1-22-2008

Designing an Autonomous Helicopter Testbed: From Conception Through Implementation

Richard D. Garcia
University of South Florida

Designing an Autonomous Helicopter Testbed: From Conception Through Implementation

by

Richard D. Garcia

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: Kimon Valavanis, Ph.D.
Abraham Kandel, Ph.D.
Steven Wilkerson, Ph.D.
MaryAnne Fields, Ph.D.
Wilfrido A. Moreno, Ph.D.
Miguel Labrador, Ph.D.
Les A. Piegl, Ph.D.

Date of Approval:
January 22, 2008

Keywords: mobile robots, aerospace control, helicopter control, helicopter reliability, modular computer systems

Dedication

I would like to dedicate this work to my family and friends who have shown an enormous level of enduring support for my endeavors. I would specifically like to thank Laura Barnes for her friendship and endless dedication throughout our paralleling struggle. Above all, I would like to thank my mother who has always supported my dreams and whose lasting dedication has always given me purpose and strength…thank you mom.

Acknowledgments

This research has been partially supported by a fellowship through the Oak Ridge Institute for Science and Education (ORISE), ONR Grants N00014-03-01-786 and N00014-04-10-487, SPAWAR, and the DOT through the USF CUTR Grant 2117-1054-02. I would personally like to thank the robotics group of the Army Research Lab at Aberdeen Proving Ground including Dr. Steven Wilkerson, Dr. MaryAnne Fields, Robert Hayes, Jim Spangler, and Harris Edge.

Note to Reader

The original of this document contains color which may be necessary for understanding certain figures. The original dissertation is on file with the USF library in Tampa, Florida.

Table of Contents

List of Tables

List of Figures

Designing an Autonomous Helicopter Testbed:
From Conception Through Implementation
Richard D. Garcia
ABSTRACT

Miniature Unmanned Aerial Vehicles (UAVs) are currently being researched for a wide range of tasks, including search and rescue, surveillance, reconnaissance, traffic monitoring, fire detection, pipe and electrical line inspection, and border patrol to name only a few of the application domains. Although small / miniature UAVs, including both Vertical Takeoff and Landing (VTOL) vehicles and small helicopters, have shown great potential in both civilian and military domains, including research and development, integration, prototyping, and field testing, these unmanned systems / vehicles are limited to only a handful of university labs. For VTOL type aircraft the number is less than fifteen worldwide! This lack of development is due to both the extensive time and cost required to design, integrate and test a fully operational prototype as well as the shortcomings of published materials to fully describe how to design and build a "complete" and "operational" prototype system.

This dissertation overcomes existing barriers and limitations by describing and presenting in great detail every technical aspect of designing and integrating a small UAV helicopter including the on-board navigation controller, capable of fully autonomous takeoff, waypoint navigation, and landing. The presented research goes beyond previous works by designing the system as a testbed vehicle. This design aims to provide a general framework that will not only allow researchers the ability to supplement the system with new technologies but will also allow researchers to add innovation to the vehicle itself. Examples include modification or replacement of controllers, updated filtering and fusion techniques, addition or replacement of sensors, vision algorithms, Operating Systems (OS) changes or replacements, and platform modification or replacement. This is supported by the testbed's design to not only adhere to the technology it currently utilizes but to be general enough to adhere to a multitude of technology that have yet to be tested.

This research will allow labs without the proper expertise to build a safe and reliable vehicle that can provide them access to real world data thus increasing the effectiveness and validity of their research. It will also allow researchers working in simulation to quickly enter into UAV development without utilizing thousands of man hours to create an unmanned vehicle.

The presented research is designed to benefit the entire UAV researching community by allowing in depth access to an area of research that has been typically classified as too expensive and too time consuming to enter.

Chapter 1
Introduction

The field of robotics, for the past few decades, has not only allowed humans to expand their abilities, it has allowed them to surpass them. This is possible through the shift from teleoperated robotics, robotic movement determined and directly controlled by a human operator, to fully autonomous robotics designed to remove or reduce the need for the human component. The ability of an autonomous system to surpass the capabilities of a human operator are obvious when one considers the limitations of the human body in locomotive speed and decision making as compared to the operating speed of the newest computer systems and the speed and precision of direct current motors.

More recently, humans have seen the capabilities of robotics expand to the flight control of both commercial and military vehicles. This shift has even begun to include highly agile and low cost Miniature Unmanned Aerial Vehicles (MUAVs). These MUAVs not only make it feasible for low budget research of control but expanded the flight package of these vehicles by allowing for inverted flight, metronomes, stall turns, flips, and rolls.

1.1 Why Helicopters?

Since the first known flight of a rotary wing vehicle in 1907 [1, 2], there has been a consistent drive to enhance this vehicle's capabilities. Although the ideology of the helicopter can be traced back to 4th century China [3], it was not until the early 1900s that the idea of a vertical takeoff and landing vehicle could be materialized. Even with Paul Cornu's first vertical flight in 1907 it would take another 46 years and the invention of the turbine engine to make these vehicles useful. In 1961 the first unmanned VTOL vehicle, DASH, was designed and utilized for naval defense [4]. Although largely consider a failure, the DASH system did open the way for the development and implementation of many more systems. Since 1961 many unmanned VTOL vehicles, including ducted fans, tilt rotors, and rotary wing vehicles, have been developed in an

attempt to broaden their overall usage. To date, the most utilized of these vehicles is the rotary wing vehicle, typically known as the helicopter.

A helicopter's most distinct advantage as an aircraft is its ability for vertical flight. This ability allows it to takeoff and land vertically but also to hover. This provides the vehicle access to areas typically restricted from fixed wing vehicles. This advantage became globally recognized with the introduction of the Bell UH-1, commonly referred to as the Heuy, into the Vietnam War. The mountainous terrain of Vietnam and constant need for troop deployment and extraction in the field made setting up and maintaining a runway for fixed wing vehicles almost impossible. The Heuy allowed for troop deployment and extraction, supply delivery, medical evacuation, and air support without requiring a landing area and could takeoff and land in manner of seconds.

Helicopters, although highly agile and highly developed, are not without their disadvantages. Helicopters are highly non-linear and heavily coupled aircrafts [5-9]. Changes in any distinct control output will have an effect on all controlled areas of the vehicle. For example, an alteration in heading will typically either add to or subtract from the amount of torque used by the main rotor. This will alter the overall head speed and thus affect the vertical thrust provided by the main rotor. Depending on the state of the vehicle, this could affect any or all of the required collective, elevator, aileron, and throttle inputs. Helicopters also have far lower airspeeds and far shorter ranges than fixed wing vehicles [10, 11]. This coupled with the harsh vibrations typical to rotary vehicles and the difficulty in safely and effectively piloting a helicopter reduced the helicopter's overall appeal. Even with the disadvantages of the helicopter it is one the safest and most effective means of transporting people and equipment to heavily congested and undeveloped areas.

With the development of MUAVs, mostly for Radio Control (RC) hobbyist, came the ability to study and simulate the effects and dynamics of a helicopter on a smaller more cost efficient scale. As these RC MUAVs became more and more popular the designs of the vehicle became more and more developed. The designers of these "toys" began to take advantage of the platform's size, compactability, and speed. Soon after, these vehicles began to be recognized for their high payload to weight ratios, maneuverability, and safety. Currently there exist unmanned RC helicopters ranging from hundreds of pounds in weight and costing hundreds of thousands of dollars [12, 13] to weighting grams costing only tens of dollars [14].

Today these RC helicopters are utilized for much more than weekend enjoyment. Today dozens of research facilities throughout the world are attempting to harness the abilities of these

low cost, highly developed, extremely agile machines. This is possible, in part, due to recent advancements in Micro-Electro-Mechanical Systems (MEMS) technology allowing for the miniaturization of many sensors and processing systems [15] used to automate the flight of VTOL vehicles. However, as of this writing less than fifteen laboratories worldwide have successfully managed to develop and implement a fully autonomous miniature helicopter. Although these vehicles have shown great advancements in the areas of controller design, filtering and fusion algorithms, mechanical design, software development, and hardware implementation; their abilities are still heavily underutilized as discussed further in Chapter 2.

1.2 Problem Statement

The problem this dissertation addresses is as follows:

Miniature unmanned vehicles are becoming popular due to their compact size, high maneuverability and high size-to-payload ratio. This is especially true with Vertical Takeoff and Landing (VTOL) vehicles due to their distinct capabilities to maneuver in any direction and to hover, even in highly confined areas. These abilities have led to research discussing the possible roles these vehicles may have in search and rescue, surveillance, traffic monitoring, fire detection, pipe and electrical line inspection, and border patrol to name only a few. Although the research has become increasingly popular, the testing of new and innovative ideas has typically been constrained to simulation and mathematical proofs. Although these testing methods are crucial when designing safe and efficient ideas, they are far from complete and many times do not hold under real world situations.

There are several limiting factors preventing testing and development on a testbed vehicle:

- *Lack of expertise:* Development of a testbed helicopter requires expertise in many areas including software design, electronic design, mechanical design, software and hardware integration, controller design, filter and fusion design, and safety piloting. Many labs and institutions do not have all these areas of expertise available and thus must seek outside help for development.
- *Lack of resources:* The time associated with selecting processing hardware, sensors, platform, and an operating system coupled with designing controllers, filter and fusion

techniques, and a software architecture along with integration of software, hardware, and platform elements are extensive.

- *Proprietary knowledge:* Commercial systems lack even the most basic flexibility. Commercial systems utilize proprietary software and hardware which severely cripples their development abilities.
- *Highly specialized systems:* Academically owned and developed systems are typically designed for in-house use and are typically tightly integrated. This type of design creates a system that is heavily specialized and only applicable to a very narrow field of development. The tight integration also prevents ideas and technologies implemented on these vehicles from being individually extracted and tested on other systems.
- *Lack of intellectual integration:* Current published works are typically incomplete and sometimes inconsistent. This is most likely due to the large number of experts required to build a system. Each expert is typically concerned with their particular work and typically glazes over the rest of the vehicle's design and any integration issues. It is also typical to see individual expert's publications span different versions of the system. Thus, the controllers described by a controls expert may have been implemented on a later version of the hardware described by the electronics expert's publication. This coupled with many expert's desires or requirements to protect the specifics of their work has prevented many institutions from duplicating published vehicles [16].

The overall motivation for this research is based on the challenge to build a complete autonomous helicopter system based on sound foundational theory that can be easily adapted for a multitude of research purposes. The objective of this dissertation is to provide the knowledge necessary to design and implement a safe and reliable UAV helicopter testbed. This work provides complete detail for selecting a platform, processing system, and sensors along with integration of the hardware onto the platform as well as controller development, data filtering and fusion, OS selection, and software integration. Great effort is placed on developing the system for easy integration of new sensors, processing hardware, software, or platforms.

Note that this dissertation focuses specifically on miniature helicopters whose characteristics typically include a small foot print, low endurance, and minimal payload capacity. Although the focus is on miniature helicopters the validity of the information is not limited to these platforms and is applicable to other vehicles including ground and full-size vehicles.

1.3 Contributions

The proposed solution presented in this dissertation is the detailed design and implementation of a miniature helicopter testbed capable of autonomous takeoff, waypoint navigation, and landing as well as describing the rationale for developmental decisions and possible alternatives.

This work benefits all areas of research involving UAV vehicles including control development, mechanical design, system integration, data mining, artificial intelligence, and vision processing by allowing researchers access to a developmental testbed that is designed to advance and reaffirm their work. Major contributions include:

- *Enabling the design and implementation of an autonomous testbed helicopter without the need for experts spanning multiple disciplines:* This is accomplished by providing all of the details from conception through implementation of a testbed helicopter including hardware selection, software design, integration and testing.
- *Alleviating the amount of time required to design and implement a testbed:* This dissertation not only provides the implementation and design of the testbed it also provides the specific hardware used to implement the information provided along with the software design and pseudocode for data acquisition, filter and fusion, and control.
- *Providing a system that utilizes zero proprietary information:* From conception this work was designed to be easily reproduced without "inside information". To adhere to this standard great effort is made to utilize as much Commercial off the Shelf (COTS) hardware as deemed possible. To increase the ease of reproduction, this dissertation provides diagrams for all non-COTS hardware. Software pseudo code is also provided along with detailed integration information.
- *Providing a design approach that facilitates a broad scope of development:* The work described in this document is designed to provide development opportunities for all areas of unmanned vehicles. This is done by utilizing hardware and software that is designed for easy modification, replacement, or removal without requiring redesign of the system.

- *Removing the possibility of information loss due to integration of multiple experts' works:* Although many individuals influenced this work the testbed described in this dissertation was conceptualized, designed, implemented, and tested by one individual person. This allows for an in-depth view of the entire testbed without information loss due to the collaboration of multiple experts.

1.4 Organization of Dissertation

The remainder of this dissertation is organized as follows: Chapter Two provides related work in the area of autonomous unmanned helicopters including both commercial and open source systems. Chapter Three presents detailed descriptions and justifications for the types of hardware utilized on the USF helicopter testbed including the platform, processing hardware, mounting hardware, and sensors. Chapter Three also details the hardware assembly of the Unmanned Systems Lab (USL) testbed. Chapter Four describes and justifies the software architecture utilized on the testbed including the OS, data acquisition, and data distribution techniques as well as the specific software processes. Chapter Five details the algorithms used to calculate state data and convert control outputs to Pulse Width (PW) values. Chapter Six is dedicated to detailing the controllers used on the USL testbed. Chapter Seven details the simulator utilized for initial testing of both the controllers and multiple calculation algorithms. Chapter Eight discusses both the testing and performance of the implemented testbed. Chapter Nine summarizes the dissertation and list several directions of future work.

It should be mentioned that the organization of this dissertation is designed to express the natural development process taken when developing and building the UAV testbed. It was designed to provide the reader with the basic steps necessary for development an implementation in an order that will reduce the overall number of pitfalls. This includes selecting a platform, selecting and configuring the onboard hardware, basic software selection and setup, algorithm development and implementation, control, and testing.

Chapter 2
Related Works

There is currently great interest in the area of UAV research including search and rescue [17-20], surveillance [21-26], traffic monitoring [27-30], fire detection [31-34], pipe and electrical line inspection [35, 36], border patrol [37, 38], and failure tolerance [39] to name only a few. Although UAVs have shown great potential in many of these areas their development as a whole has been somewhat slow. This can be attributed to the difficulty in purchasing, replicating, or developing an autonomous vehicle to validate new ideas.

2.1 Commercial UAVs

Although there exist many commercial UAV helicopters today, the work discussed here focuses on miniature UAV vehicles typically classified as class I & II by Future Combat Systems (FCS) [40]. Table 1 briefly describes a list of commercial UAVs classified as types III & IV by FCS and is only mentioned to support the claim as to the interest currently involved in UAV VTOL research.

One the most popular commercial MUAV designers today is Rotomotion. Rotomotion utilizes proprietary software and hardware to produce a strap-on autonomous control system. This control system can be purchased with a wide variety of vehicles, see Table 2, and varies from tens of thousands to hundreds of thousands of U.S. dollars for a fully autonomous vehicle. Rotomotion's strap-on control system can be mounted to any rotary vehicle capable of lifting the required equipment (1.25 lb control system, 7.2V battery, 4.8V battery, and mounting hardware). The flight controller is a self contained system and does not require a ground station or differential Global Positioning System (GPS) corrections. The control system can typically hold the vehicle's position within one meter of the commanded position [41].

Table 1: Commercial UAVs (Type III/IV)

Manufacture	Vehicle	Specifications
Bell Helicopter [42]	Eagle Eye	• Transitional vehicle (VTOL to fixed wing) • Speeds up to 200 knots • Range of 800 nautical miles • Altitude of up to 20,000 ft • Up to 3 hours of flight time
Sikorsky's [43, 44]	Cypher II	• Rotary vehicle with ability to add wings • Speeds up to 126 knots • Range of 100 nautical miles • Up to 3 hours of flight time
Northrop Grumman [45]	Firescout (MQ-8B)	• Helicopter vehicle • Speeds up to 125 knots • Range of 110 nautical miles • Altitude of up to 20,000 ft • Up to 8 hours of flight time
SAIC/ATI [46]	Vigilante (502)	• Helicopter vehicle • Speeds up to 117 knots • Range of 110 nautical miles • Altitude of up to 13,000 ft • Up to 9 hours of flight time
Bombardier [47]	CL-327 Guardian	• Counter rotating VTOL • Speeds up to 85 knots • Range of 108 nautical miles • Altitude of up to 18,000 ft • Up to 6.25 hours of flight time
Boeing [48, 49]	A160 Hummingbird	• Helicopter vehicle • Speeds up to 140 knots • Range of 1700 nautical miles • Altitude of up to 30,000 ft • Over 24 hours of flight time

Although the Rotomotion control system allows for a COTS autonomous helicopter it has several drawbacks. First, as of this writing the system cannot takeoff or land autonomously. This requires that a specially trained pilot be available for all flight tests. Second, the system utilizes proprietary software that must be configured for each individual platform. Thus each vehicle, including vehicles of the exact same model, must be setup by a Rotomotion employee making the system anything but modular. Last, the system utilizes proprietary hardware rendering the system useless for any type of software development.

Table 2: Rotomotion's COTS UAVs

Platform	Power Plant	Dry Weight	Dimensions	Cost (USD)
SR200 [50]	Gasoline 2-Stroke	55 lbs	110x30x34 in	90000**
SR100 [51]	Gasoline, Diesel, Alcohol and Electric	35 lbs	58x20x27 in	35000**
SR20 [52]	Electric	16.5 lbs	48x15x22 in	16500**
Bergen Observer*	Gasoline 2-Stroke	25 lbs	52x15x23 in	35,000
Bergen Twin*	Gasoline 2-Stroke	18 lbs	60x20x26 in	25,000

* - Not currently in production
** - Base cost

Another popular commercial producer of autonomous helicopter systems is Neural Robotics Inc. (NRI). NRI uses proprietary neural-network control software that provides stability in both hover and flight [53]. NRI's control system also utilizes real-time adaptation to the effects of wind and weight. NRI offers three platforms complete with autonomous hardware (see Table 3). Each vehicle contains a flight control system consisting of a PC/104 computer stack, Attitude and Heading Reference System (AHRS), GPS, heading-hold gyroscope, as well as proprietary avionics command/control components [54].

