O, y_I) = \begin{bmatrix} v \\ -\frac{1}{m} R(\Theta) e_3 T + e_3 g \end{bmatrix}, \tag{21}$$

where $e_3 = [0 \ 0 \ 1]^T$ and g is the acceleration due to gravity.

Inner Subsystem

The inner system is defined as

$$\Sigma_I : \begin{cases} y_I &= \begin{bmatrix} \Theta \\ T \end{bmatrix} \\ \dot{x}_I &= f_I(x_I, u_I). \end{cases} \tag{22}$$

where $y_I \in \mathbb{S}^3 \times \mathbb{R}$ is the inner output vector, $u_I = [\omega_d^{bT} \ T_d]^T \in \mathbb{R}^4$ is the inner input vector, and $x_I = [\Theta^T \ \omega^{bT} \ x_m^T]^T \in \mathbb{S}^3 \times \mathbb{R}^3 \times \mathbb{R}^{n_m}$ is the inner state vector. The vector $x_m \in \mathbb{R}^{n_m}$ is related to the dynamics of the motors and micro-controller. Since the onboard micro-controller is running much faster than our real-time controller, we assume that the motor and micro-controller dynamics are described by a set of continuous-time dynamical equations. The order $n_m \in \mathbb{R}$ will be determined during the system identification process. Because we collect system identification input-output data at a fixed sampling rate, the inner system model obtained will consist of a set of linear, discrete-time state equations to represent the actual system. The inner system consists of the SID model and the inner dynamic equations as shown in Figure 9. The vector field of the inner dynamics is defined by:

$$f_I(x_I, u_I) = \begin{bmatrix} \Psi(\Theta)\omega^b \\ I^{-1}(\tau^b - \omega^b \times I\omega^b) \\ f(x_m, u_I) \end{bmatrix}. \tag{23}$$

In the next section we obtain the inner model P_{Im} by performing system identification.

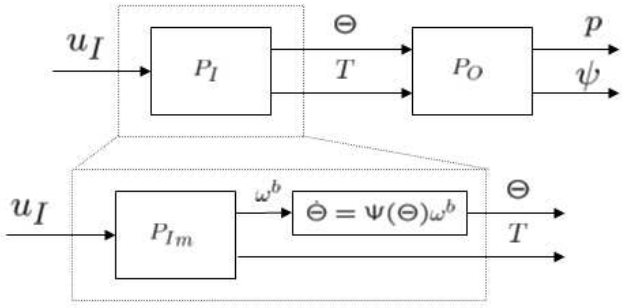


Figure 9: Inner subsystem components.

System Identification

Obtaining an accurate model for a helicopter that represents the system in different modes of operation can be extremely difficult. For this reason, a common method is to perform system identification on the vehicle during separate modes of operation and then use the appropriate model for each mode. Because our model will be moving relatively slowly, system identification (SID) is done in hover mode while exciting the inputs sufficiently enough that the subsequent change in dynamics will cover most motions we will perform during normal operation. By modelling the inner system using SID methods we avoid the complexities of trying to derive a mathematical model. In addition, the fact that we do not have access to the parameters associated with the onboard electronics, such as feedback gains, makes the internal vehicle dynamics somewhat of a mystery. In this section we first show how to linearize and discretize the continuous-time nonlinear model of the inner system. Then, a brief description of the SID process is presented, followed by an overview of the variables used, and finally the model and validation results will be discussed.

Linearization and Discretization

The standard nonlinear continuous-time system is given by

$$\dot{x} = f(x, u) \tag{24}$$

$$y = h(x) \tag{25}$$

where $x \in \mathbb{R}^n$ is the state vector, $u \in \mathbb{R}^m$ is the input vector, and $y \in \mathbb{R}^q$ is the output vector. The behavior of the system around an equilibrium point can be found by linearizing the system with respect to that point [15]. Expanding the equations in a Taylor Series about the equilibrium point (x_e, u_e), gives

$$\begin{aligned} \dot{x} &= f(x_e, u_e) + \frac{\partial f(x,u)}{\partial x}|_{x,u=x_e,u_e}(x - x_e) && (26) \\ &\quad + \frac{\partial f(x,u)}{\partial u}|_{x,u=x_e,u_e}(u - u_e) + H.O.T_f && (27) \\ y &= h(x)|_{x=x_e} + \frac{\partial h}{\partial x}|_{x_e}(x - x_e) + H.O.T_h && (28) \end{aligned}$$

where $H.O.T_f$ and $H.O.T_h$ are higher order terms in f and h respectively. Letting $\tilde{x} = x - x_e$, $\tilde{u} = u - u_e$, $\tilde{y} = y - y_e$, where $y_e = h(x)|_{x=x_e}$, we get

$$\begin{aligned} \dot{\tilde{x}} &= \dot{x} - \dot{x_e} && (29) \\ &= \frac{\partial f}{\partial x}|_{x,u=x_e,u_e}\tilde{x} + \frac{\partial f}{\partial u}|_{x,u=x_e,u_e}\tilde{u} + H.O.T_f && (30) \\ \tilde{y} &= \frac{\partial h}{\partial x}|_{x_e}\tilde{x} + H.O.T_h && (31) \end{aligned}$$

By examining the system in a sufficiently small region around (x_e, u_e), the $H.O.T._f$ and $H.O.T._h$ can be neglected. Therefore, we can drop these terms and approximate the nonlinear state equations with the following linear state equations:

$$\begin{aligned} \dot{\tilde{x}} &= A\tilde{x} + B\tilde{u} && (32) \\ \tilde{y} &= C\tilde{x} && (33) \end{aligned}$$

where $A = \frac{\partial f}{\partial x}|_{x,u=x_e,u_e}$, $B = \frac{\partial f}{\partial u}|_{x,u=x_e,u_e}$, and $C = \frac{\partial h}{\partial x}|_{x=x_e}$. This method can be extended to the discrete-time system with

$$\begin{aligned} x[k+1] &= F(x[k], u[k]) && (34) \\ y[k+1] &= H(x[k+1]) && (35) \end{aligned}$$

Following the same steps from above and letting $\tilde{x}[k] = x[k] - x_e$, $\tilde{u}[k] = u[k] - u_e$ and $\tilde{y}[k] = y[k] - y_e$ gives

$$\tilde{x}[k+1] = \frac{\partial F}{\partial x}|_{x,u=x_e,u_e}\tilde{x}[k] + \frac{\partial F}{\partial u}|_{x,u=x_e,u_e}\tilde{u}[k] + H.O.T_F \quad (36)$$

$$\tilde{y}[k+1] = \frac{\partial H}{\partial x}|_{x=x_e}\tilde{x}[k+1] + H.O.T_H \quad (37)$$

The $H.O.T_F$ and $H.O.T_H$ can once again be neglected by examining the system in a sufficiently small region around (x_e, u_e). Therefore, we can approximate the nonlinear discrete-time state equations with the following linear discrete-time state equations

$$\tilde{x}[k+1] = A\tilde{x}[k] + B\tilde{u}[k] \quad (38)$$

$$\tilde{y}[k+1] = C\tilde{x}[k+1] \quad (39)$$

where $A = \frac{\partial F}{\partial x}|_{x,u=x_e,u_e}$, $B = \frac{\partial F}{\partial u}|_{x,u=x_e,u_e}$, and $C = \frac{\partial H}{\partial x}|_{x=x_e}$

SID Process

The SID process can be divided into three stages: 1) *Data Acquisition* is performed to collect input and output data during manual or automated flight, 2) *Data Preprocessing* is then performed to obtain or compute relevant variables which are then scaled and detrended to within appropriate ranges, 3) finally, *Model Generation and Validation* using the processed IO data is performed. Since we can collect data in open- or closed-loop operation, we will use the *Direct Identification* approach [20], which uses the output of the process and input to the system in the same way for open- and closed-loop operation, ignoring any feedback and disregarding the reference signal in the identification procedure.

Data Acquisition

The first step in the SID process is IO data acquisition. We are interested in modeling the inner system whose dynamics are given in (23), specifically we will focus on the input[1] $u_I = [u_1\ u_2\ u_3\ u_4] \in [0, 255]^4$ sent from the radio transmitter to the helicopter which effects the output $y_m = [\omega_x^b\ \omega_y^B\ \omega_z^b\ T] \in \mathbb{R}^4$. The radio controls and helicopter motions can be seen in Figure 10. Each input channel from the radio transmitter has an effect on a

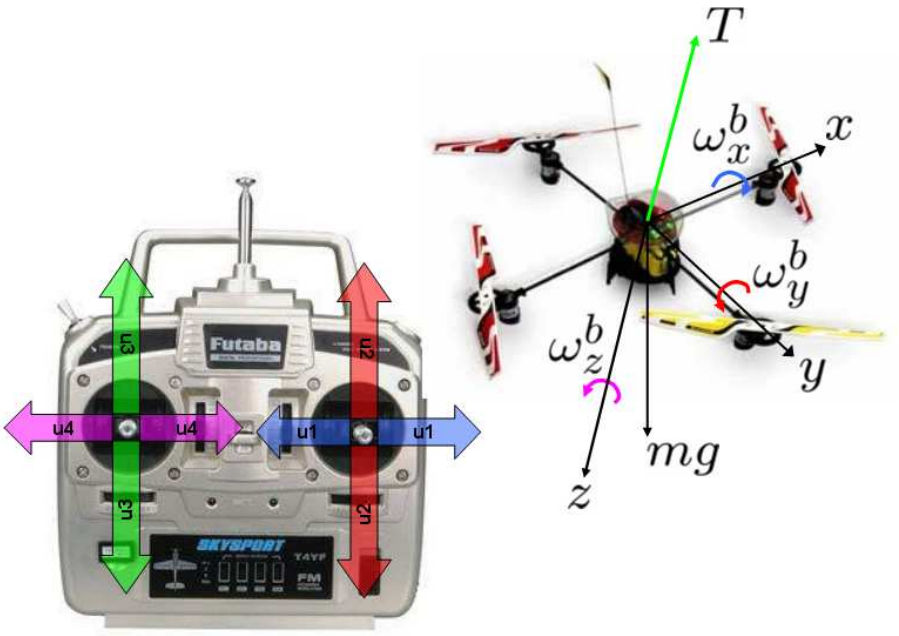


Figure 10: Radio controls and vehicle motions.

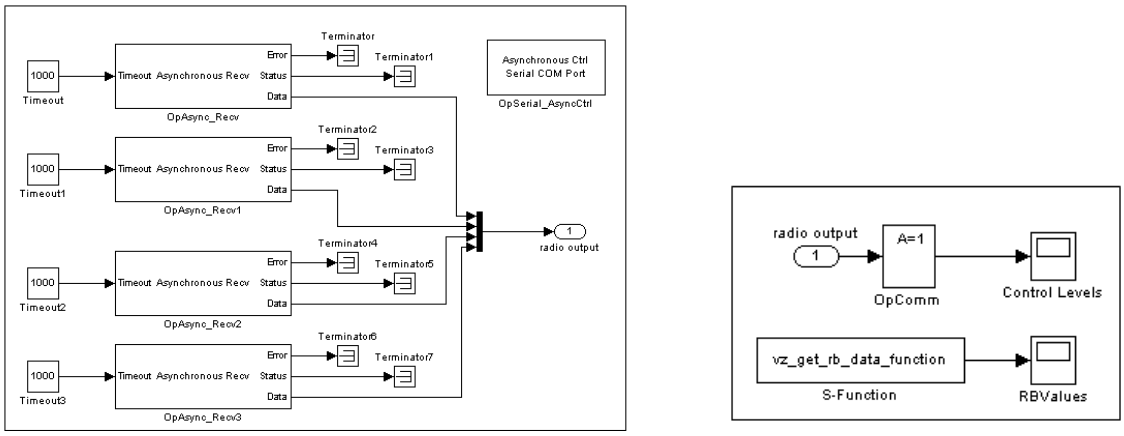
corresponding output of the helicopter. The correspondence is given as follows:

$$\begin{aligned} u_1 &\mapsto -\omega_x^b, \\ u_2 &\mapsto -\omega_y^b, \\ u_3 &\mapsto T, \\ u_4 &\mapsto \omega_z^b. \end{aligned} \tag{40}$$

The motion tracking system only provides us with measurements of the position p and the Euler angles Θ. Therefore, we can not directly measure the output values desired

[1]The radio transmitter sends an FM signal containing the four channels values (ailerons, elevator, throttle, rudder) to the helicopter. The received signal is translated into a PWM signal of four control values, each quantized by a byte $\in [0, 255]$.

for performing SID. However, due to the flatness of the model, we can obtain ω^b using (18) after differentiating Θ, and we can obtain T using (129) after twice differentiating p. Likewise, when using the model output values in simulation, we can use the same equations and derive Θ and p by integration. The IO data is aligned temporally by storing the data and time-stamp in a Simulink scope. The Simulink model is shown in



(a) Radio control signal reception

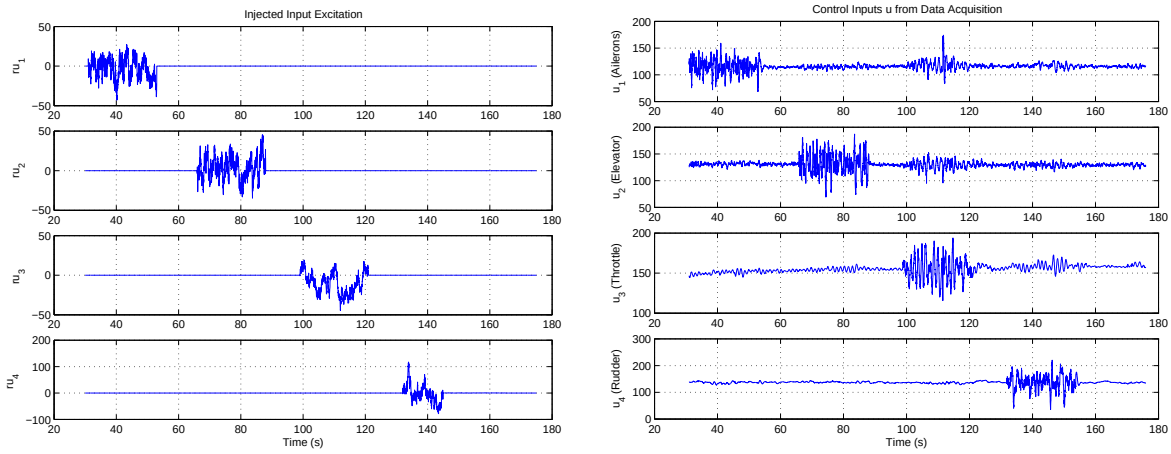
(b) Input output data capture

Figure 11: System identification data acquisition Simulink models.

Figure 11, where the S-function `vz_get_rb_data_function` is called to retrieve the rigid body data $[p^T \ \Theta^T]^T$ from the motion tracking system. Although the rigid body data is available from the motion tracking software at a rate of 100Hz, a fixed-step solver with a step size (i.e. sample step size) of $T_s = .022s$ is used during the data acquisition. The sampling rate is constrained by the bandwidth of the PWM signal sent to the radio controller, which is approximately 55Hz or 1 sample per $.018s$. The subsequent plots in this chapter detail the identification process performed on Draganflyer 1 (DF1) in closed-loop operation. Although the duration of flight was around 195 seconds, a window of 145s seconds of valid[2] data was used to perform the identification. In order to sufficiently excite the dynamics of the helicopter, we inject band-limited Gaussian noise into the control input stream. Because over-exciting the dynamics has the potential to make the system

[2]In practice the data collection starts and ends while the helicopter is resting on the ground. However, we are only interested in the data during flight. The time axis will reflect the window of time in which data is used for SID.

unstable or fly dangerously, there is a tradeoff between system safety and the range of dynamics represented by the model. The noise injection and closed loop control help to balance these conflicting issues and make the identification process easier and more productive. The noise injection signal is generated off-line and tested in simulation before being used in operation. If the noise signal is sufficient, it is saved to a file for use in the SID flight. The noise injection and augmented input control values for our SID process of



(a) Noise Injection

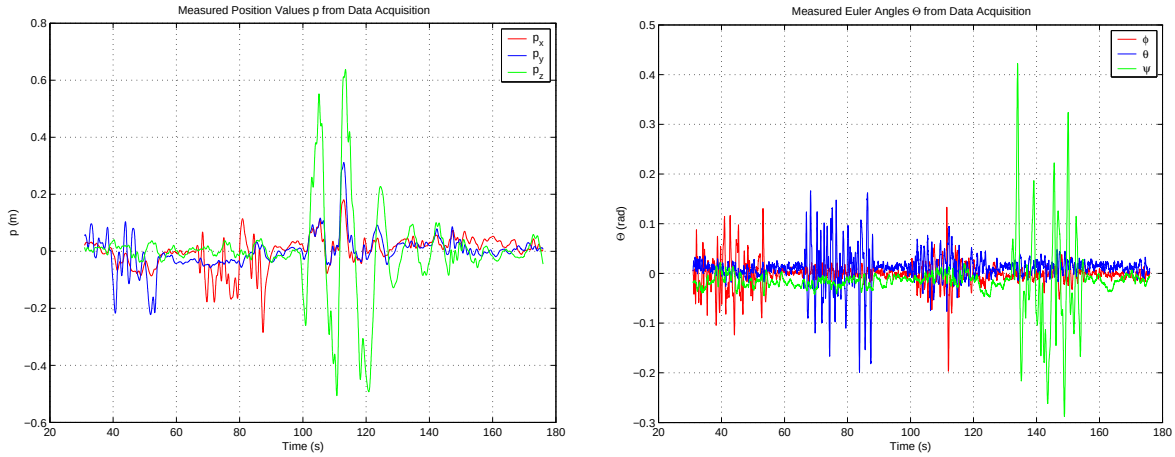
(b) Input Control Signal

Figure 12: Noise injection and control signal during closed-loop operation.

interest are shown in Figure 12. Since the controller is actively regulating the position of the helicopter, the augmented control signal is a combination of the noise injection and appropriate control reaction to position changes. The measured data values are given in Figure 13. The controller for the outer system attempts to regulate the position and heading of the helicopter about $[p_d^T \ \psi_d]^T = [0 \ 0 \ -.2 \ 0]^T$, where the units of p_d and ψ_d are m and $rads$, respectively. The real-time flight data is then saved to a file for offline data preprocessing.

Data Preprocessing

As shown in [20], data that has been collected for SID purposes may contain certain deficiencies that decrease the accuracy of the identification method. Among these are high frequency disturbances above the frequencies of interest to the dynamics, outliers or



(a) Position values

(b) Orientation values

Figure 13: Measured data collected during closed-loop operation.

gaps in the data record, and low frequency disturbances, i.e. drift and offset. Handling the low frequency disturbances can be accomplished by either pretreatment of the data, or allowing the noise model to take care of the disturbance. Several methods exist to deal with outliers and missing data including interpolation, Kalman Filter estimation, and data merging techniques. Prefiltering of the IO data can be done to remove high or low frequency disturbances that are not desirable in the modeling. An in-depth analysis of these methods is presented in Chapter 14 of [20]. In our approach we detrend and scale (i.e. pretreat) the input-output data to remove the low frequency disturbances, and then apply a low pass filter to reject the high-frequency disturbances from sensor noise.

Input-Output Data Detrending

We detrend the input data by subtracting the mean computed over the data set to obtain detrended input signals with a zero mean value. The trim values $u_e \in [0, 255]^4$ of the four control channels that keep the helicopter in stationary hover are found by averaging the set of input values that correspond to body angular velocity and translational acceleration equal to zero. In addition, because the helicopter depletes the battery during operation, the measured throttle value may have a slowly increasing drift as the controller seeks to compensate the loss in altitude. By finding a straight-line approximation of the throttle level over the entire data set, we can remove any drift that occurs. The input

data is next scaled via element-wise division by $u_{scale} = [-255 \ -255 \ 255 \ 255]^T$ to reside in the range $[-1, 1]$. The negative sign on the first two scale factors reflects the fact a given negative (positive) change in the input results in a positive (negative) change in the output. The input detrending is given as

$$\tilde{u}_I = \alpha_u(u_I) \tag{41}$$

$$= (u_I - u_{offset})./u_{scale} \tag{42}$$

where $\tilde{u}_I$ is the detrended input signal, $\alpha_u(u_I)$ is a function to scale and normalize the inputs, and ./ is element-wise division. For the output values, remember that although we measure and collect the position and orientation, we will use the body angular velocities and thrust, ω^b and T for our SID output variables. These can be computed due to the flatness of the system. Detrending is done on the orientation data to compensate for misalignment in sensor placement or vehicle center of gravity, both of which can effectively add an offset to the roll and pitch angles. Detrending Θ is necessary because both Θ and it's derivative $\dot{\Theta}$ are needed to compute ω^b, whereas only the second derivative of the position $\ddot{p}$ is needed to compute T, and therefore offsets in p will not have any effect on the thrust computation. The output values are detrended by subtracting an offset $y_{offset} = [0 \ 0 \ 0 \ mg]^T$ from the output values. The mass, m, can be found by weighing the helicopter and $g = 9.81 m/s^2$ is the acceleration due to gravity. We then scale the outputs via element-wise division by $y_{scale} = [5 \ 5 \ 5 \ mg]^T$ so that the resulting values reside in the range $[-1, 1]$ as we did with the inputs. This process is given as

$$\tilde{y}_m = \alpha_y(y_m) \tag{43}$$

$$= (y_m - y_{offset})./y_{scale} \tag{44}$$

where $\tilde{y}_m$ is the detrended output vector and $\alpha_y(y_m)$ is a function to scale and normalize the outputs. For simulating the final model, we need to be able to un-scale and remove the offset from the data generated by the model. We can obtain values in the range of

the original output data with a function given by

$$y_m = \beta_y(\bar{y}_m) \tag{45}$$

$$= \bar{y}_m * y_{scale} + y_{offset}. \tag{46}$$

Input-Output Data Filtering

We wish to filter out the high frequency noise that is associated with the sensors. We can do this with a low-pass filter, but we must first determine what an appropriate cutoff frequency is. Looking at the frequency spectrum of the input-output data gives us a better idea of what a reasonable cutoff frequency should be. We plot the frequencies using an FFT to obtain the frequency components from the time-domain information, as shown in Figures 14, 15 and 16 for inputs, body angular velocities and thrust, respectively.

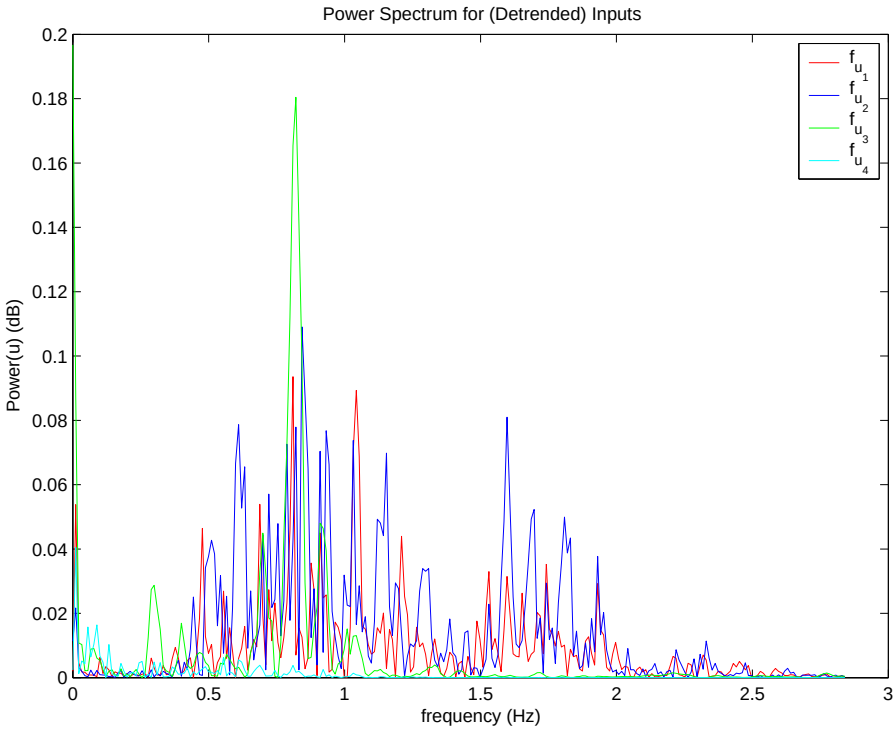


Figure 14: Frequency components of the input signals.

Based on the the results from the frequency decomposition, we choose a cutoff frequency of $f_c = 2.5$Hz. We then use this value to generate the coefficients for a 1^{st} order

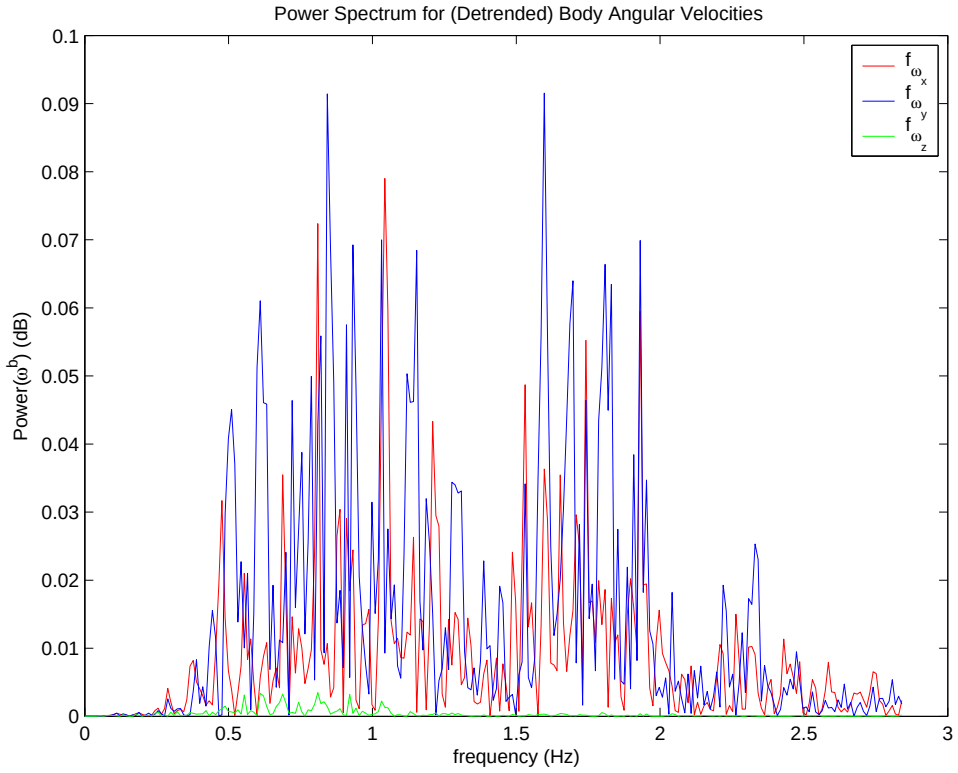


Figure 15: Frequency components of the body angular velocities.

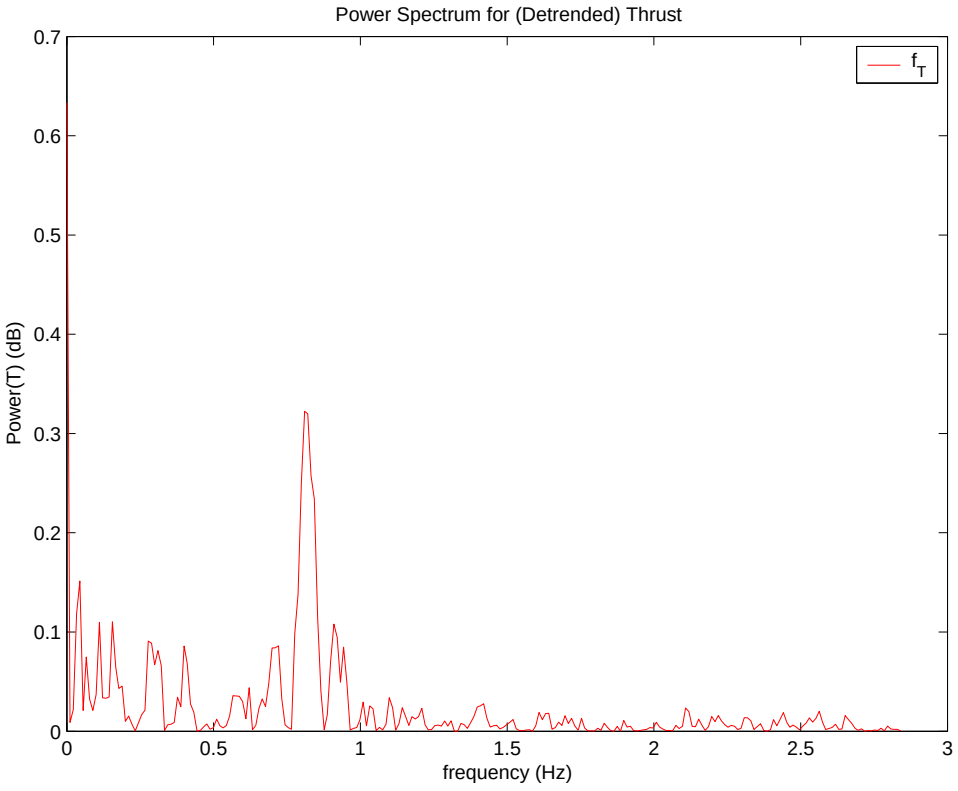


Figure 16: Frequency components of the thrust.

Butterworth filter. These coefficients are applied to a zero-phase forward and reverse digital filter[3]. After filtering in the forward direction, the filtered sequence is then reversed and run back through the filter. The result has zero phase distortion and magnitude modified by the square of the filter's magnitude response. We apply this filter to the detrended and scaled input and output values. Prefiltering the input and output data through the same filter does not change the input-output relation, however it does have an effect on the noise model[20]. But, because we do not use the noise model, this action is appropriate to remove the high frequency noise. The results of filtering the input values are shown in Figure 17. If we look closely at the plot for channel 3 of the input signal

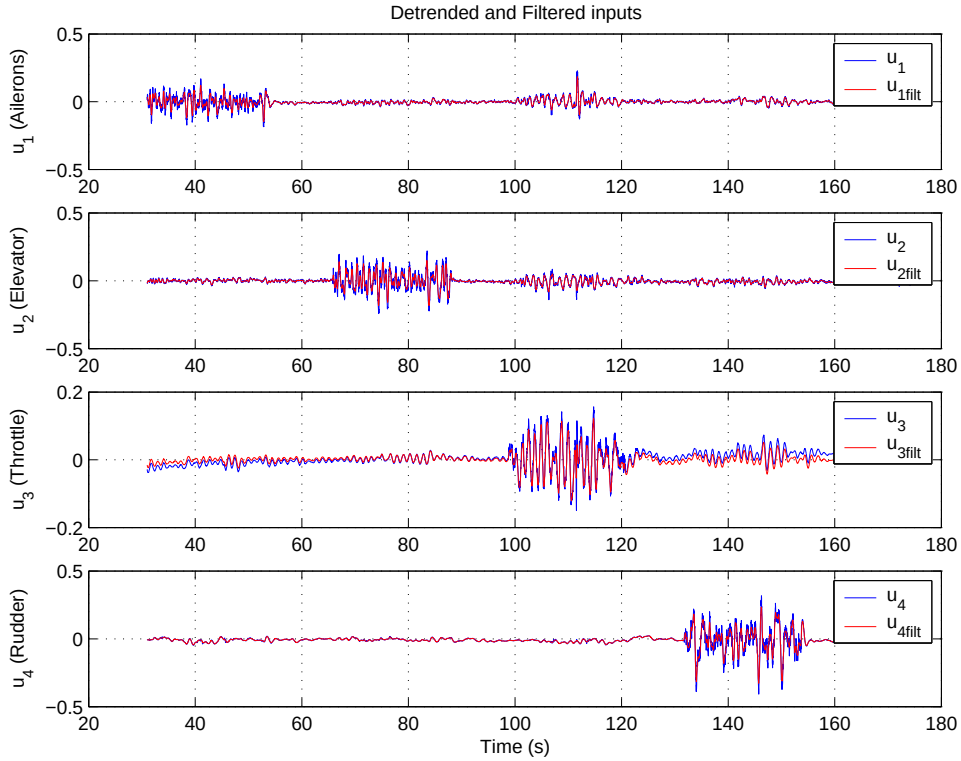


Figure 17: Input signal after detrending and filtering.

(i.e. the throttle, u_3) we can see that the slowly increasing drift has been removed in the detrended and filtered signal. The filtering results for ω^b and T are given in Figures 18 and 19, respectively. In Figure 19 we can see that the filtered thrust values start and trail below the actual data, caused by the filter initial conditions and the fact that the

[3]Uses the `filtfilt` command in MATLAB.