Although NRI's vehicles utilize state of the art software and hardware, as of this writing they are neither fully autonomous nor useful as developmental testbeds. All of the vehicles offered by NRI require input from an operator to perform any type of maneuver classifying this system more as semi-autonomous. The system also lacks the ability to be modified, adapted, or integrated by the end user. This severely diminishes the usability of the system as a testbed for most types of research.

To date there is not a commercially available autonomous VTOL vehicle that can be classified as a testbed. This comes as no surprise as it would be difficult for any business to sustain a profit on a vehicle that is entirely open source.

Table 3: NRI's COTS UAVs

Platform	Power Plant	Dry Weight	Dimensions	Cost (USD)
Explorer	Turbine	30 lbs	83 in	35000*
Express G	Gasoline	15 lbs	60 in	19800*
Express E	Electric	15 lbs	60 in	16700*

*Cost without ground station

2.2 Open Source UAVs

There currently exist several open source, RC based, autonomous helicopters. These helicopters are typically built and maintained by academic societies or government sponsored organizations and most notably include the Massachusetts Institute of Technology (MIT), Carnegie Mellon University (CMU), Stanford, Georgia Tech, University of Southern California, Berkeley, and CSIRO (see Table 4). These vehicles range in size, propulsion, payload capacity, and endurance but were all developed with one goal in mind: researching and advancing the field of autonomous helicopters. Collectively, these helicopters have shown great ability to hover [55, 56], navigate [57-60], takeoff [61], land [61-64], and track objects [34, 65-67], and have shown limited abilities to hover inverted [68], barrel roll [9, 69, 70], flip [9, 69], funnel [9, 55], stall turn [69], and pirouette [71] utilizing methods that include Proportional Integral Differential (PID), Fuzzy, H-infinity, Neural Network (NN), and Linear-Quadratic Regulator (LQR) controllers.

The most notable of all the previously mentioned autonomous helicopters is the Georgia Tech GTMax, see Figure 1. Georgia Tech's Software Enabled Control (SEC) group, as well as Berkeley, Carnegie Mellon, and various government organizations, utilize a Yamaha RMax platform, a 2-stroke horizontally opposed 246cc engine mounted to a 3.63 m long frame [13]. Georgia Tech's RMax, along with a custom developed on-board system, has shown the ability to perform waypoint navigation, autonomous takeoff and landing [72], and failure detection and correction [72]. The entire on-board system, summarized in Table 4 and detailed in [72], is powered by the RMax's on-board generator [73]. Control of the helicopter is handled by both Georgia Tech's Neural Network (NN) controllers and the proprietary Yamaha Attitude Control System (YACS).

Figure 1: Georgia Tech's Testbed Helicopter (GTMax) [74]

Table 4: Open Source UAV Helicopter Testbeds

University	Hardware and Sensors	Software
Massachusetts Institute of Technology [75-77]	• X-Cell 60 Helicopter • ISIS-IMU (100Hz and 0.02 deg/min drift) • Honeywell HPB200A Altimeter (2ft accuracy) • Superstar GPS (1 Hz)	• QNX Operating System • 13-state extended Kalman filter (state estimation) • LQR based control
Carnegie Mellon University [77-80]	• Yamaha R-Max Helicopter • Litton LN-200 IMU (400 Hz) • Novatel RT-2 DGPS (2cm accuracy) • KVH-100 flux-gate compass (5 Hz) • Yamaha laser altimeter	• VxWorks Operating System • 13-state extended Kalman filter (state estimation) • Control based on PD and H_∞ control
Stanford University [9, 55]	• XCell Tempest 91 Helicopter • Microstrain 3DM-GX1 (100Hz) • Novatel RT-2 DGPS (2cm accuracy) • DragonFly2 cameras (position est.)	• Undisclosed Operating System • 12-state extended Kalman filter (state estimation) • Differential Dynamic Programming (DDP) • Extension of Linear Quadratic Regulator (LQR)
Georgia Institute of Technology [72, 73, 81]	• Yamaha R-50 Helicopter • ISIS-IMU (100Hz and 0.02 deg/min drift) • Novatel RT-2 DGPS with 2cm accuracy • Radar and Sonar Altimeters • HMR-2300 triaxial magnetometers	• OS: QNX, VxWorks, Linux • Real-time CORBA • Object Request Broker (ORB) Architecture • 17-state extended Kalman filter (state estimation) • Neural networks control (Feedback linearization)
University of California Berkeley [71, 82]	• Yamaha R-Max & Maxi Joker • Boeing DQI-NP INS/GPS system • Novatel Millen RT-2 DGPS (2cm accuracy)	• VxWorks Operating System • No state estimation (provided by sensor) • Reinforcement Learning control
University of Southern California [83, 84]	• Bergen twin Industrial Helicopter • ISIS IMU (100Hz and 0.02 deg/min drift) • Novatel RT-2 DGPS (2cm accuracy) • TCM2-50 triaxial magnetometer • Laser altimeter (10 cm accuracy @ 10 Hz)	• Linux Operating System • 16-state Kalman filter (state estimation) • Decoupled PID based control
CSIRO [77, 85]	• X-Cell 60 Helicopter • Custom embedded IMU with compass (76 Hz) • Ublox GPS with WAAS (2m accuracy) • Stereo vision for height estimation	• LynxOS Operating System • Velocity estimation using vision • Two 7-state extended Kalman filters • Complimentary filters • PID based control
JPL [86, 87]	• Bergen Industrial Helicopter • NovAtel OEM4 DGPS (2cm accuracy) • ISIS IMU • MDL ILM200A laser altimeter • TCM2 compass	• QNX real-time OS • Behavior-based and H_∞ control • Extended Kalman filter (state estimation) • Image-based motion estimates

Although the RMax platform is highly advanced and well utilized it is not without issues. First, the RMax platform is nine feet long from tail to nose, without the blades attached, and weighs approximately 140 lbs. The shear size and weight of the vehicle and hardware limit its transportation and makes efficient deployment very difficult. Second, the RMax platform with the GPS option and a 100 meter flight ceiling has a price tag of approximately $240,000 USD. This is without the custom flight control system, Inertial Measurement Unit (IMU), ground radar, vision system, and support equipment. Last, the RMax platform does not contain an autorotation clutch. This device, installed on all modern full-size rotary aircraft, allows the platform to maneuver in the event of an engine failure. It is fairly trivial for a trained pilot to safely fly and land a VTOL vehicle containing an autorotation clutch that has had an engine failure. This is a serious safety issue and should be considered when utilizing a VTOL vehicle without an autorotation clutch.

Of all the autonomous helicopters in existence today, the GTMax is the only fully developed testbed vehicle. The vehicle was developed to ensure that multiple organizations could utilize the hardware and software for development purposes. The development of the GTMax also follows many of the ideologies that this work follows. Although similar in concept the design aspects are quite different. The GTMax has almost and order of magnitude greater payload capacity, weights an order of magnitude more than the USL testbed, and has over an order of magnitude greater cost. As such the GTMax was developed with little concern for its components size, weight, and power requirements and inherently has greater stability in flight.

Of all the UAV helicopters currently being used, Stanford's X-Cell Tempest is the only one attempting to utilize the extreme aerobatic maneuvers capable of small RC helicopters. The X-Cell Tempest has shown the ability to perform multiple flips, rolls, and nose-in and nose-out funnels [9, 71], Figure 2. To perform these maneuvers, the X-Cell Tempest utilizes the on-board system, described in Table 4, two ground mounted cameras and an off-board computer. Control is performed through multiple, on-board and off-board, Kalman filters and reinforcement learning techniques that attempt to learn an LQR task through Differential Dynamic Programming (DDP). This entire system allows the vehicle to estimate its position within 25cm and control the vehicle at a rate of 10Hz.

Figure 2: Mosaic of Stanford X-Cell Tempest Performing an Autonomous Flip [88]

Another notable autonomous unmanned helicopter is USC's Autonomous Vehicle Aerial Tracking and Reconnaissance (AVATAR) vehicle. The AVATAR's platform is a Bergen Industrial Twin helicopter utilizing a 46cc twin cylinder two cycle engine with a 10 kg payload capacity. This helicopter system is of particular interest due to its almost exclusive utilization for UAV vision research and development. The AVATAR vehicle has been used to implement visual servoing [89], visual identification of objects [65], vision based autonomous landings [62, 63], and target tracking [65, 89]. Aside from the vision algorithm development, the AVATAR has been used in the deployment of marsupial robots [90], autonomous deployment and repair of sensor networks [57], and PID control.

Although several other autonomous helicopter systems exist, see Table 4, their variations are minimal and none can be classified as purely open source testbed systems. As such, to further discuss them would be heavily repetitive and inconsequential to this work. It should be noted that all of these open source UAV helicopters, with the exception of the GTMax, appear to be developed with a specific goal in mind. This factor severely limits the scope of research obtainable through the use of these vehicles and in many cases has required the development of multiple vehicles each dedicated to an area of research.

Although the advancements in MUAV helicopters are impressive they do not begin to fully utilize the capabilities of these small agile vehicles which are capable of tick-tocs, autorotation (both normal and inverted), death spirals, funnels/tornadoes, and pirouetting flips [91]. These types of maneuvers can prove useful when attempting to avoid or evade obstacles, land without power, or simply track a highly dynamic target.

2.3 Summary

Although prior and ongoing research has shown the enormous benefits and interest in the development of MUAVs the migration of these ideas from paper to realization has been greatly limited. To realize the full benefit of these vehicles and to alleviate the gap between innovative ideas and innovative technologies there must be a medium for development and testing. This is only possible though the full disclosure of information and is the backbone of academic research.

Chapter 3
Platform & Hardware

Hardware is the building block of all unmanned vehicles and a great source of difficulty when designing a testbed vehicle. Decisions made about hardware can significantly decrease or increase the complexity and functionality of an unmanned system. For these reasons great effort is taken to effectively describe and justify all hardware, interconnections, and mounts utilized on the proposed autonomous helicopter testbed. For completeness this work also details the assembly of the vehicle to assure that the work can be effectively reproduced. Note that any hardware mentioned in this text that does not provide its specific model can be referenced in Appendix A for details.

3.1 Platform

Platform selection is crucial for developing an autonomous helicopter that fits both the testbed's current needs and future desires. There exist several variations of miniature helicopters available including turbine, electric, methanol, and gasoline. Methanol based systems are by far the cheapest versions of RC helicopter on the market but require special engine tuning, constant availability of methanol fuel, and expel a large amount oil from the exhaust. Gasoline helicopters typically have the longest runtimes for their size, 30-90 minutes, and high payload capacities, approximately 10kg, but require the operator to store gasoline and require special tuning of the carburetor. Turbine based helicopters have the highest payload to weight ratios but are the highest priced platforms available and require the purchasing and storage of jet fuel. Last, electric platforms only utilize batteries and thus do not require the handling or storage of the hazardous fuel. Electric platforms also have reduced vibrations due to the use of an electric motor. The drawbacks are that electric vehicles have only average run times and average payload capacities, approximately 15-20 minutes and 4-5 kg respectively. Note that the statistics provided above are for comparison and are based on similar sized vehicles, 90 to 120 size RC vehicles.

The platform chosen for the USL autonomous helicopter testbed is the electric Maxi-Joker II, Figure 3. The Maxi-Joker II helicopter has the following characteristics:

- Manufacturer: Joker
- Main Rotor Diameter: 1.8 meter
- Dry Weight: 4.22 kg (w/o batteries)
- Dimensions: 56x10.25x16.5 in (w/o Blades)
- Payload Capacity: 4.5 kg (after batteries)
- Endurance: 15-20 min
- Motor Battery: 37 V (10A) Lithium Polymer
- Servo Battery: 4.8 V (2.0A) NiMh
- Engine: Plettenberg HP 370/40/A2 Heli
- Speed Controller: Schulze future 40/160H

This Maxi Joker II is also equipped with a Futaba GY-401 heading hold gyro, 800mm carbon main rotor blades, 125 mm carbon fiber tail rotor blades, a Futaba S9254 tail servo, three Futaba S9250 main servos, and a Futaba R319DPS radio receiver. This platform was chosen over the previously mentioned platforms due to its cost, approximately $3000 USD ready-to-fly, desire to avoid carrying and storing explosive fuel, reduced vibrations, relatively small size, and ability to handle wind gust exceeding 20 mph.

Figure 3: Stock Joker-Maxi II Helicopter

Note that no modifications were made to the Maxi Joker II kit or any of the above mentioned equipment. The kit, motor, speed controller as well as all support equipment were assembled and setup as instructed in the manufacturer supplied manuals.

3.2 Hardware

The main hardware components of the UAV system consist of:

- Pentium M 755 2.0 GHz Processor
- G5M100-N mini-ITX motherboard
- Microstrain 3DMG-X1 IMU
- 2 Gigs of Crucial 333 MHz RAM
- Novatel Superstar 2 GPS receiver (5 Hz, 5V model)
- Microbotics Servo/Safety Controller (SSC)
- Thunderpower 11.1V 4.2Ah LiPo Battery
- Intel Pro 2200 802.11B/G Mini-PCI wireless card
- URG-04LX Hokuyo laser range finder
- IVC-200G 4channel frame grabber
- 120 Watt picoPSU-120 Power supply
- Sony FCB-EX980S module camera

A complete list of all utilized hardware is provided in Appendix A.

This configuration is used because of its high computational capabilities, various Input/Output (I/O) ports, small size, low heat emission, and cost. Figure 4 depicts the overall concept for the on-board processing system.

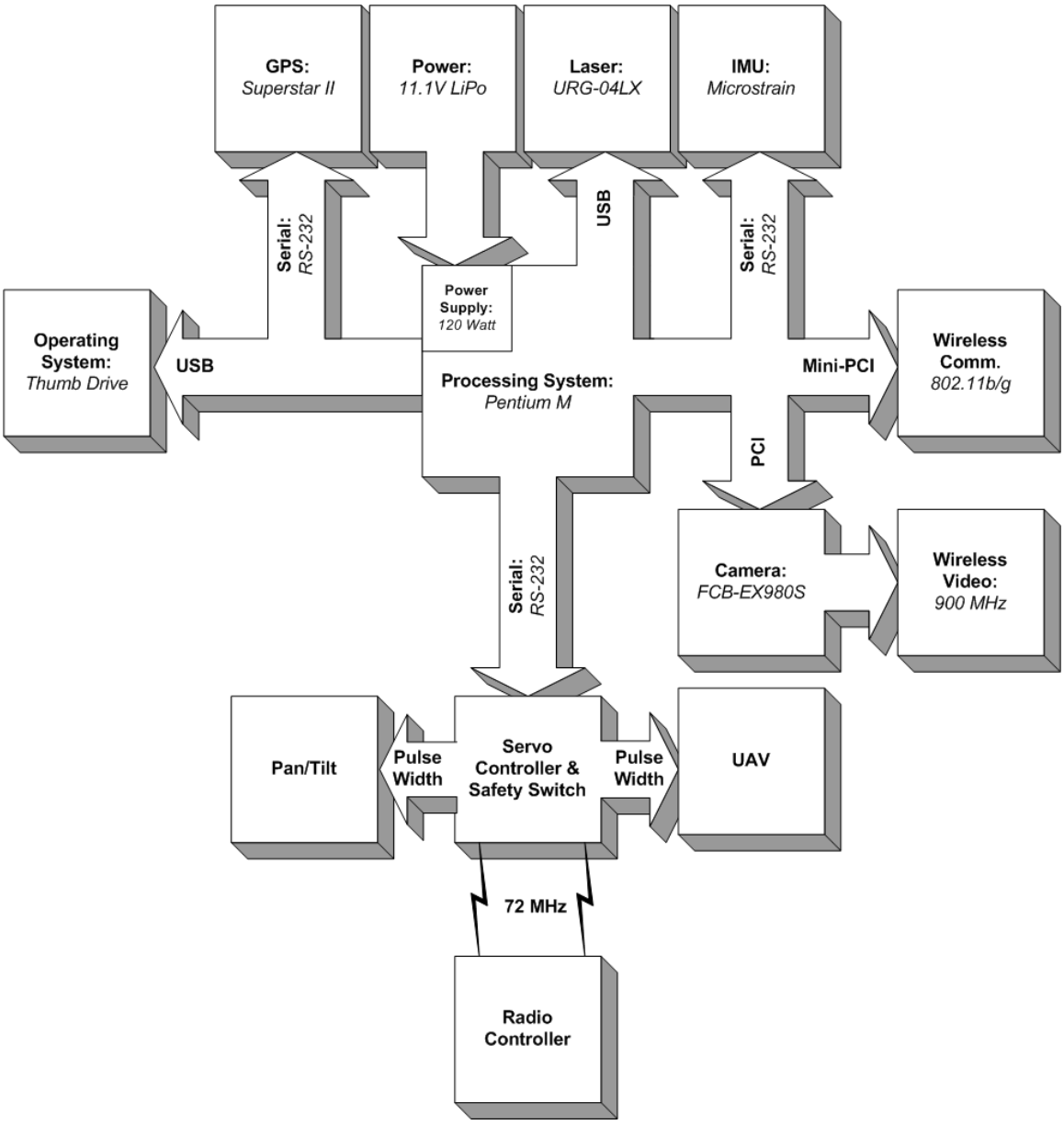

Figure 4: Conceptual Hardware Diagram

3.2.1 Enclosure

Autonomous vehicles typically have multiple components that must be protected from their natural operating environment. The most typical method of protection is to encase these items within some type of enclosure. Enclosure design is generally specific to the choice of UAV platform and sensing/computing hardware that one is attempting to utilize. A system that is

Figure 5: Empty Enclosure with EMI Foil

exposed to harsh environmental conditions must be designed and built to withstand those circumstances. Thus an enclosure designed for a turbine helicopter must be able to withstand the heat and vibration common to this type of platform.

The enclosure designed for the Joker-Maxi 2 helicopter testbed is a 21.7 x 17.5 x 6.1 cm basswood box, see Figure 5. This enclosure is specifically designed to encase and protect the processing system, GPS receiver, and safety switch. Basswood is utilized for the enclosure to ensure that the weight is kept to a minimum and that redesign due to hardware modification would be cheap, simple, and fast. The utilization of wood has two major flaws. First, wood is an extremely poor conductor of heat. Thus, the internal components had to be sufficiently resilient to heat. Second, there is no natural Radio Frequency (RF) shielding effects in wood. Through experimentation it was discovered that frequencies from the motherboard interfere greatly with GPS reception. Several variations of motherboards were tested at varying frequencies and GPS degradation varied from 20% to 100% based on the type of hardware and relative location, up to 30 inches, from the GPS antenna and receiver. To prevent Electromagnetic Interference (EMI), created from within the enclosure, from effecting components located outside of the enclosure a layer of 3M 1345 EMI foil was applied externally to the enclosure. This created a Faraday shield which prevents RF from leaving or entering the enclosure. To prevent internal EMI from affecting the GPS receiver, located within the enclosure, the receiver is also encased in EMI foil, discussed further in Section 3.3.4.