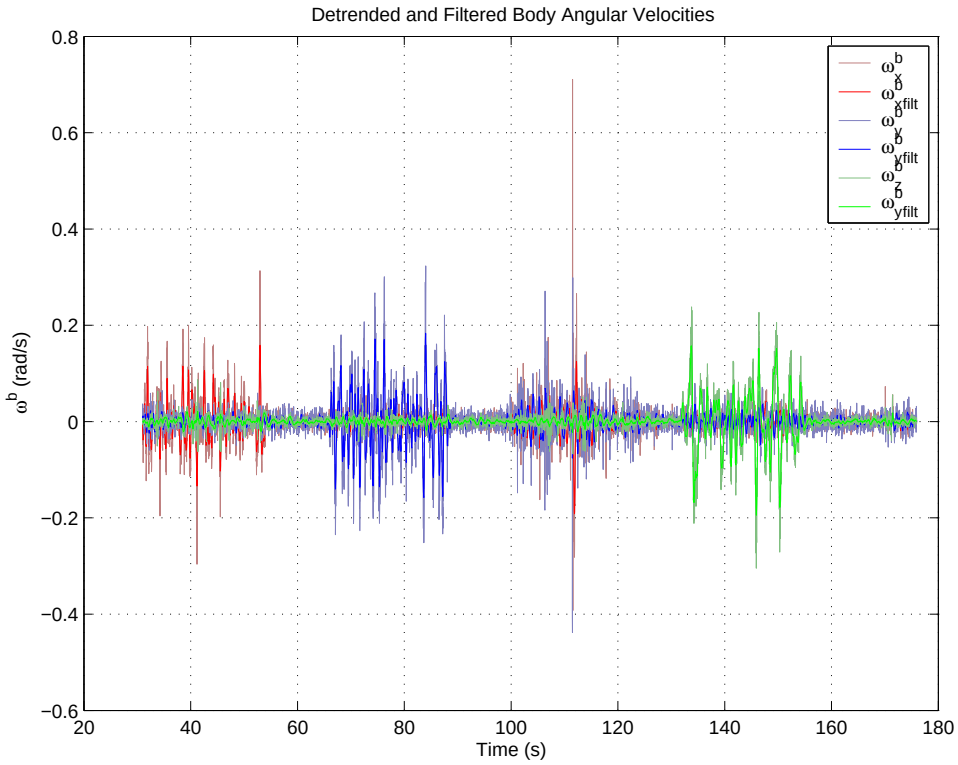


Figure 18: Detrended body angular velocities before and after filtering.

filter makes two passes. In order to discard the filter startup and end values which would make the model less accurate, we truncate the first and last 10 data points before saving the actual data to be used in the model fitting. Because computing the thrust requires calculating the first and second derivatives of the position, we see that the noise is magnified and there is considerable attenuation to the unfiltered thrust signal in the final filtered thrust signal. We thereby obtain the SID variables $\bar{u} \in [-1, 1]^4$ and $\bar{y} \in [-1, 1]^4$. A plot of the each input signal overlayed with the strongest correlated corresponding output signal ($u_1 \rightarrow y_1$, $u_2 \rightarrow y_2$, $u_3 \rightarrow y_4$, $u_4 \rightarrow y_3$) is shown in Figure 20. We can see even from a brief glance at this plot that the IO is highly correlated. In addition, we see that there exists some cross coupling most notably from u_3 to y_1,and y_2, but also from u_4 to y_1 and y_2. This effect will show up as non-zero off-diagonal elements[4] of the transfer function matrix $T(s) = C(sI - A)^{-1}B + D$, and can also be seen by looking at a plot of a step input response. Having obtained $\bar{u}_I$ and $\bar{y}_I$ we are now ready to find a model that sufficiently represents the system based on this IO data.

[4]After exchanging rows 3 and 4

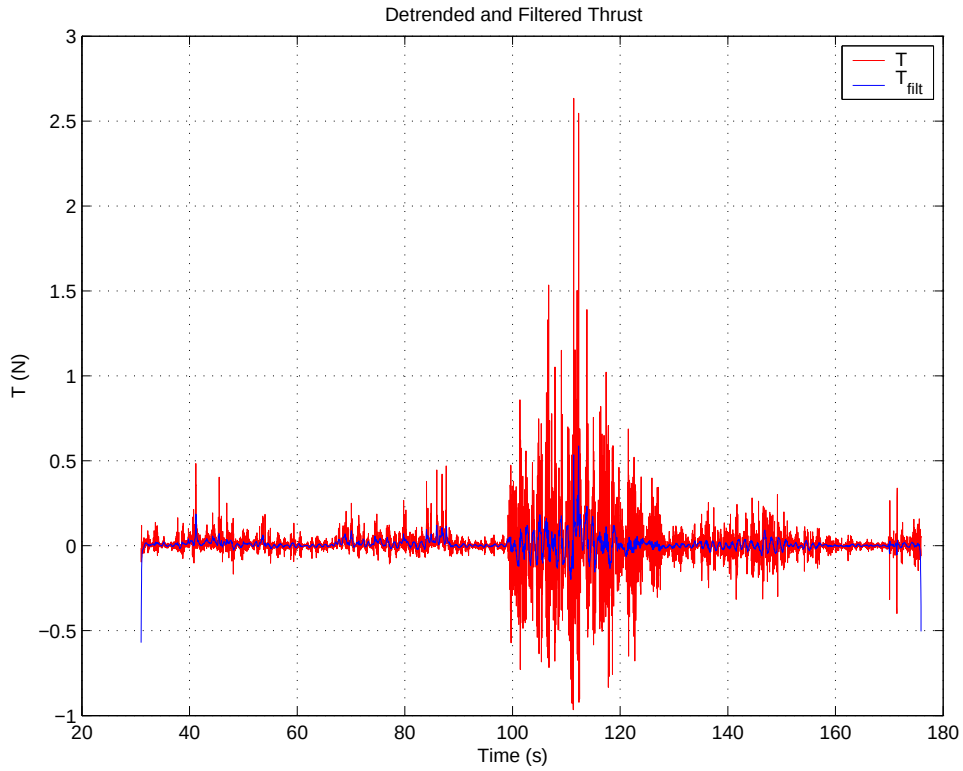


Figure 19: Detrended thrust before and after filtering.

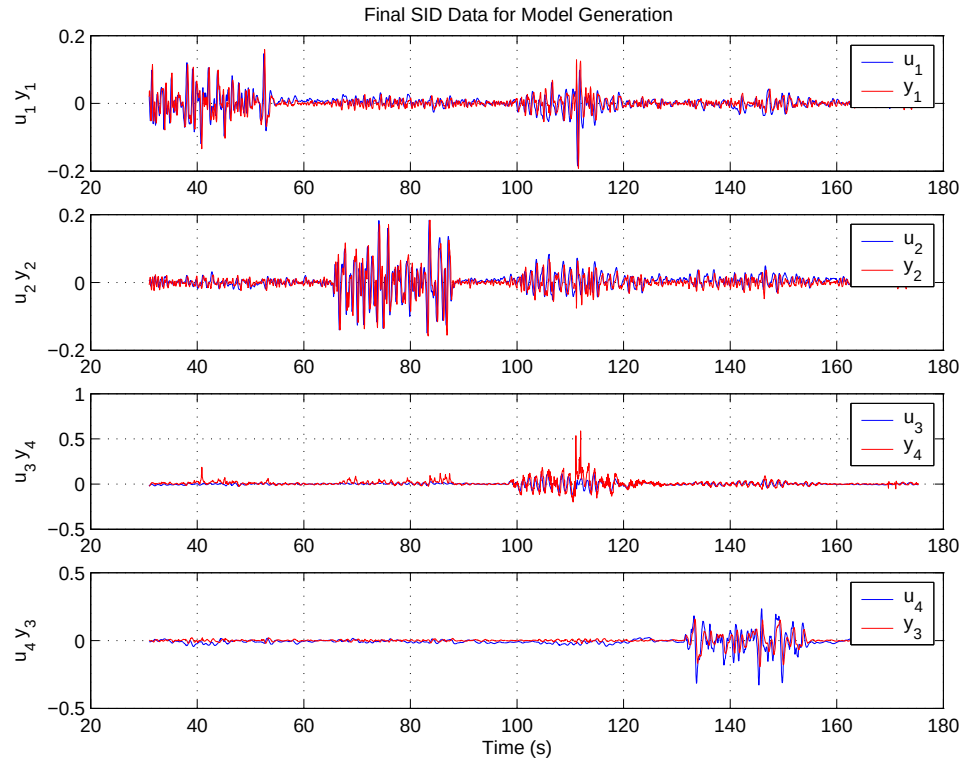


Figure 20: Preprocessed input output data for model generation.

Obtaining a Model

We generate a state space model using the `n4sid` method from the MATLAB System Identification toolbox. As shown in [20], the `n4sid` is a subspace-based method used to estimate the system matrices, A, B, C , D and K from the following state space model in the innovations form

$$\begin{aligned} x(t+T_s) &= Ax(t) + Bu(t) + Ke(t) \\ y(t) &= Cx(t) + Du(t) + e(t) \end{aligned} \tag{47}$$

The `n4sid` method obtains a least squares estimate of the A and C matrices from the IO data and appropriate weighting matrices, and then using linear regression techniques, obtains estimates of B, D, and $x(0)$. For our purposes we do not use the K matrix or $e(t)$ associated with the disturbance model and the D matrix is 0. After performing several SID iterations, it was found that for the Draganfly IV, a 6^{th}-order model generally was able to sufficiently represent the system dynamics in simulation and estimation purposes. Higher order models did not provide significant improvements in accuracy and served only to increase the computational cost. Lower order models often did not sufficiently represent the system behavior and performed poorly during the validation process.

Sixth-order Model

Using the IO data from preprocessing, we obtain the following parameters to our linear, discrete-time model based on the state space model in (47) with a step time of $T_s = .022s$, and neglecting $K(t)$ and $e(t)$:

$$A = \begin{bmatrix} 0.916 & 0.013 & -0.020 & 0.015 & 0.025259 & -0.015 \\ -0.004 & 0.840 & -0.006 & -0.017 & 0.19506 & -0.088 \\ 0.004 & 0.045 & 0.835 & -0.049 & 0.040995 & 0.176 \\ 0.011 & -0.008 & -0.001 & 0.877 & 0.021964 & 0.010 \\ 0.011 & -0.082 & -0.013 & 0.140 & 0.74224 & 0.096 \\ -0.024 & 0.019 & -0.142 & 0.151 & 0.057988 & 0.785 \end{bmatrix}$$

$$B = \begin{bmatrix} 0.0166 & 0.0406 & -0.0152 & 0.0613 \\ 0.150 & -0.022 & 0.082 & -0.042 \\ 0.068 & 0.071 & 0.090 & 0.027 \\ -0.019 & 0.007 & 0.101 & -0.014 \\ 0.057 & -0.282 & -0.052 & 0.043 \\ 0.316 & 0.310 & -0.108 & 0.013 \end{bmatrix}$$

$$C = \begin{bmatrix} -0.032 & 0.354 & 0.491 & 0.017 & 0.053 & 0.044 \\ -0.064 & -0.611 & 0.335 & 0.014 & -0.054 & 0.051 \\ 1.133 & -0.008 & 0.016 & 0.009 & 0.009 & -0.005 \\ 0.135 & 0.024 & 0.070 & 2.347 & -0.015 & -0.031 \end{bmatrix}$$

$$D = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The eigenvalues of the A matrix are given as:

$$\begin{aligned} \lambda_{1,2} &= 0.7595 \pm 0.1385i \\ \lambda_3 &= 0.9240 \\ \lambda_4 &= 0.8755 \\ \lambda_{5,6} &= 0.8394 \pm 0.1186i \end{aligned}$$

Examining the eigenvalues we see that they are all inside the unit circle, which for discrete-time systems implies that the system is stable. Although the poles all lie within the unit circle, several of the system zeros lie outside the unit circle, implying that the system is non-minimum phase. A pole-zero map of the model is shown in Figure 21.

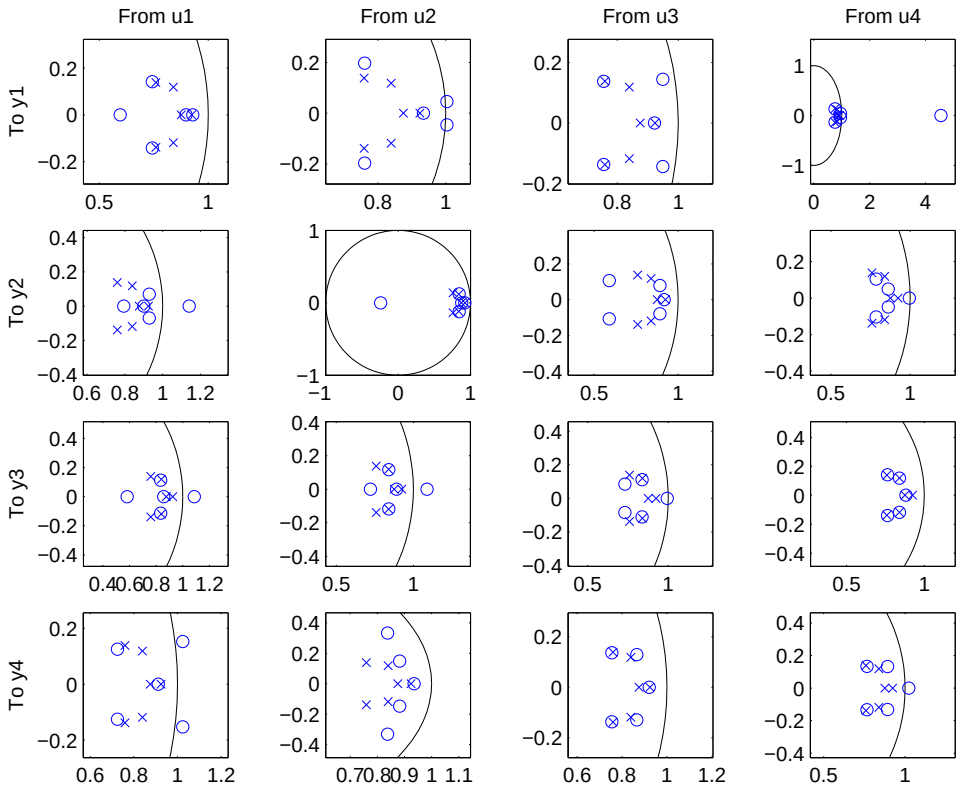


Figure 21: Pole-zero map of the model obtained via SID.

Model Validation

In order to validate the model that has been generated, we can simulate the model with inputs from the same data acquisition trial, or better yet, with data from a different data acquisition trial. We do this passing the input, $\bar{u}_I$, to the model and comparing the simulated model output y_s to the actual output $\bar{y}_m$. The validation results for the same data acquisition trial are given in Figure 22 and the results using a different validation data set are given in Figure 23. We can see that the model performs fairly well in simulating the actual output for the original data set. The fit of the model is computed by

$$fit = (1 - norm(\bar{y} - y_s)/norm(\bar{y} - mean(\bar{y}))) * 100. \tag{48}$$

In practice we have found that fit values above 30% indicate that the model in simulation will sufficiently represent the system in operation. The validation results using a different input output data set reinforce that the model is sufficient. The low fit value in y_4 is due to a very small dc offset, and upon examination of the simulated and actual data, they

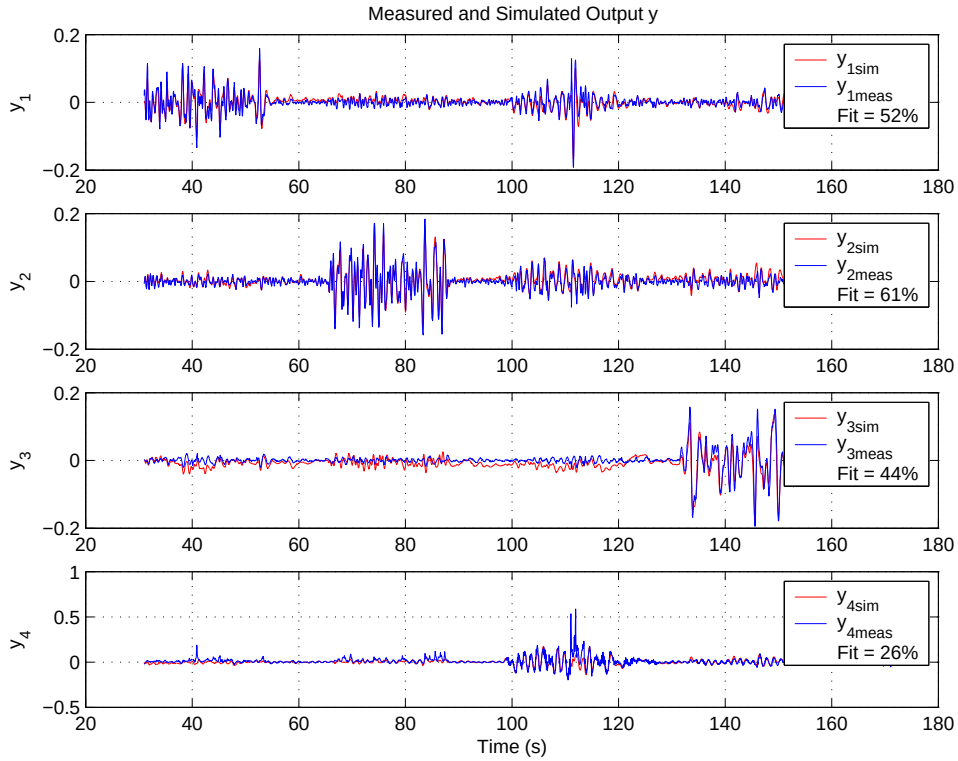


Figure 22: Model validation results using original input output data set.

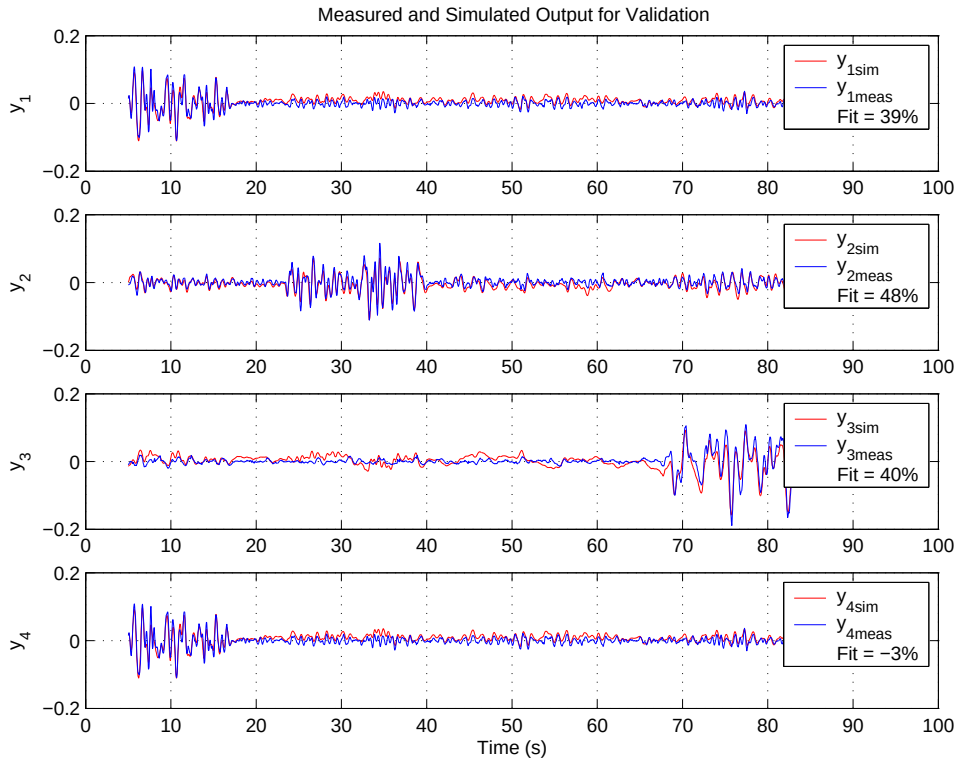


Figure 23: Model validation results using a different input output set.

are found to be reasonably close.

To see the effect of each input on the outputs, we can look at the response to a step input. We provide a (multi-)input step to the model and observe the output. The step response is given in Figure 24. From the step input we can see that the strongest input-

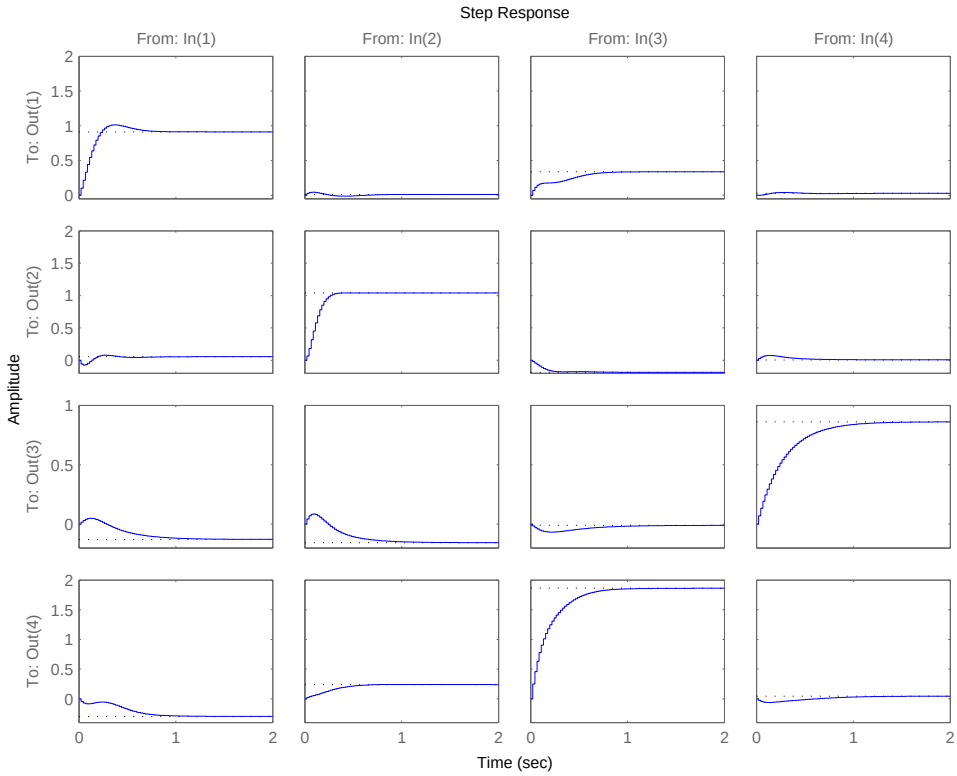


Figure 24: Multi-input step response for the model generated via identification.

output connections are as follows: $u_1 \rightarrow y_1$, $u_2 \rightarrow y_2$, $u_3 \rightarrow y_4$, $u_4 \rightarrow y_3$. In addition, we can see some of the couplings between the other input-output pairs, although these are relatively small for the most part, however they do highlight the coupled nature of the onboard electronics and rotor dynamics.

Incorporating the Model

After obtaining a suitable model, we are ready to perform simulation and use the SID results in our Kalman Filter design. Because the model is a discrete-time model, we use a zero-order hold to hold the output between time steps. The input is sampled at

the same rate as the controller, ie. every 22 ms. A block diagram of the model P_{Im} as used in simulation is shown in Figure 25. Note that we must use our scaling functions

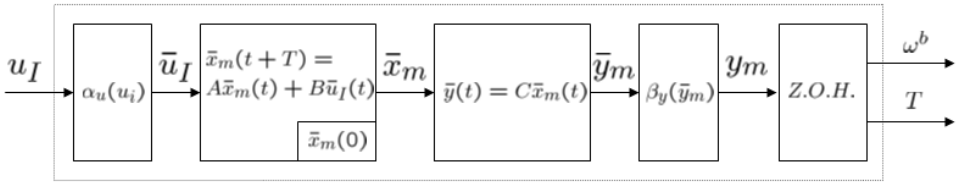


Figure 25: Block diagram of the model used in simulation.

to get appropriate data ranges on the input and output. The output of the inner system $[\Theta^T \ T]^T$ becomes the input to the outer system. We obtain the inner system input by integrating the equation $\dot{\Theta} = \Psi(\Theta)\omega^b$ to obtain Θ and combining it with T at each time step.

Extended Kalman Filter

The controller we will design requires online computation of the first and second derivatives of position to obtain the thrust. One drawback to this approach is that taking the derivatives of a noisy signal tends to magnify the noise. Passive filtering online has the effect of introducing phase lag and thereby changing the nature of the controller. One way to address this issue is with the use of state estimators that incorporate the input, information about the noise, and the dynamics of the system in order to generate a real-time filtered estimate of the states. In addition, using a state estimator provides us with states that would otherwise be unobservable (ie. the inner dynamics of the helicopter), and are therefore available for state-feedback control. An Extended Kalman Filter (EKF) will be employed to provide state estimates for feedback control purposes. The EKF is a form of the Kalman Filter that has been "extended" to nonlinear dynamical systems, and is developed using a two-stage process of prediction and correction [24], with the assumption that the measurement noise is Gaussian. The EKF uses information about the model in addition to current input values to compute the state estimates. A block diagram of the EKF and the system model is shown in Figure

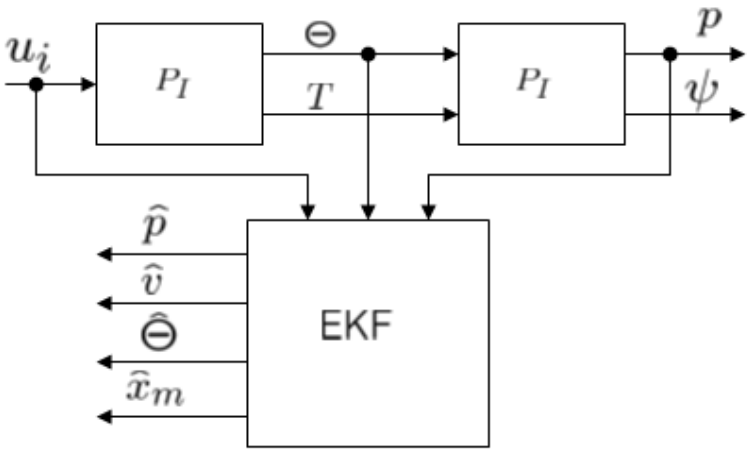


Figure 26: Block diagram of the EKF and system models.

26. We now derive the equation for generating the state estimates. The system dynamics for a nonlinear, time-invariant, continuous-time system can be described by the following state and measurement equations:

$$\Sigma : \begin{cases} \dot{x}(t) &= f[x(t), u(t)] + w(t) \\ z(t) &= h[x(t)] + v(t) \end{cases} \tag{49}$$

where $x \in \mathbb{R}^n$ is the state, $u \in \mathbb{R}^l$ is the control input, $w \in \mathbb{R}^p$ is process noise and $z \in \mathbb{R}^m$ and $v \in \mathbb{R}^m$ are the measurement and measurement noise, respectively. Given a nominal input $u^*(t)$, we can compute the nominal state $x^*(t)$, and nominal output $z^*(t)$ which satisfy the following equations:

$$\dot{x}^*(t) = f[x^*(t), u^*(t)] \tag{50}$$

$$z^*(t) = h[x^*(t)] \tag{51}$$

Expanding $f[x(t), u(t)]$ and $h[x(t)]$ in a Taylor Series about the nominal values, $x^*(t)$ and $u^*(t)$, yields

$$\begin{aligned} f[x(t), u(t)] &= f[x^*(t), u^*(t)] + F_x[x^*(t), u^*(t)](x(t) - x^*(t)) \\ &\quad + F_u[x^*(t), u^*(t)](u(t) - u^*(t)) + H.O.T. \end{aligned} \tag{52}$$

$$h[x(t)] = h[x^*(t)] + H_x[x^*(t)](x(t) - x^*(t)) + H.O.T. \tag{53}$$

where $F_x \in \mathbb{R}^{n\times n}$, $F_u \in \mathbb{R}^{n\times l}$ and $H_x \in \mathbb{R}^{m\times n}$ are Jacobian matrices given by

$$F_x[x^*(t), u^*(t)] = \left(\begin{array}{ccc} \partial f_1/\partial x_1 & \cdots & \partial f_1/\partial x_n \\ \vdots & \ddots & \vdots \\ \partial f_n/\partial x_1 & \cdots & \partial f_n/\partial x_n \end{array} \right) \Bigg|_{x(t)=x^*(t), u(t)=u^*(t)} \tag{54}$$

$$F_u[x^*(t), u^*(t)] = \left(\begin{array}{ccc} \partial f_1/\partial u_1 & \cdots & \partial f_1/\partial u_l \\ \vdots & \ddots & \vdots \\ \partial f_n/\partial u_1 & \cdots & \partial f_n/\partial u_l \end{array} \right) \Bigg|_{x(t)=x^*(t), u(t)=u^*(t)} \tag{55}$$

$$H_x[x^*(t)] = \left(\begin{array}{ccc} \partial h_1/\partial h_1 & \cdots & \partial h_1/\partial x_n \\ \vdots & \ddots & \vdots \\ \partial h_m/\partial x_1 & \cdots & \partial h_m/\partial x_n \end{array} \right) \Bigg|_{x(t)=x^*(t)} \tag{56}$$

Prediction Equation

Our first step in the filter design is to derive the prediction equation. Note we have neglected the noise term as no prediction about the noise can be made. Characteristics about the noise will be used later in the correction equations. Neglecting the "higher-order terms" we can write equation (52) as

$$\begin{aligned} f[x^*(t), u^*(t)] &= F_x[x^*(t), u^*(t)]x^*(t) + F_u[x^*(t), u^*(t)]u^*(t) \\ &\quad +\{f[x(t), u(t)] - F_x[x^*(t), u^*(t)]x(t) \\ &\quad -F_u[x^*(t), u^*(t)]u(t)\} \end{aligned} \tag{57}$$

If $x^*(t), u^*(t)$ is sufficiently close to $x(t), u(t)$ then we can write (57) as

$$f[x^*(t), u^*(t)] \simeq F_x[x^*(t), u^*(t)]x^*(t) + F_u[x^*(t), u^*(t)]u^*(t) \tag{58}$$

The Kalman Filter provides estimates of the state at time t using knowledge about the system dynamics and the current measurement. We will use the form $\hat{x}(t|t_k)$ to denote the estimate of the state at time t based on information up until time $t = t_k$, and will use the shorthand notation $\hat{x}(k+1|k)$ to represent $\hat{x}(t|t_k)$. Define

$$\delta\hat{x}(t|t_k) \quad = \quad \hat{x}(t|t_k) - x^*(t), \quad for\ t \in [t_k, t_{k+1}]. \tag{59}$$

By linearizing the state equation about $\hat{x}(k|k)$ at each time step t_k, then it can be shown that

$$x^*(t) = \hat{x}(t|t_k), \quad \forall t \in [t_k, t_{k+1}]. \tag{60}$$

Therefore $\delta\hat{x}(t|t_k) = 0$, and by substituting (60) into (50) it can be shown that

$$\hat{x}(k+1|k) = \hat{x}(k|k) + \int_{t_k}^{t_{k+1}} f[\hat{x}(t|t_k), u^*(t)]dt. \tag{61}$$

This is called the *EKF prediction equation.* Note we have used the fact that $x^*(t_{k+1}) = \hat{x}(k+1|k)$.

Now, since we evaluate the Jacobian matrices associated with the linearization at $x(t) = x^*(t_k), u(t) = u^*(t_k)$, and since $u^*(t)$ is fixed over the time interval $[t_k, t_{k+1}]$ then for $t \in [t_k, t_{k+1}]$, $F_x[x^*(t), u^*(t)]$ and $F_u[x^*(t), u^*(t)]$ are constant and can be expressed as

$$F_x[x^*(t), u^*(t)] \quad = \quad F_x[\hat{x}(k|k), u^*(t_k)] \tag{62}$$

$$F_u[x^*(t), u^*(t)] \quad = \quad F_u[\hat{x}(k|k), u^*(t_k)] \tag{63}$$

which we will write as F_x and F_u. In addition, $H_x[x^*(t), u^*(t)]$ will be evaluated at the predicted value $\hat{x}(k+1|k)$ and is therefore also constant over the time interval. It can therefore be expressed as

$$H_x[x^*(t)] \quad = \quad H_x[\hat{x}(k+1|k))] \tag{64}$$

which we will write as H_x for short. Then, we can write the approximated linearized equation as

$$f[\hat{x}(k|k), u^*(t_k)] \simeq F_x\hat{x}(k|k) + F_u u^*(t_k) \tag{65}$$

A solution to this equation is

$$\hat{x}(k+1|k) \quad = \Phi(t_{k+1}, t_k) \quad \hat{x}(k|k) + \int_{t_k}^{t_{k+1}} \Phi(t_{k+1}, \tau) F_u u^*(t_k) d\tau \tag{66}$$

where

$$\Phi(t_{k+1}, t_k) = e^{F_x T_s} \tag{67}$$

and $T_s = t_{k+1} - t_k$. We will use $\Phi(k+1, k)$ to represent $\Phi(t_{k+1}, t_k)$. The matrix exponential $e^{F_x T_s}$ can be written as the Taylor series

$$e^{F_x T_s} = I + F_x T_s + F_x^2 \frac{T_s^2}{2} + F_x^3 \frac{T_s^3}{3!} + \cdots \tag{68}$$

Then for sufficiently small values of T_s, we have the following term that will be used in our prediction and correction equations:

$$e^{F_x T_s} \simeq I + F_x T_s \tag{69}$$

thus we will use

$$\Phi(k+1, k) = I + F_x T_s \tag{70}$$

Applying this result to the integration term in (66) and using the fact that $u(t) = u^*(t)$ is constant for $t \in [t_k, t_{k+1}]$, by integrating term by term it can be shown that

$$\begin{aligned}\int_{t_k}^{t_{k+1}} \Phi(t_{k+1}, \tau) F_u d\tau &= \int_{t_k}^{t_{k+1}} I + F_x(t_{k+1} - \tau) F_u d\tau \\ &\simeq F_u T_s + F_x F_u t_{k+1} T_s - F_x F_u \int_{t_k}^{t_{k+1}} \tau d\tau \\ &\simeq F_u T_s + F_x F_u \frac{T_s^2}{2} \simeq F_u T_s. \end{aligned} \tag{71}$$

We will denote $F_u T_s$ as $\Psi(k+1, k)$. Applying this result yields

$$\hat{x}(k+1|k) = \Phi(k+1, k)\hat{x}(k|k) + \Psi(k+1, k)u^*(t_k) \tag{72}$$

$$= [I + F_x T_s]\hat{x}(k|k) + F_u T_s u^*(t_k) \tag{73}$$

which is our closed-form prediction equation.