3.2.2 Servo/Switch Controller

Typical MUAVs are controlled by Pulse Width Modulation (PWM) servos. To autonomously control an RC vehicle hardware must be present that can communicate efficiently and effectively with both the processing system and PWM servos. It should also be noted that typical servos require constant refresh to hold position. Thus servo control hardware must be capable of refreshing positions at an appropriate rate, typically 50 Hz or more.

Besides control, one must also be aware of the safety concerns inherent to developing and testing a UAV, especially helicopters. It is inevitable that a software or hardware bug will cause the vehicle to endanger itself and possibly others. For this reason it is necessary that any autonomous vehicle be equipped with either a takeover or stop switch. For the purpose of this research a stop switch is defined as a device that removes the vehicle's ability to function. This type of device can simply cut power to the motor or immediately destroy the vehicle. Although a stop switch is effective, it is much more cost effective and typically safer to utilize a takeover switch. A takeover switch allows a human, or possibly secondary system, to immediately take full control of the vehicle. Note that safety devices such as the takeover and stop switch must be functional in the event of a failure. Thus, these safety devices must be designed in manners that allow them to function even if the rest of the system does not. These types of safety devices will not only increase safety for the user but also for those occasional onlookers.

To adhere to these requirements the USL Joker-Maxi II helicopter is equipped with the Microbotics Servo/Switch Controller (SSC). This controller allows the on-board processing system to output servo commands, via RS232, to all PWM devices used on the vehicle. The SSC is designed to allow a single switch on any common radio controller to take immediate control of the vehicle. The SSC also allows the on-board processing system to query the state of control, autonomous or human, and the current PWM signals to and from the SCC.

The interface to the SSC is a 44 pin high density D-sub connector. This interface initially presented somewhat of a design issue. Initially interface with the device was accomplished via a custom made cable. To allow for a varying number of PWM devices the cable had to be designed to allow for the maximum number of possible servos. This created a cable that had multiple unused connections hanging from the side of the enclosure. This became both an aesthetic problem and a safety concern as connections were easily crossed during hardware maintenance or upgrades. To prevent these types of issues from occurring, a custom interface board was designed. This board simply presents a row of standard 3-pin servo pinouts which can

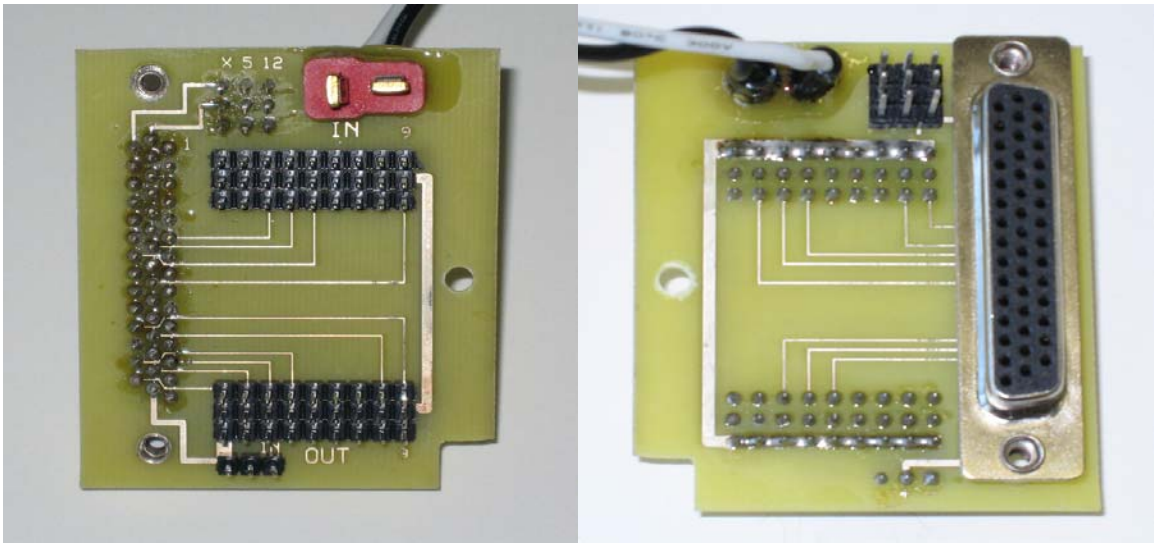

Figure 6: Front (left) and Back (right) of the SSC Interface Board

be utilized as desired. This board is also utilized to provided external interfaces for the processing system (both power and data), external sensors (power only), and safety switch (power only), see Figure 6. A schematic of the SSC interface board is given in Appendix B.

Due to the safety issues surrounding UAV research and the importance of manual takeover control, the SSC interface board is designed to support a secondary battery dedicated to the SSC. This design provides a human operator with complete control even in the event that the main power supply fails. The USL Maxi-Joker II utilizes a small 11.1V 0.5Ah LiPo battery for the SSC power supply. This battery can supply power to the SSC for dozens of flights before requiring recharge, see Chapter 8.

3.2.3 Orientation & Position Sensors

Vehicle state data, typically orientation, velocity, and/or position, are crucial to controlling any autonomous vehicle. Although there are numerous ways to obtain a system's orientation, velocity, and positional information, only a select few are widely accepted and heavily utilized.

The first of these methods includes utilization of a complete sensor suite. This method typically utilizes multiple sensors that directly provide the end user with all desired data. Although optimal, this type of method is rarely feasible. The barrage of sensors required to obtain the appropriate data at the desired rate typically exceeds either the payload of the vehicle

or the abilities of current technology. A typical example of a complete sensor suite for a fixed wing vehicle would include an Inertial Measurement Unit (IMU), GPS, and pitot sensor. These sensors can directly supply the position, velocity, and heading of the vehicle.

A second method is to utilize a sub-optimal sensor suite and attempt to calculate the desired values from the provided information. An example of this is utilizing an IMU to directly sense the orientation and accelerations of the vehicle and then attempting to calculate the position and velocity of the vehicle through integration of the accelerations. This type of method can significantly reduce the size and complexity of the hardware components. Although obviously beneficial, this type of method is limited by the abilities of the calculations. A second order integration to determine position from acceleration data, even from an extremely accurate sensor, will allow error to grow exponentially and typically diverge within a matter of seconds. Although this type of error can be reduced through filtering and fusion, the drawbacks should be taken into account when considering this method.

Third, it is mathematically possible to derive orientation, position, and velocity without sensors by tracking the control inputs into the system and deriving the necessary data through a mathematical model of the vehicle. Although theoretically possible, controlled systems typically contain uncontrollable and highly dynamic inputs (wind, gravity variation, barometric pressure, temperature, etc) that greatly affect the accuracy of the vehicle model. It is possible, however, to utilize a single sensor such as a GPS to dynamically tune the vehicle's model and thus correct for uncontrolled or unknown inputs. The usability of this method is greatly dependent on the accuracy of the vehicle's model as well as the accuracy and data rate of the sensor utilized for tuning.

To satisfy the need for orientation, velocity, and positional data required to successfully control the USL testbed helicopter a Microstrain 3DMG-X1 IMU and Superstar II GPS receiver (5Hz model) were chosen, Figure 7. The Microstrain IMU allows the user access to orientation (Euler angles or Quaternions), accelerations, and angular rates at a rate of up to 100Hz. The sensor is capable of sending both raw and gyro stabilized data and can be hard iron calibrated to account for variations in the magnetic field caused by surrounding equipment. The Superstar II GPS receiver provides latitude, longitude, and altitude (from sea level) position at 5 Hz. The Superstar II GPS is also Wide Area Augmentation System (WAAS) capable which allows the system to receive correction data without being locked to a ground station. Velocities are obtained through calculations using both the GPS and IMU supplied data and are described in detail in Section 5.4.

Figure 7: Superstar II GPS (left) and Microstrain 3DMG-X1 IMU (right)

3.2.4 Takeoff & Landing Sensor

Although positional data provides information regarding the location of a vehicle, it cannot supply positional details about other objects. This becomes an issue when attempting to autonomously takeoff or land a vehicle. Takeoffs and landings require that the locations of external objects such as the ground or landing platforms be known with a high degree of accuracy. This information allows the controlling system to know when certain maneuvers are and are not appropriate. An example of this would be to limit the lateral and longitudinal movements when the vehicle is within inches of the ground. Excessive movements could cause a tail or main rotor strike which would severely damage the vehicle.

Obtaining information regarding the vehicle's relative position to a landing surface is typically determined in one of two ways. First, the vehicle can simply assume that the altitude of the landing surface is known. This could be as simple as assuming that the landing surface is at the same altitude as its takeoff location. Also assuming that the vehicle's positional data is extremely accurate the vehicle could simply mark an altitude as ground level. This altitude could then be directly used to guide the vehicle through the takeoff and landing routines. Although this is the simplest method its assumptions are rarely valid.

The second method is to allow the vehicle to sense the location of the landing surface. This type of range data can be acquired using stereo vision, optic flow, infrared sensors, sonar, radar, planar lasers, etc. Vision algorithms such as optic flow and stereo vision have shown great advancements in recent years but are still considered highly error-prone in outdoor dynamic

environments [20, 62, 83]. Sonar and infrared type sensors are by far the lightest and cheapest of these devices but only provide distance in a single dimension and are commonly disrupted by uneven terrain, color, and texture. Radar by far provides the widest range of distance measurements but due to its technology is typically too heavy for use on a MUAV vehicle. Scanning/Planar lasers provide multiple distance readings over a two dimensional plane but are typically disrupted by both direct sunlight and heavy vibration. Scanning/Planar laser are also the most expensive of the range devices.

Although the USL vehicle utilizes GPS to determine altitude, the accuracy of this device is highly restrictive. GPS devices similar to USL's have shown altitude errors of three to five meters [92]. This level of accuracy prevents the vehicle from being able to safely and consistently land under computer control. To assure that the USL testbed is capable of both takeoff and landing it is equipped with the URG-04LX scanning laser, see Figure 8. This sensor provides the processing system with range information vital to both takeoff and landing. The laser is capable of providing range data of up to four meters with single millimeter resolution at 10 Hz. This data allows the testbed to directly determine its relative position to the ground and allows the system the capability to land and takeoff at varying locations.

Figure 8: Hokuyo URG-04LX Laser Range Finder

3.2.5 Electrical Power

Battery selection, although fairly limited, does contain a few pitfalls that must be addressed. Currently several variations of battery chemistry are available on the market. The most popular of these are Nickel-cadmium (NiCa), Nickel-metal hydride (NiMh), and Lithium Polymer (LiPo). NiCa is the oldest, most widely available, and cheapest of these chemistries but provides the lowest power to weight ratio. NiMh is a somewhat newer technology, provides greater power to weight ratios than NiCa, and is priced moderately higher than NiCa. LiPo has by far the highest power to weight ratio, three times greater than NiMh and four time greater than NiCa [13], but is the most expensive of the three chemistries and potentially explosive when used improperly.

The USL testbed helicopter utilizes four distinct batteries, Figure 9. The first and largest of the four batteries is responsible for powering the helicopter's electric motor. The utilized battery is a 37V 10Ah LiPo and weights approximately 2.19 kg. Due to the strict payload limitations of electric MUAV helicopters, they are typically limited to the use of LiPo batteries. Also note that this battery was specifically ordered to fit within the stock enclosure provided by the Joker-Maxi II helicopter.

The second battery utilized by the USL testbed is a 4.8V 2.0Ah NiMh which provides power directly to all servo motors and to the RC receiver. Note, that this battery could have been removed by utilizing the 5V output from the processing system to power the afore mentioned devices. This was avoided for two main reasons. First, digital high speed high torque servos draw a large amount of current when transitioning. This current is constantly in fluctuation and can cause a short, but significant, drop in voltage. This drop in voltage may adversely affect other components on the testbed. Second, setting up the system in this manner would directly link the operation of the vehicles servos to the operation of the processing system. In the event that the processing system's battery became depleted during flight the safety pilot would be unable to retake control of the vehicle. For these reasons the 4.8V is solely responsible for powering the servos and radio receiver.

The third battery used on the USL testbed is an 11.1V 0.5Ah LiPo. This battery is only responsible for powering the SSC. Although the SSC could also have been powered directly by the processing system's power supply, it was decided, for safety, that the SSC have its own power supply. This assures that the SSC will operate properly regardless of the state of the processing system or its battery.

Figure 9: USL Testbed Batteries

The last battery on the USL testbed is an 11.1V 4.2Ah LiPo battery. This battery is used to power the on-board processing system and all internal and external sensors.

3.2.6 Data Processing Board

Of all the components to be selected for the development of a MUAV helicopter testbed the most difficult is the main processing board. This is mainly due to the abundant variations of form factors and types and number of available peripherals. Processing boards come in many shapes and sizes and can be COTS or designed in-house. Popular form factors of processing boards include PC-104, Mini and Nano Integrated Technology Extended (ITX), and Advanced Technology Extended (ATX). Along with variations of form factor, one must decide on the desired types and number of peripherals. Popular peripherals include Universal Serial Bus (USB), Recommended Standard 232 (RS232), Recommended Standard 422 (RS422), Transistor-Transistor Logic (TTL), parallel, Firewire (IEEE1394), Ethernet, Super Video Graphics Array (SVGA), infrared, Peripheral Component Interconnect (PCI), and Mini-PCI, to name a few.

Although each form factor for the main processing board, from a high level, essentially performs the same functions, a hasty choice may significantly increase the effort required to install, develop software for, and power. PC-104 based processing systems are at least 9.0 x 9.6 cm printed circuit boards that are stacked for inter-connectivity. Each individual board typically,

but not always, has a single function including I/O interface boards, CPU boards, power supply boards, and wireless networking boards. These allow the user to mix and match the desired abilities of the overall processing system. PC-104 boards also tend to have more support for hard-realtime OSs. Although their size and modularity make PC-104 form factors very attractive, they do tend to utilize less than state-of-the-art technology and are typically more expensive than the other available form factors. ATX, although the most heavily utilized form factor, is generally reserved for desktop type computers and is typically too large, too heavy, and requires too much power to operate on a MUAV. Mini, 17 x 17 cm, and Nano, 12 x 12 cm, ITX form factors appear to be the medium between the PC-104 and ATX form factors. These boards provide a more energy efficient and compact form factor than the ATX while providing more state-of-the-art technology than PC-104.

When deciding on peripheral interfaces one should generally take into account the type of equipment used on MUAVs and the overall most common interfaces. Since most sensors designed today utilize either RS-232 or USB, it would beneficial to assure that the chosen processing board have several of these integrated into the system. Peripheral choice, although important, is typically eased by the abundance of converters and adapters available that allow the user to plug almost any device into any port.

The USL testbed is equipped with a G5M100-N Mini-ITX motherboard, Figure 10. This motherboard was chosen based on its type and number of peripheral interfaces (Table 5), support for a low power processor (Pentium M), overall size, and low cost.

Table 5: G5M100-N Interface Support

Port Type	# Available	Interface Type
USB	6	4x 5 Pin Standard, 2x Board Pinout
Serial	3	2x RS232, 1x RS232/RS422
Ethernet	2	RJ45
VGA	1	VGA
PS2	2	1x Keyboard, 1x Mouse
PCI	1	PCI Slot
Mini-PCI	1	Mini-PCI
IDE	2	1x 40 Pin IDE, 1x44 Pin IDE
RAM	2	PC 2700

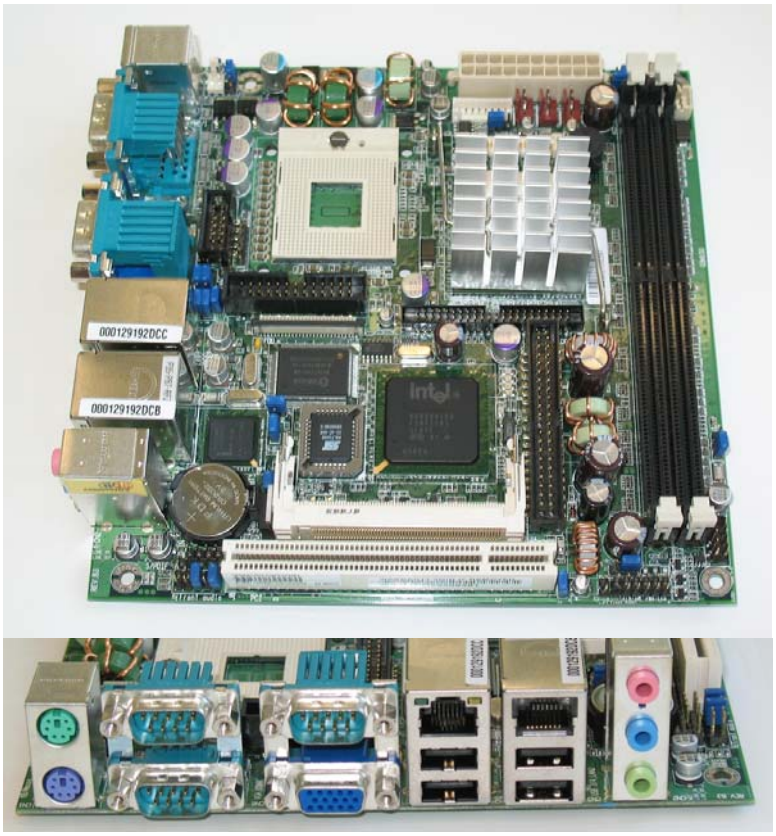

Figure 10: G5M100-N Mini-ITX Motherboard's Top View (top) and Front Panel (bottom)

3.2.7 Communication

For the purpose of this dissertation communication refers to any wireless transfer of information from the UAV to any disconnected system. This includes data transferred via the radio controller, on-board processing system, and video transmitter. Although there are hundreds of variations of communication protocols, only a few are commercially available and even fewer are not restricted by the Federal Communications Commission (FCC) [93].