Correction Equation

We next present the *EKF correction equation*, which is given as

$$\begin{aligned}\hat{x}(k+1|k+1) &= \hat{x}(k+1|k) \\ &\quad + K(k+1)\{z(k+1) - \hat{z}(k+1|k)\} \end{aligned} \tag{74}$$

where

$$\hat{z}(k+1|k) = h[\hat{x}(k+1|k)]. \tag{75}$$

Note that we must also compute the EKF gain matrix, $K(k+1)$. This is an $n \times m$ Kalman gain matrix specified by the set of relations

$$\begin{aligned} P(k+1|k) &= \Phi(k+1,k)P(k|k)\Phi^T(k+1,k) \\ &\quad +Q(k+1,k) \qquad (76) \\ K(k+1) &= P(k+1|k)H_x^T(k+1)[H_x(k+1) \\ &\quad P(k+1|k)H_x^T(k+1)+R(k+1)]^{-1} \qquad (77) \\ P(k+1|k+1) &= [I-K(k+1)H_x(k+1)]P(k+1|k) \qquad (78) \end{aligned}$$

where $I \in \mathbb{R}^{n\times n}$ is an Identity matrix, $P(k+1|k+1) \in \mathbb{R}^{n\times n}$ is the error covariance matrix, and $Q(k+1,k)$ is the covariance of $w(k)$ which is defined as $Q(k+1,k) \triangleq E\{w(k)w^T(k)\}$, where $E\{\cdot\}$ is the linear expectation operator. Also note that we use the predicted estimate, $\hat{x}(k+1|k)$ in equations (76)-(78). In implementing the EKF equations, $Q(k+1,k)$ and $R(k+1)$ can be initialized by constant diagonal matrices $Q \in \mathbb{R}^{p\times p}$ and $R \in \mathbb{R}^{m\times m}$. Also note that we have used $\Psi(k+1,k)$ and $R(k+1)$ to represent parameters in our EKF equations. We also use the symbols Ψ and R in the system dynamic equations to describe the mapping and rotation matrix. It should be clear from the context whether we are referring to the EKF parameters or the system dynamics values. We use the closed-form solutions in our implementation of the EKF predictor-corrector equations. The steps are provided in Appendix B.

Application

In order to implement the EKF, we need to combine the nonlinear continuous time model and the linear discrete time inner model to obtain a complete estimate of the system states which we present in this section. Recall from Chapter II that we have the following system composed of inner and outer subsystems as shown in Figure 27.

The inner system state vector and output vector are $x_I = [\Theta^T \ \omega^{bT} \ x_m^T]^T$ and $y_I = [\Theta^T \ T]^T$, respectively. The linear discrete-time model of the inner system P_{Im}, with state x_m, was obtained by performing SID. The state and output equations for the inner model

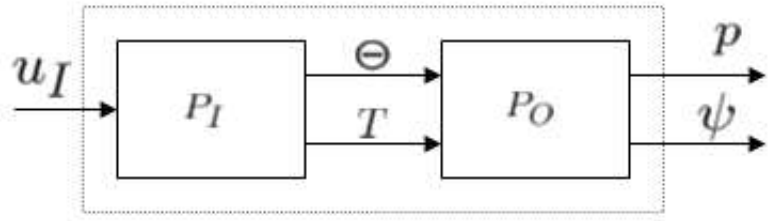


Figure 27: Inner and outer system.

are given by:

$$x_m[k+1] = Ax_I[k] + B\bar{u}_I[k] \tag{79}$$

$$\bar{y}_m[k+1] = Cx_I[k+1] \tag{80}$$

where $x_m \in \mathbb{R}^{n_m}$ is the inner state , $\bar{u}_I \in [-1,1]^4$ is the control input, $\bar{y}_m \in [-1,1]^4$ is the output equation, $A \in \mathbb{R}^{n_m \times n_m}$, $B \in \mathbb{R}^{n_m \times 4}$ and $C \in \mathbb{R}^{4 \times n_m}$. In the development of the inner plant model, the input-output data was normalized before performing System Identification. When using the model, the input and output values are converted back to the operating ranges using equations (41), (43) and (45). The output of the inner system y_I can be obtained from the output of the discrete-time model y_m, the conversion functions and the dynamic equations. Thus we can write the inner model output as

$$y_m = \begin{bmatrix} \omega_x^b \\ \omega_x^b \\ \omega_x^b \\ T \end{bmatrix} = \beta_y(Cx_m) \tag{81}$$

The output of the inner system y_I is used as the input to the outer system, i.e. $u_O = y_I$. The nonlinear continuous-time outer state, output and input vectors are given by

$$x_O(t) = \begin{bmatrix} p(t) \\ v(t) \end{bmatrix} \tag{82}$$

$$y_O(t) = \begin{bmatrix} p(t) \\ \psi(t) \end{bmatrix} \tag{83}$$

$$u_O(t) = \begin{bmatrix} \Theta(t) \\ T(t) \end{bmatrix} \tag{84}$$

The dynamics are given by

$$\dot{x}_O(t) = \begin{bmatrix} \dot{p}(t) \\ \dot{v}(t) \end{bmatrix} = \begin{bmatrix} v(t) \\ \frac{1}{m}(e_3 mg - R(\Theta(t))e_3 T(t)) \end{bmatrix} \tag{85}$$

which we write as

$$\dot{x}_O(t) = f_O[x_I(t), u_O(t)] \tag{86}$$

$$y_O(t) = h_O[x_I(t)] \tag{87}$$

We wish to combine the inner and outer system models to form the augmented state $x \in \mathbb{R}^6 \times \mathbb{S}^3 \times \mathbb{R}^{n_m}$ where

$$x = \begin{bmatrix} x_O \\ x_I \end{bmatrix} = \begin{bmatrix} p \\ v \\ \Theta \\ x_m \end{bmatrix}. \tag{88}$$

In addition, we will have the measurement vector used by the estimator given as

$$z = \begin{bmatrix} p \\ \Theta \end{bmatrix}. \tag{89}$$

where $z \in \mathbb{R}^3 \times \mathbb{S}^3$ will be sampled at fixed intervals. For the prediction equation we compute $\Phi(k+1,k)$ and $\Psi(k+1,k)$ for the augmented state based on the current state estimate $\hat{x}(k|k)$. Because we have a continuous-time model for the outer system, we must be careful when combining it with the inner discrete-time model. We will use the techniques presented in the previous section along with the prediction equations for the inner system to derive the necessary parameters for the EKF. For the outer system prediction equations we have

$$\hat{x}_O(k+1|k) = \Phi_O(k+1,k)\hat{x}_O(k|k) + \Psi_O(k+1,k)u_O(k). \tag{90}$$

Recall that we use

$$\Phi(k+1,k) = I + F_x T_s \tag{91}$$

$$\Psi(k+1,k) = F_u T_s \tag{92}$$

The prediction equation for a discrete-time linear system is given by the system dynamics. So for the inner system we can write

$$\hat{x}_i(k+1|k) = A\hat{x}_I(k|k) + B\bar{u}_I(k). \tag{93}$$

In addition, we have

$$y_m(k+1|k) = Cx_I(k+1|k), \tag{94}$$

$$y_I(k+1|k) = y_{scale} * y_m(k+1|k). \tag{95}$$

We will combine the inner and outer system dynamic equations to come up with augmented matrices. these matrices to get the combined prediction and correction equations for the EKF. Since $u_O = y_I = \beta_y(Cx_I)$, we obtain the augmented $\Phi(k+1,k)$ and

$\Psi(k+1,k)$ matrices given as

$$\Phi(k+1,k) = \begin{bmatrix} I + F_{x_O} T_s & F_{u_O} T_s y_{scale} C \\ 0 & A \end{bmatrix} \tag{96}$$

and

$$\Psi(k+1,k) = \begin{bmatrix} 0 \\ B \end{bmatrix} \tag{97}$$

where $T_s = .022s$ is the time interval $t_{k+1} - t_k$. In addition, we obtain the augmented matrix associated with the measurement as

$$H_x = \begin{bmatrix} \frac{\partial y_O}{\partial x} & \frac{\partial y_I}{\partial x} \end{bmatrix} = \begin{bmatrix} H_{x_O} & 0 \end{bmatrix} \tag{98}$$

where $H_{x_O} \in \mathbb{R}^{6\times(9+n_m)}$. We now have all the equations necessary for performing estimation using the EKF for state feedback.

CHAPTER III

REAL-TIME CONTROLLER DESIGN

Two-stage Controller

The controller consists of two stages. In each stage, a nominal system and a perturbation are considered, and the controller is designed so that the perturbed system is stable. A block diagram of the two-stage controller and helicopter model is shown in Figure 28.

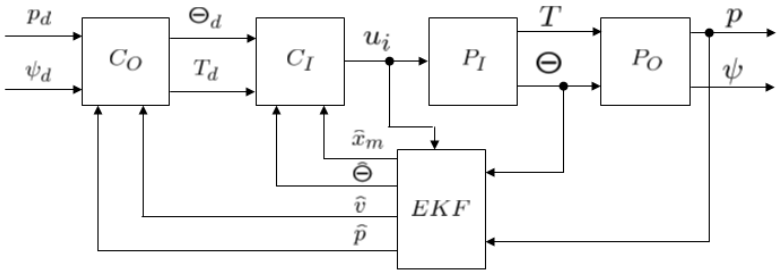


Figure 28: System with feedback control.

Inner System State Feedback Controller

We first focus on the inner controller C_I as shown in Figure 28. The output of the inner controller directly drives the helicopter inner model P_I, whose state vector is $x_I = [\Theta^T \ \omega^{bT} \ x_m^T]^T$. We will design a state feedback controller, utilizing the available state estimates from the EKF to drive $\omega^b \to \omega_d^b$ and $T \to T_d$. We will derive w_d and T_d later in the chapter. A major advantage in using the SID model and the EKF is that the model generated is guaranteed to be observable and controllable. We can therefore use the inversion theory for discrete-time systems to generate the control signal. Consider the linear discrete-time inner system model given by

$$x[k+1] = Ax[k] + Bu[k], \tag{99}$$

$$y[k+1] = Cx[k+1]. \tag{100}$$

We construct the following vector of output signals:

$$Y_n = [y(0)^T \; y(1)^T \; \cdots y(n-1)^T]^T, \quad n = 6. \tag{101}$$

which we can then write as

$$Y_n = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix} x(0) + \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ CB & 0 & 0 & \cdots & 0 \\ CAB & CB & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ CA^{n-1}B & CA^{n-2}B & \cdots & \cdots & 0 \end{bmatrix} \begin{bmatrix} u(0) \\ u(1) \\ u(2) \\ \vdots \\ u(n) \end{bmatrix} \tag{102}$$

We consider a nominal output vector Y_n^*, state vector x^* and input vector u^* that satisfy the dynamic equations. At steady state, we can write $Y_n^* \simeq Y_n$, $x^* \simeq x(0)$ and $u^* \simeq u(0)$. Therefore we get

$$Y_n^* = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix} x^* + \begin{bmatrix} 0 \\ CB \\ CAB \\ \vdots \\ CA^{n-1}B + CA^{n-2}B + \ldots \end{bmatrix} u^* \tag{103}$$

$$\tag{104}$$

$$= \underbrace{\begin{bmatrix} C & \vdots & 0 \\ CA & \vdots & CB \\ CA^2 & \vdots & CAB + CB \\ \vdots & \vdots & \vdots \\ CA^{n-1} & \vdots & CA^{n-1}B + CA^{n-2}B + \ldots \end{bmatrix}}_{\Lambda} \begin{bmatrix} x^* \\ u^* \end{bmatrix}. \tag{105}$$

Now we have

$$\begin{bmatrix} x^* \\ u^* \end{bmatrix} = \Lambda^{-1} Y_n^*, \tag{106}$$

where we can take the pseudoinverse of Λ given by $\Lambda^{-1} = (\Lambda^T \Lambda)^{-1} \Lambda^T$, because the model generated from the SID process is guaranteed to be observable. We populate the nominal output with the desired values $y_d = [\omega_d^{bT} T_d]$, therefore we may write

$$\begin{bmatrix} x^* \\ u^* \end{bmatrix} = \underbrace{(\Lambda^T \Lambda)^{-1} \Lambda^T \begin{bmatrix} I_4 \\ \vdots \\ I_4 \end{bmatrix}}_{\Lambda^*} y_d, \tag{107}$$

where I_4 is a 4×4 identity matrix. Using the model parameters obtained by performing SID we can compute Λ^{-1} and Λ^* which we obtain as

$$\Lambda^* = \begin{bmatrix} -0.0131 & -0.0135 & 0.8786 & -0.0036 \\ 0.8009 & -1.1124 & -0.0663 & 0.0053 \\ 1.3937 & 0.7760 & 0.0980 & -0.0042 \\ -0.0409 & -0.0036 & -0.0534 & 0.4247 \\ 0.1106 & -0.3796 & 0.1860 & -0.0569 \\ 0.5595 & 0.7432 & -0.1210 & -0.0931 \\ 1.0310 & 0.0274 & -0.0246 & -0.1850 \\ -0.0272 & 0.9360 & -0.0113 & 0.0988 \\ 0.1653 & -0.1199 & -0.0301 & 0.4946 \\ 0.1506 & 0.1721 & 1.1542 & -0.0041 \end{bmatrix} \tag{108}$$

We next consider designing a controller for the system

$$\dot{x}^* = Ax^* + Bu^*. \tag{109}$$

Letting $x_e[k] = x[k] - x^*$ and using the control law given by

$$u[k] = u^* - K(x[k] - x^*), \tag{110}$$

it can be shown that

$$\begin{aligned} x_e[k+1] &= (A - BK)x_e[k] && (111) \\ &= Ax_e[k] - Bu_e[k], && (112) \end{aligned}$$

where

$$u_e[k] = -Kx_e[k] \tag{113}$$

then by appropriate selection of K, we can drive the error dynamics $x_e[k+1]$ to zero. We can always use this method since the SID process is guaranteed to generate a model that is controllable. We use a deadbeat controller and choose the poles of $(A - BK)$ to lie on the origin, however we also could have used linear quadratic regulator (LQR) to drive the system with the cost function given by

$$J = \int_0^\infty (x^T Q x + u^T R u) d\tau, \tag{114}$$

for which tools exist to find the minimum realization. Using one of these methods we can drive the output $[\omega^{bT} T]^T \rightarrow [\omega_d^{bT} T_d]^T$. Next we consider the outer system controller that will generate the desired trajectories for the inner system controller.

Outer System Nonlinear Controller

We will design two components in the outer system controller the first will generate the reference trajectories for the inner controller and the second will drive the position p to p_d. Consider the inner dynamics, Σ_I. The output of the system is $y_I = [\Theta^T \; T]^T$ and we specify the desired output as $y_{id} = [\Theta_d^T \; T_d]^T$. Having assumed that the inner controller

will drive $(\omega^b, T) \rightarrow (\omega_d^b, T_d)$ as $t \rightarrow \infty$, we can drive (Θ) to (Θ_d). Using equation (18), namely $\dot{\Theta} = \Psi(\Theta)\omega^b$, we replace ω^b with ω_d^b in order to derive the controller. Since $\Psi^{-1}(\Theta)$ exists except at the singularities, we can apply feedback linearization for control design. Next define

$$\dot{\Theta} = v_\Theta \tag{115}$$

$$\omega_d^b = \Psi^{-1}(\Theta)v_\Theta \tag{116}$$

where $v_\Theta \in \mathbb{R}^3$ is the Euler angle velocity vector. Choosing $v_\Theta = \dot{\Theta}_d - K_\Theta(\Theta - \Theta_d)$ where $K_\Theta \in \mathbb{R}^{3\times3}$, the error dynamics of the Euler angles can be described by

$$\dot{z}_\Theta = A_\Theta z_\Theta \tag{117}$$

where $z_\Theta = \Theta - \Theta_d$ and $A_\Theta = -K_\Theta \in \mathbb{R}^{3\times3}$. By choosing the appropriate matrix K_Θ, we can ensure that the equilibrium point $z_\Theta = 0$ is exponentially stable. Hence, we see that $\Theta \rightarrow \Theta_d$ as $t \rightarrow \infty$. However, the statement is true only if $\omega^b = \omega_d^b$. We will show that $\Theta \rightarrow \Theta_d$ as $t \rightarrow \infty$ only if $\omega^b \rightarrow \omega_d^b$ as $t \rightarrow \infty$.

Define the perturbed system as

$$\dot{z} = f(z,t) + \rho. \tag{118}$$

We will show that this system is stable. First we assume that the *equilibrium point* $z = 0$ of the nominal system

$$\dot{z} = f(z,t) \tag{119}$$

is exponentially stable and $\rho \rightarrow 0$ as $t \rightarrow \infty$. Based on the *Lyapunov Theorem for Exponential Stability and its Converse* [28], the system is *locally* exponentially stable if

and only if there exists a *Lyapunov function* $v(z,t)$ such that *for all* $z \in B_h$, and $t \geq 0$

$$\alpha_1\|z\|^2 \leq v(z,t) \leq \alpha_2\|z\|^2 \tag{120}$$

$$\frac{\partial v(z,t)}{\partial t} + \frac{\partial v(z,t)}{\partial z} f(z,t) \leq -\alpha_3\|z\|^2 \tag{121}$$

$$\left\| \frac{\partial v(z,t)}{\partial z} \right\| \leq \alpha_4\|z\| \tag{122}$$

for some constants $\alpha_1, \alpha_2, \alpha_3, \alpha_4, h > 0$, where B_h is a ball of radius h centered at $z = 0$.

Theorem III.0.1. *Let $z = 0$ be an exponentially stable equilibrium point of the nominal system (119). Let $V(z,t)$ be a Lyapunov function of the nominal system that satisfies (120),(121) and (122). Suppose that the perturbation term ρ of the equation (118) satisfies $\rho \to 0$ as $t \to \infty$, then the solution of the perturbed system $z \to 0$ as $t \to \infty$.*

Proof. The derivative of $V(z,t)$ along the trajectories of the perturbed system (118) satisfies the inequality

$$\dot{V}(z,t) \leq -\alpha_3\|z\|_2^2 + \alpha_4\rho\|z\|_2$$

Using (120), we can show that $\dot{V}(z,t)$ is restricted to

$$\dot{V}(z,t) \leq -\frac{\alpha_3}{\alpha_2}V(z,t) + \frac{\alpha_4}{\sqrt{\alpha_1}}\rho\sqrt{V(z,t)}$$

Letting $W(t) = \sqrt{V(z,t)}$ and rewriting the inequality yields

$$\dot{W}(t) \leq -\sigma W(t) + \beta\rho$$

where $\sigma = \frac{\alpha_3}{2\alpha_2}$ and $\beta = \frac{\alpha_4}{2\sqrt{\alpha_1}}$. Thus,

$$W(t) \leq e^{-\sigma(t)}W(0) + \int_0^t \beta e^{-\sigma(t-\tau)}\rho(\tau)d\tau$$

Since $\rho \to 0$ as $t \to \infty$, we have

$$W(t) \to 0 \text{ as } t \to \infty.$$

Hence,

$$z \to 0 \text{ as } t \to \infty$$

since $\|z\|_2 \leq \frac{1}{\sqrt{\alpha_1}} W(t)$ □

Therefore, provided the nominal system is exponentially stable, the system is extremely robust in terms of exponential stability to any vanishing perturbation.

Lemma III.0.2. *Consider the dynamics of the Euler angles that satisfy (18). Given the control design* $\omega_d^b = \Psi^{-1}(\Theta)(\dot{\Theta}_d - K_\Theta(\Theta - \Theta_d))$*, if* $\omega^b \to \omega_d^b$ *as* $t \to \infty$ *then* $\Theta \to \Theta_d$ *as* $t \to \infty$*.*

Proof. Rewrite (18) as

$$\dot{\Theta} = \Psi(\Theta)\omega_d^b + \Psi(\Theta)(\omega^b - \omega_d^b),$$

then substitute ω_d^b into the equation. Thus, we have

$$\dot{z}_\Theta = A_\Theta z_\Theta + \rho_\Theta \tag{123}$$

where the perturbation vector $\rho_\Theta = \Psi(\Theta)(\omega^b - \omega_d^b)$. Hence, if $\omega^b \to \omega_d^b$ as $t \to \infty$, then $\rho_\Theta \to 0$ as $t \to \infty$. Since the nominal system $\dot{z}_\Theta = A_\Theta z_\Theta$ is exponentially stable, we can apply Theorem III.0.1 to show that $z_\Theta \to 0$ as $t \to \infty$. Therefore, we have shown that $\Theta \to \Theta_d$ as $t \to \infty$. □

Consider the outer system whose dynamics are given by (21) and whose input is the output of the inner system, $y_I = [\Theta^T \; T]^T$. The outer system output is the position vector, $y_O = [p^T \; \psi]^T$. Since we know that $(\omega^b, T) \to (\omega_d^b, T_d)$ as $t \to \infty$ is guaranteed by the inner system, we are interested in designing a controller to drive p to p_d and Θ to Θ_d.

We will base the outer controller design on the concept of differential flatness, which was originally defined by Martin Fliess. A system is differentially flat if we can find a set of outputs (equal in number to the inputs) such that all states and inputs can be determined from these outputs without integration as shown in [22]. Given some system with states $x \in \mathbb{R}^n$ and inputs $u \in \mathbb{R}^m$ then the system is flat if there can be found

outputs $y \in \mathbb{R}^m$ of the form

$$y = h(x, u, \dot{u}, \ldots, u^{(r)}), \tag{124}$$

such that

$$x = \varphi(y, \dot{y}, \ldots, y^{(q)}), \tag{125}$$

$$u = \alpha(y, \dot{y}, \ldots, y^{(q)}). \tag{126}$$

Many classes of systems commonly used in nonlinear control theory are flat, including any system that can be linearized by change of coordinates, static feedback transformations, or dynamic feedback transformations. Flatness indicates that the nonlinear structure of the system can be exploited for designing control algorithms for motion planning, trajectory generation, and stabilization. As shown in [17], the outer system Σ_O is *differentially flat* with respect to the output $(p_x,\ p_y,\ p_z,\ \psi)$. Therefore, by definition, the state and input trajectories can be written as algebraic functions of the output trajectories and their derivatives. We next show how this is done. As mentioned before we have the outputs for the outer system from (20). We rewrite the dynamic equation from (21) as

$$\underbrace{\begin{bmatrix} \ddot{p}_x \\ \ddot{p}_y \\ \ddot{p}_z \end{bmatrix}}_{\ddot{p}} = e^{\hat{z}\psi} e^{\hat{y}\theta} e^{\hat{x}\phi} \begin{bmatrix} 0 \\ 0 \\ -T/m \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \tag{127}$$

and performing some algebraic manipulations gives us

$$e^{-\hat{z}\psi} \begin{bmatrix} \ddot{p}_x \\ \ddot{p}_y \\ \ddot{p}_z - g \end{bmatrix} = e^{\hat{y}\theta} e^{\hat{x}\phi} \begin{bmatrix} 0 \\ 0 \\ -T/m \end{bmatrix}. \tag{128}$$

Computing the 2-norm of both sides yields

$$T = m\sqrt{(\ddot{p}_x)^2 + (\ddot{p}_y)^2 + (\ddot{p}_z - g)^2}. \tag{129}$$

Using (128) again we can write

$$e^{-\hat{z}\psi}\begin{bmatrix} \ddot{p}_x \\ \ddot{p}_y \\ \ddot{p}_z - g \end{bmatrix} - \frac{m}{T} = e^{\hat{y}\theta}e^{\hat{x}\phi}\begin{bmatrix} s\theta c\phi \\ -s\phi \\ c\phi c\theta \end{bmatrix}. \tag{130}$$

It can then be shown that

$$\phi = \sin^{-1}\left(\frac{-\ddot{p}_x \sin\psi + \ddot{p}_y \cos\psi}{T/m}\right) \tag{131}$$

$$\theta = \text{atan2}\left(\frac{\ddot{p}_x \cos\psi + \ddot{p}_y \sin\psi}{-T\cos\phi/m}, \frac{\ddot{p}_z - g}{-T\cos\phi/m}\right) \tag{132}$$

$$\psi = \psi \tag{133}$$

where $\phi, \theta \neq \pm\pi/2$.

Therefore, there exists a smooth mapping from $(\ddot{p}, \psi)$ to (Θ, T):

$$\begin{aligned} \Pi: \quad \mathbb{R}^3 \times \mathbb{S} &\rightarrow \mathbb{S}^3 \times \mathbb{R} \\ (\ddot{p}, \psi) &\mapsto (\Theta, T) \end{aligned}$$

defined by the equations (129), (131), (132) and (133).

Next we replace (Θ, T) with (Θ_d, T_d) for deriving the controller. Given the mapping Π, instead of using (Θ_d, T_d) for design, we focus on $(\ddot{p}_d, \psi_d)$. Since $\ddot{p}_d$ describes the desired second derivative of the position vector, we can use the vector to specify the desired position dynamics. Define an error vector as $z_p = [p^T \ v^T]^T - [p_d^T \ 0]^T$. If we choose

$$\ddot{p}_d = -K_v\dot{p} - K_p(p - p_d), \tag{134}$$

where $K_p, K_v \in \mathbb{R}^{3\times 3}$, the error dynamics of the position can be described by

$$\dot{z}_p = A_p z_p \tag{135}$$

where

$$A_p = \begin{bmatrix} 0_{3\times 3} & I_{3\times 3} \\ -K_p & -K_v \end{bmatrix}$$

where $0_{3\times 3}$ is a 3×3 zero matrix and $I_{3\times 3}$ is a 3×3 Identity matrix. By appropriately choosing the matrices K_p, K_v, we can ensure that the equilibrium point $z_\Theta = 0$ is exponentially stable. Therefore, given $(\ddot{p}_d, \psi_d)$, we can derive the desired input by using

$$(\Theta_d, T_d) = \Pi(\ddot{p}_d, \psi_d) \tag{136}$$

If $(\Theta, T) = (\Theta_d, T_d)$ for $t \geq 0$, we have $p \to p_d$ as $t \to \infty$. Similarly, we can show that $p \to p_d$ as $t \to \infty$ only if $(\Theta, T) \to (\Theta_d, T_d)$ as $t \to \infty$.

Lemma III.0.3. *Consider the dynamics of the outer system given in (21). Given the control design $(\Theta_d, T_d) = \Pi(\ddot{p}_d, \psi_d)$ where $\ddot{p}_d = -K_v\dot{p} - K_p(p - p_d)$ from (134). If $(\Theta, T) \to (\Theta_d, T_d)$ as $t \to \infty$ then $p \to p_d$ as $t \to \infty$.*

Proof. Rewrite (21) as

$$\begin{bmatrix} \dot{p} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} v \\ \frac{1}{m}R(\Theta)e_3T_d + e_3g + \frac{1}{m}(R(\Theta)e_3T - R(\Theta)e_3T_d), \end{bmatrix}$$

then substitute $(\Theta_d, T_d) = \Pi(\ddot{p}_d, \psi_d)$ into the equation to give

$$\dot{z}_p = A_p z_p + \rho_p, \tag{137}$$

where the perturbation vector is given by

$$\rho_p = \begin{bmatrix} 0_3 \\ \frac{1}{m}(R(\Theta)e_3T - R(\Theta_d)e_3T_d) \end{bmatrix}.$$

Hence, if $(\Theta, T) \to (\Theta_d, T_d)$ as $t \to \infty$, then $\rho_p \to 0$ as $t \to \infty$. Since the nominal system $\dot{z}_p = A_\Theta z_p$ is exponentially stable, we can apply Theorem III.0.1 to show that $z_p \to 0$ as $t \to \infty$. Therefore, we have shown that $p \to p_d$ as $t \to \infty$. □

Parameter Tuning

Once we have designed the control law, tuning the parameters will consist of two parameter tuning stages done in simulation and then again in implementation. Because the model we obtain from system identification is not perfect and does not model the noise, there will be differences in the simulation and implementation. Using the model parameters (A, B, C, D) obtained from SID we design a deadbeat controller for the inner system and place the poles of $(A - BK)$ at the origin. This involves solving for K. We can then generate the nominal values for our controller from the inversion theory. Based on our model parameters, we get the following result for K:

$$K = \begin{bmatrix} -2.5587 & 7.0763 & 10.2035 & -20.8095 & -0.7268 & -1.9013 \\ -4.2786 & -8.1504 & -2.2246 & 15.0945 & 0.5334 & 5.3280 \\ 5.2256 & -1.9796 & -7.3453 & 22.3769 & 2.3690 & 1.6205 \\ 19.7891 & 5.7049 & -0.3287 & -6.6071 & -0.2105 & -4.4321 \end{bmatrix} \tag{138}$$

which gives us all the components for the inner controller. Next we tune the components of the outer system K_Θ, K_v and K_p. The results obtained in simulation are given next. The results for K_Θ are

$$K_\Theta = \begin{bmatrix} -8 & 0 & 0 \\ 0 & -8 & 0 \\ 0 & 0 & -2 \end{bmatrix}. \tag{139}$$

The outer system position control gains are set to

$$K_p = \begin{bmatrix} -5 & 0 & 0 \\ 0 & -5 & 0 \\ 0 & 0 & -3 \end{bmatrix} \tag{140}$$

and

$$K_v = \begin{bmatrix} -3 & 0 & 0 \\ 0 & -3 & 0 \\ 0 & 0 & -2 \end{bmatrix}. \tag{141}$$

The simulation results for applying step inputs to the desired position are shown in Figure 29. Note that the state estimate is plotted along with the measured and desired position

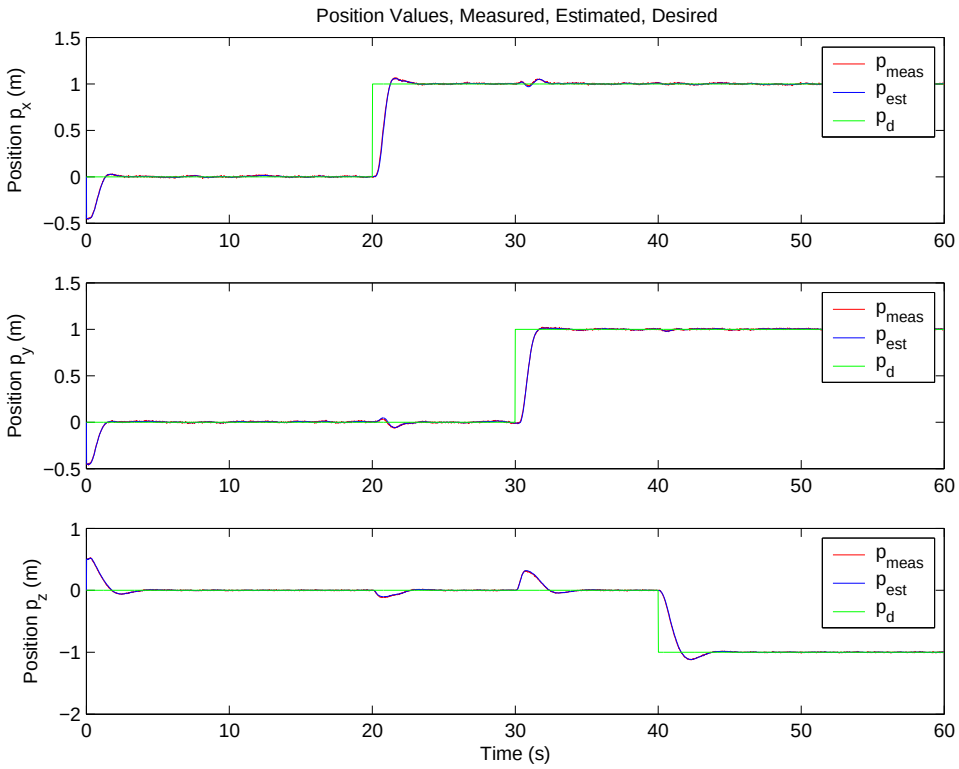


Figure 29: Simulation response to position step input.

values. We can also see the coupled nature of the system in Figure 29 by looking at the change in p_z given a transition in p_x or p_y. The simulation results for the Euler angles, velocity, thrust, body angular velocities and inputs are given in Figures 30 through 34. In the simulation, we add a Gaussian random noise signal with zero mean and variance equal to [1; 1; 1; .1; .1; .1] to the measurement vector to simulate the sensor noise on the measurement $[p^T \ \Theta^T]^T$.