To date, publicly accepted and commercially available forms of wireless communication include aeronautical radio navigation, maritime mobile, TV and FM broadcasting, and satellite. Although all of these communication devices are available commercially, many frequencies typically require special authorization by the FCC and are restricted in the locations in which they can be used. Furthermore, the FCC typically restricts the power outputs of devices that do not require FCC permission for operation.

For hobbyist type RC aircraft the typical frequency is 72 MHz, ranging from 72.010 MHz (channel 11) to 72.990 MHz (channel 90). This frequency is utilized by the radio controller to transmit PWM request to the vehicle. This data is received by the radio receiver and converted into PWM signals distributed to the appropriate servos. In the case of the USL testbed, the 72 MHz frequency is also used to grant or remove control of the vehicle to and from the on-board computer. More recently there have been several models of radio controllers and receivers that utilize a 2.4 GHz frequency. The 2.4 GHz frequency is also utilized by the 802.11 and Bluetooth protocols as well as many cordless phones. Make note that the wide use of the 2.4 GHz frequency could potentially cause radio control failure and should be utilized with caution in industrial or residential areas.

Utilization of distinct wireless frequencies for video transmission is currently extremely popular on UAVs. This is mainly due to the bandwidth required to wirelessly transmit high quality streaming video. To date there are several non-regulate variations of wireless video transmitters being utilized by MUAVs. These devices typically operate on the 900 MHz, 2.4 GHz, and 5.0 GHz frequencies and are typically used to relieve bandwidth limitations imposed on the 802.11 protocol. The main differences between the varying frequencies are the average operational distances, which also varies based on power output, and the amount of bandwidth available.

It should be noted that is very typical to see a variety of communication protocols used beyond the frequency domain. Critical information such as object identification and internal state data must not be lost during transmission. This data can be sent utilizing protocols that guarantee the receipt of data in the correct order, i.e. TCP. For other information, such as streaming video, it may be more critical to receive the most current data rather then delaying it to assure that previous transmissions were received. This type of data is typically sent utilizing non-guaranteed protocols such as UDP. Non-guaranteed protocols also offer the advantage of heavily reduced overhead but do not guarantee that data is received or is in the correct order.

The USL testbed is equipped with an Intel Pro 2200 Mini-PCI wireless card that allows the on-board processing system to communicate with any computer network operating on the 802.11 protocol, Figure 11. The testbed is also equipped with a 72 MHz receiver that receives commands from the radio controller. The last piece of communication hardware present on the USL testbed is the wireless video transmitter. This device transmits all video, via the 900 MHz 100 mW transmitter, to any receiver within broadcast range.

Figure 11: Intel Pro 2200 with External Antenna and Pigtail

3.2.8 Camera

Although an on-board camera is not required for effective autonomous flight it would be severely limiting to a developmental testbed to not include one. Commercial cameras are available that utilize multiple interfaces including Firewire (IEEE1394), USB, and composite and vary greatly in size, weight, accuracy, and cost. When developing a MUAV testbed one must take into account several limiting factors of camera during the selection process.

First, due to the payload limitation of aerial vehicles one must consider the payload loss for a particular camera. This includes the weight of the camera and the weight of the extra electrical power required to operate the camera. One must also consider the mounting location of the camera. If the camera is mounted towards an extremity of the vehicle it must be compensated for with a counter weight which will further deplete the payload of the vehicle.

The second item that must be considered is the camera's ability to function correctly in its operational environment. Aerial vehicles typically experience high frequency vibrations and large gravitational forces as well as operate in varying light intensity environments. One issue to consider is the camera's ability to compensate for varying light intensities. Several models of cameras have automatic iris compensation that attempt to control the amount of light entering the lens while others attempt to compensate for light exposure through software. One must also be aware that aerial vehicles typically do not operate at fixed altitudes. As such the designer must be aware that the camera may need to be periodically zoomed or focused.

Last, one must consider the interface utilized by the camera. Firewire cameras have become extremely popular due to their compact size, high data rates, and ability to easily daisy chain multiple cameras. Although very well suited for MUAVs, Firewire cameras are typically

expensive, somewhat difficult to interface with, and are not always supported by the OS. USB cameras allow for a very common interface and are typically moderately priced but are difficult to interface with in most OSs. Composite cameras are by far the most heavily commercialized cameras. Composite cameras are typically small and fairly inexpensive but require frame grabbing hardware to utilize correctly.

The USL helicopter testbed is equipped with a Sony FCB-EX980S block camera and IVC-200G frame grabber, Figures 12 and 13. The block camera supports auto stabilization, 26x optical zoom, low-light operation, and iris/zoom/focus control via an RS232 interface all in a 230 gram 5 x 5.7 x 8.3 cm (wxhxd) package. Interface with the device requires both a conversion board and frame grabber. The conversion board, model IFB-EX232, allows for powering of the camera and RS232 based control. The PCI frame grabber allows for data acquisition at 30 frames per second on four channels simultaneously. Note that the frame grabber is shipped with standard BNC connectors as the interface. These connectors were removed and replaced with female RCA connectors, see Figure 13.

Figure 12: Sony FCB-EX980S Camera, Interface Board, and Video Transmitter

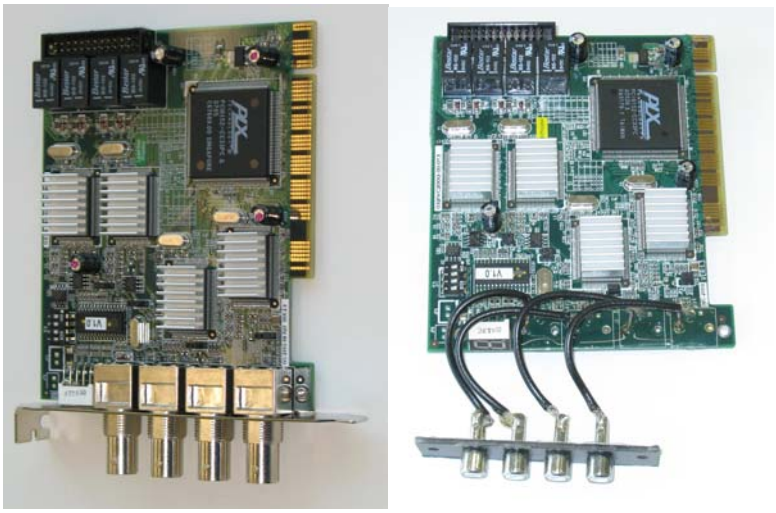

Figure 13: Frame Grabber Before (left) and After (right) Interface Alteration

3.2.9 Data Storage

Data storage, in reference to the proposed testbed, describes the area or device where the OS, data acquisition software, control software, and collected data are stored. The most common data storage devices include magnetic drives, magnetic disks, and solid state memory. Although magnetic drives and disk are the cheapest, per gigabyte, they require mechanical devices to read and store data. This not only limits the speed at which data can be stored and retrieved, it requires that the device be virtually stable at all times to prevent damage to the device. This coupled with the size and weight of magnetic devices typically prevents their usability on miniature autonomous vehicles.

Solid state memory, due to its size, weight, and operational characteristics, is the typical utilized device for data storage on MUAVs. One major short coming of solid state memory is its degradation due to usage. Individual sectors of solid state memory are typically only good for a few hundred thousand writes. Although this seems sufficient for almost any system one must consider that many OSs utilize permanent storage as virtual memory and may perform thousands of write operations during a single procedure.

Data storage for the USL testbed is handled utilizing both solid state memory, in the form of a USB thumbdrive, and volatile Random Access Memory (RAM). The solid state memory is used to store the OS and any software necessary to make the system functional, including device drivers and communication protocols. The testbed is first booted from the USB drive where the

operating system is copied to a RAM drive, sometimes referred to as a virtual drive. From this point all data storage is performed on the virtual drive. This allows the USB drive to be removed from the system after bootup preventing its loss or damage during vehicle operation. This also allows for a modular hardware design where alterations to the platform, sensors, or software only require the system to be rebooted utilizing a properly configured USB drive.

3.2.10 Hardware Chassis

The hardware chassis refers to the frame to which all of the hardware is mounted. This is of special interest as the chassis is responsible for protecting the hardware as well as providing the necessary frame for successful operation. There are several key features that one must be aware of when designing a chassis for sensors and processing hardware.

First, helicopter based platforms naturally create high frequency vibrations caused by the motor, main rotor, and tail rotor. This vibration, if not isolated, will create noise, error, and possibly damage to hardware mounted to the platform. For these reasons it is advisable that some type of vibration isolation or reduction be built into the chassis. This would be similar to the suspension system built into modern vehicles to reduce road vibration.

Second, as previously mentioned MUAVs have low payload capabilities thus a hardware chassis must balance the desire for stability and protection with the lifting capabilities of the platform. This can be accomplished through the use of alloys such as titanium or aluminum.

Last, sensors such as magnetometers, typically used to sense heading, are heavily influenced by magnetic fields and ferrous materials. This can include sensors, batteries, actuators, electrical current, and metal objects. With this in mind one must not only assure that sensors, like magnetometers, are mounted in areas that isolate them from interference but that the hardware and chassis used to mount these sensors do not create interference. Typically materials that do not adversely affect magnetometers are Styrofoam, wood, plastic, aluminum and brass.

The USL helicopter utilizes custom designed aluminum skids for the hardware chassis, see Figure 14. Aluminum allows the chassis to be lightweight and sturdy without adversely affecting the Microstrain IMU utilized on the vehicle. The tubular design also adds to the strength of the chassis. The chassis contains two distinct layers of vibration isolation mounts. The first layer separates the chassis from the platform utilizing four rubber isolation mounts. The second layer isolates the processing system and sensors from the chassis again using rubber isolation mounts. Specific dimensions for the chassis are provided in Appendix C.

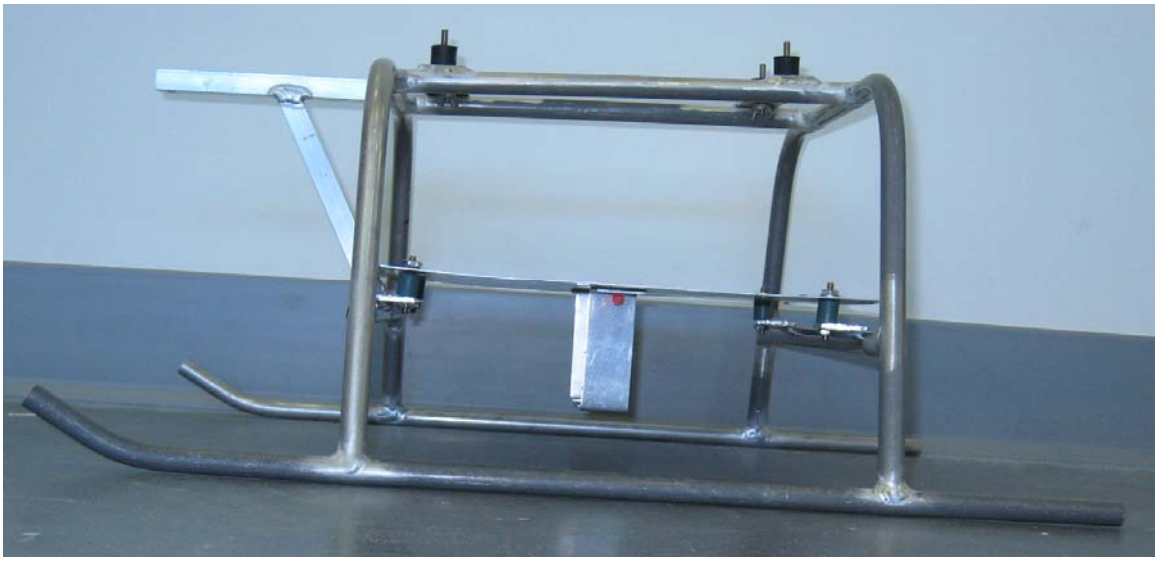

Figure 14: USL Hardware Chassis

One should also note the mounting location of the Microstrain IMU. The IMU is suspended two inches below the enclosure. The benefits of choosing this mounting location are three fold. First, this location separates the sensor from the rest of the system which in turn reduces the possibility of magnetic interference sensed by the IMU's magnetometers. Second, for the IMU to provide the most accurate angular rates, accelerations, and heading data it should be mounted as close to the rotational axis of the vehicle as possible. The chosen mounting location allows the IMU to be mounted directly below the main shaft of the helicopter which is also the rotational axis for heading changes. Last, vibration isolators typically require a minimum load to successfully isolate vibrations. The extreme light weight of the IMU is not sufficient to isolate the helicopters vibrations using the desired materials. Thus this mounting location allows the IMU to utilize the weight of the enclosure and all of its components to create the necessary load for sufficient vibration isolation.

3.2.11 Pan/Tilt

To adhere to the design of a highly usable developmental testbed the USL helicopter is equipped with a pan/tilt unit that allows the camera direction to be moved without altering the position and heading of the helicopter, see Figure 15. The pan/tilt is custom built and integrated

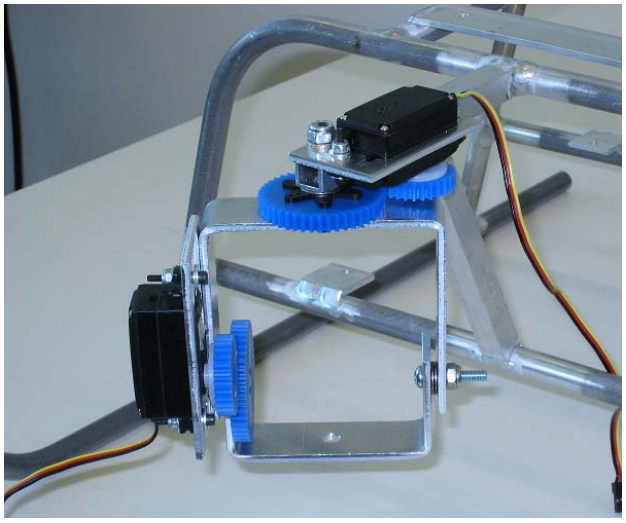

Figure 15: Pan/Tilt

into the hardware chassis. The pan/tilt is equipped with two low profile Futaba servos and allows the camera more than 60 degrees of motion in both pan and tilt. Design specifications for the pan/tilt are provided in Appendix D.

3.3 Assembly

To assure that the USL development testbed can be fully replicated this section details the assembly of the hardware and describes system components that may have not been detailed in the previous sections. This section does not detail the assembly of the platform as it is described in great detail in the manufacture provided assembly and setup manuals. This section also assumes that the user has some basic knowledge about computer assembly including how to properly handle Electrostatic Discharge (ESD) and basic safety.

To simplify the assembly of the testbed the hardware components are divided into groups. Assembly within each group is described in great detail. Several of the following sub-sections will include an overview diagram detailing the hardware described within that section (green) and connections with other groups of hardware (blue) described in other sub-sections. Note, the red blocks within the diagram are not discussed in detail since they fall outside the scope of this dissertation.

3.3.1 Enclosure

As the enclosure is the storage and interface unit for the majority of the hardware it is the most logical place to begin assembly descriptions. The enclosure consists of a custom built wooden box sealed with EMI foil. A schematic providing all measurements for the enclosure can be found in Appendix E.

Assembly of the custom box is first performed by cutting out the six sides of the box from a sheet of basswood. The six sides include the lid (225 x 181 mm), bottom (225 x 178 mm), faceplate (61 x 225 mm), back (58 x 225 mm), and two short sides (58 x 175 mm). One of the long sides is then stenciled and cut to be used as the faceplate for the enclosure, refer to Appendix E for measurements. This faceplate is the medial interface for all outside devices. The four sides and bottom are then assembled and glue together with standard wood glue. Note that the back and two sides are assembled on top of the bottom plate making the inside dimensions of the box 219x175x58 mm.

The next step in assembling the enclosure is to coat the box using EMI foil. Although any method will work one must assure that there is a continuous electrical circuit covering the entire box. Thus, our method includes having as few distinct strips of foil as possible. The enclosure uses one continuous sheet of foil to cover the faceplate, bottom, back, and top of the enclosure. This method also serves as a virtual hinge for the lid to open and close. To assure that the foil does not tear with persistent opening and closing of the lid the foil is reinforced with masking tape along the hinge. A second sheet of foil is used to coat the remaining sides providing a second coat on the bottom of the enclosure. The last step in coating the enclosure is to use three small strips of EMI foil to seal the three unconnected sides when the lid is closed. This last step is only performed when the hardware has been installed and the enclosure is ready to be mounted to the chassis, discussed in Section 3.3.5.

3.3.2 Laser

Once the enclosure has been assembled it can be fitted with hardware. The first piece of hardware to be mounted to the enclosure is the laser. Due the sensitivity of the laser and the desire to prevent damage to this sensor it is mounted directly to the enclosure. This allows the laser to take advantage of the two layers of vibration isolation used to stabilize the enclosure and IMU.

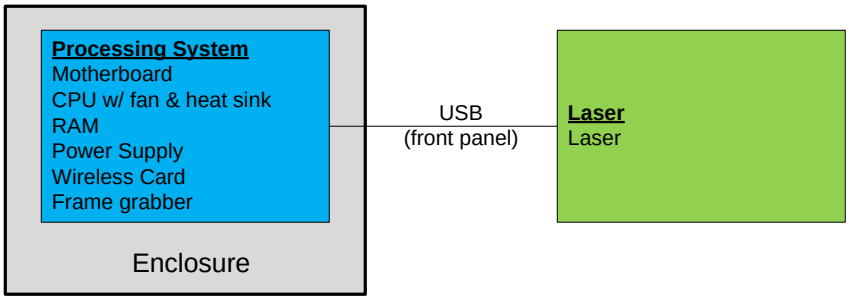

Figure 16: Laser Hardware Connection Diagram

The first step in mounting the laser is to drill two small mounting holes diagonally along the left side of the box. The laser can then be mounted using two flush mount bolts. To assure the mounting hardware does not back off over time a small drop of blue loctite is applied to both bolts. Once the laser is correctly mounted to the side of the enclosure two small pieces of electrical tape are used to cover the exposed bolt heads.

The laser will ultimately be connected to the processing system via a mini-B to Series "A" USB cable plugged into one of the four available ports located on the enclosure's faceplate, see Figure 16. Power for the laser is supplied via the 5V power output available on the SSC interface board and is transmitted via a custom power cable (Futaba-J male to PHR-8).

It is noteworthy to mention that URG-04LX Laser is designed for indoor use. Through experimentation it was determined that the laser could provide correct results outdoors. The only noticeable failure was an intermittent shutdown of the laser during extreme exposures to direct sunlight. To account for this problem a layer of black electrical tape is placed over the top half of the laser. This shielded the laser from direct sunlight and removed the intermittent shutdown.

3.3.3 Processing System

The next step in equipping the enclosure with hardware is to assemble and mount the processing system. The processing system hardware consist of the motherboard, CPU with fan and heatsink, system memory, power supply, wireless card, 2.4 GHz antenna w/ cable, frame grabber, power switch, and flexible PCI extension cable. Figure 17 details the connection diagram used for the processing system hardware.

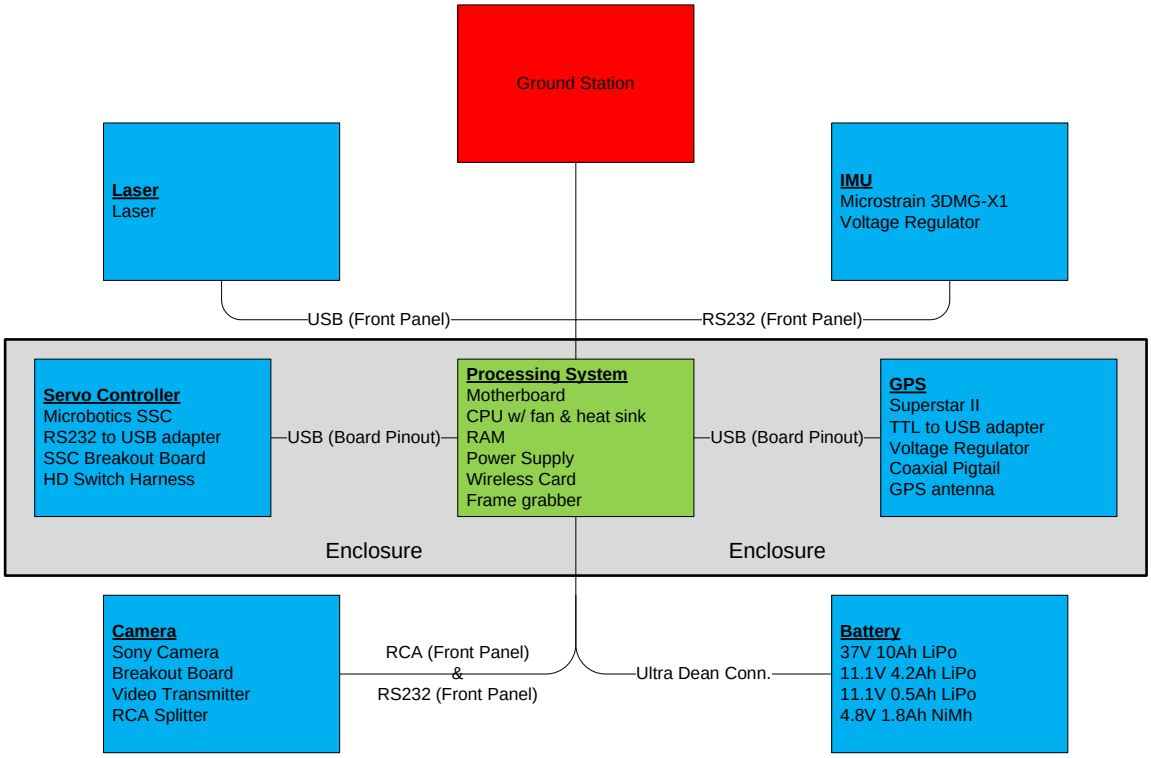

Figure 17: Processing System Hardware Connection Diagram

The first step in assembling the processing system is to install the major components onto the motherboard. This consists of installing the CPU, system memory, mini-PCI wireless card, heatsink, and fan to the mini-ITX motherboard. Due to the standard snap in interfaces of the motherboard, the assembly of the CPU, system memory, and mini-PCI wireless card consist of snapping them into the correct slot. Installation of the fan and heatsink is also straight forward but one must assure that a thermal paste is applied between the CPU and heatsink to assure proper conduction of heat.

Once this is complete the motherboard can be mounted within the enclosure. This is performed by angling the board, faceplate first, into the enclosure and sliding the front panel of the motherboard into the enclosure faceplate. Adhering the motherboard to the enclosure requires drilling holes at the four mounting locations. This can be done using a hand tool to assure that the motherboard is not damaged. Since the interior of the enclosure is nonconductive it can be affixed without spacers. This is done using four flat headed plastic bolts and four nuts. Note that the bolts should protrude into the box. Either a small drop of hot glue or a second set of nuts can be used to assure that the mounts do not loosen over time.

Once the processing system has been mounted, the 2.4 GHz wireless antenna, cable, and main power switch should be installed. The wireless cable is simply plugged into the main antenna connector on the mini-PCI card and then mounted to the right side of the enclosure. The antenna can now simply be screwed on. The power switch is mounted by forcing it through the opening in the enclosure's faceplate and is naturally held in place by its retention clips. The power switch should then be plugged into the back of the motherboard.

The power supply, although not electronically modified, did receive several connector alterations to fit our constraints and thus should be modified before being installed. First the DC input connector must be removed. It will later be replaced with an Ultra Dean connector, Section 3.3.5, but at this point should be left without a connector. Second, the outer most LP4 connector is removed and replaced with two male Futaba-J connectors. These Futaba connectors corresponded to the 12V and 5V power outputs on the enclosures faceplate. At this point the power supply can be installed simply by plugging it into its appropriate interface on the motherboard.

The next step in assembly is to install the PCI frame grabber. As mentioned in Section 3.2.8 the BNC connectors, standard on the PCI frame grabber, were removed and replaced with RCA connectors. This is done to reduce the overall size and weight of the board as well as to provide a more common interface. This is accomplished by cutting both the signal and ground wires on the back of the BNC connectors just above the PCB. Once this is complete a flat head screwdriver is placed under the BNC connector and used to pry it off of the frame grabber. The mounting rods, ground pins, and signal pins were then heated and removed from the board. Next, four individual strips of coaxial cable were soldered to both the frame grabber and a strip of four phono jacks.

To reduce the amount of unused space within the enclosure it was determined that the PCI frame grabber be installed horizontally. Initially a 90 degree PCI riser was selected but due to the configuration of the RAM and the desired compactness of the enclosure it was insufficient. Thus a 3in flexible PCI riser was selected. This allowed the PCI frame grabber to be mounted horizontally but also to be shifted into a more compact position.

Physical installation of the frame grabber first included plugging the PCI riser cable into both the frame grabber and motherboard. The cable is then shaped to allow the frame grabber to slip in-between the RAM and front panel components. To assure that the frame grabber did not sit directly on the motherboard a small, non-conductive, piece of foam is hot glued to the underside of the frame grabber. This piece of foam is designed to act as a standoff between the

frame grabber and the chipset heatsink on the motherboard. Next, the four phono jacks were mounted to the faceplate of the enclosure using two plastic screws and nuts. Last, a 1.5 cm wide brace is epoxied across the enclosure. This assures that the frame grabber is held in place during operation.

3.3.4 GPS

With the processing system successfully mounted within the enclosure the GPS hardware can now be installed. The GPS hardware consists of a Superstar II GPS receiver, a TTL to USB converter, a 5V 1A voltage regulator with heatsink, and pigtail cable, Figure 18.

The Superstar II GPS is a 5V receiver with two main connectors, a 20 pin terminal strip (Samtec TMM-110-03-T-D) and an MCX. The terminal strip connector is the interface for output data and power connection. Correct functionality of the receiver requires that a regulated 5 volts be supplied to the main power pin and, if utilizing a powered antenna, to the antenna pin. Power for this connector is supplied via a 3 pin 5V 1A regulator. At this point the regulator should be wired with two sets of power and ground wires. The first set should be wired to the receiver and should supply a regulated 5 volts. The second set of wires should be left unconnected but will be directly wired to the main battery in Section 3.3.5. The regulator will be wired this way to allow the GPS to function regardless of the state of the processing system. This allows the GPS to gather satellite data before powering the processing system and allows the

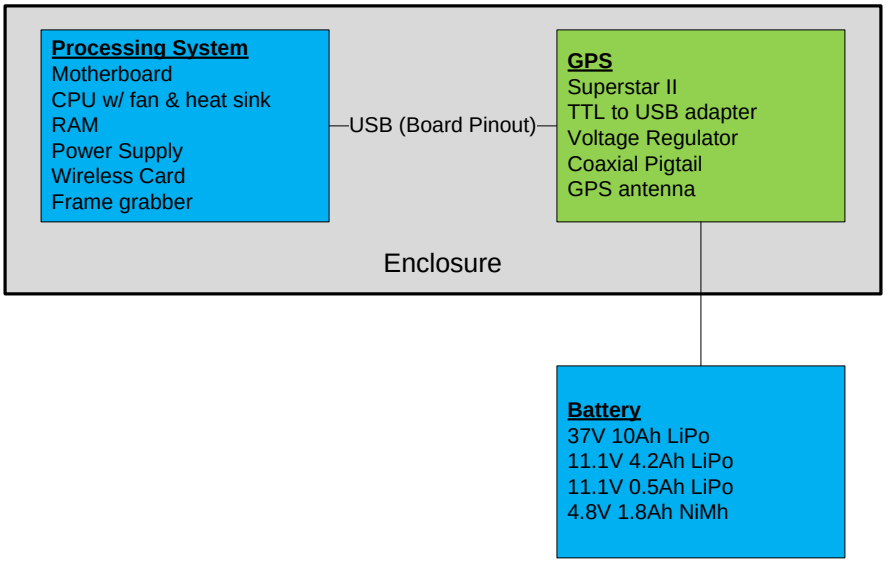

Figure 18: GPS Hardware Connection Diagram

receiver to retain a satellite fix in the event the processing system must be restarted. The regulator is then attached to a 1.5 x 3.5 x 0.2 cm sheet of aluminum used as a heatsink. To assure that regulator's pins did not wear with age they were coated with epoxy. Last, the regulator and heatsink were sealed with PVC heat shrink to protect them from possible short circuiting.

The transmit, receive, and data ground pins on the receiver's terminal strip are used for interfacing the receiver and the processing system. Since the processing system cannot directly interface with the TTL communication standard used by the receiver, all communication is performed through a TTL to USB converter. The transmit, receive, and ground pins are wired directly to the TTL to USB converter board. This converter connects directly to one of the available USB pinouts on the back of the motherboard. To interface the converter (mini-B USB) to the motherboard's pinouts (2 x 5 pin header) an adapter cable was manufactured. This cable utilizes a 2x5 pinout connector, supplied with the motherboard, and a mini-B USB cable. The mini-B USB cable is first cut leaving approximately six inches of cable. The 2x5 pinout connector is then removed from the supplied cable and soldered to the open end of the mini-B USB cable. Note that the 2x5 pinout connecter supplies interfaces for two USB devices. The second interface will be utilized in Section 3.3.5.

The second connection required for operating the GPS receiver is the antenna connector. Due to the desire to have all external devices connect to the faceplate of the enclosure, a 6 in MCX to SMA female pigtail is used to connect the GPS receiver to the faceplate. The GPS antenna is affixed to the horizontal fin on the tail boom of the helicopter using servo tape and can be connected to the enclosure after the chassis is assembled in Section 3.3.6.

Now that the GPS receiver is wired, it must be shielded from radio interference. The first step in shielding the receiver is to encase the receiver in PVC heatshrink and seal the ends with electrical tape. The only opening is a small hole at the front of the receiver just largest enough to allow the interface cables to pass through. This step provides a layer of protection for the circuitry. Next, the receiver is coated with EMI foil. The EMI foil is wrapped around the receiver and the excess is folded over on both ends to prevent as much RF leakage as possible. Note that there must be some overlap on the ends of the foil so that a good circuit can be made. Last the receiver is again encased in PVC heat shrink which is again sealed on the ends by electrical tape. This provides a layer of protection for the circuitry external to the receiver. An assembled GPS system is detailed in Figure 19.

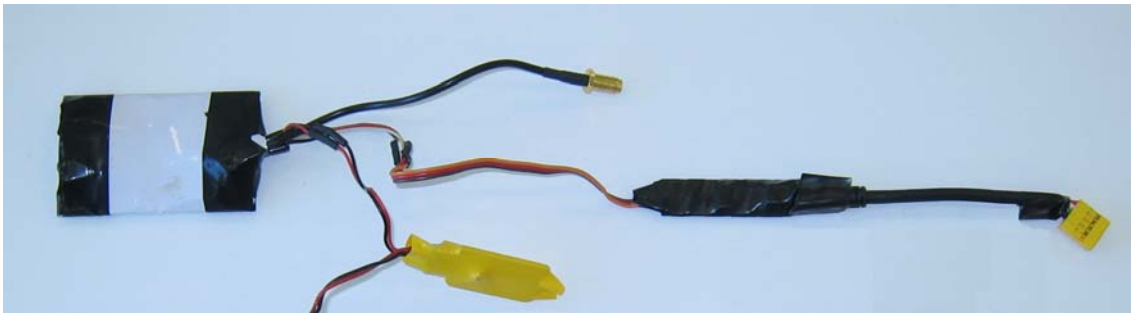

Figure 19: Partially Assembled GPS System

Last the receiver is mounted to the back left corner of the enclosure. This can be done by using two small strips of double sided foam tape.

3.3.5 Servo Controller

The last item to be mounted within the enclosure is the servo controller hardware, Figure 20, which consist of the Microbotics SSC, an RS232 to USB converter, and the custom two-layer PCB, referred to as the SSC interface board, described in Section 3.2.2. For simplicity, this section will refer to numbered blocks that represent sections on the SSC interface board. These blocks are detailed in the interface guide shown in Figure 21.

The first step in this part of the assembly is to solder all of the required headers and connectors to SSC interface board. Note that there are four distinct types of connectors that must

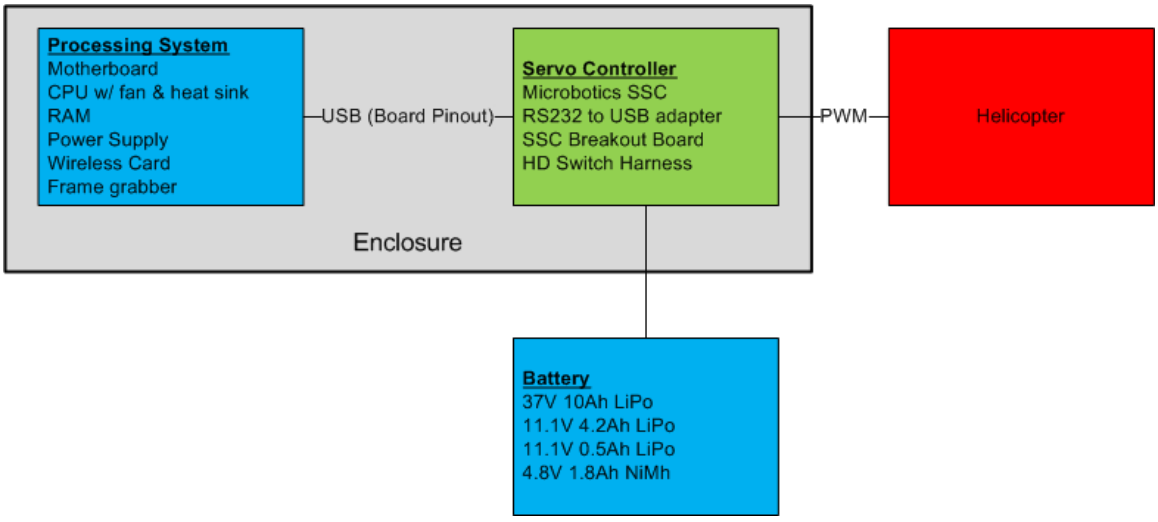

Figure 20: Servo Controller Hardware Connection Diagram

be soldered to the interface board: single sided headers, double sided headers, a male Ultra Dean connector, and a 44 pin HD connector. This assembly first solders all single sided headers, blocks 5, 7 and 8. Note that the pins should be accessible on the outer side of the SSC interface board, refer to Figure 6 for visual details. These connections represent the servo inputs from the radio receiver (block 5), the outputs to the servos (block 7), and the SSC power input (block 8).

Next, the three sets of double sided headers are soldered to the board, blocks 2 and 3. Double sided headers, for the purpose of this work, represent headers that have pins for connections on both sides of the board. The three sets of double sided headers represent a 12 volt power output (right pins of block 3), 5 volt power output (left pins of block 3), and the RS232 communication connection to the SSC (block 2). The two power outputs will allow the processing system's power supply to power external devices. Internally, the communication connection is used to interface the SSC with the processing system. Externally, the communication connection is used to reprogram the SSC if the need arises. Note that the external connection is typically capped. This prevents the connection from accidentally being used as a power supply but also acts as a reminder to the user that the internal interface, used by the processing system, must not be in use when the SSC is being reprogrammed. Having both connections to the communication connector active at one time may damage internal and/or external hardware.