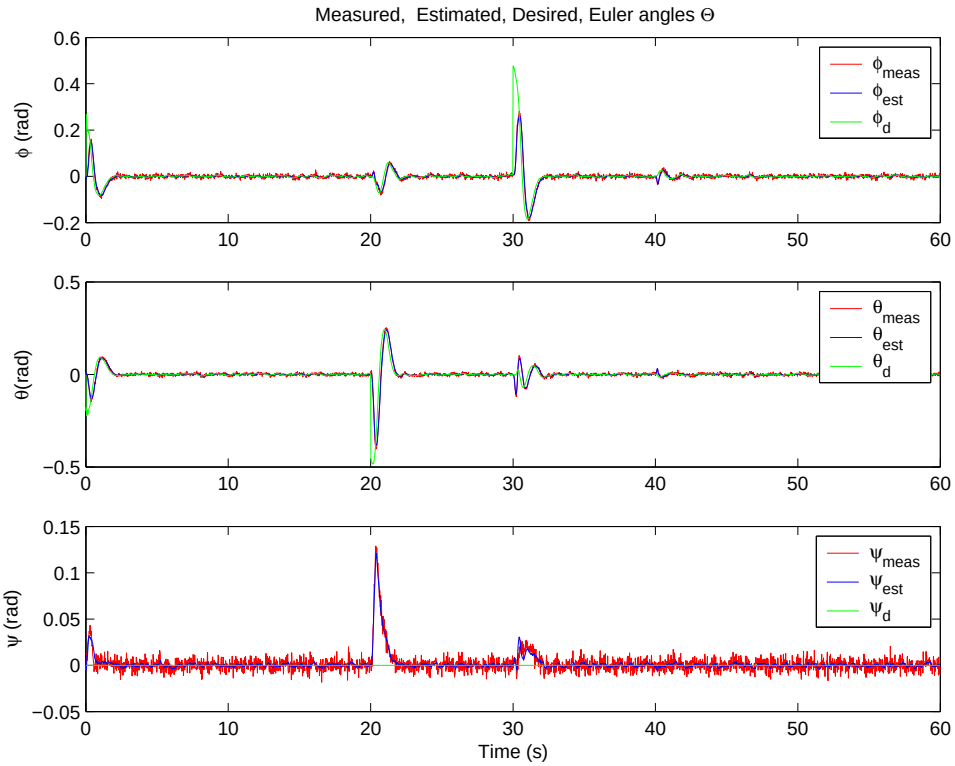


Figure 30: Plot of Euler angle values.

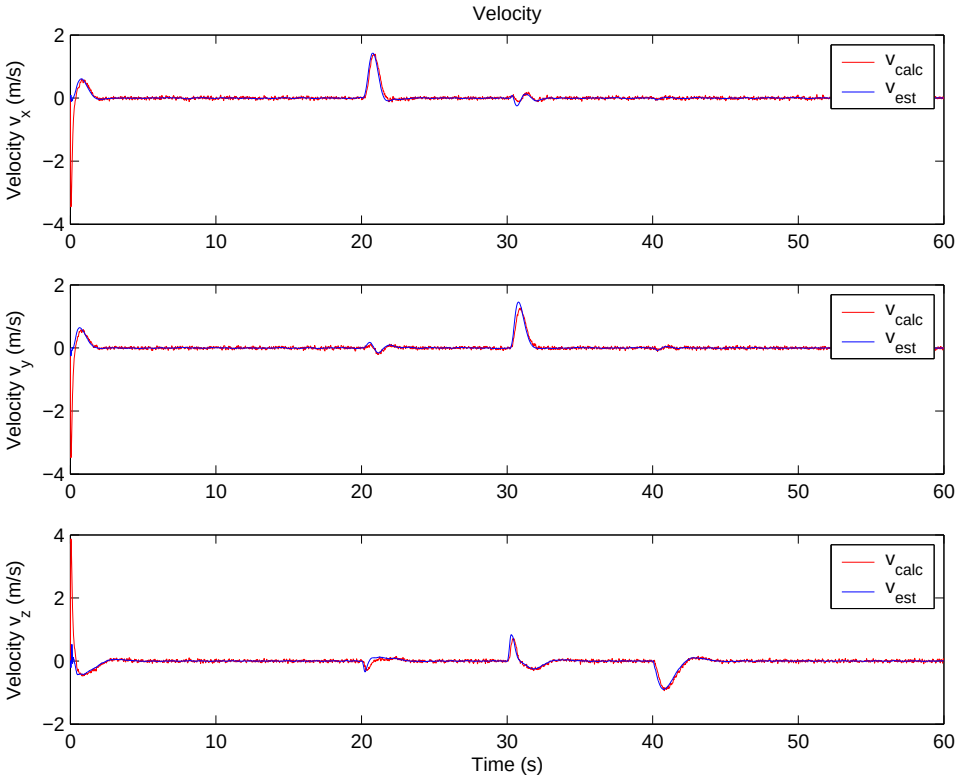


Figure 31: Plot of velocity values.

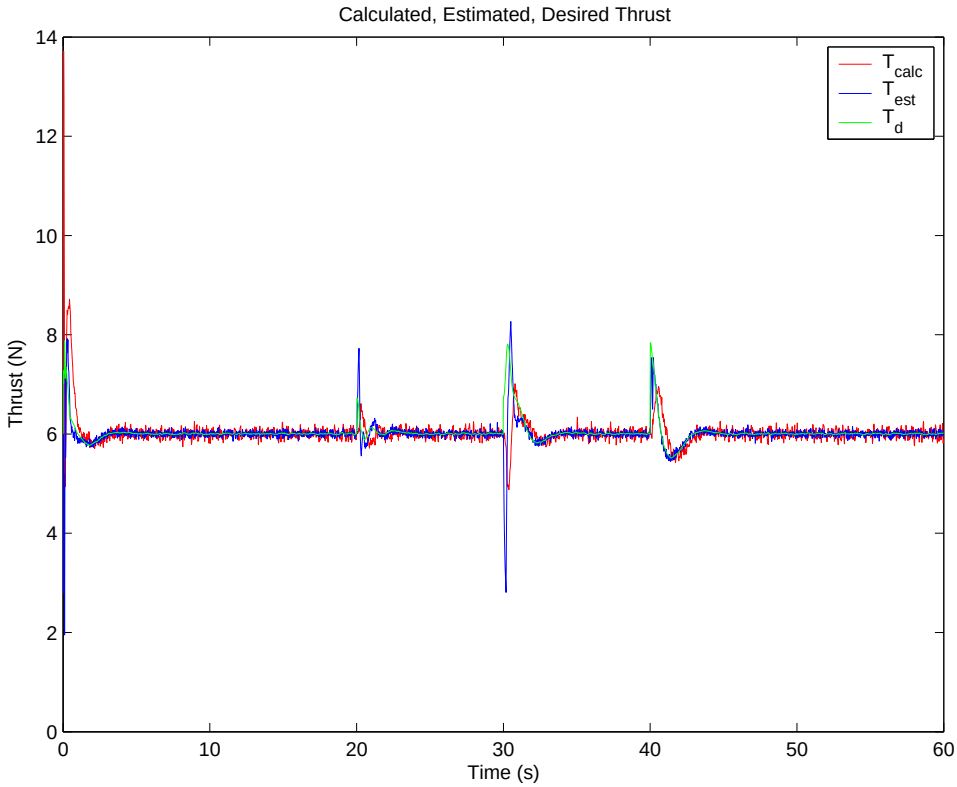


Figure 32: Plot of thrust values.

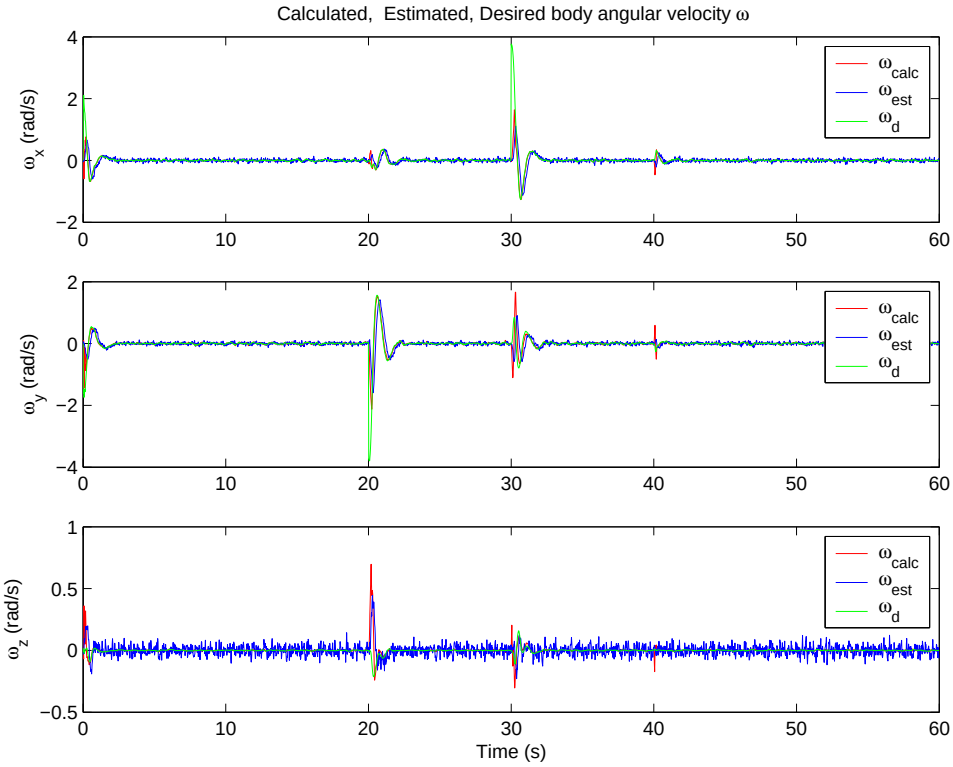


Figure 33: Plot of ω values.

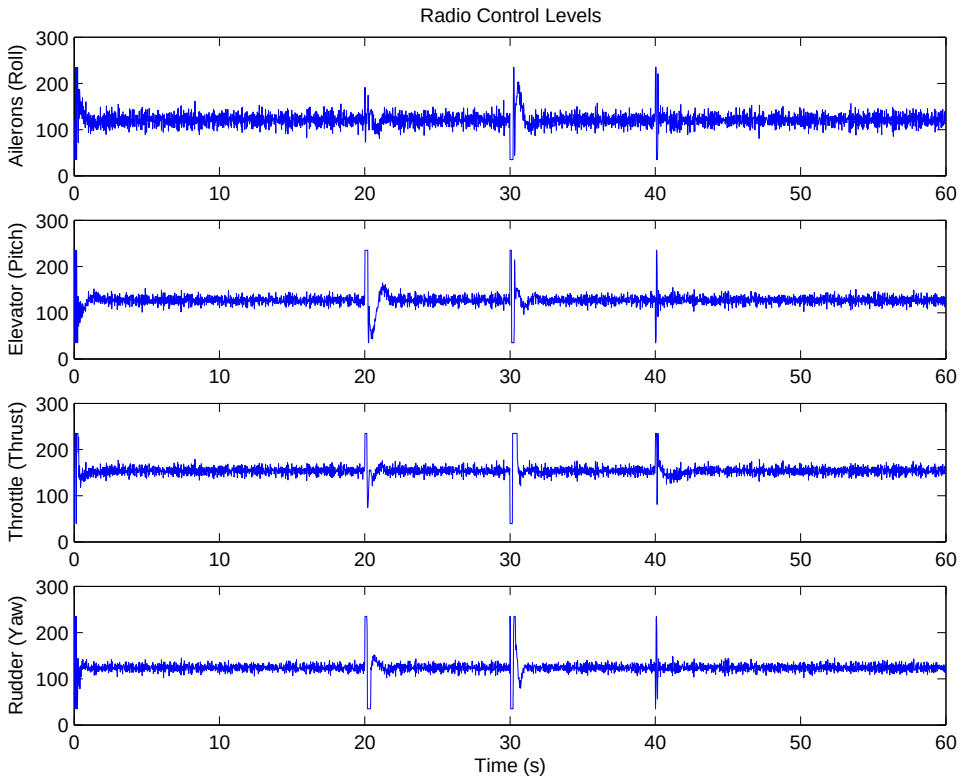


Figure 34: Plot of input values.

State Feedback Control Implementation

Next we implement the controller in our real-time system. Using the same values that were derived in simulation we set the helicopter to hover in one place. In addition we add a ramp-up and ramp-down stage to the throttle input to force a slow takeoff and landing. This happens at $t = 10$s and $t = 20$s. The results for the real time flight are given below. Comparison of the results of the hover in real-flight to a simulated hover is shown in Figure 41.

As the Draganflyer is flying the battery slowly depletes. The Draganflyer will sustain flight for 5-8 minutes depending on the amount of manuevering. As it uses the battery power, additional throttle is needed to compensate the loss in battery power. We end up with a slow drift away from the desired position as seen in Figures 42and 41. In addition, we can see from the plot that the system is sensitive to invalid sensor data, which can occur for instance, if an led marker on the helicopter is occluded and the measurement signal has a "blip". When an invalid data point is measured, the system will prolong the error effect as the EKF estimates try to re-converge to the actual states again.

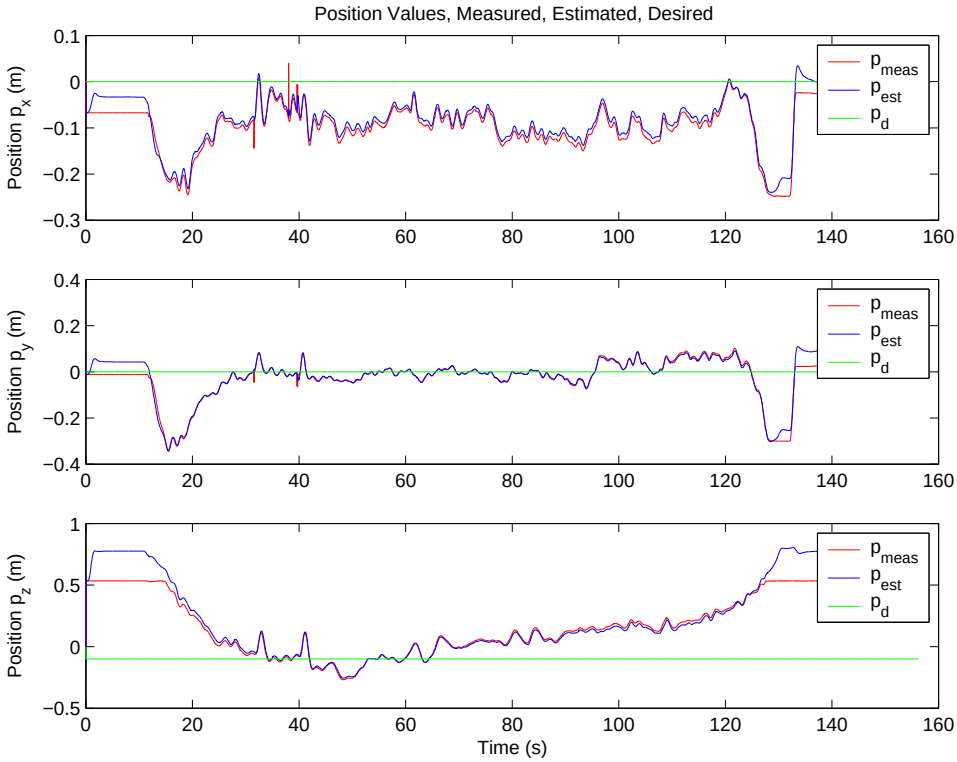


Figure 35: Implementation plot during hover.

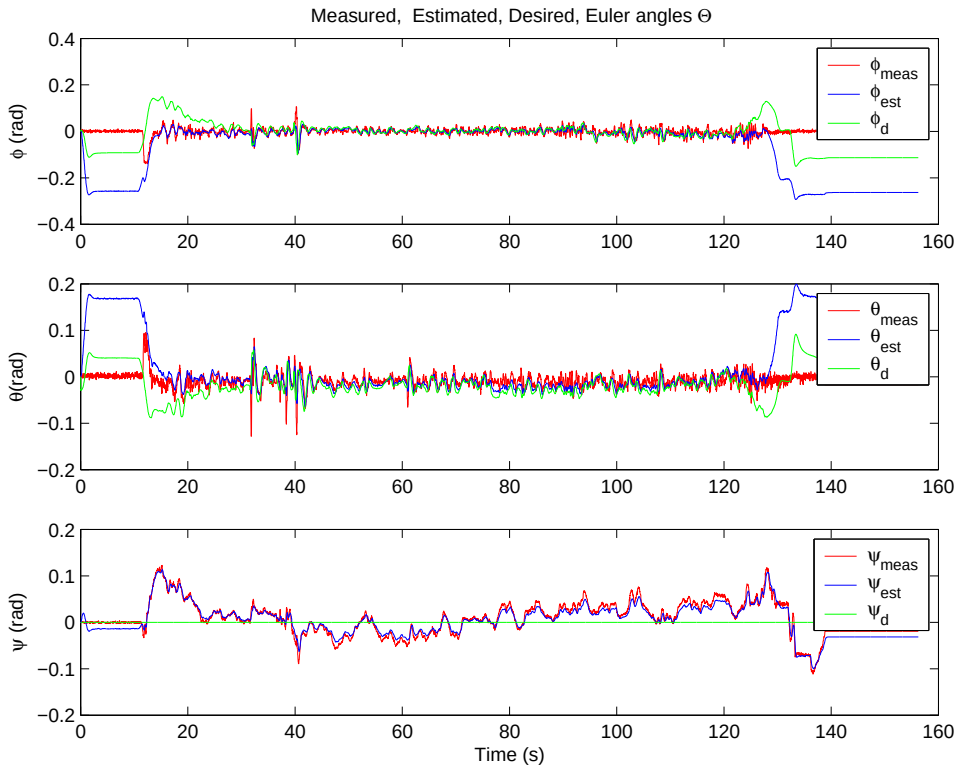


Figure 36: Plot of Euler angle values.

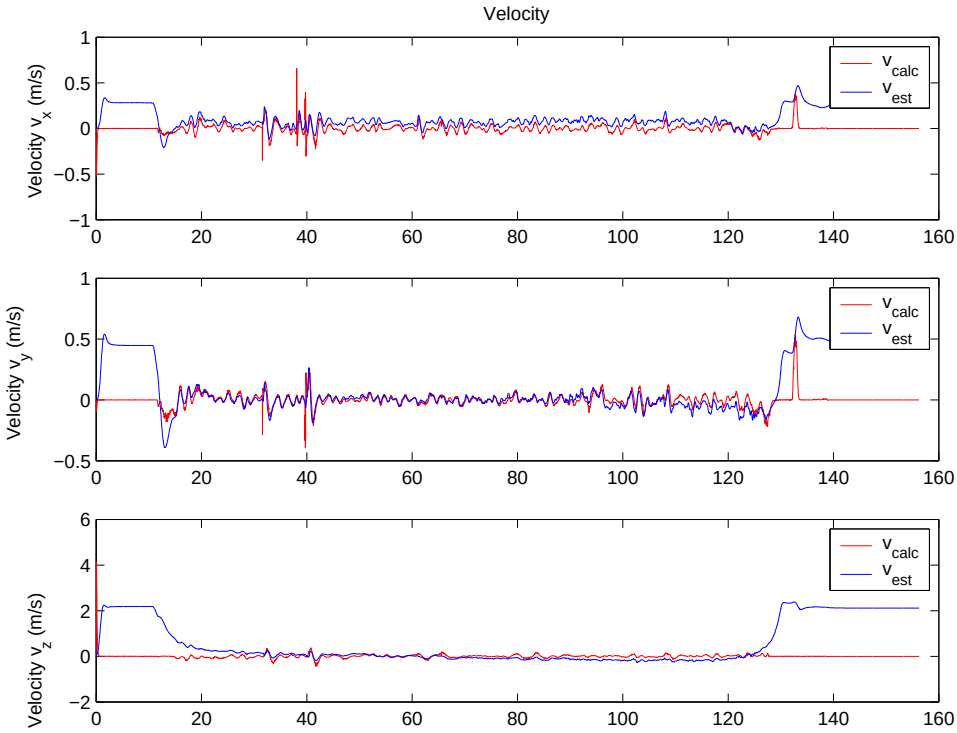


Figure 37: Plot of velocity values.

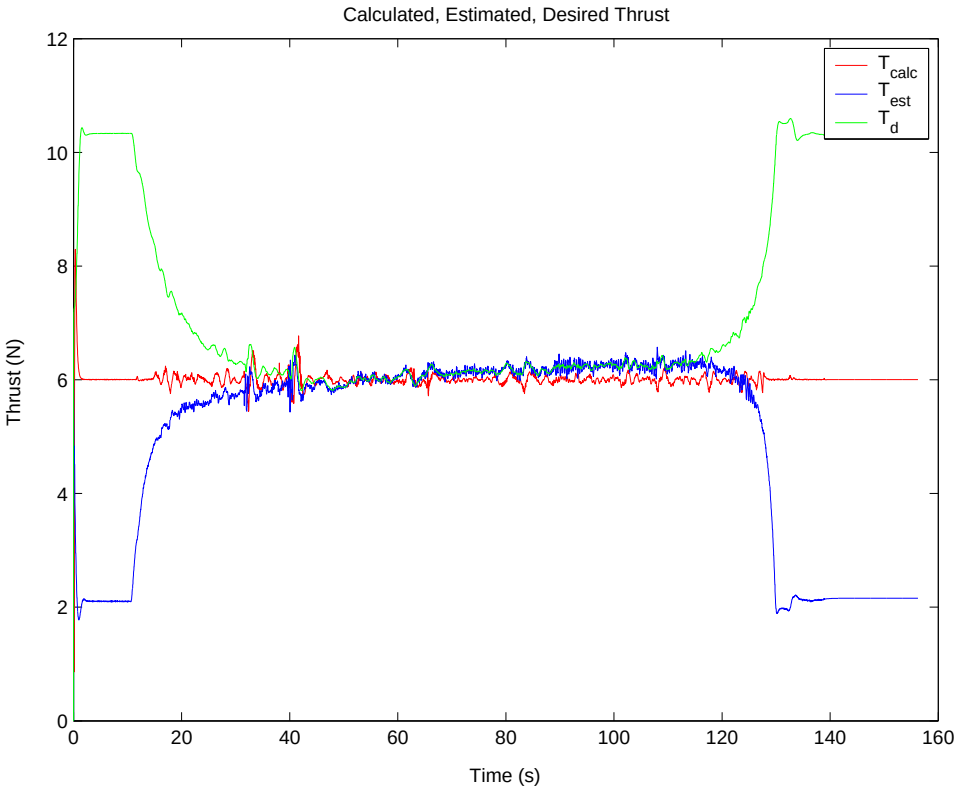


Figure 38: Plot of thrust values.

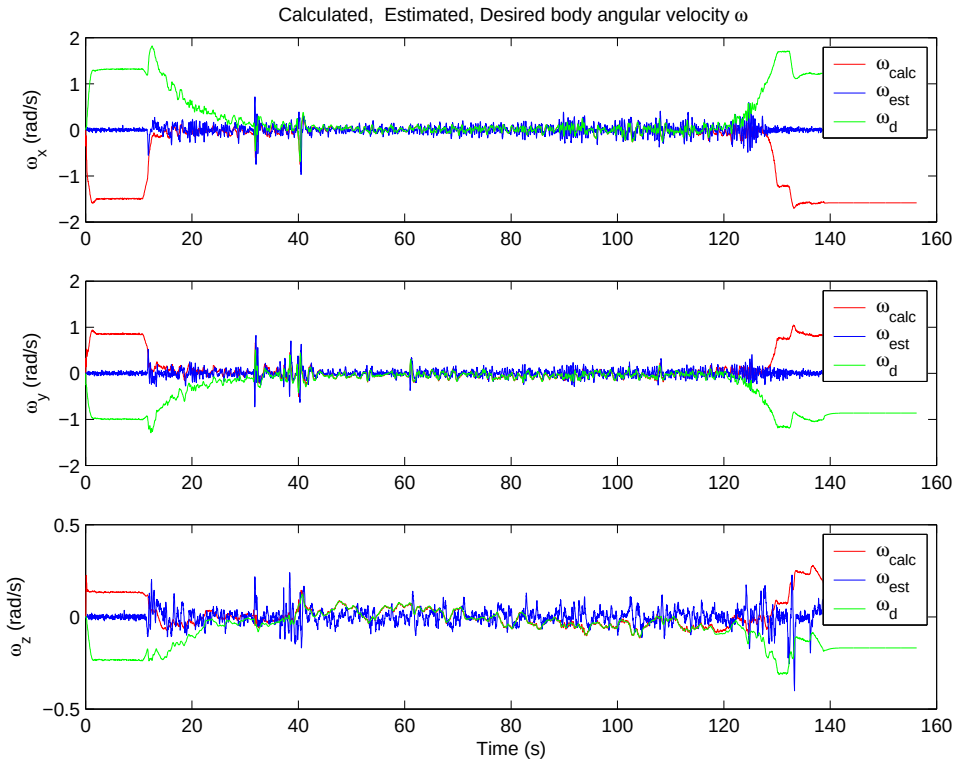


Figure 39: Plot of ω values.

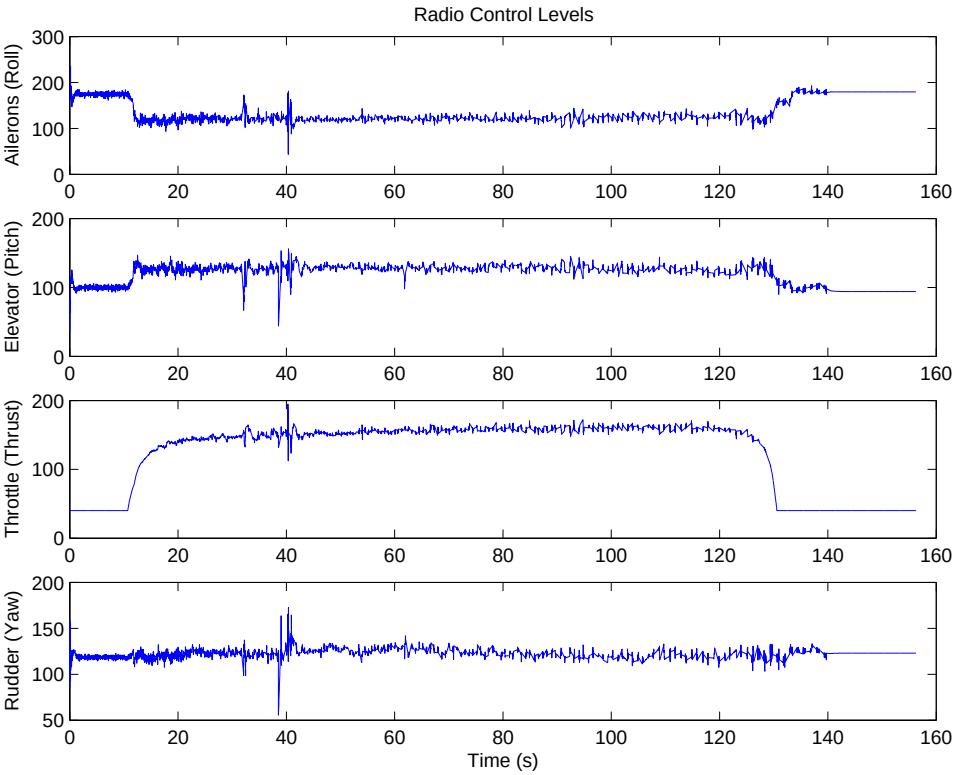


Figure 40: Plot of input values.

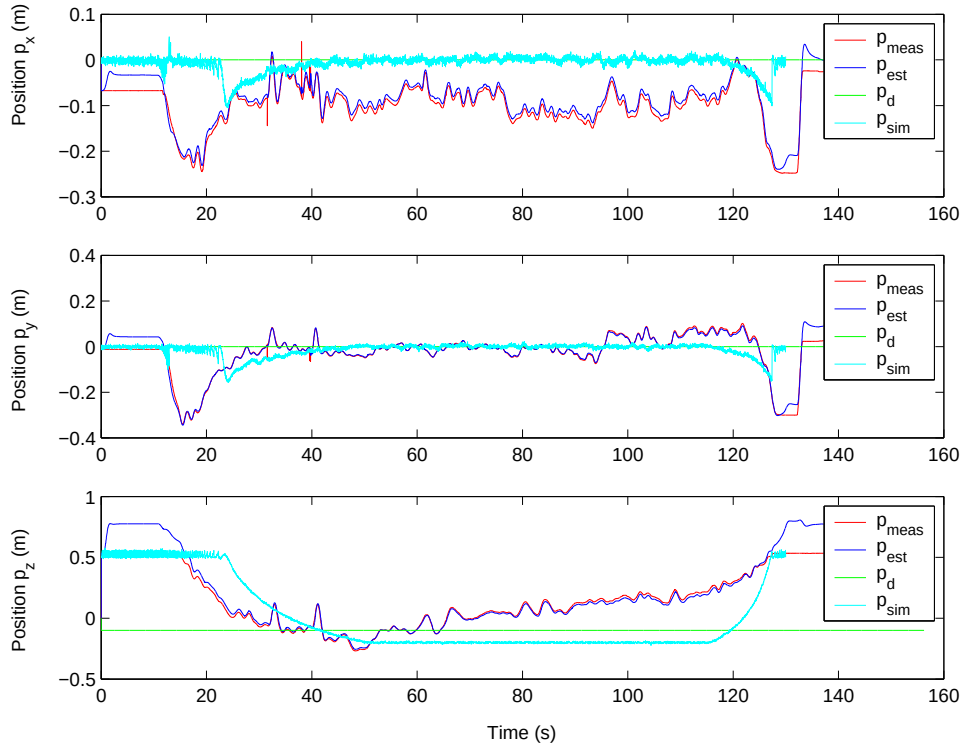


Figure 41: Comparison of simulated vs real flight hovering.

Since we have shown that the system can stabilize in hover mode using the state-feedback controller, we are interested to see the response for changes in the position reference. The results for the real time flight are given below. Looking at the results of the response to changes in reference input, we see that the system does not track the desired z-position well when making a transition. It is possible that with a more accurate model and a more refined or optimal controller, the system performance would be adequate. However, based on the performance of the feedback controller we determine that a more robust and less sensitive controller should be used to perform the multi-vehicle coordination. A more basic controller that had already been tested and tuned is used to control the multiple vehicle coordination to simplify the design and provide more performance reassurance. By using a PI controller with $T - T_d$ as the error vector, we can solve the battery depletion issue. In addition, because during hover the helicopter is close to the equilibrium point, we can design a proportional controller to drive w_d^b and make the assumption that the onboard controller can drive $(w^b \rightarrow w_d^b)$. The equation for w_d^b is given as $w_d^b = \Psi^{-1}(\Theta)(-K_\Theta(\Theta - \Theta_d)$. In addition, for small values of Θ, $\Psi^{-1}(\Theta)$

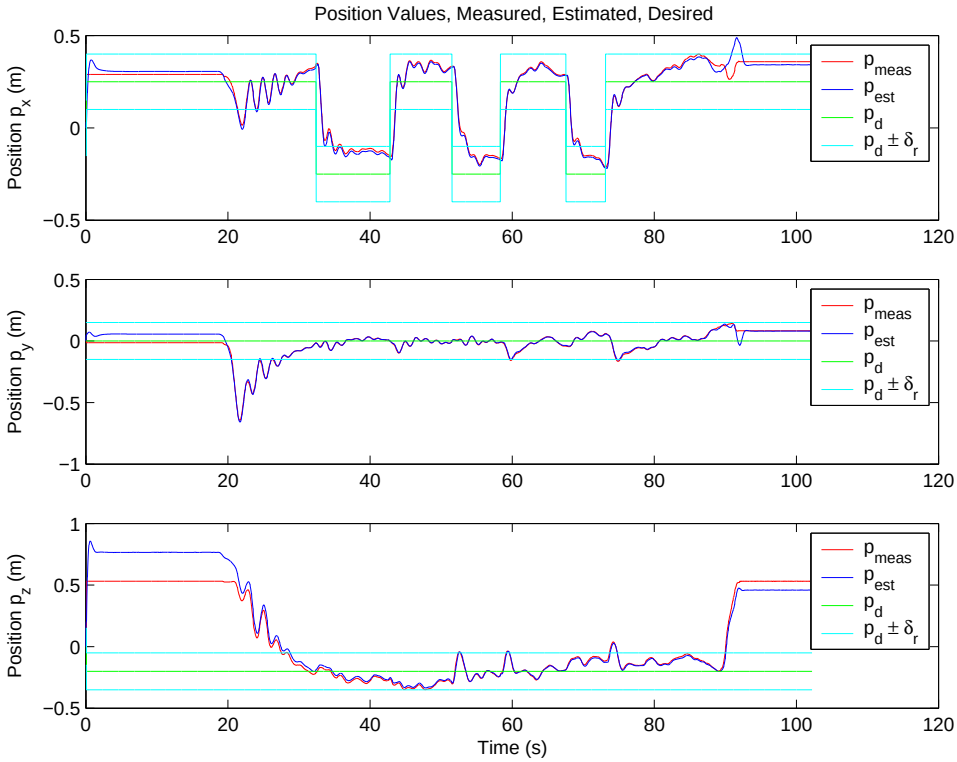


Figure 42: Implementation plot during hover.

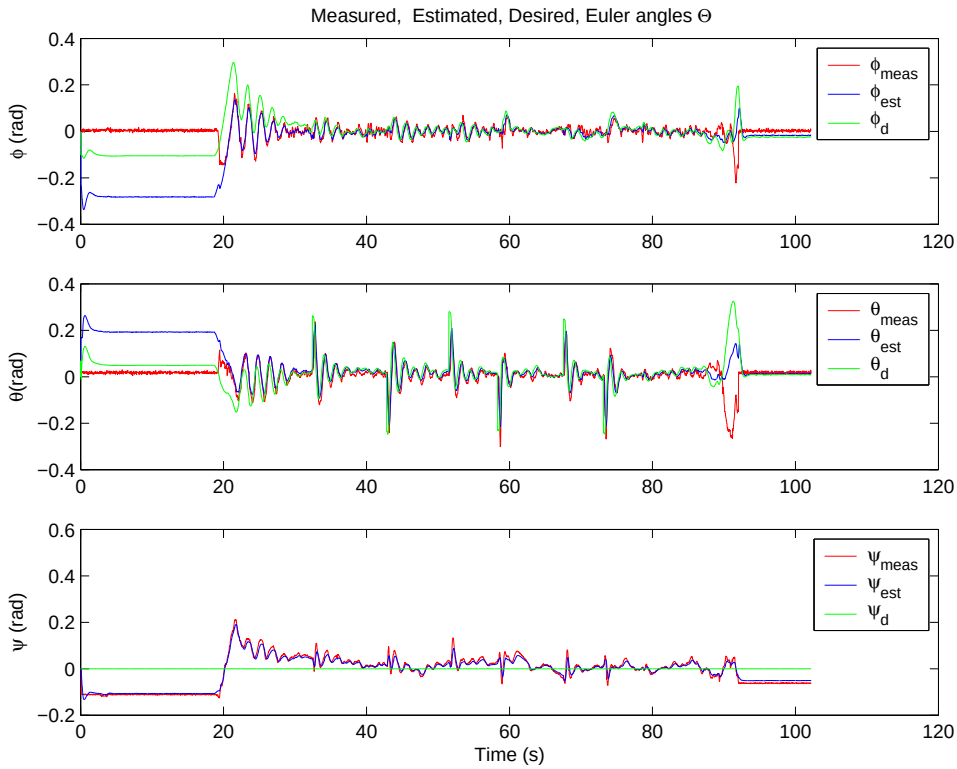


Figure 43: Plot of Euler angle values.

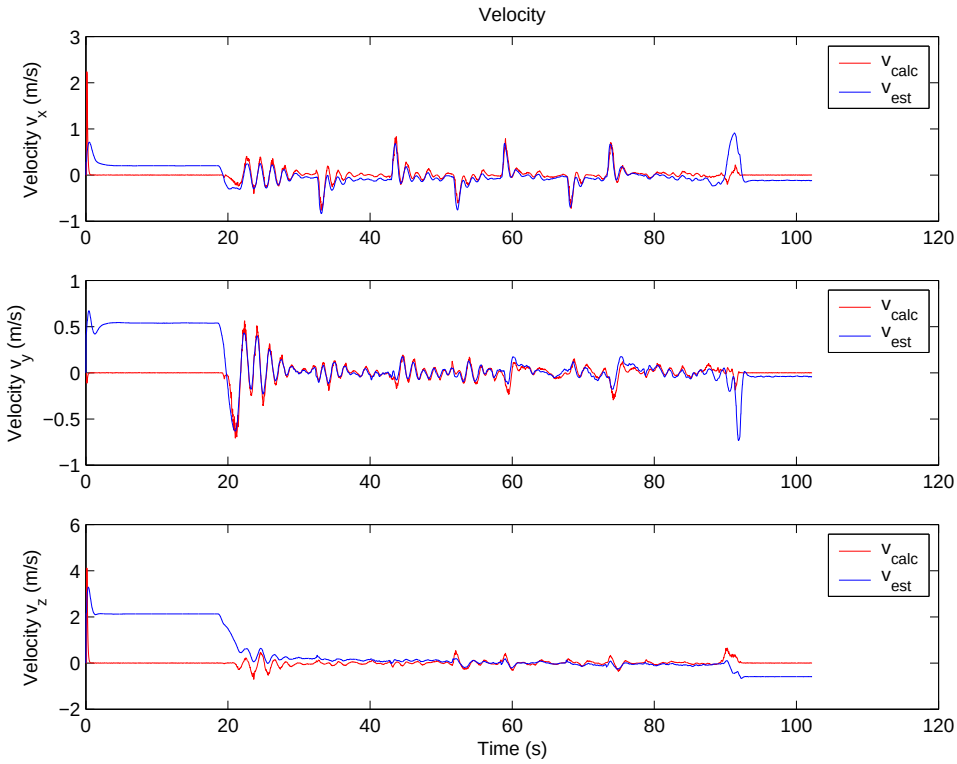


Figure 44: Plot of velocity values.

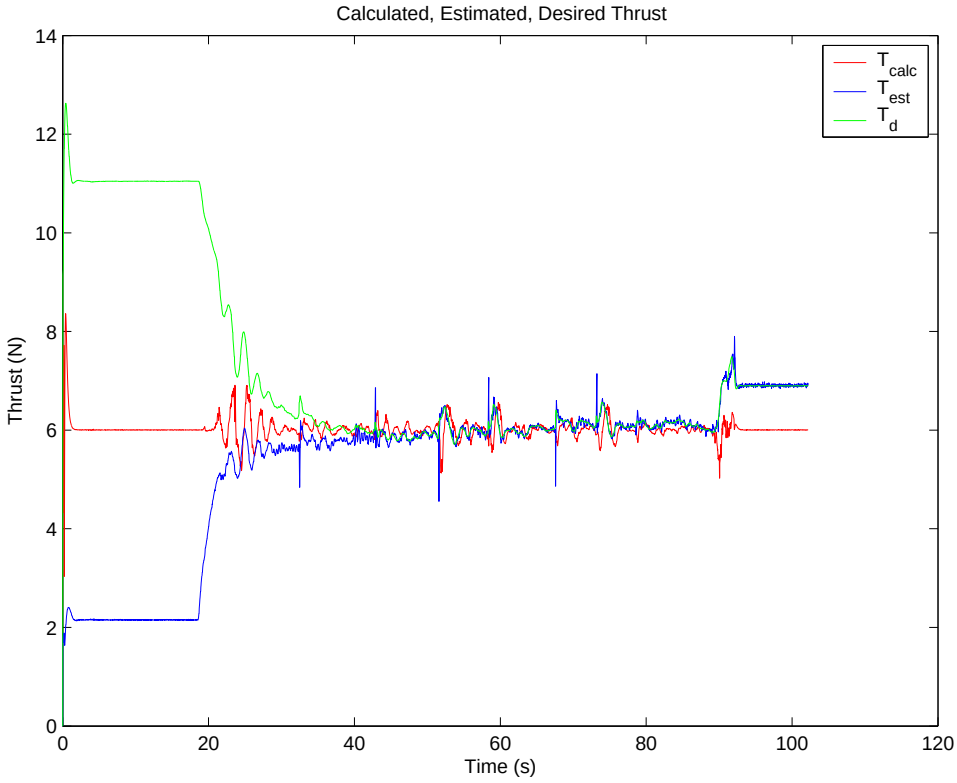


Figure 45: Plot of thrust values.

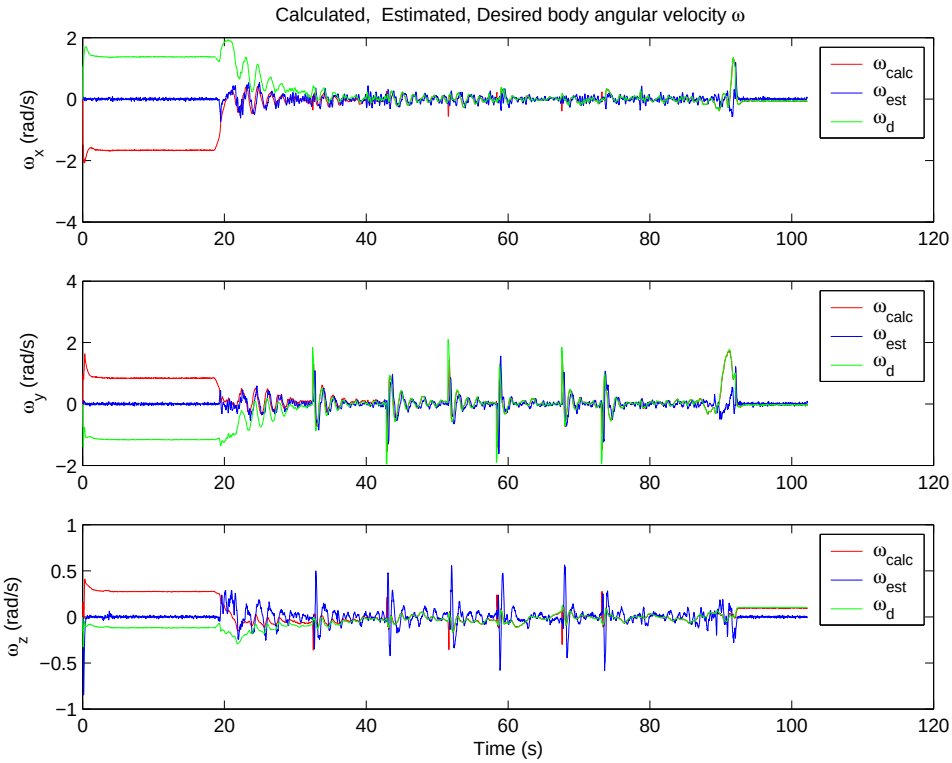


Figure 46: Plot of ω values.

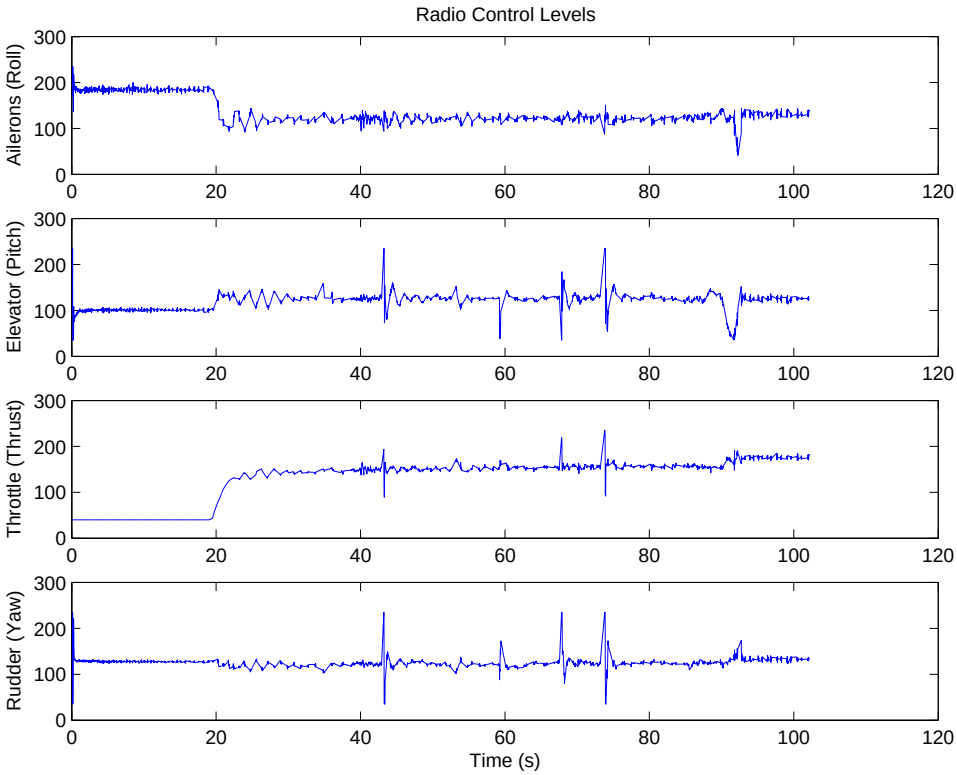


Figure 47: Plot of input values.

is close to identity matrix. We use the same form of the outer controller for controlling the position and generating Θ_d and T_d. Tuning this more basic controller in simulation results in

$$K_\Theta = \begin{bmatrix} -3 & 0 & 0 \\ 0 & -3 & 0 \\ 0 & 0 & -2 \end{bmatrix} \tag{142}$$

$$K_T = -.2 \tag{143}$$

The step response for Θ_d and T_d are given in Figures 48 and 49.

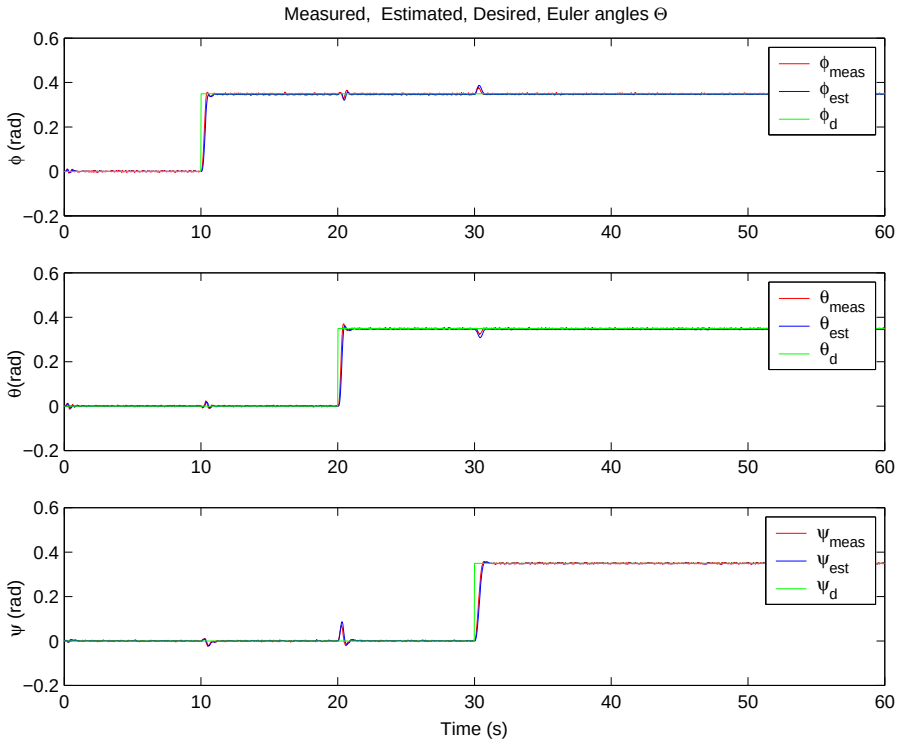


Figure 48: Inner controller Θ step response.

Rise times for ϕ, θ, ψ and T are .35s, .4s, 1s and .6s respectively.

The outer controller parameters are now designed. We design the outer controller parameters to yield a response that is slower than the inner system by a factor of 5. This means a response between 1.75s to 5s. The simulation model for the outer controller is shown in shown in Figure 50. The outer controller parameters are found to be

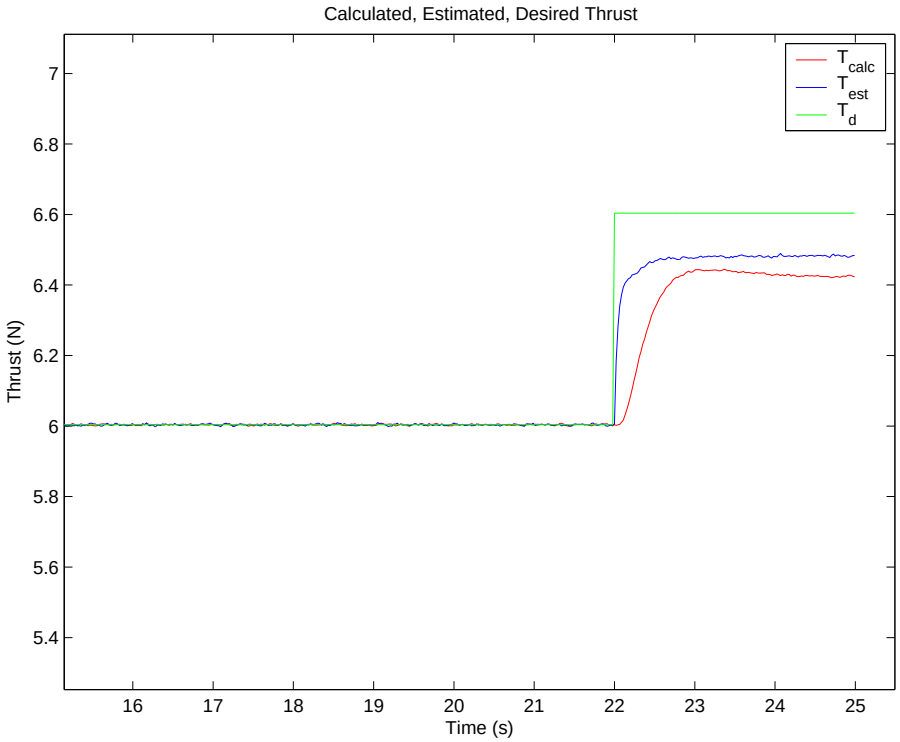


Figure 49: Inner controller T step response.

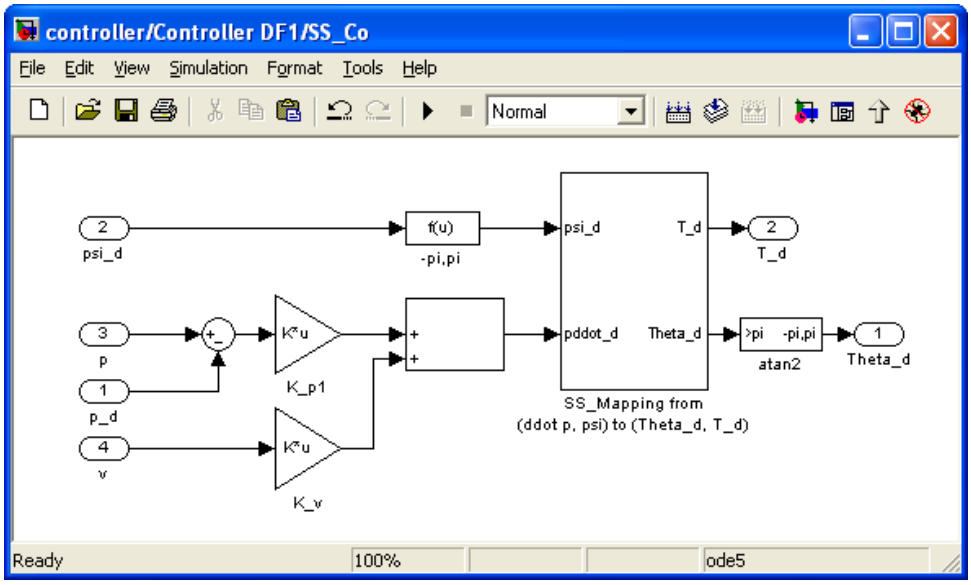


Figure 50: Outer controller simulation model.

$$K_p = \begin{bmatrix} -2 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & -.35 \end{bmatrix} \tag{144}$$

$$K_v = \begin{bmatrix} -2 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & -.75 \end{bmatrix} \tag{145}$$

A plot of the values given a step input for p_d are given in Figure 51. Rise times for p_x,

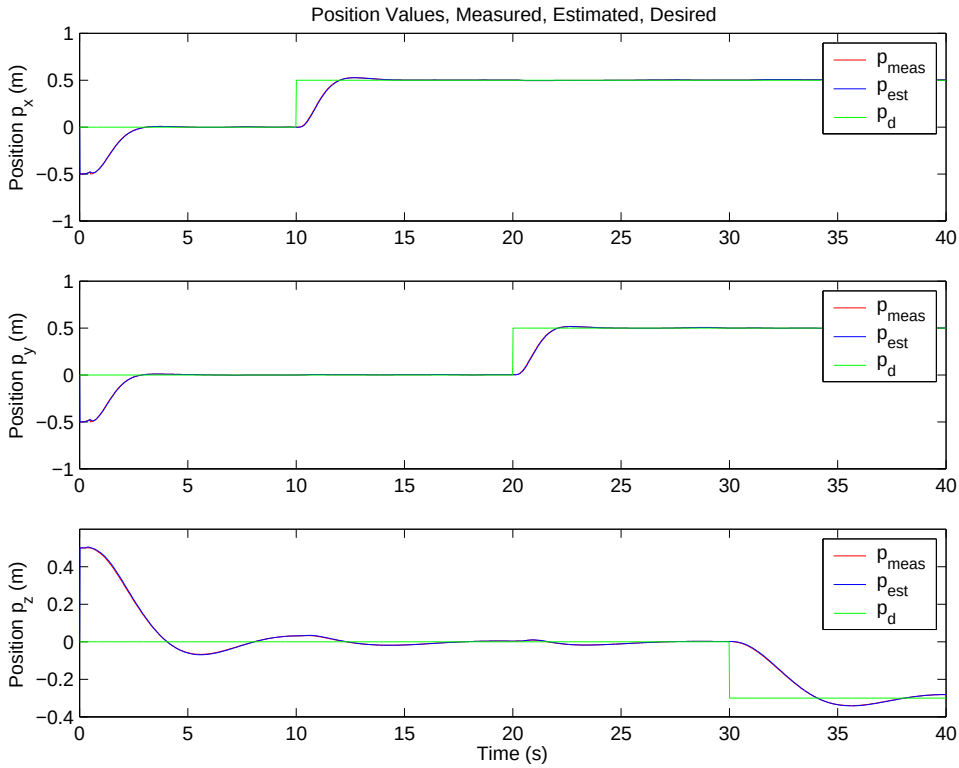


Figure 51: Outer controller step response for p.

p_y, and p_z are 2.5s, 2.5s and 4s respectively which matches with our design goal.

Upon implementing the controller with the parameters tuned in simulation, it is found that there is considerable overshoot with measured and desired values for the inner controller, namely with Θ. Possible reasons for this are the processing and radio signal delays not represented in the simulation and discrepancies between the model and actual system. Therefore, we reduce the gain K_Θ. In addition, we address the battery discharge

problem and steady state offset in p_z by adding the integral term K_{TI} to the thrust error. The final gains are tuned to the following values:

$$K_\Theta = \begin{bmatrix} -1.5 & 0 & 0 \\ 0 & -1.5 & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{146}$$

$$K_T = -.15 \tag{147}$$

$$K_{TI} = -.1 \tag{148}$$

Gains for the outer controller are changed to

$$K_p = \begin{bmatrix} -2 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{149}$$

$$K_v = \begin{bmatrix} -1.5 & 0 & 0 \\ 0 & -1.5 & 0 \\ 0 & 0 & -.5 \end{bmatrix} \tag{150}$$

We apply a step input to the position reference while the helicopter is hovering. In order to get the helicopter to hover first, we slowly increase the throttle while providing a position reference input. The results in the figures below show the vertical takeoff, three step inputs and the landing. Also note that due to the North-East-Down frame assignment, vertical lift is in the negative z direction. The results for applying a step input during hover are shown in Figures 52, 53, 54 and 55.

Outer system response times are approximately 2.4s, 3s, and 3.5s for p_x, p_y, and p_z which meets our specifications. A comparison between the position values for the simulated and measured step responses is given in Figure 56. Here the gain values designed in implementation have been used in the simulation. This controller will be used in the final implementation of the multi-robot coordination.

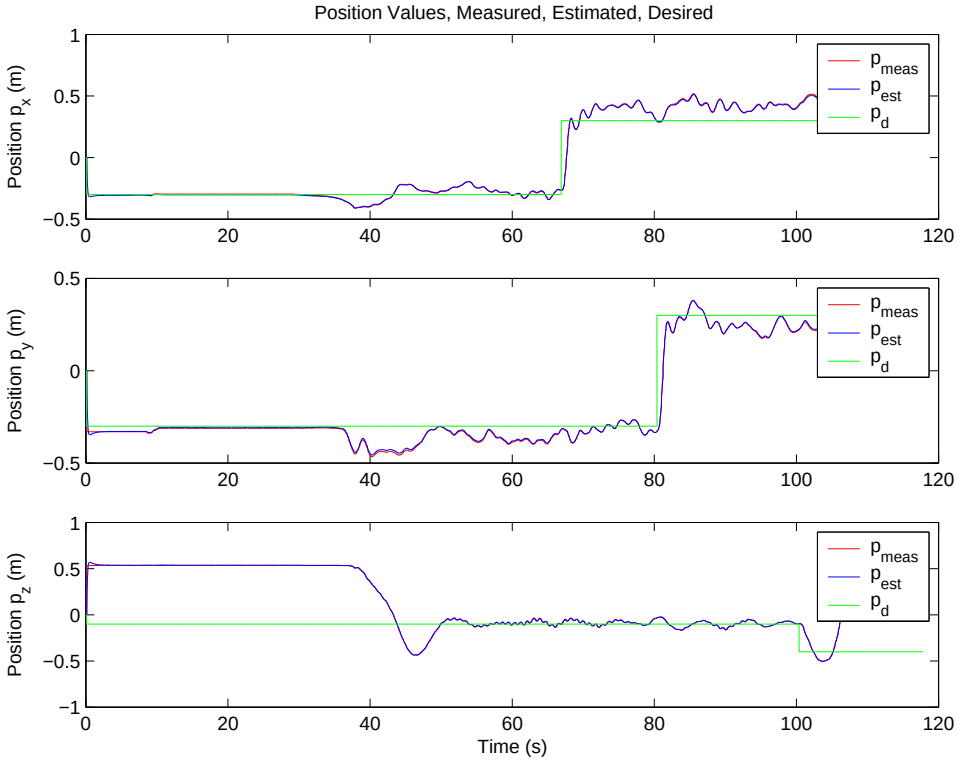


Figure 52: Position values for controller implementation.

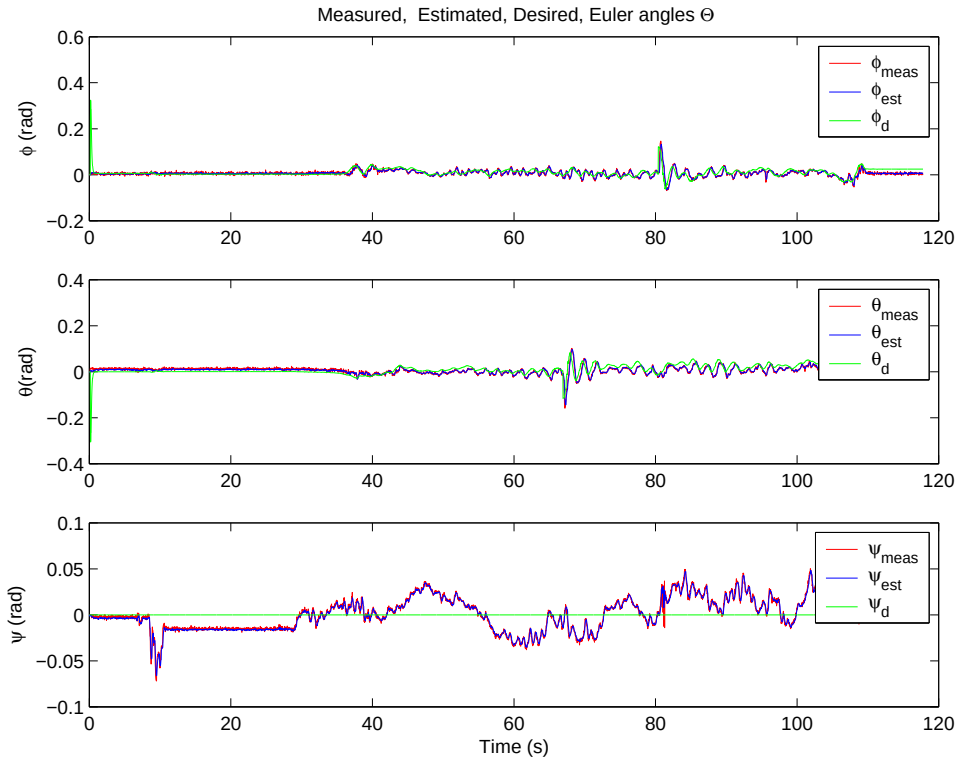


Figure 53: Euler angle values for controller implementation.

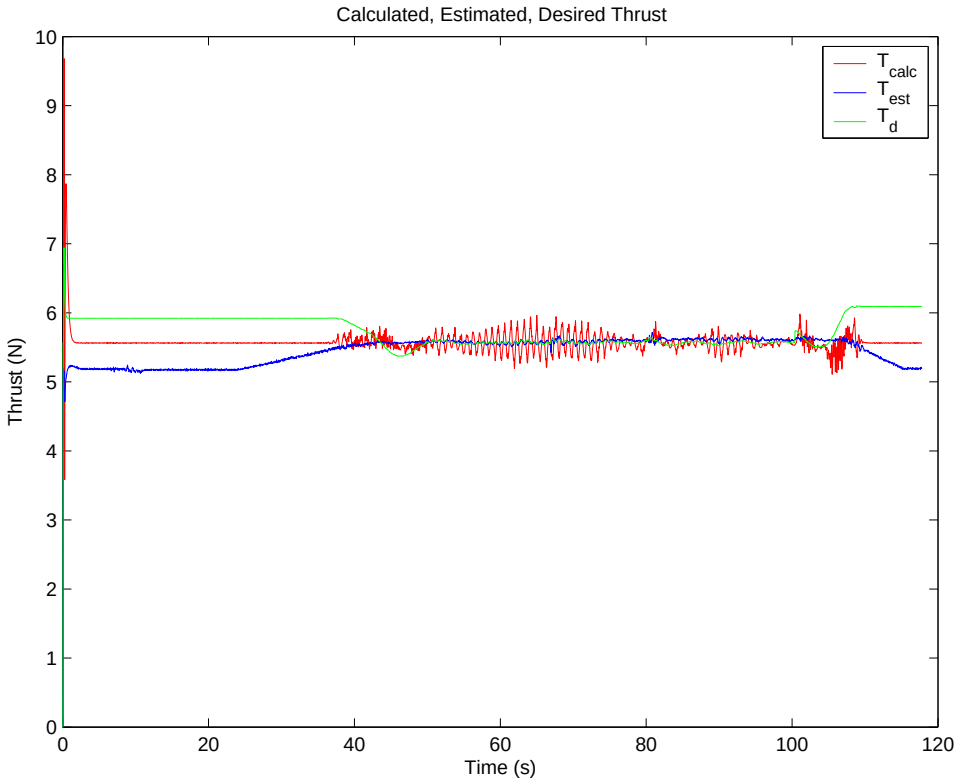


Figure 54: Thrust values for controller implementation.