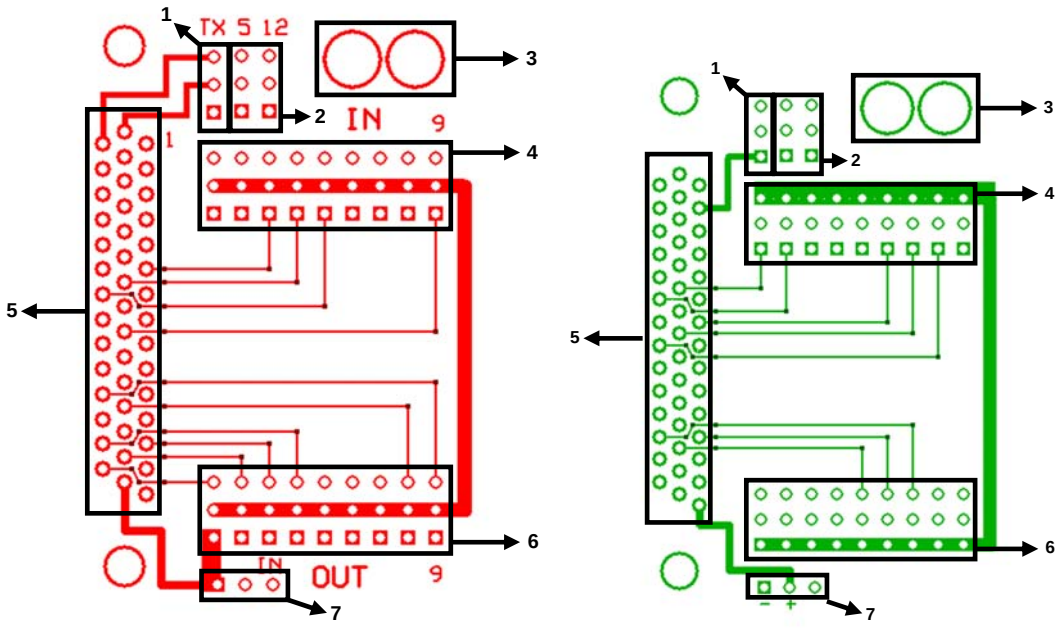

Figure 21: Interface Guide for the Front (left) and Back (right) of the SSC Interface Board

The last two items soldered to the interface board are the 44 Pin HD connector (block 6) and the Ultra Deans connector (block 4). The 44 Pin HD connector is simply slid through the back of the interface board and each individual connection soldered. Once this is complete the excess ends of the pins can be clipped off. The Ultra dean's connector is connected to both the main power input for the processing system's power supply and the input wires for the GPS's voltage regulator, described in the previous section. This can be done by either splicing the voltage regulator wires into the power supply's input wires or simply by soldering both sets of connections to the Ultra Dean connector. Note that the wires should first be passed through the back of SSC interface board and then soldered to the Ultra Dean. Once this is complete the Ultra Dean can be epoxied to the interface board to assure that it remains secure.

The SSC can now be attached to the interface board and installed in the enclosure. This is done using three plastic bolts and two nuts. Note that at least one of the plastic bolts should extend through the interface board and into the SSC. This will prevent the SSC from slipping out of the interface board during operation.

The last step in installing the SSC hardware is to setup the communication link between the SSC and the processing system. This communication operates through the SSC interface board via the 3x1 communication header. Interface between the processing system and interface board is handled by a custom cable utilizing an RS232 to USB adapter. To assure that the interface cable is as small as possible the D-sub connector on the adapter is removed. A male Futaba-J connector is then wired to the adapter to be used for the RS232 communication. Last, the open set of connections on the 5x2 connector, described in Section 3.3.4, are wired to the USB communication side of the adapter. Note, that both the GPS and SSC communication adapters are wrapped with electrical tape to reinforce their connections and protect their circuitry.

Now that the SSC hardware is properly installed the two Futaba-J power outputs from the processing system can be plugged into the SSC interface board. Any loose cables that may drift into the fan or become entangled should now be ziptied. The enclosure can now be sealed as described in Section 3.3.1.

3.3.6 Chassis

The next step in assembling the helicopter testbed is to outfit the platform with the custom aluminum chassis. The chassis was designed with two main goals: to reduce the vibrational forces on the enclosure and sensors, and to protect the hardware.

The chassis is strictly designed from aluminum tubing with the exception of the mounting plate for the enclosure, also aluminum. This design allows the chassis to take advantage of the lightweight nature of aluminum and the tensile strength of tubular constructed materials. Aluminum is also a non-ferrous metal and will not adversely affect the magnetometers located within the IMU. The chassis should be assembled by a fabrication shop capable of welding aluminum and bending 1/2 inch aluminum tubing. Specific measurements for every component of the chassis can be found in Appendix C.

Once the chassis has been fabricated the first step in assembly is to install all eight rubber isolation mounts. The larger of the two isolation mounts is mounted to the top of the chassis and will ultimately be mounted to the helicopter. The smaller of the two mounts is installed on the four tabs located within the chassis. Next, the enclosure mounting plate should be installed. This is a 300x200 mm aluminum plate with eight mounting holes. Four of the mounting holes are used to connect the plate to the isolation mounts. Note, that you should not over tighten the isolation mounts when bolting down the plate. If the mounts are over tightened the rubber will twist within the mount and it may tear. The other four holes in the mounting plate are used for installing the IMU mounting brackets.

The IMU mounting brackets are two small S-shaped brackets that allow the IMU to be mounted away from the rest of the testbed's hardware. Each mounting bracket has two holes on one end, corresponding to the enclosure plate's mounting holes, and one hole on the other end used to mount the IMU. Installation of the brackets includes installing three small strips of double sided servo tape between each bracket and the mounting plate and then looping a ziptie through the mounting holes and pulling it tight.

Now that the chassis is assembled the enclosure can be installed. The enclosure is installed using four small pieces of Velcro placed on the bottom of the enclosure and top of the mounting plate. Note that the enclosure's installation should allow for two inches of clearance between itself and the square tubing at the front of the chassis. This area will later be used to mount the processing system's battery. Effort should also be placed in assuring that the enclosure is centered, as much as possible, under the main shaft of the helicopter.

The last step in installing the chassis is to mount it to the helicopter. Two small adapter plates were manufactured to allow the Joker Maxi-II to mount to the custom chassis. These plates simply bolt to the stock mounting holes on the helicopter and then extend off of the side of the helicopter to be mounted to the chassis. This design supports remounting of the chassis to

various helicopters. Again, assure that when mounting the helicopter to the chassis that the rubber isolation mounts are not over tightened.

Now that the helicopter, chassis, and enclosure are integrated the helicopter servos can be wired to the enclosure. Servos can simply be plugged into the output block, block 6 in Figure 21, of the SSC interface board. Make note that the wiring harness for the servos may not be long enough to reach the enclosure. If this is the case an appropriate size servo cable extender can be purchased or hand made. Since the USL testbed helicopter utilizes a heading hold gyro, two servo connections are required for tail rotor control. In order from pins one through nine (left to right) the connections are: left servo, front servo, throttle, heading hold gyro (3 wires), heading hold gyro (1 wire), right servo, pan servo, and tilt servo. Note that the signal wire, usually white on standard servos, must be on the top row of the connectors.

3.3.7 IMU

Now that the IMU mounting brackets have been installed on the chassis the IMU hardware can be installed. The IMU hardware consist of the Microstrain GDMG-X1 IMU and a 6V 2.5A voltage regulator, see Figure 22.

The IMU should be mounted to the chassis brackets using two brass bolts, washers, and nuts. Brass is used to mount the IMU as it is a non-ferrous metal and will not adversely affect the IMU. Assure that the communication port for the IMU is pointed towards the front of the helicopter. Verify that the IMU's X axis, identified by the communication port, is running as parallel as possible to the helicopter's frame. This will assure that heading measurements are accurate.

The supplied cable can now be attached to the IMU. To adhere to our standard of power connecters the supplied connector for the IMU cable is removed and replaced with a three pin

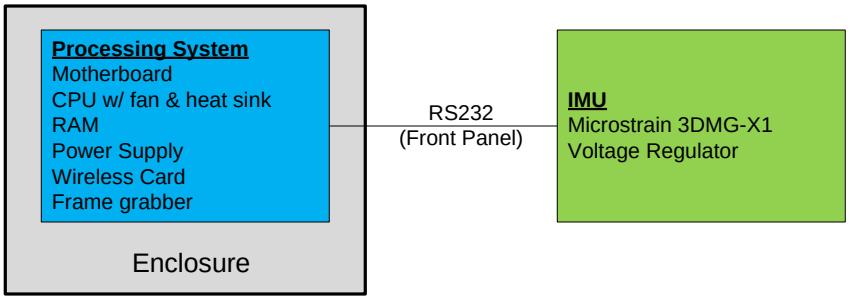

Figure 22: IMU Hardware Connection Diagram

female Futaba-J connector. Power to the IMU is supplied through the Medusa 6V regulator. The regulator output is fitted with both a female Futaba-J connector and a male Futaba-J connector. The male output is used to supply power to the IMU and the female output is used to supply power to the camera, discussed in Section 3.3.9. The input for the regulator is fitted with two male Futaba-J connectors. One of the input connectors plugs directly into the 12V power output from the SSC interface board. The second input connector is plugged into the female Futaba-J connector used to power the video transmitter discussed in Section 3.3.9.

3.3.8 Pan/Tilt

The pan/tilt for the testbed mounts directly to the front of the chassis utilizing the chassis's two small strips of square tubing. This location allows the pan/tilt to take advantage of one layer of vibration isolation. The pan/tilt can be assembled and mounted any time after the chassis has been complete.

The pan/tilt assembly consist of two servo brackets, two camera brackets, four gears, two servos, and three servo arms, see Figure 23. The schematics for the all four brackets and the models of the gears can be found in Appendices D & A respectively. Note that all three servo arms are supplied with the servos.

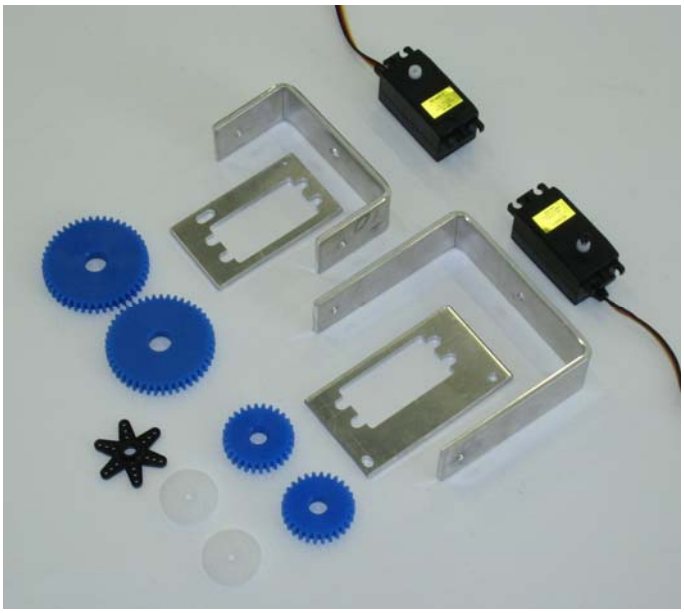

Figure 23: Pan/Tilt Hardware

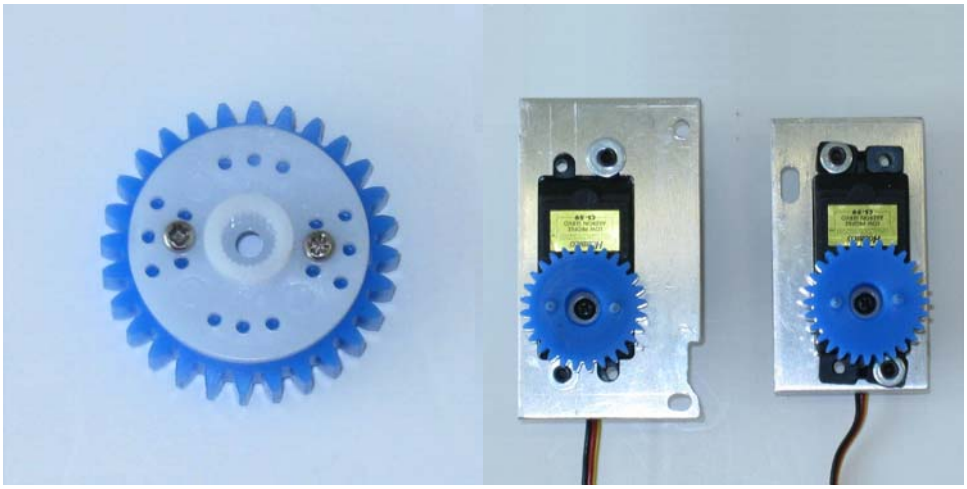

Figure 24: Assembled Pan/Tilt Gears (left) and Assembled Servo Brackets (right)

The first step in assembling the pan tilt is to mount the servos to there respective brackets. This first consist of inserting the four rubber standoffs, supplied with the servo, into each of the four mounting holes. Each servo is then mounted to the bracket using two bolts, two nuts, and four washers, see Figure 24 for orientation.

The next step in assembly is to tap the two small mounting holes on the large camera bracket. These 2.5mm holes should be tapped for a 3mm metric bolt. These mounts will later be used to mount the left servo bracket. Once this is complete the two small gears should be mounted the servos.

Mounting the gears to the servos first requires that the gears be mounted to the supplied servo arms, see Figure 24. This is done by placing the servo arms, face down, on top of one of the small gears and then screwing two small screws through the servo arm into the gear. Assure that the screws do not protrude out the other side of the gear. The gears can now be placed on the servos and fastened with the servo supplied bolt. Note that all four gears were purchased with small mounting extrusions on one side. These extrusions were sanded off to assure that flush mounts could be achieved.

The camera brackets can now be assembled as shown in Figure 25. This is done using two bolts, seven washers, and two nylon nuts. Note that the bolts should only be tightened enough to ensure that the brackets are secure and should not restrict their rotational movement. Once the camera brackets are assembled the two large gears should be installed. Both gears are installed and secured using epoxy. To ensure that the epoxy adheres correctly to both the gear and mounting location it should be lightly sanded with rough sandpaper. The first gear is epoxied

Figure 25: Assembled Pan/Tilt Camera Brackets

to the inside of the small camera bracket over the bolt that extends inward, see Figure 26. This is done by placing a thin coat of epoxy on the inside of the small camera bracket and then sliding the gear over the washers. The washers should fit snugly inside of gear. The gear should then be clamped to the bracket and allowed to dry for several hours.

The second large gear is epoxied to the top of the large camera bracket, see Figure 26. A thin layer of epoxy should be applied to the outer side of the bracket and then the gear should be clamped in place. Assure that the mounting hole in the bracket is aligned with the center hole of the gear. The epoxy should then be allowed to set for several hours.

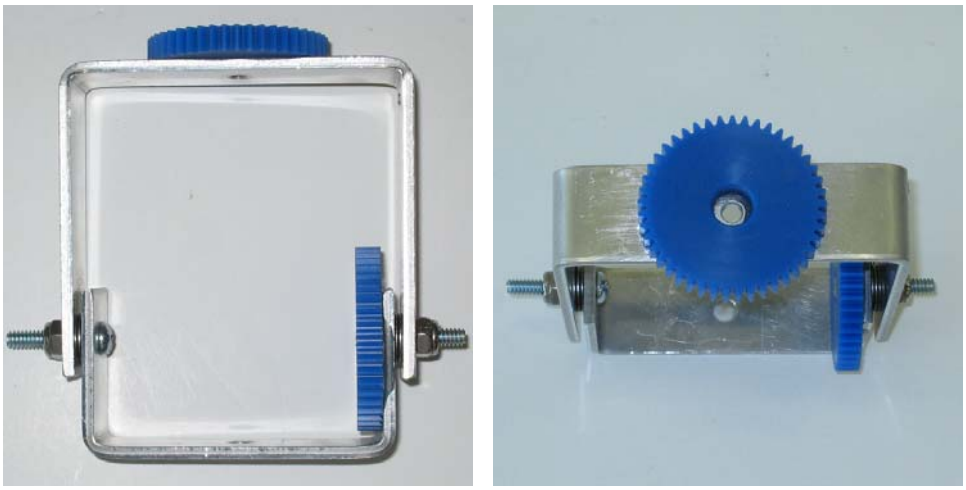

Figure 26: Assembled Pan/Tilt Camera Brackets with Gears

Once the gears are installed the left servo bracket should be installed. This is done using two small 3mm bolts and a small amount of loctite. Assure that the length of the mounting bolts do not interfere with the motion of the lower camera bracket. Before tightening down the servo bracket the servo gear and bracket gear must be correctly meshed. This is done to prevent binding during rotations and is done by placing a sheet of paper between the two gears and then pressing them together. Once the bracket is tightened the sheet of paper can be removed.

The last step in assembling the pan/tilt is to mount the top servo bracket. To do this the hardware must be mounted to the chassis. This is done using one large and one small bolt, two washers, the remaining servo arm, and two nylon nuts. The large bolt is first passed through the upper camera bracket and gear. The servo arm, hub towards the gear, and a washer are then slipped over the bolt. The servo arm acts as a brace for the bolt and assures that it is centered in the middle of the gear. The bolt is then passed through the square tubing on the chassis and then through the large hole in the top servo bracket. Note that this bolt should only be tightened enough to secure the pan/tilt. Over tightening this bolt will prevent the pan/tilt from moving and will ultimately damage the servos. The small mounting hole on the top servo bracket should then be fasted to the chassis. At this point one should assure that the two upper gears are correctly meshed and then fully secure the small mounting bolt. Figure 27 details the fully assembled pan/tilt.

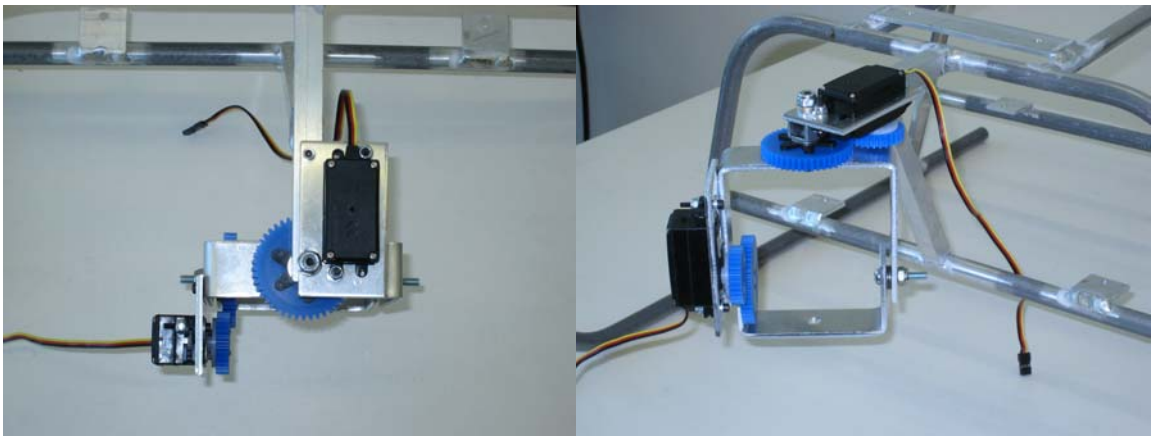

Figure 27: Fully Assembled Pan/Tilt Mounted to Chassis

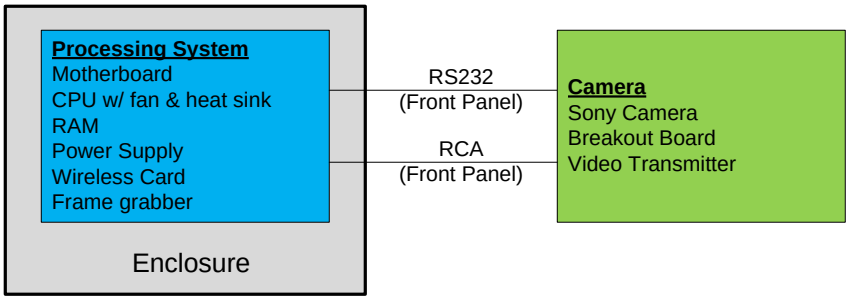

Figure 28: Camera Connection Diagram

3.3.9 Camera

The Camera hardware consist of the Sony block camera, interface board, video transmitter, and a custom RCA splitter, see Figure 28.