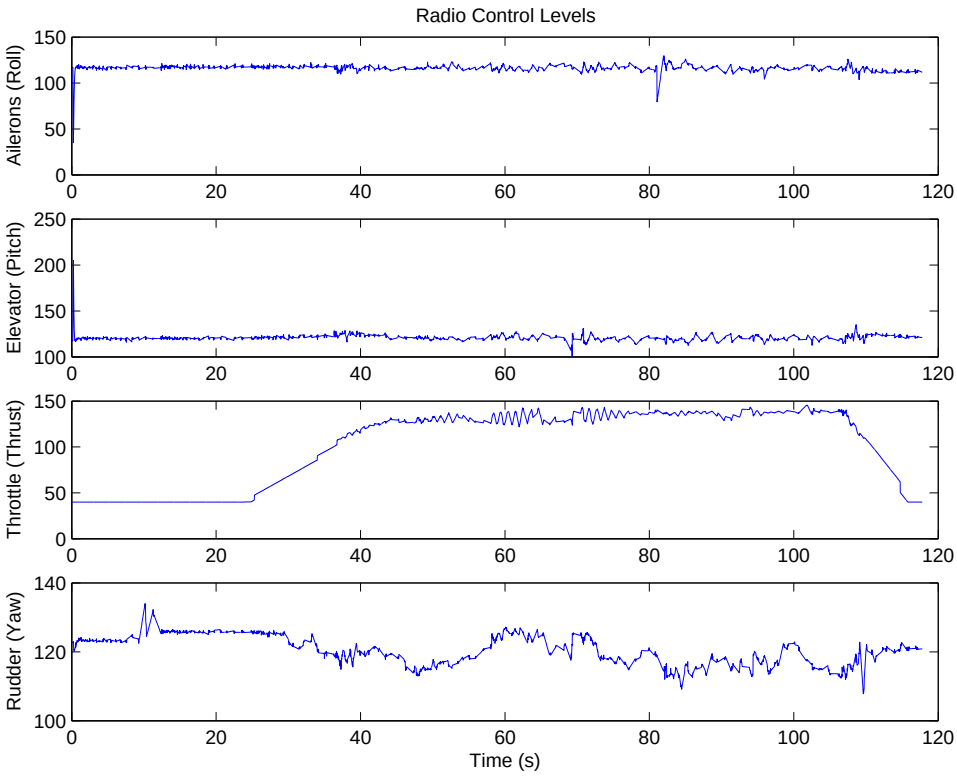


Figure 55: Radio control signals for controller implementation.

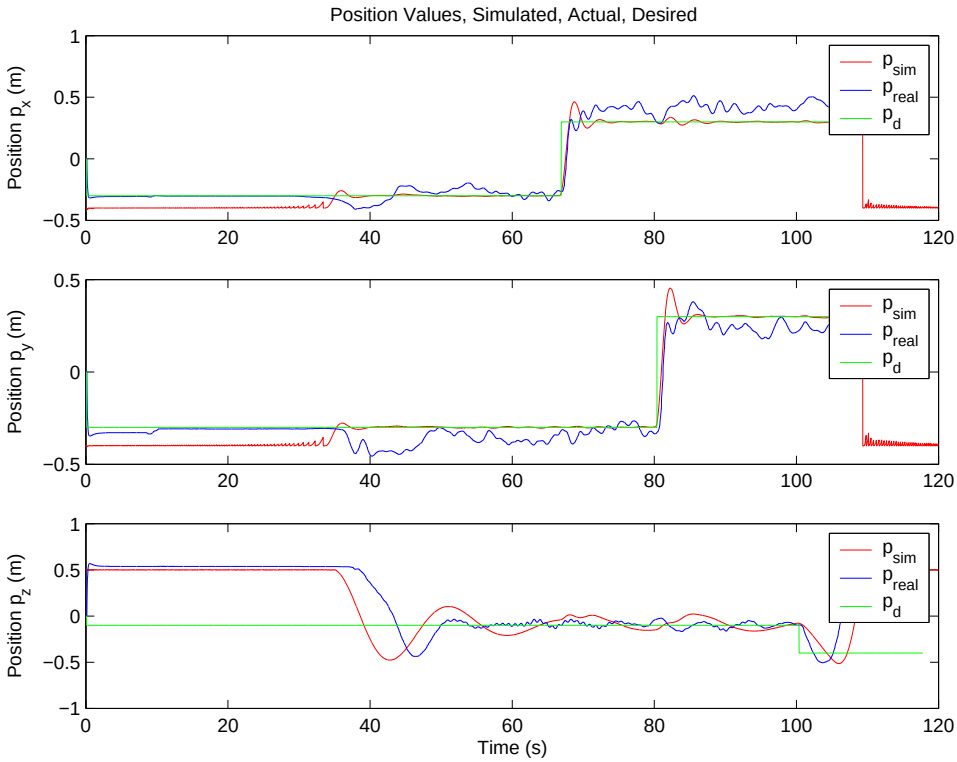


Figure 56: Comparison of simulated and actual flight with step input.

CHAPTER IV

HYBRID SYSTEM ARCHITECTURE

Hybrid systems have been used successfully to model the behavior of dynamical systems that exhibit both continuous state and discrete state dynamics. Several well-known examples are the bouncing ball, thermostat and two-tank systems. In these systems, within each discrete state of operation, the system may behave differently and can be described by a specific set of dynamic equations. A fundamental problem we wish to solve in this paper is the coordination of multiple robots in such a way that certain specifications are met, such as the avoidance of collisions and completion of waypoint navigation.

Hierarchical System Design

Our design method involves a model based approach that is motivated by the desire to be able formally specify a multi-vehicle mission and be able to make guarantees about the system performance. The multi-stage modeling approach is shown in Figure 57. Therefore, we design the system with information in mind about the bottom layer

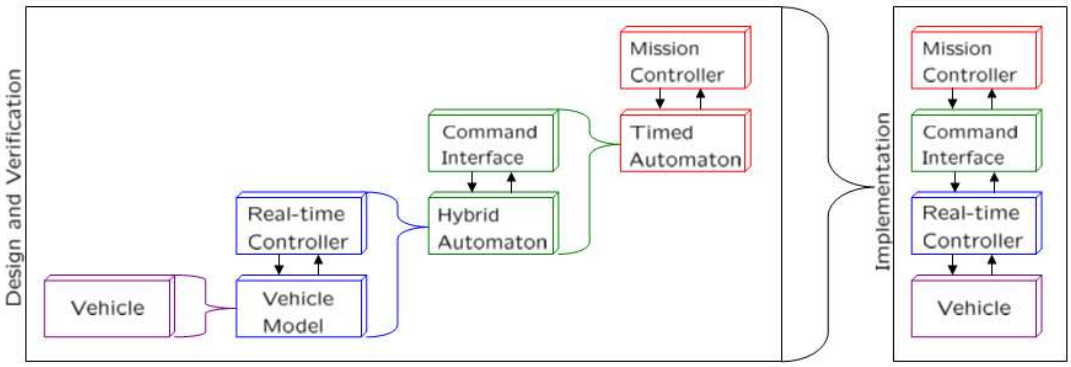


Figure 57: Multi-stage modeling approach.

(vehicle) and the top level (mission). We choose a specification language CTL, that is understandable to humans and computers, thus forming an interface, and generate a timed automaton (TA) to represent the multi-vehicle system. A TA is a finite state machine that has been extended by the use of clock variables. Except for the clocks, the TA is mainly a discrete event system. In order to bridge the gap between this discrete system

and our real-time vehicle, we design a hybrid automata (HA) to represent the vehicle system which can be viewed as a multi-modal system. It can be shown that the TA is bisimular to the HA, meaning that specifications checked on the TA will hold for the HA, and the two systems will give the same input-output data. In addition, because bisimilar system preserve branching time properties, the branching time calculation in the symbolic verification will cover the trajectories exhibited by the real-time hybrid system. [1]

The TA is developed using a tool called UPPAAL, a software program used to verify the specification requirements in computational tree logic (CTL) and generate trajectories if any exist that satisfy the specifications. Our design of the TA is motivated by fact that we wish to be able to generate trajectories for complex missions while guaranteeing that safety and performance specifications are met. By extracting the trajectory information from the verification results we can provide the real-time hybrid-system with commands that will meet the specifications that have been verified. A major benefit to this design is that at each stage of the design we construct a model that bisimulates the combined components lower in the hierarchy, so that the TA bisimulates the hybrid automata/command interface subsystem, the hybrid automata bisimulates the real-time controller/vehicle model subsystem and so on. Therefore we can guarantee that specifications verified using the TA will hold for the constructed system.

Timed Automata

A TA is a finite-state machine that has been extended to include real-valued clocks, which proceed simultaneously and measure the amount of time elapsed since being reset. We will use a network of TA to bisimulate the hybrid system and the verification tool UPPAAL to test certain specifications about the system. The network, which consists of the robots, the environment and the control, communicate over synchronization channels. We adopt the convention used in [2] to define the TA.

Definition 4.2 *(Timed Automaton):* A timed automaton (TA) is a tuple $\mathcal{A} = (L, l_{init}, E, I, V)$ defined over actions Act, propositions $\mathcal{P}$, clocks $\mathbb{C}$ and guards $\mathcal{B}(\mathbb{C})$ where:

[1] For a detailed review of bisimilar systems refer to [9] or [10].

- L is a set of control locations;
- l_{init} is the initial control location;
- $E \subseteq L \times \mathcal{B}(C) \times Act \times 2^{\mathbb{C}} \times L$ is a finite set of edges;
- $I \ : \ L \rightarrow \mathcal{B}(\mathbb{C})$ is the invariant condition of the control location;
- $V \ : \ L \rightarrow 2^{\mathcal{P}}$ assigns a proposition to each location, where $2^{\mathcal{P}}$ is the power set of$\mathcal{P}$.

Essentially, a TA is a hybrid system in which the clock dynamics are defined by $\dot{c}_j = 1, \forall c_i \in \mathbb{C}$ and can accept only non-negative values.

Partitioning the Physical Space

Because we wish to manage the multi-vehicle motion in the physical environment, we decompose the continuous state space $X \subseteq \mathbb{R}^3$ in which the vehicles move into a finite number of cells by a partition $\pi = \{X_i\}_{i=1}^{n}$. The dimensions are identical for each X_i and were determined by examining the behavior of the closed-loop system. The partition satisfies

$$X = \bigcup_{n} X_i \tag{151}$$

$$\emptyset = X_i \cap X_j, \forall i \neq j \tag{152}$$

UPPAAL represents the partition as a 3D array, in which elements can be assigned a value of 1 if the physical location is occupied or currently reserved for transition purposes or a 0 if the cell is unoccupied.

Vehicle Process

We consider a group of H_m helicopters confined to the physical environment $X \subseteq \mathbb{R}^3$. The helicopters can move in positive and negative directions along the $x-$, $y-$, and $z-$axes between adjacent cells. Each helicopter motion behavior is assigned a discrete control location, so we have
$L = \{INIT,\ STOP,\ M\_PX,\ M\_MX,\ M\_PY,\ M\_MY,\ M\_PZ,\ M\_MZ\}$. In addition, the temporal motion behavior is defined by $\tau \in [\tau_1, \tau_2]$, the possible time to transition

from one cell to another. In the TA for the helicopter, a clock is assigned to each robot in order to represent the time it takes for the physical system to complete the motion command. Once a transition is made in the vehicle TA, it will remain in that state until the clock value falls within τ. The model checker tests the specification by evaluating all possible transitions within that time range, thereby representing the non-deterministic nature of the physical system. The TA model of the helicopter, $\mathcal{A}_H$, is given by:

- $L = \{INIT,\ STOP,\ M\_PX,\ M\_MX,\ M\_PY,\ M\_MY,\ M\_PZ,\ M\_MZ\}$
- l_{init} is the initial control location;
- $E \subseteq L \times \mathcal{B}(C) \times Act \times 2^{\mathbb{C}} \times L$ is the set of edges,

  $e_0 = (INIT,\ INIT)$

  $e_1 = (INIT,\ STOP)$

  $e_2 = (STOP,\ \texttt{movePX?},\ M\_MX)$

  $e_3 = (M\_MX,\ \tau_1 < c < \tau_2,\ c = 0,\ STOP)$

  $e_4 = \ldots$
- $I\ :\ L \rightarrow \mathcal{B}(\mathbb{C})$ is the invariant condition of the control location;

The automaton associated with the helicopter is shown in Figure 58. Initially, the au-

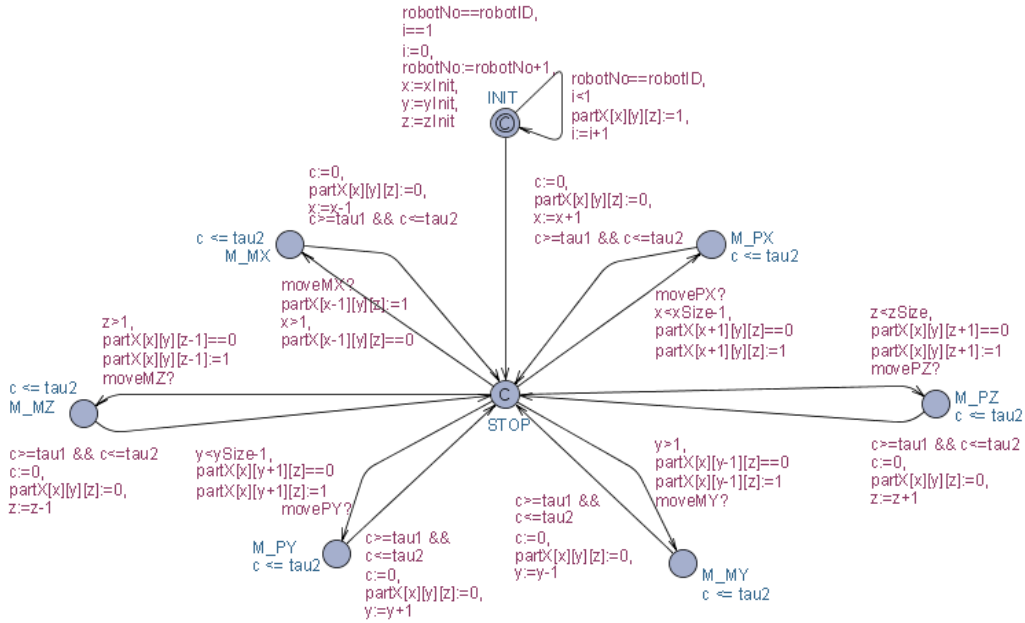


Figure 58: Automaton associated with helicopter vehicle.

tomaton starts in the $INIT$ location, transitions to $STOP$ and then is free to move

accordingly. Multiple vehicles can be created by instantiation within the UPPAAL system editor. Therefore, scaling the system is very straightforward.

Control Process

The helicopter automaton synchronizes with the control automaton shown in Figure 59 through the six synchronization channels. The vehicle process location is changed

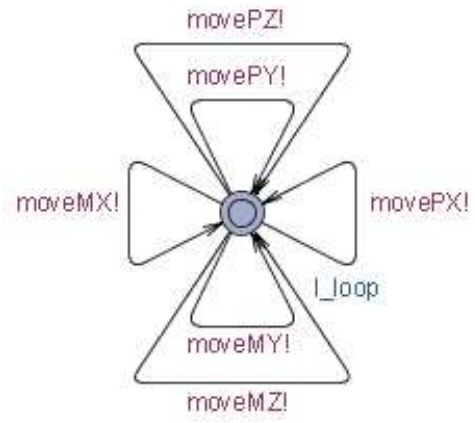


Figure 59: Automaton associated with control motions.

when control signals are received from the control process. In addition, all automata have access to the state-space partition represented as a 3D array, `partX[x][y][z]`, stored in a shared memory location as a global variable. Vehicles are prevented from occupying spaces that are currently occupied and spaces that are being used to move to or from another cell.

Specification Verification

We are concerned with verifying whether a system satisfies certain specifications. In order to make guarantees about the behavior of a given system, it is preferable to use a formal method that can be tested, proven and repeated. One such method involves the use of temporal logics, specifically computational tree logic (CTL) to reason about the time-changing state of the system. CTL enables us to inquire about a wide range of branching time properties. Given a system containing multiple autonomous vehicles and

its environment modeled as a network of TA, we can obtain a set of motion commands that will satisfy the required specifications such as collision avoidance and arrival to desired waypoints. CTL specifies the ability to eventually reach a destination and avoid collisions as liveness and safety properties, respectively. Although we cannot check these properties simultaneously, by using a correct-by-construction approach in modeling the TA and then using the UPPAAL model checker to determine the liveness property we can verify guarantee that if a path exists, the vehicles will reach their destinations while avoiding collision. We are concerned with this specification for the purposes of this paper. We therefore pose the following CTL query in UPPAAL given a team of two vehicles `H1` and `H2` with initial positions `H1.pinit` and `H2.pinit`:

```
E<>(H1.x==H1.xd and H1.y==H1.yd and H1.z==H1.zd and
    H2.x==H2.xd and H2.y==H2.yd and H2.z==H2.zd)
```

As part of the symbolic verification of the CTL query, a trace file is generated that contains all the transition events and intermediate states as well as the branching clock value. We parse this trace file to obtain a sequence of motion commands $\mathscr{L}$ associated with the discrete command locations of the control process.

Hybrid Architecture

Next, we show how the complete hybrid architecture is constructed from the vehicle model, real-time controller, command interface and mission controller.

Vehicle Model

We begin by modeling the physical system as shown in Chapter 2. This is done by collecting input-output data from the vehicle in manual or closed-loop operation and performing system identification. The input to the plant is $u = [u_1\ u_2\ u_3\ u_4]^T \in [0, 255]^4$. The output from the vehicle model is the state vector $x = [p^T\ v^T\ \Theta^T x_m^T]^T \in \mathbb{R}^6 \times \mathbb{S}^3 \times \mathbb{R}^6$ obtained from the EKF state estimator.

Real-time Controller

Based on the vehicle model, we design the real-time controller in simulation and further tune the controller parameters in actual flight as shown in Chapter 3. The real-time controller accepts as a high-level commands the reference position $r = [p_d^T \psi_d]^T \in \mathbb{R}^3 \times \mathbb{S}$, and based on the state of the vehicle, x, computes the control signal, u and returns the continuous position, p to the command interface.

Hybrid Automaton

In order to model the different behaviors associated with the hybrid automata, we will use a descriptive language as shown in [21].

Definition 4.2 *(Hybrid Automaton):* A hybrid automaton H is a collection $H = (Q, X, f, \Sigma, Init, E, G, R)$ where

- $Q = \{q_1, q_2, \ldots\}$ is a set of discrete states;
- $X = \mathbb{R}^n$ is a set of continuous states;
- $f(\cdot)$:$Q \times X \rightarrow R^n$ is a vector field;
- Σ is a set of events $\{\sigma_1, \sigma_2, \ldots\}$
- $Init \subseteq Q \times X$ is a set of initial states;
- $E \subseteq Q \times Q$ is a set of edges;
- $G(\cdot) : E \rightarrow P(X)$ is a guard condition;
- $R(\cdot, \cdot) : E \times X \rightarrow P(X)$ is a reset map.

$P(X)$ denotes the power set of X, and we refer to $(q, x) \in Q \times X$ as the state of H. Hybrid automata therefore describe the possible continuous state evolutions of x stating from $(q_0, x_0) \in Init$ flowing according to the differential equation

$$\dot{x} = f(q_0, x), \tag{153}$$

$$x(0) = x_0. \tag{154}$$

while q remains constant.

For our system we have the hybrid automata defined by $H = (Q \times X, f, \Sigma, Init, G, R)$ where

- $Q = \{q_i\}_{i=0}^{6}$ is a set of discrete states;
- $X \subseteq \mathbb{R}^3$ is the physical state space in which the vehicle moves;
- $p = [p_x\ p_y\ p_z]^T \in X$ is the continuous position of the vehicle
- $Init$ is the initial set (q_0, p_0);
- $\Sigma = \{\sigma_i\}_{i=0}^{6}$ is a set of events; The vector field
- f is defined by:

$$f(q,x) = \begin{cases} [0\ 0\ 0]^T & \text{if } q = q_0, \\ [\mu_x\ 0\ 0]^T & \text{if } q = q_1, \\ [-\mu_x\ 0\ 0]^T & \text{if } q = q_2, \\ [0\ \mu_y\ \ 0]^T & \text{if } q = q_3, \\ [0\ \ -\mu_y\ \ 0]^T & \text{if } q = q_4, \\ [0\ 0\ \mu_z]^T & \text{if } q = q_5, \\ [0\ 0\ \ -\mu_z]^T & \text{if } q = q_6. \end{cases} \tag{155}$$

where $.2 <= \mu_x, \mu_y, \mu_z <= 1$ are based on our travel cell times. The guard is given by $G(q_i, q_j) = X \times \sigma_j$, and the reset by $R(q_i, q_j, x) = x$. The hybrid automaton that models our real-time system is shown in Figure 60.

Let us now look at the components of the hybrid system. Once the real-time controller is designed, the next step is designing the command interface. In order to do this we must determine the parameters of the closed-loop (i.e. controller and vehicle) system. In our final design we partition the 3-D state space into a collection of discrete locations or cells (actually 3-D polyhedra). To determine the dimensions of these cells we study the behavior of the closed-loop system. Let β_{δ_r} be a ball of radius $\delta_r \in \mathbb{R}$ centered at a given reference position $p_d \in \mathbb{R}^3$ with respect to the inertial frame. We wish to determine δ_r

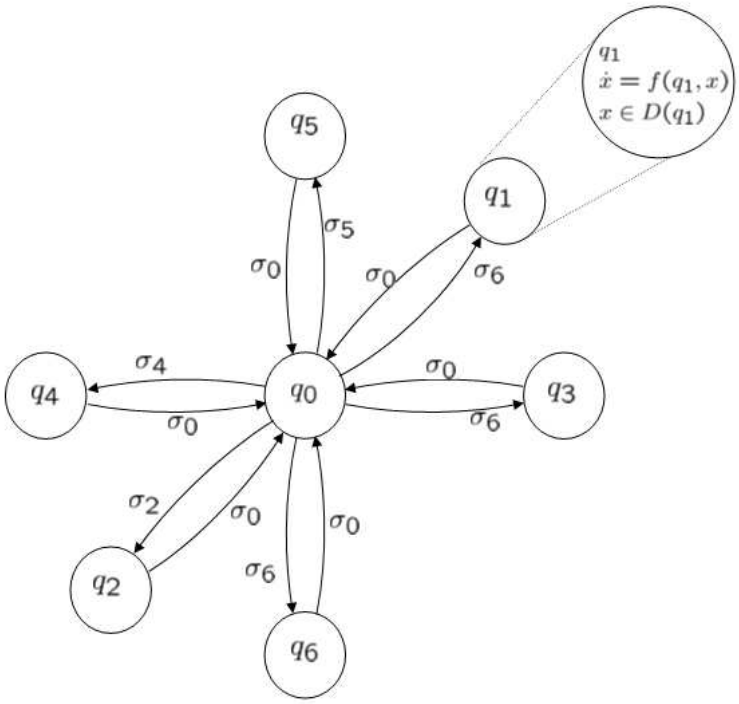


Figure 60: Hybrid automaton that models the real-time system.

such that the position of the vehicle (i.e. the c.g.) remains inside β_{δ_r} during hover. This constraint is given as

$$\|p - p_d\| < \delta_r. \tag{156}$$

We find by δ_r experimentally by flying the helicopter in hover mode at a fixed reference point and looking at the deviation from that point. A plot showing one of these trials is shown in Figure 61. After several trials, δ_r is determined to be 12.5cm. This means that given a fixed reference point, the c.g. of the helicopter will stay within a ball of radius 12.5cm centered at that reference point. Note that 12.5cm is a conservative determination as most of the trials resulted in flight regions of approximate radius equal to 5cm. However, due to disturbances and possible offsets caused by c.g. changes or twisting of the helicopter frame, 12.5cm is a reasonable conservative determination. Although the position of the c.g. will remain inside β_{δ_r}, the actual helicopter occupies a much larger space. We compute the total space occupied during flight by adding δ_r to the physical dimensions of the helicopter. The dimensions of the helicopter are 75cm×75cm×15cm (l×w×h). Therefore, we will use a cell size of 100cm×100cm×40cm (x×y×z) as shown in Figure 62. We can now use this information to design the command interface.

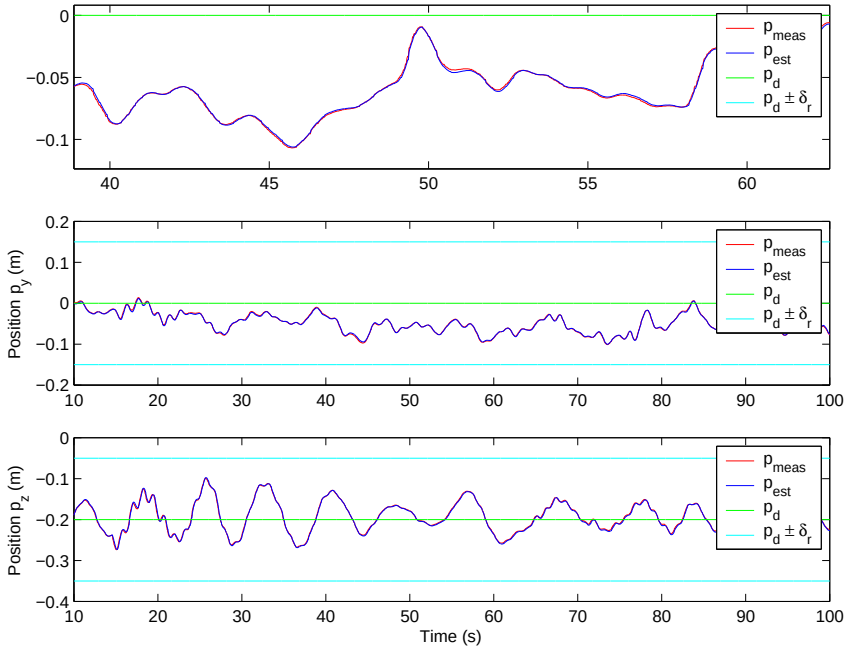


Figure 61: Determining δ_r by experimentation.

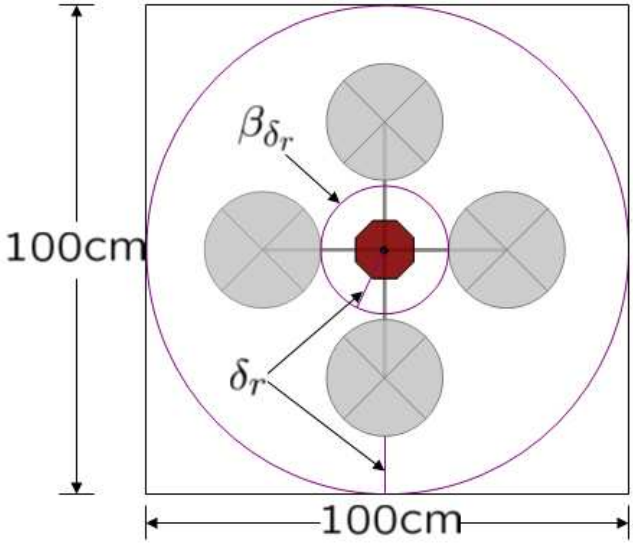


Figure 62: Operating region for helicopter in hover.

Command Interface

The command interface is a finite state machine (FSM) that accepts high-level commands $\sigma \in L$, where $L \in \{l_0, l_1, l_2, \ldots, l_6\}$, is a set of motion commands and $\mathscr{L} \in L$ is a finite sequence of motion commands extracted form the verification trajectory. The Simulink model of the command interface is shown in Figure 63. In Figure 63, d_{x1}, d_{y1},

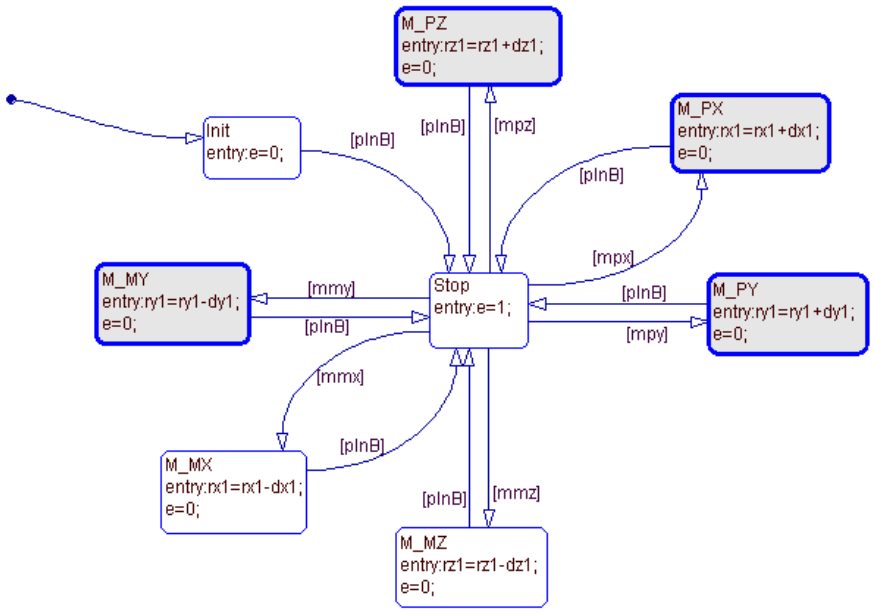


Figure 63: Finite state machine of the command interface.

and d_{z1} are the cell dimensions obtained from the hybrid automata; r_{x1}, r_{y1}, and r_{z1} are the vehicle reference coordinates sent to the real-time controller (we set $\psi_d = 0$); e is an event indicating that the vehicle has reached the desired reference point; $pinB$ is a guard condition defined by

$$pinB = \begin{cases} 1 & \text{if } \|p - p_d\| < \delta_\rho \\ 0 & \text{if } \|p - p_d\| \geq \delta_\rho \end{cases}, \tag{157}$$

where $\delta_\rho \in \mathbb{R}$ and $0 < \delta_\rho < \delta_r$. The other guard conditions mpx, mmx, mpy, mmy, mpz, and mmz are determined by the high level motion command σ as

$$mpx = \begin{cases} 1 & \text{if } \sigma = 1 \\ 0 & \text{if } \sigma \neq 1 \end{cases} \tag{158}$$

$$mmx = \begin{cases} 1 & \text{if } \sigma = 2 \\ 0 & \text{if } \sigma \neq 2 \end{cases} \tag{159}$$

$$mpy = \begin{cases} 1 & \text{if } \sigma = 3 \\ 0 & \text{if } \sigma \neq 3 \end{cases} \tag{160}$$

$$mmy = \begin{cases} 1 & \text{if } \sigma = 4 \\ 0 & \text{if } \sigma \neq 4 \end{cases} \tag{161}$$

$$mpz = \begin{cases} 1 & \text{if } \sigma = 5 \\ 0 & \text{if } \sigma \neq 5 \end{cases} \tag{162}$$

$$mmz = \begin{cases} 1 & \text{if } \sigma = 6 \\ 0 & \text{if } \sigma \neq 6 \end{cases} . \tag{163}$$

The command interface acts as a mid-level position controller, receiving commands from the mission controller and translating them into reference coordinates for the real-time controller to follow. In developing the command interface, we determined the parameters necessary to create the hybrid automata. We now need the temporal properties associated with the combined system to create the TA, in particular we need to determine the travel time range $\tau = [\tau_1, \tau_2]$ representing the minimum and maximum time for the vehicle to go from one cell region to another. In particular, we seek the time range for the vehicle to travel from anywhere in β_{δ_r}, a ball of radius δ_r centered at one cell origin to anywhere in β_{δ_ρ}, a ball of radius δ_ρ centered at an adjacent cell origin as shown in Figure 64. Again, we determine the parameters by experimentation, and after several trials we determine that the time range to travel between two adjacent cells as define above is $\tau = [1.25s,\ 4.5s]$. An example trial is shown in Figure 65. We are now ready to design the TA that will bisimulate the hybrid system. Finally, the mission controller

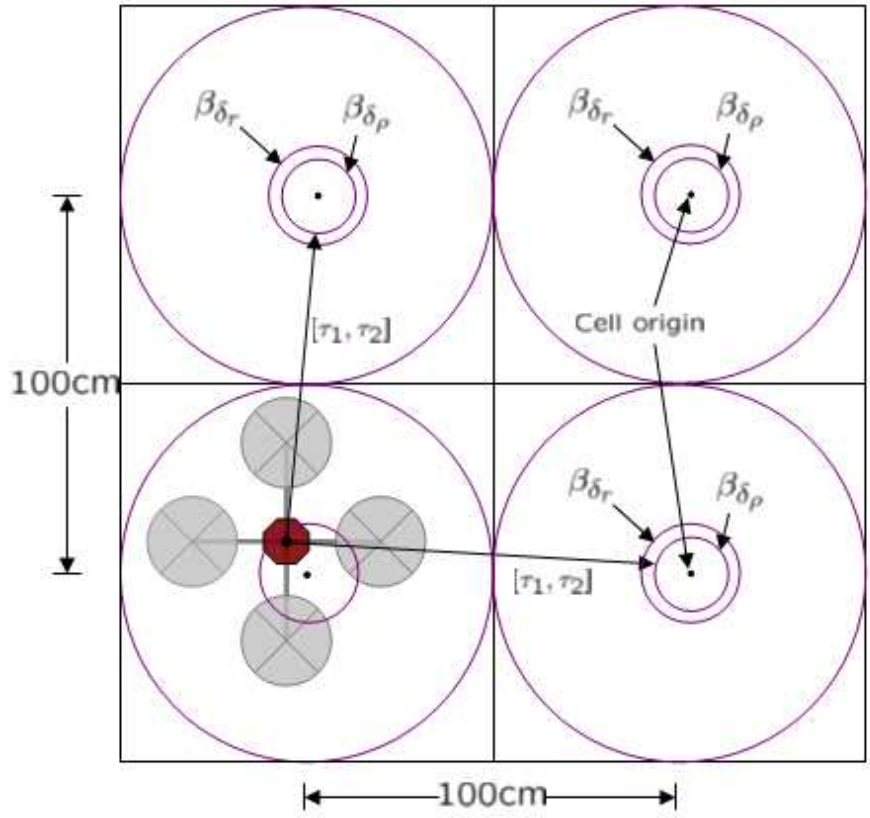


Figure 64: Vehicle cell travel time determination.

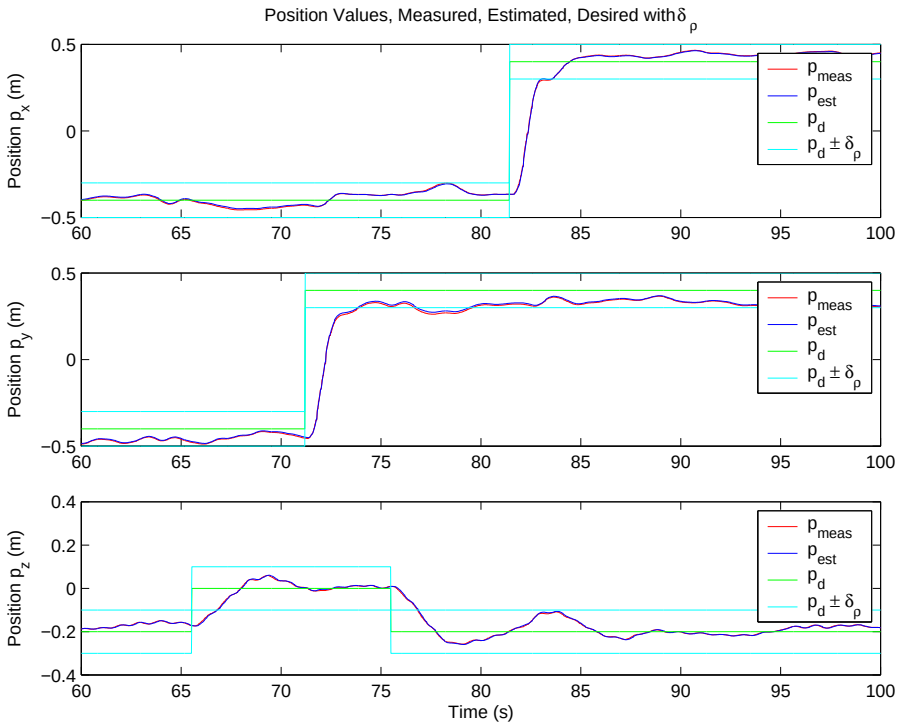


Figure 65: Vehicle cell travel time experimentation.

will be based on the trajectories generated by the verification of the TA against the CTL specifications.

Mission Controller

Based on the sequence $\mathscr{L}$ obtained from parsing the verification trace file, (explained subsequently in the TA section) we develop the mission controller, which is essentially a finite state machine as a command dispatcher that feeds the sequence of motion commands to the command interpreter. The command interpreter returns an event $e_i \in [0, 1]$ signalling that vehicle H_i has arrived at its desired location (within a ball of radius δ_{rho} centered at the desired position). Additionally, the dispatcher must synchronize the motion commands of all robots. It does this by only releasing a new motion command signal when all vehicles have reached their target locations for the current motion command iteration. A vehicle that has received a new command to stop will have already reached its desired position and will signal this to the mission controller. The mission controller receives a sequence $\mathscr{L}$ of motion commands and dispatches a motion command signal σ_i to each vehicle when all have reached their target locations, which they signal with the event e_i. The simulation model is shown in Figure 66. The mission controller has three

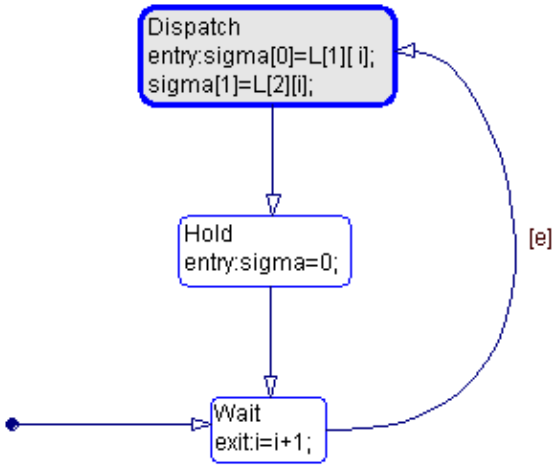


Figure 66: Mission controller finite state machine.

states for dispatching the current motion command, resetting the sigma value, and then

waiting for all vehicles to reach their target locations. The complete hybrid system consisting of the (discrete) multi-modal mission controller, command interface, closed-loop (continuous) real-time controller and vehicle system is shown in Figure 67. The signals

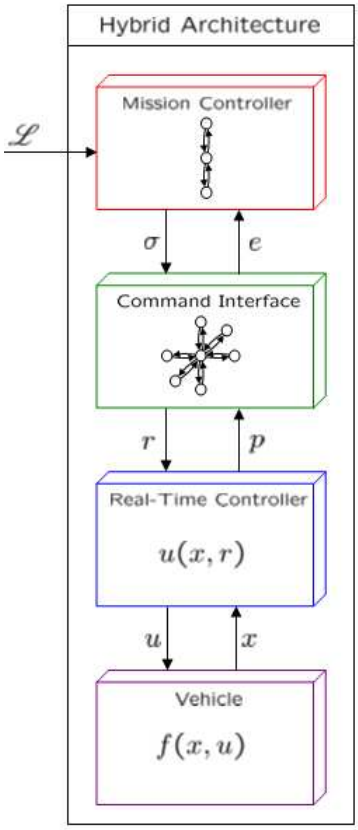


Figure 67: Multi-level hybrid architecture.

associated with the Hybrid System are given as:

- $L \in \{0, 1, 2, \ldots, 6\}$: is a set of motion control commands;
- $\mathscr{L} \subset L$: is a finite sequence of motion commands;
- $\sigma_i \in L$: is a control command for the i^{th} vehicle;
- $r_i \in \mathbb{R}^3 \times \mathbb{S}$: is the i^{th} vehicle reference value $r = [p_d\ \psi_d]^T$;
- $p_i \in \mathbb{R}^3 \times \mathbb{S}$: is the i^{th} vehicle continuous position $p = [p_x\ p_y\ p_z\ \psi]^T$;
- $\delta_\rho \in \mathbb{R}$: is the radius of a ball centered at r_i
- $e_i \in \{0, 1\}$: is an event signalling the i^{th} vehicle has reached its target such that$\|p - r\| < \delta_\rho$
- $u \in [0, 255]^4$: is the radio control signal to the vehicle
- $x \in \mathbb{R}^6 \times \mathbb{S}^3 \times \mathbb{R}^6$: is the state vector for the plant model *where* $x = [p^T\ v^T\ \Theta^T\ x_m^T]^T$ $v \in \mathbb{R}^3$ is the translational velocity of the center of mass, $\Theta \in \mathbb{S}^3$ is the Euler angle vector$\Theta = [\phi\ \theta\ \psi]^T$ $x_m \in \mathbb{R}^6$ is the linear discrete-time inner model state vector

Multi-Vehicle Coordination Results

The resulting sequence $\mathscr{L}$ associated with the above query, given `H1.pinit`=$[1\ 1\ 1]^T$, `H2.pinit`=$[2\ 2\ 1]^T$, `H1.pd`=$[2\ 2\ 2]^T$, and `H2.pd`=$[1\ 1\ 0]^T$ is:

$$\mathscr{L} = \begin{bmatrix} 1\ 0\ 3\ 0\ 5\ 0\ \ldots\ 0 \\ 0\ 2\ 0\ 4\ 0\ 6\ \ldots\ 0 \end{bmatrix}$$

Therefore, helicopter 1 (`H1`) will move forward, right, down and helicopter 2 (`H2`) will move backward, left, up. The results for actual flight with two helicopters are shown in Figures 68 and 69. In the actual flight, the trajectory navigation starts at 70 seconds after an initial mode to begin the hover; this can be seen in the plot for p_z. A 2-D projection onto the $x - y$ plane is shown in Figure 70. In order to understand the significance of the verification and the validity of the bisimulation, we can look at the

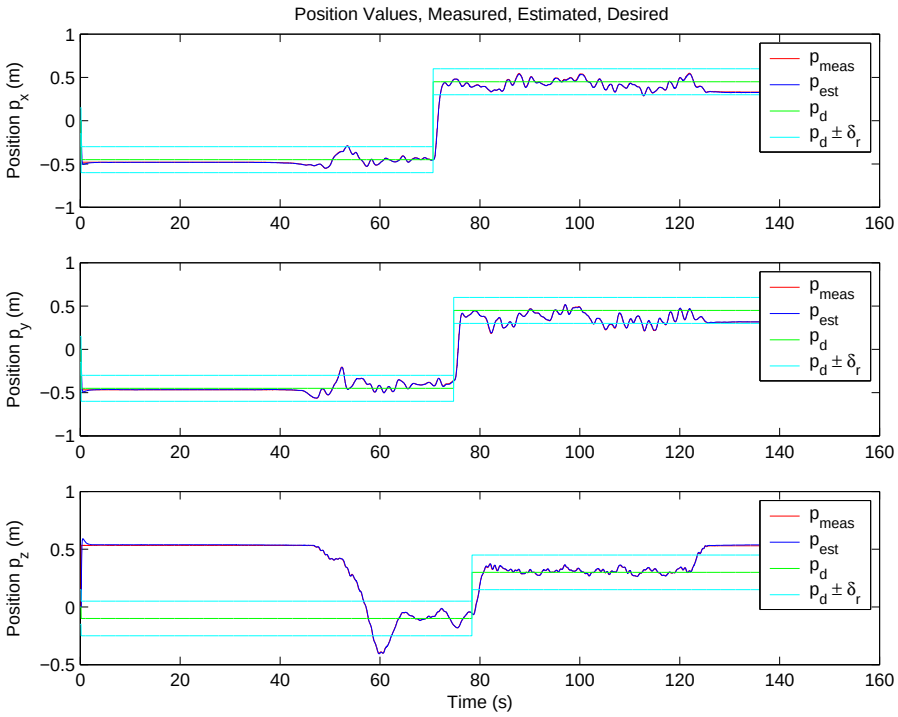


Figure 68: H1 position flight results.

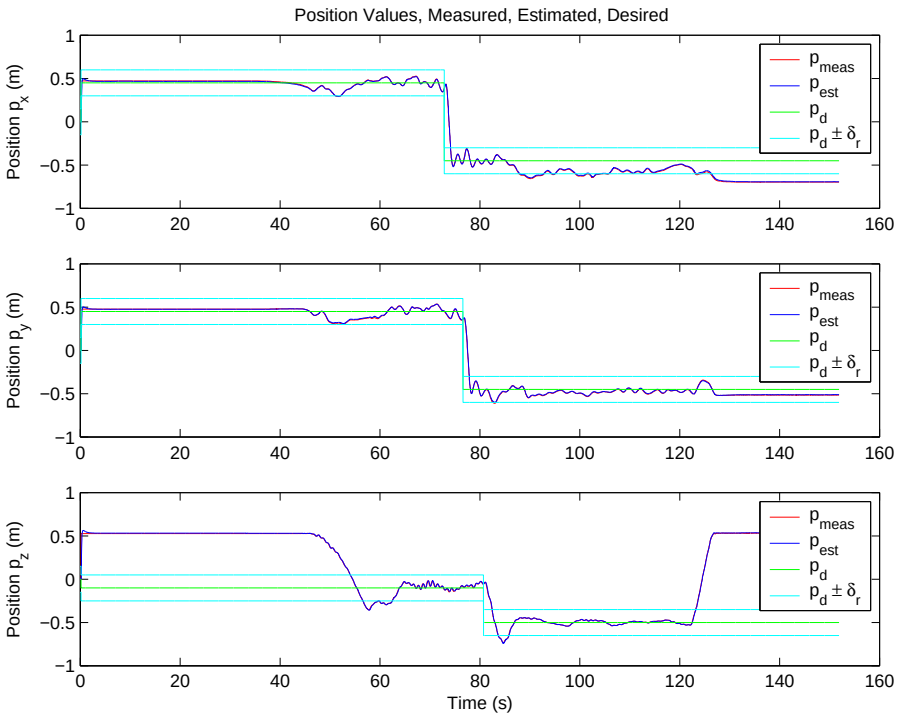


Figure 69: H2 position flight results.

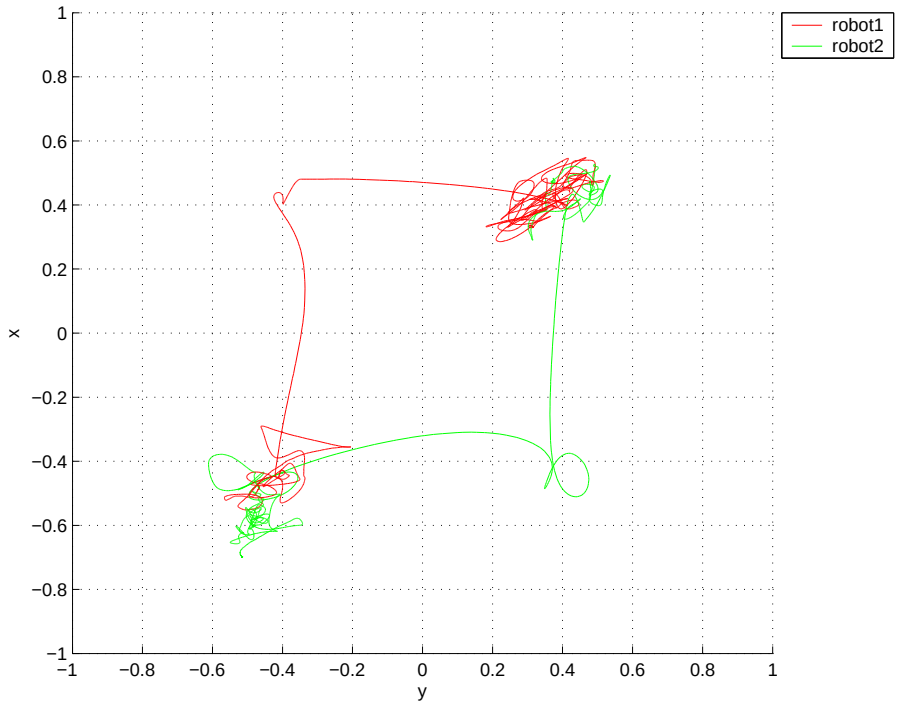
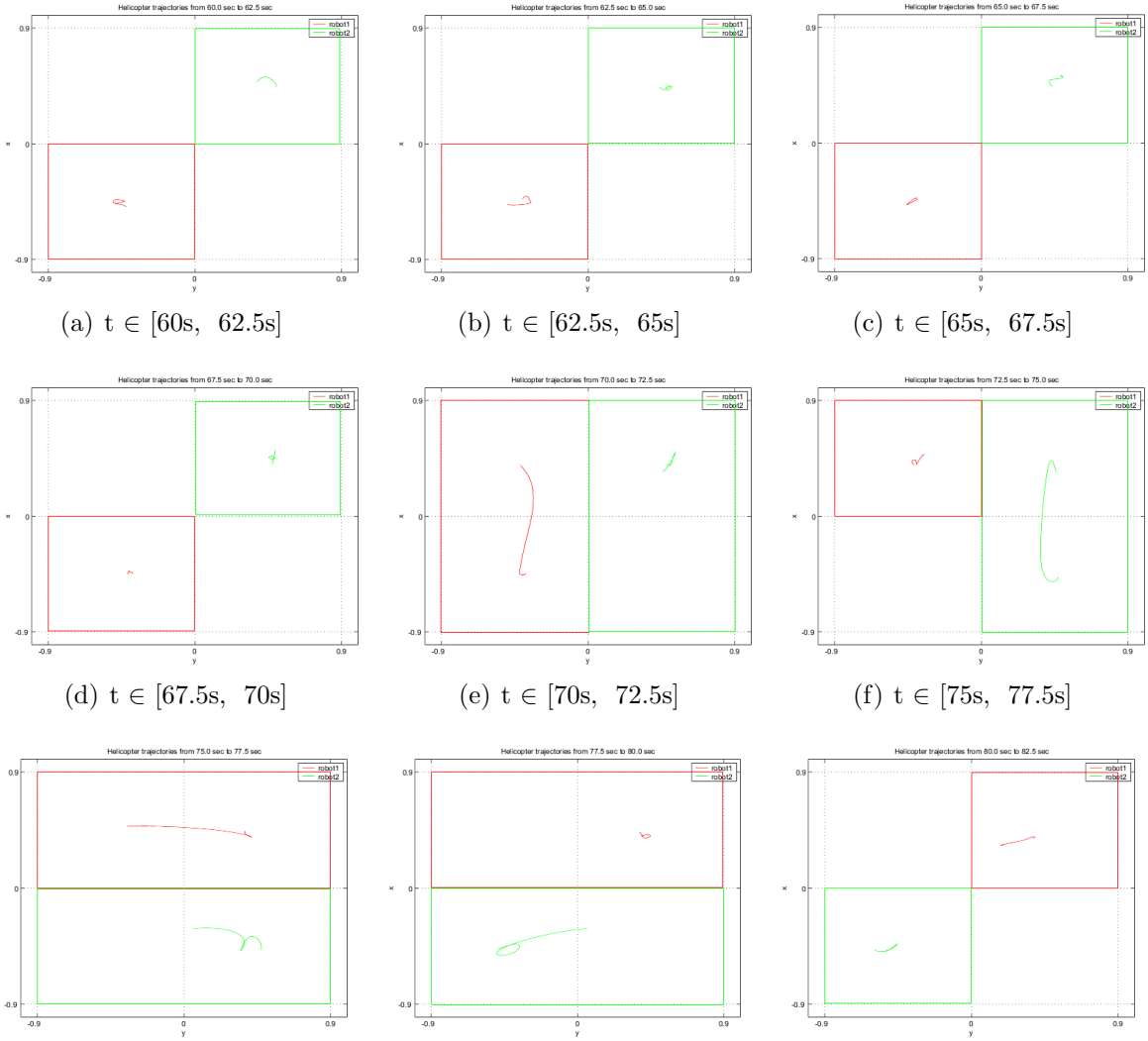


Figure 70: Projection of H1 and H2 flight position onto x-y plane.

timing that is provided in the trace file of the verification results. Because UppAal can only handle integer values for guard conditions, we have chosen to use a time range of $\tau = [1s,\ 5s]$ instead of the original $\tau = [1.5s,\ 4.5s]$. Since our new range covers the old range, the bisimulation still holds, in fact it is a more conservative approximation. Looking at the sequence and given these new times, we should see that the first transition (for both vehicles) should take place in the range $[70s,\ 75s]$, the second transition during $[72s, 80s]$, and the third transition between $[73s,\ 85s]$. Transition times (to get to the new cell) for `H1` are 72.25s, 76.15s, and 80s, and for `H1` are 74.2s, 77.9s, and 82.6s. This can be seen in Figures 68 and 69 and also in Figure 71. Thus, we see that the branching times generated from the symbolic verification of the specification accurately cover the actual times measured in real flight, demonstrating the validity and power of this approach. Although this trajectory is fairly simple, as long as the model obeys the temporal properties determined during the development of the hybrid automata, a more complex trajectory involving many more vehicles can be generated and guarantees based on the specifications will be honored.



(a) t ∈ [60s, 62.5s] (b) t ∈ [62.5s, 65s] (c) t ∈ [65s, 67.5s]

(d) t ∈ [67.5s, 70s] (e) t ∈ [70s, 72.5s] (f) t ∈ [75s, 77.5s]

(g) t ∈ [77.5s, 80s] (h) t ∈ [80s, 82.5s] (i) t ∈ [82.5s, 85s]

Figure 71: Progression of flight trajectory and acceptable cell occupation.

CHAPTER V

VANDERBILT EMBEDDED COMPUTATION PLATFORM FOR AUTONOMOUS VEHICLES

Real-time systems must react to events in the environment within precise time specifications. A large number of systems rely on computer control, in fact real-time computing is critical in areas such as chemical and nuclear plant control, automotive systems, telecommunications and flight control systems [4]. Despite this diverse application domain, real-time computing is often misunderstood, and design and development methodologies tend to be either ad hoc or based on some heuristic approach. Often, control programs are written as one large piece of assembly code, with customized timers, device drivers, and interrupt priorities. Although programs produced using this approaches may run efficiently, there are several disadvantages to this method including

- **Laborious programming.** Trying to design and develop high level control programs using assembly can be onerous and result in code inefficiency.
- **Code illegibility.** The resulting code is often unreadable and incomprehensible to anyone but the original programmers.
- **Distribution and maintenance problems.** Errors can be introduced when distributing the code manually and maintenance often proves tedious and difficult.
- **Time constraint verification.** Without formal hardware and software analysis methods, the verification of time constraints becomes extremely difficult.

The end result is that control programs can become unpredictable, and appear to work at the outset, while failure may be lurking around the corner.

The ECSL group is currently doing work on the coordination and control of multiple autonomous aerial vehicles, specifically small RC helicopters. Controlling a helicopter in a confined environment is a time-critical control challenge. Stable control requires a system that is capable of performing reliably in real-time. Unlike ground-based mobile robots, a helicopter cannot stop and wait for late or missing control commands as this

would prove catastrophic. In an effort to eliminate some of pitfalls associated with an ad hoc design, we have designed the VECPAV platform to enable the rapid development and deployment of autonomous aerial vehicles. There are two major components to the architecture, an integrated HW/SW system for rapid and code generation and distribution, and a hybrid-system/model-based design methodology for verification and generation of vehicle coordination. Both components are highly automated, reducing the burden on programmers and hastening the pace of design, development and implementation. The hybrid systems/model-based design for development and verification was detailed in the Chapter IV. In this chapter we give an overview of the integrated programming environment and system infrastructure that enables a fast, flexible and reliable design process. The process is highlighted in Figure 72. After the design has been completed, we can

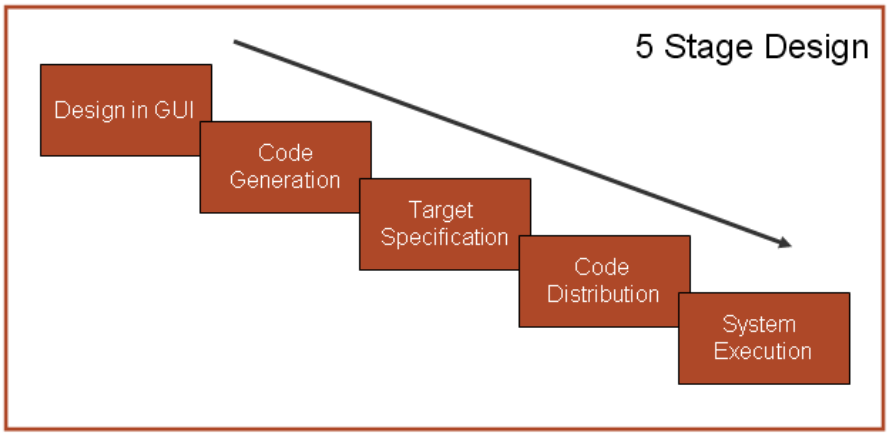


Figure 72: VECPAV automated design flow.

compile, assign targets, load and execute code automatically with just the click of a mouse.

Highly Automated Code Development and Distribution

As the backbone of our systems integration and code development we use the RT-Lab system from Opal-RT Technologies. Opal-RT Technologies provides a complete range of Hardware-in-the-Loop (HIL) simulation services and products - from distributed real-time technologies to turnkey engineering simulators - specializing in applications where plant model fidelity and fault tolerance requirements push rapid control prototyping and

HIL testing to their limits. The system consists of software to automate the editing, compilation, distribution and execution of control code, in addition to the hardware consisting of several stations (nodes) running on the QNX real-time operating system. In addition, RT-Lab handles all the communication and synchronization between QNX nodes. This allows the developer to focus on the control design and high level system functionality and reduces the tedious job of compilation, translation, distribution, and execution to simply a few clicks of a mouse. Operations are carried out through the use of the RT-Lab main console as shown in Figure 73. From this console the developer has

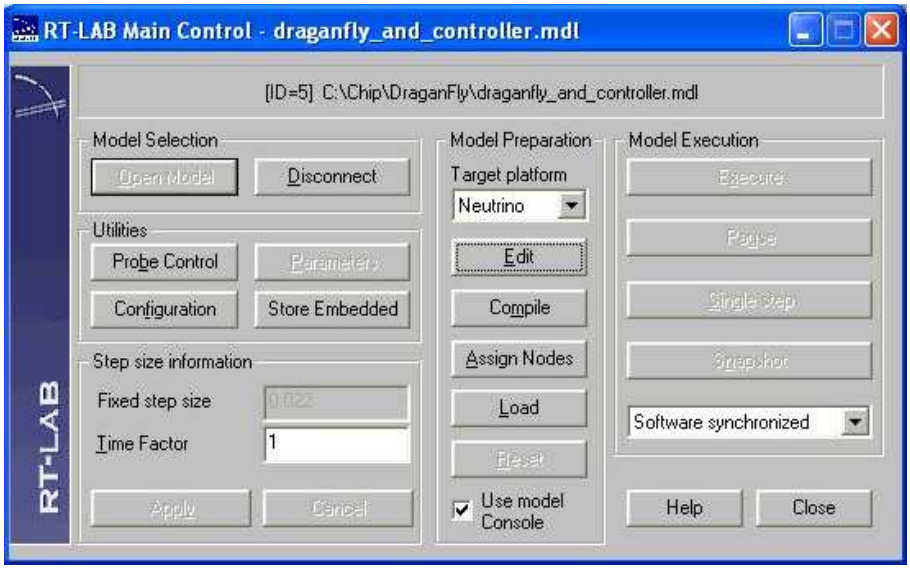


Figure 73: RT-Lab main console.

at his command the ability to edit the model in Simulink, compile, assign target nodes, distribute code, and begin control execution. All development is done in Simulink in MATLAB and there is no need to worry about C coding or low-level timing constraints. This provides for a very streamlined and rapid design. The system hardware setup is shown in Figure 74. The system is setup as follows. The draganflyer is equipped with sensors that are detected by the PTI Motion Tracker. The motion tracker continuously sends the 3-D position of all the sensors to the VZSoft/VZanalyzer programs running on a Windows machine, Boxx. These programs calculate the position and rotation of the vehicle with respect to a pre-configured reference frame and pass this information to the controllers running on the QNX machines. The QNX machines get this data, derive the

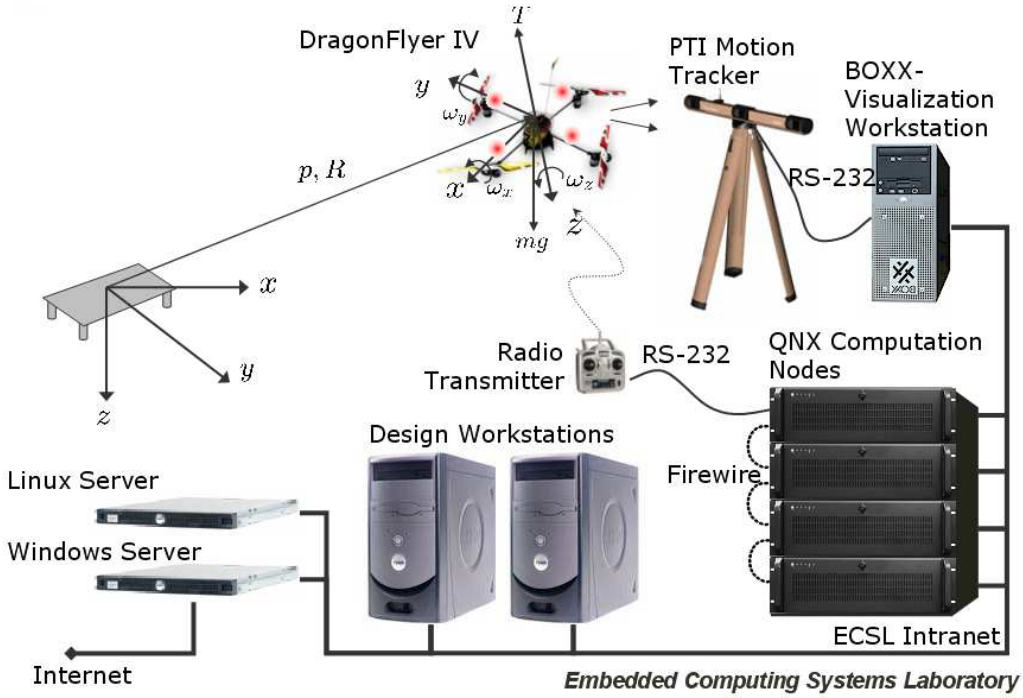


Figure 74: VECPAVs overall system setup.

appropriate correction based on the desired position and orientation, and send a signal to the radio transmitter that is controlling the helicopter. Thus the control loop is closed. We will now go into detail about each of the subsystems.

Vehicle System Setup

The vehicle that we focus on in this paper is the Draganfly IV. The Draganflyer IV can be equipped with a camera or additional sensors which make it ideal as an indoor search and rescue vehicle or for use in confined spaces where maneuverability is crucial. Due to the affordability, ease of construction and repair and durability of the vehicle, the popularity of the Draganflyer in the research and development of flight systems for small-unmanned aerial vehicles is increasing. For instance researchers at Stanford [11] have demonstrated sustained outdoor autonomous flight with a Draganflyer. A vision based stabilization and output tracking control method for the Draganflyer was presented by Altug et al. [1], a constraint model-based predictive controller for longitudinal and lateral trajectory control is derived by Cheng et al. [5] and modeling of the Draganflyer is shown by McKerrow [23].

The Draganflyer is designed as an integrated flight vehicle. Onboard electronics map the radio transmitter commands to changes in rotor speed in addition to providing additional stability functions. The remote control system consists of a 4-channel Futaba ®transmitter, using a conventional FM control signal. The Draganflyer has all the maneuverability of a normal helicopter. It solves the induced moment problem by rotating two sets of rotors clockwise and two sets counter-clockwise. Let the rotors be numbered clockwise 1 through 4, starting with the front rotor. Clockwise (counterclockwise) yaw is produced by increasing(decreasing) rotors 2,4 and decreasing(increasing) rotors 1,3. In this way the altitude remains the same because the overall thrust has not changed. Roll (pitch) is produced by varying speeds on left/right (front/back) rotors. Additional thrust can be generated by increasing all rotors equally. This is depicted in Figure 75. The led markers that perform the localization are powered by the same onboard battery

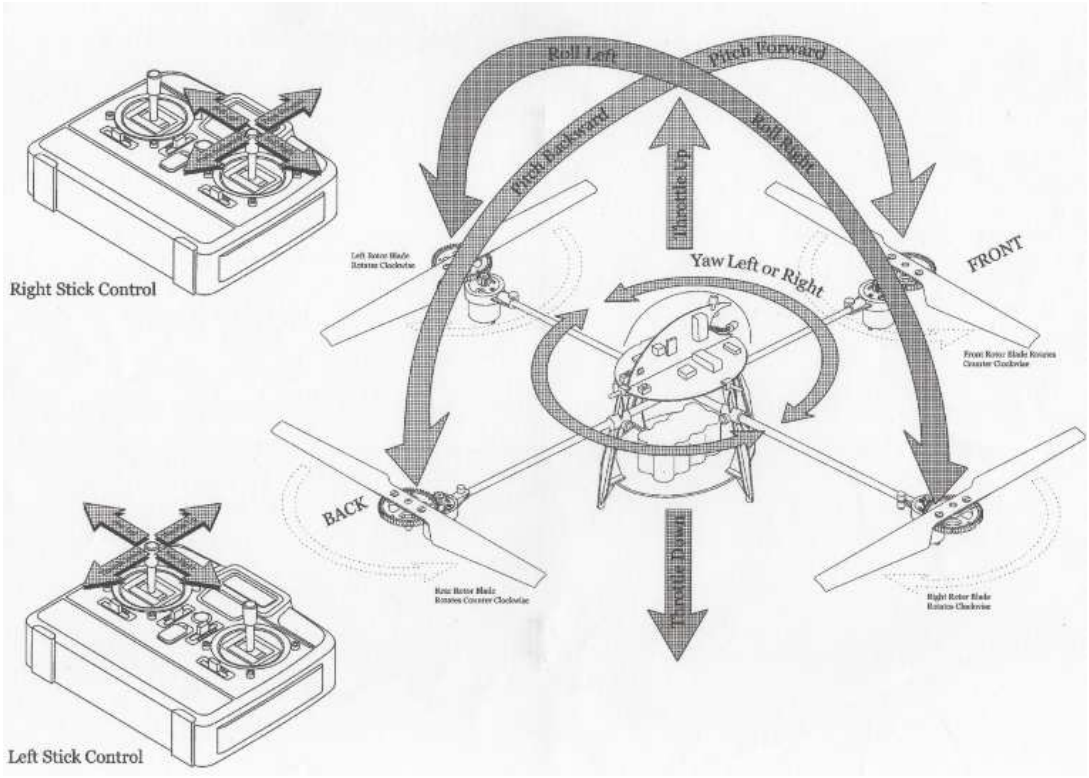


Figure 75: Draganfly motion generation.

used to power the Draganflyer. The onboard sensor system consists of the led markers, a receiver and a control module that receives commands from the PTI tracking system

when enabled through the use of the VZSoft program. The system is completely wireless and the receiver and control module fit neatly onboard next to the Draganflyer battery. The marker setup is shown in Figure 76.