Assembly of the camera hardware first consists of integrating the interface board with the camera. This is first done by attaching the two supplied camera cables to both the interface board and camera. The remaining two cables are then plugged into the interface board and will be wired later in this section. Now that the interface board is fully connected it can be mounted to the camera for security and modularity. This is done using electrical tape to secure the interface board to the top of the camera. The camera and interface board are then heat shrunk together, see Figure 29. Assure that both sets of unconnected cables are still accessible after heatshrinking.

Next, the camera cables must be wired to supply power, control, and data output. Identification of the appropriate wires can be done using the interface board's supplied wiring

Figure 29: Assembled Camera Module

guide. The two power wires are first wired to a male Futaba-J connector and then plugged into the female output of the 6V regulator described in Section 3.3.7. The signal ground, transmit, and receive wires are connected to a 9 pin D-sub female hub and then plugged into one of the available serial ports on the enclosure. Last the two data outputs, signal and ground, are wired to a male RCA connector.

Due to wireless bandwidth constraints it was decided that the camera's video be transmitted via a video transmitter. To interface the camera with both the video transmitter and on-board processing system an RCA splitter can be made or purchased. This splitter consists of a female phono jack wired to two male RCA connectors. The phono jack is plugged directly to the camera's RCA connector and the two male RCA connectors are plugged into the first phono jack on the enclosure and the input for the video transmitter.

The video transmitter is mounted to the enclosure's mounting plate just behind the enclosure using servo tape. The high gain 900MHz antenna is then connected and protrudes out the back of the chassis. Power for the video transmitter is supplied by a coaxial (size M) to female Futaba-J cable which is plugged into the free male connector described in Section 3.3.7.

3.3.10 Radio Control Receiver

The radio control receiver is responsible for transmitting control signals from the safety pilot to the SSC and is responsible for granting or denying control to the on-board processing system.

The receiver is mounted just in front of the network ports on the enclosure's mounting plate. Servo tape is used to secure the receiver to the plate. The last step of this installation is to hook the receiver channels to the enclosure's input channels (block 4 in Figure 20). This requires seven male to male servo cables. In order from pins one through nine (left to right) on the SSC interface board the connections are: channel 8, 1, 2, 3, 4, 5, and 6. For power, the DSC channel on the receiver is connected to the servo power switch on the Joker Maxi-2. Note that contrary to the output channels, described at the end of Section 3.3.6, the signal wires must be on the bottom row of the connectors.

3.3.11 Battery

The last pieces of hardware to be mounted to the USL testbed are the batteries. The battery hardware consists of a 37V 10Ah LiPo battery, 11.1V 4.2Ah LiPo battery, 11.1V 0.5Ah LiPo battery, and 4.8V 2.0Ah NiMh battery, Figure 30.

The 37V battery is responsible for powering the platform's main motor and is composed of two heat shrunk 18.5V 10Ah batteries. This battery fits into the frame of the Joker Maxi-2 and is secured from the rear by a small Velcro strap. To supply the platform's Electronic Speed Controller (ESC) with the required 37V a small adapter cable was manufactured in-house. This cable puts the two 18.5V batteries in series and supplies the correct voltage to the ESC.

The 11.1V 4.2Ah battery is used to power both the GPS receiver and the processing system which in turn is responsible for providing power to the remaining sensors. This battery is mounted, using Velcro, to the enclosure's mounting plate. It is placed just between the enclosure and the square tubing towards the front of the chassis. Due to the location of this battery a small extension cable must be used to reach the SSC interface board. For safety, a small low voltage alarm is wired directly into this extension cable. This alarm constantly monitors the battery and warns the operator when the voltage is reaching a critical level.

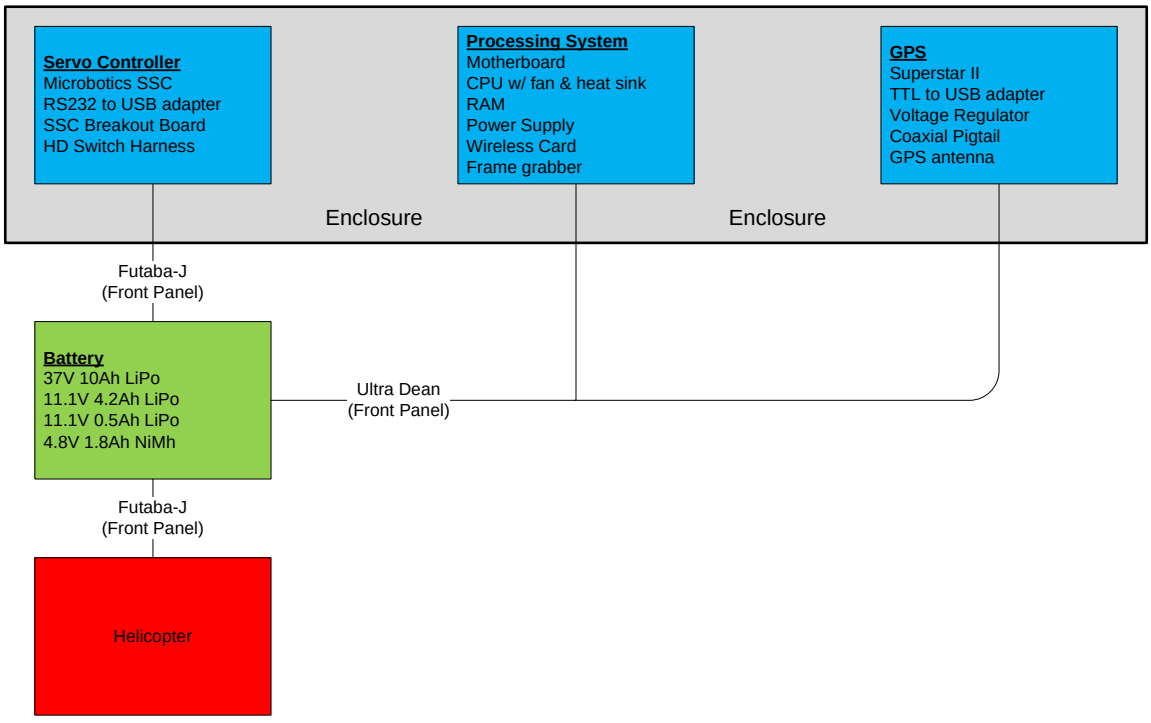

Figure 30: Battery Connection Diagram

The 11.1V 0.5Ah battery is solely used to power the SSC. This battery is equipped with a 3 pin male Futaba-J connector and is mounted to the top of the enclosure using Velcro. The battery is then connected directly to a HCAM2761 HD power switch. This switch is mounted to the chassis using two zipties. The power output connector of the switch is then connected to the SSC power connector on the enclosure's faceplate.

Last, the 4.2V 2.0Ah NiMh battery is solely used to power the servo actuators throughout the testbed. This includes powering the platform's control servos and the pan/tilt servos. This battery is plugged into the Futaba radio receiver via the Maxi Joker-2's servo power switch. Power is then naturally routed from the radio receiver to the SSC interface board where it is distributed to all servo connections. A complete assembly of the testbed is detailed in Figure 31.

Before concluding this section, it is noteworthy to mention that LiPo batteries can catch fire and explode if not handled properly. This includes insuring that the individual cells do not fall below 3.0V per cell or rise above 4.2V per cell. If a cell swells or smokes it should immediately be considered a fire hazard and disposed of properly.

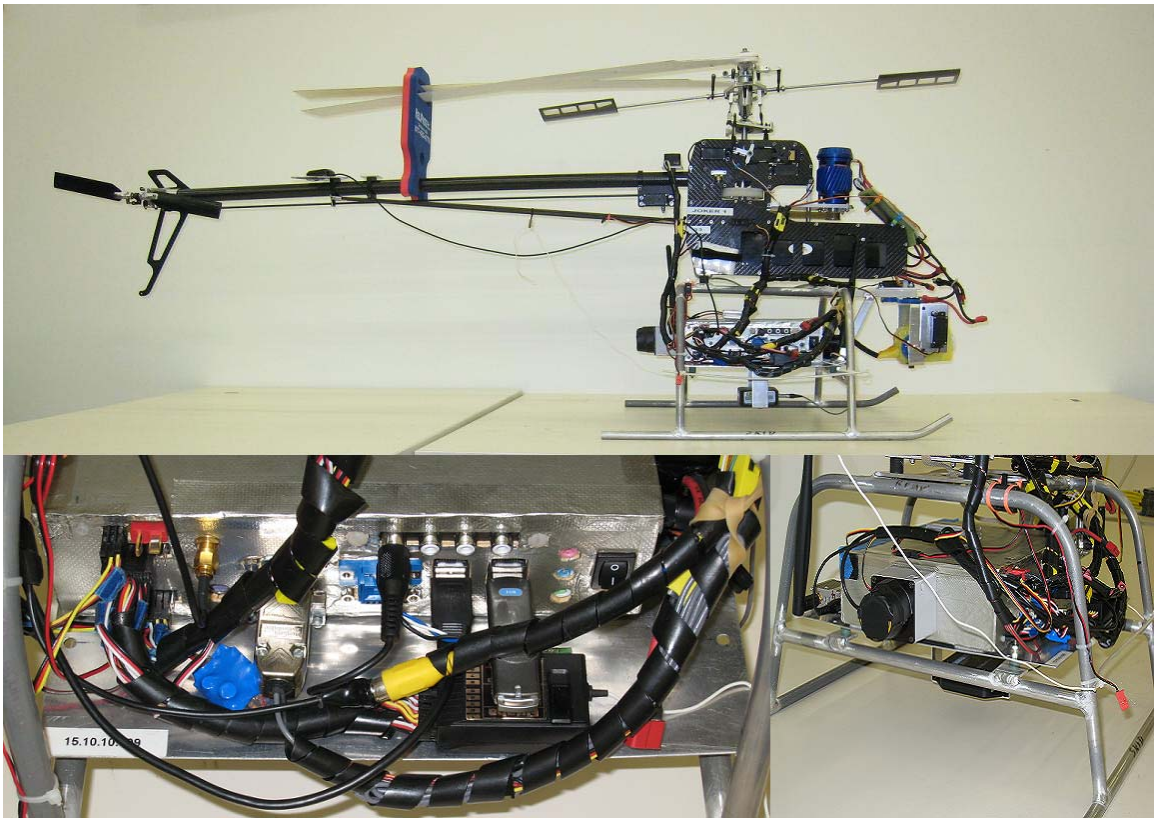

Figure 31: Mosaic of the Completely Assembled USL Testbed Helicopter

Chapter 4
Software Architecture

Although the hardware is an integral part of designing an autonomous vehicle testbed it is of very little use without corresponding software. The software architecture described in this chapter covers the overall structure of the software developed for the USL testbed. It also discusses the details of the individual processes and how these processes interact to form a single entity. To ensure comprehension most sections are supported by high level pseudocode available in Appendix G. Note that this section does assume some level of comprehension with OS installation and operating in the Linux environment.

4.1 Operating System

The OS is the backbone of any software architecture and provides a base for all supporting software. Although almost any modern OS would be sufficient, the USL testbed has been strictly operated by Linux distributions. Linux was chosen due stability, ease of modification, and heavy use in both the academic and industrial arenas. The utilized distributions have included multiple versions of Slackware, versions 10.0 through 12.0, and Gentoo. The currently utilized distribution, and the one detailed in this research, is the Slackware 12.0 distribution. Slackware was chosen due to ease of installation and ability to install and boot from a USB thumbdrive. Also note that for the USL testbed the OS operates completely out of a RAM drive. Due to the physical limitations of RAM, this requires that the OS be minimal in size to allow for sufficient RAM for operating processes.

4.1.1 Development System

To ensure that a version of the operating system and supporting software could easily be modified and tested, a developmental computer was created. This developmental system is an exact replica of the enclosure built in the previous Chapter. The only modifications are the

installation of magnetic drive via the 40 pin Integrated Drive Electronic (IDE) slot and a Compact Disc Read-Only Memory (CD-ROM) drive via the 44 pin IDE slot.

Setting up a development system has three advantages. First, it provides a backup of the foundational software and is available in the event that the USB thumbdrive is damaged. Second, it allows us to develop and test new software on an exact replica of the testbed's enclosure without having to constantly copy a new software image onto the thumbdrive. Last, it allows us to co-develop the software for multiple testbeds that utilized this enclosure, which will briefly mentioned in Chapter 9.

4.1.1.1 Installation

Once the development system is assembled, the magnetic drive is formatted for two 5 gigabyte partitions. The first partition, now referred to as installation *A*, is installed with Slackware's base and networking packages. The second partition, now referred to as installation *B*, is installed with Slackware's base, applications, developmental, kernel source, libraries, and networking packages. This setup allows us to have a complete installation for development and a minimal installation that can be copied to a thumbdrive. To limit the overall size of the OS on the thumbdrive several sub-packages were removed during installation. This is accomplished using the menu prompting option for installation and simply required unchecking the appropriate boxes. A list of the removed packages is available in Appendix F. Note that the majority of the removed packages are mail programs, raid software, and printer drives. These were deemed useless for the current configuration and thus wasted space.

Installation, via a Slackware installation DVD, first required setting up the file system. Our installation used the ext2 file system and a 4092 byte inode. Each installation utilizes a single partition which is defined to be the root of the file system. Once this is complete, the desired packages can be selected, as mentioned above, and installed. From this point on in the installation all of the default values were selected. Note that the networking configuration is not setup in installation and will be configured later in this chapter.

4.1.1.2 Kernel Setup

Although Slackware provides a pre-built 2.6 kernel it does need to be recompiled to support the computer's wireless card. Although kernel compiling and installation are beyond the scope of this work, an overview of the process is described.

The kernel source, for the large Slackware installation, is automatically installed and placed in */usr/src/*. From within the kernel source directory, the *make menu* command can be performed to provide a graphical interface for updating the kernel. The only required modification to the kernel is to include the Intel Pro 2200 package as a module. The kernel and modules can now be compiled and installed.

Note that the default kernel provides much more support than is necessary to fully operate the testbed's hardware. The kernel can be configured to remove these unnecessary packages thus providing a smaller OS. To correctly operate the USL testbed any kernel design must support serial devices, USB devices, ACM devices, SSH, networking, and wireless devices.

Although the kernel now supports the installed wireless card, it will not function correctly without the appropriate firmware. This firmware is publicly available from Sourceforge (*sf.net*) and should be downloaded and installed into the */lib/firmware* directory.

Now that the large installation of Slackware is prepared, the kernel, kernel modules (located in */lib/modules*), and firmware must be copied from installation *B* to installation *A*. This can be done by mounting partition *A* and copying the kernel image, typically *bzImage*, modules, and firmware to their respective locations on installation *A*.

Note that the boot loader should be edited to provide access to both partitions. This should be done by editing */etc/lilo.conf* and assuring that an option exists for both installations using the new kernel. Once complete the changes should be committed to the Master Boot Record (MBR) by running the *lilo* command.

4.1.1.3 Networking Setup

The last step in configuring the base software is to setup the communication. To allow the system the freedom to operate without an access point it was decided that communication be performed using an ad-hoc network. It was also desired that an automated network reconfiguration be available on the testbed. To support this, the open source development of

Mobile Mesh was selected (available at *www.mitre.org/work/tech_transfer/mobilemesh*). This software automatically determines the most appropriate route to any client on a dynamic network.

Once Mobile Mesh is downloaded it should be uncompressed and compiled. Installation instructions are provided with Mobile Mesh and typically only require three commands to compile and install (*make depends*, *make*, *make install*). It should be mentioned that some compliers will display error messages when attempting to compile Mobile Mesh. This is due to redundant statements placed within two C classes. These can be repaired simply by deleting the four redundant statements (*Debug::* on line 41 of UtDebug.h and *String::* on lines 103-105 in UtString.h).

Now that Mobile Mesh is installed it should be configured to operate on the development system. This first requires modifying the */etc/Mobile Mesh/mmrp.conf* file. This file contains a single line that identifies which networking device to utilize, typically *eth0* or *eth1*. This line should be modified to point to the wireless device's identifier which can be obtained by running the *iwconfig* command. Next, the Internet Protocol (IP) address, broadcast, and netmask mush be setup. This can be done through the *ifconfig* command. The essid and mode, which must be Ad-Hoc, should also be setup using the *iwconfig* command. Note that the essid must be set the same for all vehicles and computers requiring intercommunication and should not be the same as any wireless router within operating range. The specific networking setup for the USL testbed can be seen in Table 6. Last the Mobile Mesh software should be initiated using the *mmdiscover –i [device]* and *mmrp* commands respectively.

Through experimentation it was determined that the wireless identifier is inconsistent from motherboard to motherboard. To account for this a small script is added to */ect/rc.d/rc.local* and is performed towards the end of each bootup cycle. This script determines the wireless identifier, modifies the *mmrp.conf* file, setups up the IP, broadcast, netmask, essid, and mode, and then initiates Mobile Mesh. This provides wireless connectivity immediately upon completion of the boot process.

Table 6: USL Testbed's Network Configuration

Ethernet Port	Essid	IP	Mode	Netcast	Broadcast
eth1	Vision	15.10.10.109	Ad-Hoc	255.255.255.0	15.10.10.255

Now that the development partition is setup the appropriate files must be copied from installation *B* to installation *A*. This includes copying the */etc/Mobile Mesh* directory, *mmdiscover* and *mmrp* commands from */bin*, and any scripts that setup Mobile Mesh on boot to their corresponding locations.

4.1.2 Booting from the USB

Once installation *A* is fully functional its image can be copied to a USB device. This USB device will store all of the software necessary to initialize the testbed.