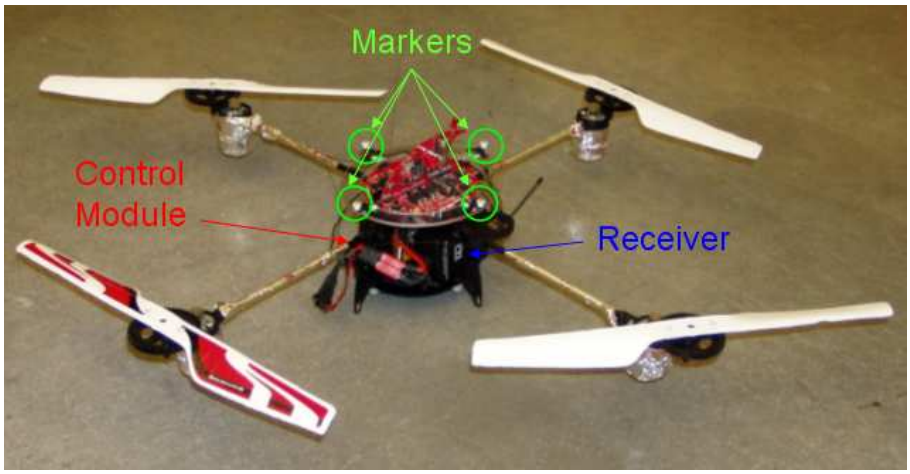


Figure 76: Draganfly marker setup.

Motion Tracking System

Localization of the helicopter vehicles is done using an active-optical motion capture system originally designed to track human motion. The PTI motion tracking system, developed by Phoenix Technologies Inc., repeatedly captures the 3D positions of led markers affixed to the helicopter. The 3D position of each marker is processed to provide position and orientation of the helicopter with an update rate of 100 Hz.

VZSoft

There are two user interfaces provided as part of the PTI system. The first is VZSoft, in which the operator can set the world coordinate frame and enable individual markers for tracking. VZSoft also permits the real-time capture of motion data for later playback and analysis. The user screen is shown in Figure 77. In Figure 77 in the left window pane, we can see the two sets of markers associated with two helicopters. In order to get the 3-axis rotation information, we need at least three markers attached to the helicopter.

Figure 77: PTI VZSoft graphical user interface.

In our case we use four markers, aligned with the struts of the helicopter as can be seen in Figure 77.

VZAnalyzer

The second user interface is the VZAnalyzer program, in which a rigid body representation is created by selecting markers that are currently enabled. In addition to the 3-axis rotation, we also obtain the position of the center of the rigid body. The VZAnalyzer user screen is shown in Figure 78. Visible in the user window are the coordinate reference frames assigned to each rigid body and also the world coordinate frame (all in North-East-Down configuration). Rotational units are in degrees and position coordinates are in millimeters. From this screen the user can also zoom in, zoom out, pan and rotate to change the viewing perspective. In addition, multiple rigid bodies can be created which allows us to easily scale the project to include many vehicles. The motion tracking and display is done in real-time and therefore movements of the helicopter can be seen simultaneously on the user screen.

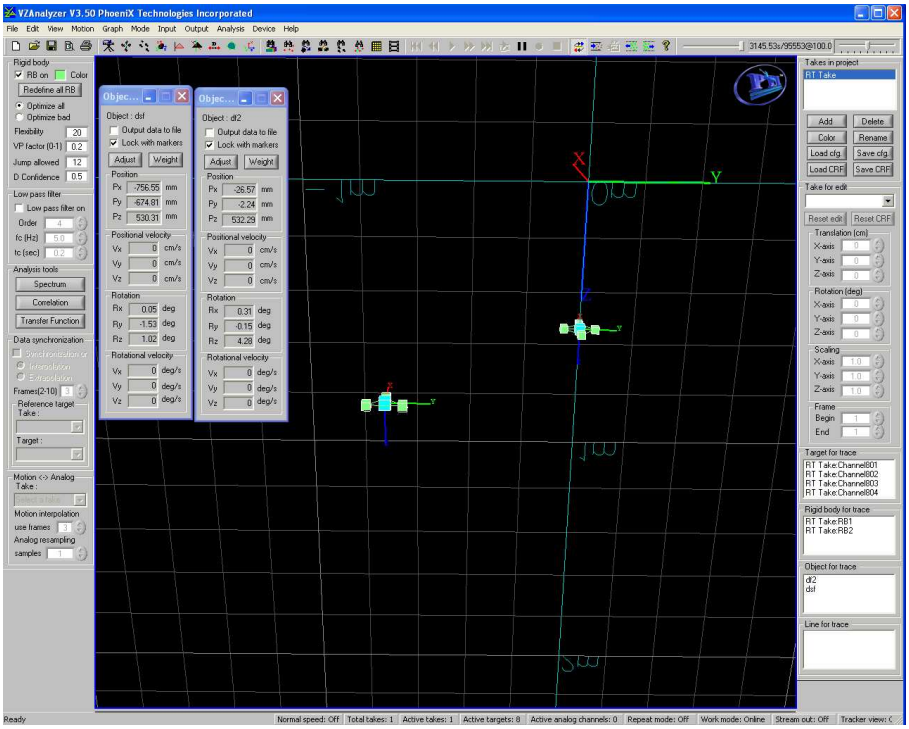


Figure 78: PTI VZAnalyzer graphical user interface.

Matlab Control Design

Our control design is done in Simulink. In order to obtain the rigid body data in Matlab from VZAnalyzer we use an S-function. This provides an array containing position and rotation values for as many objects have been created in VZAnalyzer. The top-level Simulink model for a two-controller implementation is shown in Figure 79. Since we have two helicopters, the model consists of two controller system blocks, `SM_Control` and `SS_Control_2`, two sending blocks `SS_Send1` and `SS_Send2`, and the user console `SC_Control`. At execution time, the control and sending blocks reside on the QNX nodes, while the console block runs on a windows machine (BOXX) running the PTI motion tracking software. Target node assignment is left to the user. After connecting to the model in Figure 79 via the RT-Lab main console, we can compile, distribute, and execute the real-time code. The deployment is shown in Figure 80.

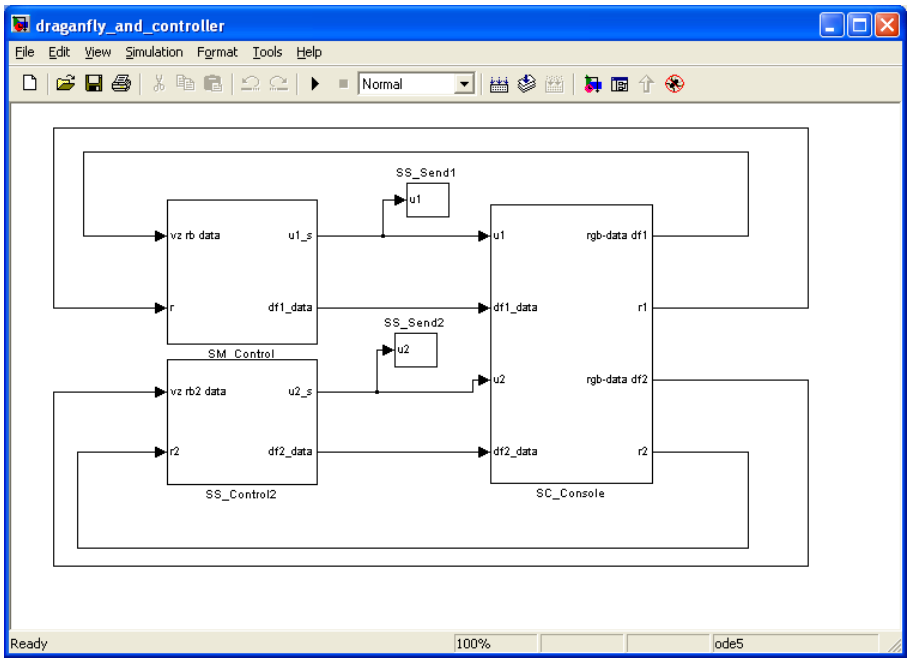


Figure 79: Top level controller Simulink model.

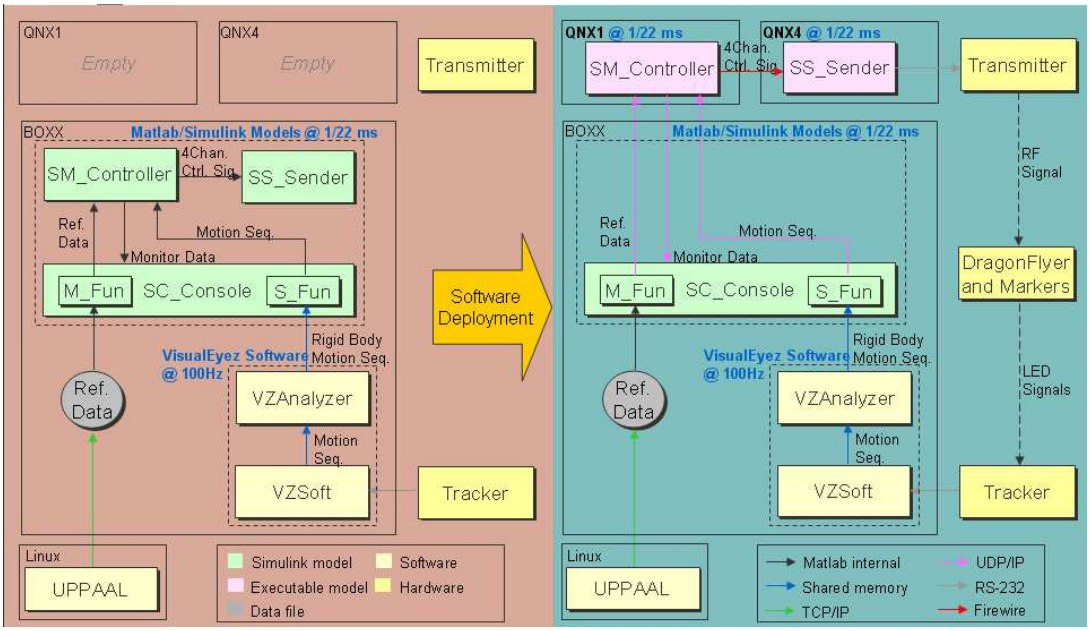


Figure 80: Code deployment stage.

CHAPTER VI

CONCLUSION

Conclusions

In this thesis, a method for the coordination and control of multiple autonomous robots is presented using a hybrid systems based design and a model-based approach. It has been shown that motion commands based on a set of specifications can be generated for the real-time system. Furthermore, these motion commands have been verified with a model checker to meet certain specifications by using a timed automata to represent the real-time system thus guaranteeing that the specifications will hold. We design the system hierarchically using information about the top and bottom level in order to "meet-in-the-middle." At the lowest level, we obtain a discrete-time linear model by capturing input-output data and performing system identification and an Extended Kalman Filter (EKF) to generate estimates of the states for filtering and feedback control purposes. At the highest level we design a timed automaton that can be checked for certain safety and performance specifications. Connecting these levels is a hybrid automata that represents the discrete modes and continuous dynamics of the real-time system and is bisimular to the timed automata. Therefore we can guarantee that the if the specifications hold for the timed automata, they will hold for our hybrid system.

Summary of Contributions

1. We demonstrated that we can represent a real-time control application by modeling it as a hybrid system. By proper construction, there exists a bisimular timed automata by which we guarantee certain real-time performance specifications.

2. We demonstrate the use of combined linear and nonlinear controllers using the results of a system identification process and a Kalman Filter for state estimates.

3. We show how an Extended Kalman Filter can be constructed to estimate the states of a system whose model is described by both nonlinear continuous-time and linear discrete-time linear dynamic equations.

4. We demonstrate the model based design for bridging the gap between high-level mission objectives often specified in a formal language and low level dynamic system whose behavior is often described by continuous-time dynamic equations.

5. We construct and demonstrate a method for the rapid development and deployment of embedded system designs.

Future Work

There are many ways in which this research could be extended and improvements on the project made.

1. The project can be extended to include swarms of vehicles flying in a virtual environment with obstacles and boundaries. Complex mission objectives such as formation configuration and mass vehicle transit could be incorporated in the mission specification. It will be interesting to see how the system performs with the number of vehicles scaled up.

2. The generation of motion commands is currently produced via the verification process offline and then transferred to the system at run-time. This should be automated so that commands can be generated in real time given a changing environment and the occurrence of unforseen events. This will however, involve the development of a more complex model to represent the physical environment.

3. Although the controller we have developed performs adequately for current operations, it is possible that more robust and effective controller could be developed using modern control theories. As of this writing, frequent calibration of angle offsets and trim values is needed, or the the helicopter exhibits a slight steady state position offset. This could be addresses using a model-based predictive controller

to adjust to changes in the environment and changes in the vehicle dynamics as the helicopter is flying.

APPENDIX A

SYMBOLS

$m(kg)$	: Total mass of the robot	
$g(Nm/s^2)$	: Acceleration due to gravity, $g = 9.81m/s^2$	
p(m)	: Position* of c.g., $p = [p_x\ p_y\ p_z]^T$	$p \in \mathbb{R}^3$
$v(m/s)$	: Velocity* of c.g., $v = [v_x\ v_y\ v_z]^T$	$v \in \mathbb{R}^3$
$\Theta(rad)$	: Orientation* (Euler angles), $\Theta = [\phi\ \theta\ \psi]^T$	$\Theta \in \mathbb{S}^3$
$\phi(rad)$	: Euler roll angle	$\phi \in \mathbb{S}$
$\theta(rad)$	: Euler pitch angle	$\theta \in \mathbb{S}$
$\psi(rad)$	: Euler yaw angle	$\psi \in \mathbb{S}$
$\omega^b(rad/s)$	: Body angular velocity $\omega = [\omega_x^b\ \omega_y^b\ \omega_z^b]^T$	$\omega^b \in \mathbb{R}^3$
$T(N)$	: Thrust acting along body z-axis	$T \in \mathbb{R}$
$R(\Theta)$	: Rotation matrix	$R \in SO(3)$
$\Psi(\Theta)$	: Mapping function	$\Psi : \mathbb{S}^3 \rightarrow \mathbb{R}^{3\times3}$
x_m	: Σ_I state vector	
	$x_m[k+1] = Ax_m[k] + B\bar{u}_I[k]$	$x_m \in \mathbb{R}^{n_m}$
u_I	: Tx input $u_I = [u_1\ u_2\ u_3\ u_4]^T$	$u \in [0, 255]^4$
$u_1(byte)$	: Radio Tx ailerons signal (roll)	$u_1 \in [0, 255]$
$u_2(byte)$	: Radio Tx elevator signal (pitch)	$u_2 \in [0, 255]$
$u_3(byte)$	: Radio Tx throttle signal (throttle)	$u_3 \in [0, 255]$
$u_4(byte)$	: Radio Tx rudder signal (yaw)	$u_4 \in [0, 255]$
Σ_I	: Inner System	
Σ_O	: Outer System	
$\bar{u}_I$	: Σ_I input $\bar{u}_I = (u_I - u_{scale})/u_{offset}$	$\bar{u}_I \in [-1, 1]^4$
$\bar{y}_m$	: Σ_I output $\bar{y}_m = Cx_m = [\bar{y}_{m1}\ \bar{y}_{m2}\ \bar{y}_{m3}\ \bar{y}_{m4}]^T$	$\bar{y}_m \in [-1, 1]^4$
y_I	: Σ_I output $y_I = [\omega^T T]^T$	$y_I \in \mathbb{R}^4$
*	: relative to inertial frame	

x_O	: Σ_O state $x_O = [p^T v^T \Theta^T x_m^T]^T$	$x_O \in \mathbb{R}^6 \times \mathbb{S}^3 \times \mathbb{R}^n$
u_O	: Σ_O input $u_O = [p_d^T \psi_d]^T$	$u_O \in \mathbb{R}^3 \times \mathbb{S}$
y_O	: Σ_O output $y_O = [p^T \psi]^T$	$y_O \in \mathbb{R}^3 \times \mathbb{S}$
n_m	: Inner system model order	
$T_s(s)$	: Fixed sample time interval	$T_s \in \mathbb{R}$
EKF Variables		
z	: EKF measurement vector $z = [p^T \Theta^T]^T$	$z_F \in \mathbb{R}^3 \times \mathbb{S}^3$
$x(k)$	: Augmented State at time $t = t_k$, $x = [x_O^T x_I^T]^t$	$x \in \mathbb{R}^6 \times \mathbb{S}^3 \times \mathbb{R}^{n_m}$
$\hat{x}(k\|k)$	: Estimate of $x(k)$ at time $t = t_k$	$x \in \mathbb{R}^6 \times \mathbb{S}^3 \times \mathbb{R}^{n_m}$
$f_O(x, u)$	: Outer system dynamics	$f_O \in \mathbb{R}^6 \times \mathbb{S}^3 \times \mathbb{R}^{n_m}$
$h_O(x)$	: Outer system measurement	$h_O \in \mathbb{R}^3 \times \mathbb{S}$
F_{x_O}	: Jacobian (w.r.t x) from f_O	$F_x \in \mathbb{R}^{9\times9}$
F_{u_O}	: Jacobian (w.r.t u) from f_O	$F_u \in \mathbb{R}^{9\times4}$
H_{x_O}	: Jacobian (w.r.t x) from h_O	$H_x \in \mathbb{R}^{6\times9}$
H_{u_O}	: Jacobian (w.r.t u) from h_O	$H_u \in \mathbb{R}^{6\times6}$
$\Phi(k+1, k)$	: Parameter for augmented state	$\Phi \in \mathbb{R}^{(9+n_m)\times(9+n_m)}$
$\Psi(k+1, k)$	: Parameter for augmented state	$\Psi \in \mathbb{R}^{(9+n_m)\times4}$
$\Phi_O(k+1, k)$	: Parameter for outer system	$\Phi_O \in \mathbb{R}^{9\times9}$
$\Psi_O(k+1, k)$	: Parameter for outer system	$\Psi_O \in \mathbb{R}^{9\times4}$
$R(k+1)$	: Measurement error covariance	$R \in \mathbb{R}^{6\times6}$
$Q(k+1)$	: System noise covariance	$Q \in \mathbb{R}^{(9+n_m)\times(9+n_m)}$

APPENDIX B

SYSTEM IDENTIFICATION SPECIFICS

System Identification Variables

The variables used during system identification are shown in Table B.

Table B.1: Variables Used in the System Identification

System ID Variables			
Data Acquisition	$u \in [0, 255]^4$ $p \in \mathbb{R}^3$ $\Theta \in \mathbb{S}^3$ $\Delta \in \mathbb{R}$ $T_f \in \mathbb{R}_+$	$u = [u_1\ u_2\ u_3\ u_4^T$ $p = [p_x\ p_y\ p_z]^T$ $\Theta = [\phi\ \theta\ \psi]^T$ $\Delta = .022s$ $t \in [0, T_f]$	Tx Channels Position Euler Angles Time Step Acq. Duration
Data Processing	$v \in \mathbb{R}^3$ $a \in \mathbb{R}^3$ $\omega^b \in \mathbb{R}^3$ $T \in \mathbb{R}$	$v = \dot{p}$ $a = \dot{v}$ $\omega^b = \Psi^{-1}(\Theta)\dot{\Theta}$ T	Spatial Velocities Spatial Acceleration Body Ang. Velocities Thrust
SID of Inner System	$u \in [0, 255]^4$ $y \in \mathbb{R}^4$ $k \in [2, K-2]$	$u = [u_1\ u_2\ u_3\ u_4]^T$ $y = [\omega_x^b\ \omega_y^b\ \omega_y^b\ T]^T$ $K = T_f/\Delta$	SID Input SID Output Step Index

In computing the trim values, we consider the following conditions:

1. Rigid body subject to body forces $f^b \in \mathbb{R}^3$ and torques $\tau^b \in \mathbb{R}^3$ applied at the center of mass.

2. The trim conditions ($x_e \in \mathbb{R}^3$, $u_e \in [0, 255]^4$) are derived when $f^b(x_e, u_e) = 0$, $\tau^b(x_e, u_e) = 0$. Using the Newton-Euler equations leads to the following criteria for

evaluating the trim conditions:

$$a \cong 0_3$$

$$\dot{\omega}^b \cong 0_3$$

$$\omega^b \cong 0_3$$

We let $B(0, \epsilon) \in \mathbb{R}^3$ be a ball of radius ϵ centered at the origin, so that for our analysis:

$$a \in B(0, \epsilon_1)$$

$$\dot{\omega}^b \in B(0, \epsilon_2)$$

$$\omega^b \in B(0, \epsilon_3)$$

where $\epsilon_1 > 0$, $\epsilon_2 > 0$ and $\epsilon_3 > 0$. The trim conditions are determined via experimentation.

Jacobian Matrices

Jacobian matrices for linearizing about the nominal state are given as

$$F_{x_o} = \begin{bmatrix} \frac{\partial \dot{p}}{\partial p} & \frac{\partial \dot{p}}{\partial v} & \frac{\partial \dot{p}}{\partial \Theta} \\ \frac{\partial \dot{v}}{\partial p} & \frac{\partial \dot{v}}{\partial v} & \frac{\partial \dot{v}}{\partial \Theta} \\ \frac{\partial \dot{\Theta}}{\partial p} & \frac{\partial \dot{\Theta}}{\partial v} & \frac{\partial \dot{\Theta}}{\partial \Theta} \end{bmatrix}$$

where

$$\frac{\partial \dot{p}}{\partial p} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad \frac{\partial \dot{p}}{\partial v} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

$$\frac{\partial \dot{p}}{\partial \Theta} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \frac{\partial \dot{v}}{\partial p} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \frac{\partial \dot{v}}{\partial v} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

$$\frac{\partial \dot{v}}{\partial \Theta} = \begin{bmatrix} \frac{-T}{m}(-s\phi s\theta c\psi + c\phi s\psi) & \frac{-T}{m}(c\phi c\theta c\psi) & \frac{-T}{m}(-c\phi s\theta s\psi + s\phi c\psi) \\ \frac{-T}{m}(-s\phi s\theta s\psi - c\phi c\psi) & \frac{-T}{m}(c\phi c\theta s\psi) & \frac{-T}{m}(c\phi s\theta c\psi + s\phi s\psi) \\ \frac{-T}{m}(-s\phi c\theta) & \frac{-T}{m}(-c\phi s\theta) & 0 \end{bmatrix},$$

$$\frac{\partial \dot{\Theta}}{\partial p} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \frac{\partial \dot{\Theta}}{\partial v} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

$$\frac{\partial \dot{\Theta}}{\partial \Theta} = \begin{bmatrix} \omega_y c\phi t\theta - \omega_z s\phi t\theta & \omega_y s\phi / c^2\theta + \omega_z c\phi / c^2\theta & 0 \\ -\omega_y s\phi - \omega_z c\phi & 0 & 0 \\ \omega_y c\phi / c\theta - \omega_z s\phi / c\theta & \omega_y s\phi t\theta / c\theta + \omega_z c\phi t\theta / c\theta & 0 \end{bmatrix},$$

and

$$F_{u_o} = \begin{bmatrix} \frac{\partial \dot{p}}{\partial \omega} & \frac{\partial \dot{p}}{\partial T} \\ \frac{\partial \dot{v}}{\partial \omega} & \frac{\partial \dot{v}}{\partial T} \\ \frac{\partial \dot{\Theta}}{\partial \omega} & \frac{\partial \dot{\Theta}}{\partial T} \end{bmatrix}$$

where

$$\frac{\partial \dot{p}}{\partial \omega} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \frac{\partial \dot{v}}{\partial \omega} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad \frac{\partial \dot{\Theta}}{\partial \omega} = \Psi(\Theta),$$

$$\frac{\partial \dot{p}}{\partial T} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \frac{\partial \dot{v}}{\partial T} \begin{bmatrix} \frac{-1}{m}(c\phi s\theta c\psi + s\phi s\psi) \\ \frac{-1}{m}(c\phi s\theta s\psi - s\phi c\psi) \\ \frac{-1}{m}(c\phi c\theta) \end{bmatrix}, \quad \frac{\partial \dot{\Theta}}{\partial T} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

and

$$H_{x_o} = \begin{bmatrix} \frac{\partial p_I}{\partial p} & \frac{\partial p_I}{\partial v} & \frac{\partial p_I}{\partial \Theta} \\ \frac{\partial \Theta}{\partial p} & \frac{\partial \Theta}{\partial v} & \frac{\partial \Theta}{\partial \Theta} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\frac{\partial \dot{v}}{\partial x_i} = \begin{bmatrix} y_{scale_4} c_{41} \frac{1}{m}(c\phi s\theta c\psi + s\phi s\theta) & \dots & y_{scale_4} c_{46} \frac{1}{m}(c\phi s\theta c\psi + s\phi s\theta) \\ y_{scale_4} c_{41} \frac{1}{m}(c\phi s\theta s\psi - s\phi c\psi) & \dots & y_{scale_4} c_{46} \frac{1}{m}(c\phi s\theta s\psi - s\phi c\psi) \\ y_{scale_4} c_{41} \frac{1}{m}(c\phi c\theta) & \dots & y_{scale_4} c_{46} \frac{1}{m}(c\phi c\theta) \end{bmatrix},$$

$$\frac{\partial \dot{\Theta}}{\partial x_m} = \begin{bmatrix} sc_1 c_{11} + sc_2 c_{21} s\phi t\theta + sc_3 c_{31} c\phi t\theta & \dots & sc_1 c_{1n} + sc_2 c_{2n} s\phi t\theta + sc_3 c_{3n} c\phi t\theta \\ sc_2 c_{21} c\phi - sc_3 c_{31} s\phi & \dots & sc_2 c_{2n} c\phi - sc_3 c_{3n} s\phi \\ sc_2 c_{21} s\phi / c\theta + sc_3 c_{31} c\phi / c\theta & \dots & sc_2 c_{2n} s\phi / c\theta + sc_3 c_{3n} c\phi / c\theta \end{bmatrix},$$

$$\frac{\partial \dot{x}_i}{\partial p} = \begin{bmatrix} 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \end{bmatrix}, \frac{\partial \dot{x}_m}{\partial v} = \begin{bmatrix} 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \end{bmatrix}$$

$$\frac{\partial \dot{x}_m}{\partial \Theta} = \begin{bmatrix} 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \end{bmatrix}, \frac{\partial \dot{x}_m}{\partial x_m} = \begin{bmatrix} A_{11} & \dots & A_{1n_m} \\ \vdots & \ddots & \vdots \\ A_{n_m 1} & \dots & A_{n_m n_m} \end{bmatrix},$$

EKF Computation Steps

1. Given T, Q, R, $\hat{x}(0|0)$, $P(0|0)$

   For each k:

2. Compute F_x, F_u

3. Compute $\Phi(k+1,k)$, $\Psi(k+1,k)$

   (a) $\Phi(k+1,k) = I + F_x[\hat{x}(k|k), u^*(t_k)]T$

   (b) $\Psi(k+1,k) = F_u[\hat{x}(k|k), u^*(t_k)]T$

4. Prediction equation

   (a) $\hat{x}(k+1|k) = \{I + F_x[\hat{x}(k|k), u^*(t_k)]T\}\hat{x}(k|k) + F_u T[x(k|k, u^*(t_k)])$

5. Compute $P(k+1|k)$

   (a) $P(k+1|k) = \Phi(k+1,k)P(k|k)\Phi^T(k+1,k) + Q$

6. Compute H_x

7. Update Kalman gain matrix K

   (a) $K(k+1) = P(k+1|k)H_x^T\{H_x P(k+1|k)H_x^T + R]\}^{-1}$

8. Update error covariance matrix

(a) $P(k+1|k+1) = [I - K(k+1)H_x]P(k+1|k)$

9. Correction equation

(a) $\hat{x}(k+1|k+1) = \hat{x}(k+1|k) + K(k+1)\{z(k+1) - h[\hat{x}(k+1|k), u^*(t_{k+1})]\}$

BIBLIOGRAPHY

[1] Erdinç Altug, James P. Ostrowski, and Robert E. Mahony. Control of a quadrotor helicopter using dual camera visual feedback. *The International Journal of Robotics Research*, 24(5):329–341, 2005.

[2] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[3] Gerd Behrmann, Alexandre David, and Kim G. Larsen. A tutorial on uppaal. In *Formal Methods for the Design of Real-Time Systems*, number 3185 in Lecture Notes in Computer Science, pages 200–236. Springer Verlag, 2004.

[4] Giorgio C. Buttazzo and Giorgio Buttanzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[5] M. Cheng and M. Huzmezan. A combined mbpc/2 dof hinf controller for quad rotor unmanned air vehicle. *AIAA Atmospheric Flight Mechanics Conference and Exhibit*, pages 329–341, 2005.

[6] John J. Craig. *Introduction to Robotics: Mechanics and Control*. Addison-Wesley Publishing Company, Inc., Reading, MA, 1989.

[7] Neal P. Curtin. Gao report to the chairman, subcommittee on tactiacl air and land forces, committee on armed services, house of representatives. Technical report, Unites States General Accounting Office, March 2004.

[8] Greg Goebel. Early us target drones, http://www.vectorsite.net/twuav01.html.

[9] E. Haghverdi, P. Tabuada, and G. Pappas. Bisimulation relations for dynamical and control systems, 2002.

[10] W.P.M.H. Heemels, B. De Schutter, and A. Bemporad. Equivalence of hybrid dynamical models. *Automatica*, 37(7):1085–1091, July 2001.

[11] Gabe Hoffmann, Dev Gorur Rajnarayan, Steven L. Waslander, David Dostal, Jung Soon Jang, and Claire J. Tomlin. The stanford testbed of autonomous rotorcraft for multi agent control (starmac). In *Proceedings of the 23rd Digital Avionics Systems Conference*, November 2004.

[12] http://en.wikipedia.org/wiki/Daedalus.

[13] http://inventors.about.com/library/inventors/blda vinci.htm.

[14] http://www.aerofiles.com/wright bio.html.

[15] H.K. Khalil. *Nonlinear Systems*. Prentice Hall, New Jersey, 2002.

[16] T. J. Koo, G. J. Pappas, and S. Sastry. *Multimodal Control of Constrained Nonlinear Systems*. IEEE Press, 2002.

[17] T. John Koo and Shankar Sastry. Output tracking control design of a helicopter model based on approximate linearization. In *CDC '98: Proceedings of the 37th IEEE Conference on Decision and Control*, pages 3635–3640, December 1998.

[18] T.John Koo, Yi Ma, and Shankar S. Sastry. Nonlinear control of a helicopter based unmanned aerial vehicle model. January 2001.

[19] J. Gordon Leishman. The bréguet-richet quad-rotor helicopter of 1907. *2001 AHS International Directory*.

[20] Lennart Ljung. *Nonlinear Systems: Theory for the User*. Prentice Hall, Upper Saddle River, New Jersey, 1999.

[21] John Lygeros. Lecture notes on hybrid systems. Technical report, Department of Electrical and Computer Engineering, University of Patras, Feb. 2004.

[22] Ph. Martin, R. M. Murray, and P. Rouchon. Flat systems, equivalence and trajectory generation.

[23] Phillip J. McKerrow. Modelling the draganflyer four-rotor helicopter. In *ICRA '04: Proceedings of the 2004 IEEE International Conference on Robotics & Automation*, pages 3596–3601, April 2004.

[24] Jerry M. Mendel. *Lessons in Estimation Theory for Signal Processing, Communications, and Control*. Prentice-Hall, Upper Saddle River, NJ, 1995.

[25] Richard M. Murray, S. Shankar Sastry, and Li Zexiang. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Inc., Boca Raton, FL, USA, 1994.

[26] Office of the Secretary of Defense. Unmanned aerial vehicle roadmap. Technical report, Department of Defense, December.

[27] M.M. Quottrup, T. Bak, and R. Izadi-Zamanabadi. Multi-robot planning: A timed automata approach. In *ICRA '04: Proceedings of the 2004 IEEE International Conference on Robotics & Automation*, pages 4417–4422, April 2004.

[28] Shankar Sastry. *Nonlinear Systems: Analysis, Stability and Control*. Springer-Verlag, New York, NY, 1999.