The first step is to create an image of installation *A*'s file system. This is done by creating an empty file system large enough to hold the entire partition. This can be done using the *dd* command to create the file and then formatting the file for ext2 using the *mke2fs* command. This file system should then be mounted using the *-o loop* option. The file system can now be loaded with data. This should be done by copying, *cp -a*, the bin, boot, dev, etc, home, lib, mnt, root, sbin, usr, and var directories from the small partition to the mounted file system. The sys, proc, and tmp directories should also be created in the file system but should not be copied. To assure that the boot process is accomplished without error two files must be edited within the new file system, */etc/rc.d/rc.S* and */etc/fstab*. Fstab must be changed to prevent it from attempting to mount a hard drive. This can be done by changing the root mount point from */dev/hda1* to */dev/fd2*. This will force the OS to believe it is operating off of a floppy device. *Rc.S* must be edited to ensure that the ramdisk loads correctly and that the system does not attempt to utilize swap memory. This is done by removing the entire section in *rc.S* responsible for checking the root partition. This section begins with the "*# Test to see if the root partition is read-only...*" line and concludes with "*fi # Done checking root file system*"

The new file system is now completely populated and should be unmounted and compressed using gzip and the *-c -9* options. This compressed image should be loaded onto a USB thumbdrive, along with the kernel image.

The last step in creating a bootable USB device is to install the bootloader. To prevent from having to format the USB partition, the Syslinux bootloader was chosen for the USL testbed. This specific bootloader supports booting from File Allocation Table (FAT) partitions which are the most common default formats for USB thumbdrives. Installation first includes creating a *syslinux.cfg* file on the USB stick. This file contains the booting and kernel

instructions. For clarity the USL *syslinux.cfg* file is provided in Appendix G. Last the *syslinux* command should be performed on the device, i.e. *syslinux /dev/sda1*.

The USB is now fully configured to boot into a ramdisk on any machine supporting USB boot. Note that booting from a USB device is an option that must be setup in the Basic Input/Output System (BIOS) of the motherboard.

4.1.3 Operating System Comments

It should be noted that the bootup time is directly associated with the size of the OS. The configuration shown here will take between 180 and 200 seconds to boot. This bootup time can be significantly reduced if the OS is loaded using the USB 2.0 protocol. Although the BIOS for the USL motherboard only supports the USB 1.1 protocol, it is still possible to significantly reduce the bootup time. This can be done by creating a custom initrd that only loads the absolute minimal software for boot. Once booted the OS, which does support USB 2.0, can uncompress and load the remaining sections of the OS. This method has been tested on the USL testbed using the Gentoo distribution and is capable of booting the full OS in approximately 30 seconds.

As the custom initrd is not required for the testbed to operate correctly its setup is not discussed. The inclusion of a custom initrd is only discussed as to provide information into alternative methods.

4.2 Source Code Architecture

Source code, for the purpose of this work, describes all of the in-house developed software used to interface with and control the USL testbed. All of the source code for the USL testbed is developed in the C programming language.

A graphical depiction of the architecture for the source code utilized on the USL testbed can be seen in Figure 32. The levels within the figure describe the directory structure, level one being the upper most directory. The color breakdown of Figure 32 is as follows:

- Green – folder name
- Blue – source and header files within that folder
- Yellow – miscellaneous files that will be individually described in the text
- Orange – subfolders

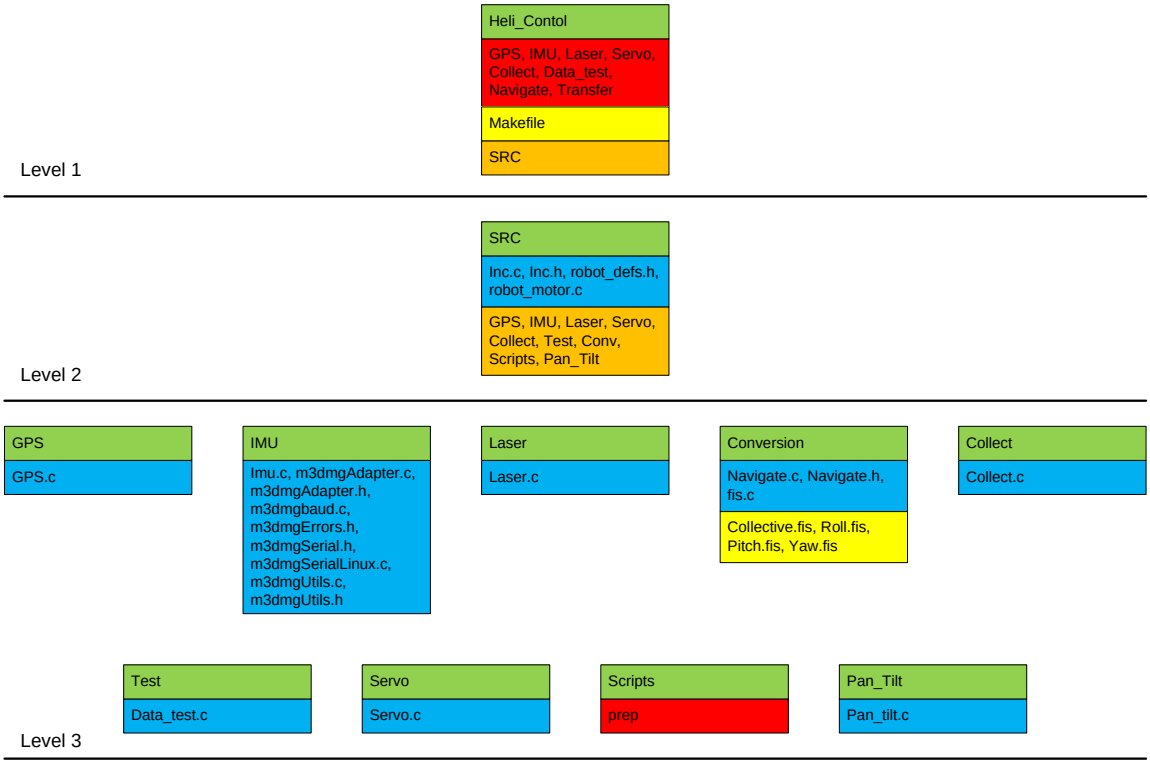

Figure 32: Directory and File Structure for the USL Testbed

From conception the software for the USL testbed was designed to be highly modular and to support an operating structure that could be dynamically modified. Modularity in the design allows for code reuse, quick integration, and ease of understanding. Dynamic modification simply means that the system is able to remove or add functionality during operation. To support these design requirements, software is developed as a set of processes that run concurrently and pass information through shared memory structures. By utilizing this method of development a monitoring process can start and stop processes as needed by the testbed. An example of this would be a situation where the process controlling the vehicle is operating at a lower than optimal rate. At this point a monitoring process could shutdown lower priority processes freeing up both memory and CPU for the higher level process. Although this work does not utilize the ability to dynamically alter the software structure, support for it is maintained throughout software development.

It should also be mentioned that all of the developed software is written and tested on a soft real-time OS. This means that the OS attempts to minimize delay but does not guarantee run time. Although there is no guarantee of run time there is a reasonable expectation that process

will be handled efficiently and should operate within some boundaries. Using this assumption many of the processes were designed to attempt to operate within a given window of time. If they fail to operate within this window they report a warning detailing how excessive their delay is. This is performed using sets of timestamps. Processes request a timestamp before sleeping and then sleep for a fraction of the desired delay time. When the process wakes it compares the current system time to the retained timestamp. If the amount of time passed is within the operating threshold the process performs its required operation. If the amount of time passed is less than the threshold the process again sleeps for a fraction of the remaining time. Last, if the amount of time exceeds the threshold a warning is presented to the operator.

Source code for the testbed is typically kept on the ground station's laptop, a Dell Latitude D820. This laptop utilizes a Fedora Core 6 distribution of Linux and is the primary source for non-OS software modification. During field testing the source code can be modified, compiled, and uploaded directly to the testbed. Note that the GCC compiler, version 4.1.0, is utilized to compile all of the software before it is uploaded to the testbed.

4.2.1 Level 1

The Level 1 structure of the software architecture is the root of all source code utilized on the USL testbed. This folder contains a makefile (*Makefile*), one folder (*SRC*), all compiled executables (*GPS*, *IMU*, *Laser*, *Servo*, *Collect*, *Data_test*, and *Navigate*) and one script (*transfer*).

The makefile is responsible for compiling all of the executables and is operated by typing the *make* command from this directory. This process will enter all specified subfolders and individually compile each process. Once this is complete, all of the compiled executables will be placed in the *Heli_Control* directory of Level 1. Although the design of a makefile falls outside the scope of this work the utilized makefile on the USL testbed is provided in Appendix G for completeness.

When desired, all the necessary executables, scripts, and fuzzy inference files, discussed later in this chapter, can be uploaded to the testbed utilizing the *transfer* script. This script simply copies the specified files to a predetermined remote destination. The contents of this script can also be viewed in Appendix G.

4.2.2 Level 2

The second level of the source code structure contains a single directory, SRC. This directory and its subfolders contain all of the developed source code for the USL testbed. The only source files directly contained within this folder are the *Inc.c*, *Inc.h*, *Robot_defs.h*, *Robot_motor.c* files. Note that pseudocode for each of these files is available in Appendix G.

The *Inc.c* file contains only two functions which are responsible for retrieving the current time and determining the amount of time passed between two timestamps. Note that for Linux, time is determined by the amount of time that has passed since midnight on the 1st of January of 1970. For accuracy, all time is computed using microsecond precision.

The *Inc.h* file contains declarations of the two functions in Inc.c and all of the shared memory structures utilized throughout the source code.

Robot_motor.c contains four functions responsible for communicating with the SSC. These four functions are responsible for setting up a connection between the processing system and the SSC, sending the desired servo positions to the SSC, calculating the SSC checksum, and disconnecting from the SSC.

The last source file, *Robot_defs.h*, contains the four function declarations for *Robot_motor.c* and multiple constant definitions. These constants define the pulse width limits and neutral values for the servos and several values for setting up SSC communication. The SSC communication values include the number of servos being controlled and two masking values for masking higher order bits.

4.2.3 Level 3

Level 3 is the lowest level of the source code architecture and contains all of the remaining source code for the USL testbed. Note that each directory within this level, with the exception of the scripts directory, represents a distinct executable that will be created by the makefile in Level 1. This allows modification of a single process to be constrained to a single folder.

Although the specifics of each individual file will not be described here, Appendix G does provide details through comments and pseudocode. The remainder of this chapter will detail the individual processes created by these files.

4.3 GPS Process

The GPS process is responsible for reading and parsing data from the GPS receiver as well as placing the data in shared memory. This process will also inform the user in the event that a significant change has occurred in the accuracy of positional data. Significant changes in accuracy are determined by the type of lock acquired and are always one of three states, no lock (zero), lock without WAAS correction (one), lock with WAAS correction (two). Note that the firmware for the GPS receiver is configured, using Starview, to continuously output the NMEA GPGGA message at 5Hz.

The GPS process's main goal is to continually update eight individual values in shared memory. The first seven of these values represent the latitude, longitude, latitude direction, longitude direction, altitude, number of satellites, and type of lock as reported by the GPS receiver. The last value recorded to shared memory is a count variable. This variable increments with each newly received string of data from the receiver. This counter allows processes accessing the GPS data's shared memory to determine if the information available is newer than the information already gathered by that particular process.

The order of operation for the GPS process is first to setup a serial communication connection with the GPS receiver. Once the connection is established the process will create a shared memory location for the GPS data and a corresponding semaphore for controlling mutual exclusion. This semaphore, as well as all semaphores discussed in this chapter, assures that no other process can access a particular piece of shared memory when it is being accessed by any another process. This prevents processes from retrieving partially modified data. The GPS process then enters an infinite loop where it will continuously read, parse, and store GPS data. The only modification performed on the parsed data is that the latitude and longitude are divided by one hundred. This modification formats the latitude and longitude into decimal hours. After each successful post to shared memory the process sleeps for 0.18 seconds. Make note that specific values for sleeping are simply fractions of the time between available data, detailed in Section 4.2.

4.4 IMU Process

The IMU process is responsible for accessing the Microstrain IMU. This process gathers Euler angles, angular rates, and accelerations at approximately 80 Hz and posts their values, along with an associated time stamp, to shared memory.

The order of operation of this process is to first setup a serial connection to the IMU. Once this is performed this process will create the appropriate shared memory and semaphore. The process then enters and infinite loop where it will request the appropriate data. Once the data is received the system time is requested. The sensor readings and timestamp are then posted to shared memory. It should be mentioned that, unlike other processes described in this chapter, there is no sleep request in the main function. This was deemed unnecessary as a sleep process is already built into the data acquisition function supplied by the manufacture.

As the manufacture supplied source code is not developed by USL it will not be detailed in this work but is available directly from Microstrain. The main function which interfaces with the Microstrain supplied source code is detailed in Appendix G. To fully utilize the IMU a single function is added to the manufacture supplied source code. This additional function is also detailed in Appendix G.

During testing it was discovered that the manufacturer supplied source code contained an error that periodically caused the device to fail. This failure was experienced when the IMU was not the first device connected to by the OS or if the device was not properly disconnected from. This error was located in the *receiveData* function in the *m3dmgSerialLinux.c* file. Line 319 which reads *n = read(3, &inchar, 1);* is attempting to read data from a hard coded file descriptor for the IMU. This line is changed to read *n = read(portHandle, &inchar, 1);* which attempts to read data from the file descriptor that is set when the device is opened.

4.5 Laser Process

The laser process is responsible for gathering range data from the laser to any object below the vehicle. This information will be used for both takeoff and landing.

Testing of the laser reveled that it is prone to shutting down in the event of heavy external light or after an intense shock. This is noticed during several consecutive takeoff and landing attempts and in direct sunlight. Although these procedures are designed to prevent damage to the laser they do not restart the laser to a functional state when the disturbance has resided. To

account for these issues the source code is written with the ability to shutdown and reset the laser in the event of a failure.

The order of operation of the laser process is first to setup a serial communication with the laser. Since communication with the laser is performed via USB, this is accomplished through one of the available ACM ports (*/dev/ttyACM?*). Once communication has been established the process creates both shared memory and a semaphore for the laser data. The laser is then restarted to assure the current state of the laser is known. The process will then request that the laser output range data continuously. This will provide the laser process with range data at 10Hz. Since the laser is capable of retrieving range data at angles far beyond what is currently needed, the data request indicates that only the 16 degrees directly below the laser should be retrieved. This reduces computational time and the required storage area. The process then enters an infinite loop where it will continuously parse the incoming data and place it in shared memory. After each successful receipt of data the process sleeps for 0.09 seconds.

If, at any point, the laser fails to function correctly the software will attempt a recovery process. The specifics of this process depend on the specific failure message received. Generally, a failure will cause the software to request that the laser restart. If the restart is unsuccessful the user is notified of the failure. Note that unrecoverable failures will cause the vehicle to refuse autonomous landing, discussed further in Section 4.10.4.

4.6 Servo Process

The Servo process is responsible for generating packages which will request that the SSC move individual servos to particular locations. It is also responsible for retrieving and parsing state data from the SSC. This data allows processes to determine if the human safety pilot is in control or if the on-board processing system is in control.

Unlike previous processes, the servo process creates three separate shared memory locations and three corresponding semaphores. The first two shared memory locations are dedicated to the position requests for the helicopter's servos and the pan/tilt's servos. This is done to allow pan/tilt controlling software and vehicle controlling software to be separated. This allows software to move the pan/tilt without blocking the process that updates the vehicle's servo commands. The third shared memory location is dedicated to SSC state data. This shared memory location stores a single variable which identifies who is providing vehicle control, the human operator or the on-board computer.

This process first sets up the serial communication, shared memory, and semaphores. Once this has been performed the process loads shared memory with neutral values for all of the servos. This prevents residual data in the shared memory from causing unspecified movements in the servos. The process then enters its infinite loop. Once inside the loop the process will poll both sets of shared memory for servo positions and then generate a data package. A complete list of details for SSC package structures can be found in the manufacture provided manual. Once the package is sent to the SSC, the process polls the input buffer for state input from the SSC. Since the SSC data output and input are at different rates, 50Hz and approximately 100Hz respectively, state data may or may not be available. Thus, the process attempts to read one byte from the file descriptor. If data is not immediately available the read request will fail and the process will abandon its attempt to retrieve state data. If the read is successful the process will read the entire data package and parse it for state information. This information is then updated in shared memory. At the end of each loop the process sleeps for approximately 0.01 seconds.

4.7 Pan_Tilt Process

The pan/tilt process was developed as a functional template. The process is fully operational but is only coded to hold the pan/tilt in a neutral position. This was done to support quick integration of vision code that may need to control the pan/tilt. This program, in its empty template state, will only connect to the shared memory location created by the servo process and output pan and tilt neutral positions.

4.8 Collect Process

The Collect process is responsible for data collection during operation of the testbed. This process collects the testbed's position, orientation, accelerations, approximated velocities, control outputs, and SSC state from shared memory and commits them to a file, data.txt, on the RAM drive. Data is collected and stored at approximately 4Hz.

The execution order of this process first connects to shared memory and then creates an empty data.txt file. Once this is complete the process creates a signal handler responsible for closing the text file in the event that the process is terminated. The process then enters an infinite loop that will retrieve data from shared memory, create a timestamp, and output that data to the file. After each successful write to the output file the process will sleep for 0.2 seconds.

4.9 Test Process

The Test process is designed to allow the user to test shared memory and sensors prior to starting the autonomous navigation software. This allows the user one final inspection to assure that data is being gathered and looks feasible. During execution this process will output, to the screen, the position, orientation, angular rates, and laser range being collected and posted at 10Hz.

The execution order of this process first connects to all of the utilized shared memory locations and then enters an infinite loop. While in this loop the process will gather all of the required data and output that data to the standard I/O interface, typically the screen. Note that the process only outputs a single laser range value for each loop. Once the process has gathered all of the laser range data requested it will then average that data and output it to standard I/O. This is done for both readability and to smooth any erroneous values from the sensor. After each successful output the process sleeps for 0.1 seconds.

4.10 Navigate Process

The navigate process is the focal point for all of the software developed for the USL testbed and is the last process to be started before autonomous flight can begin. This process sets up and interfaces with the fuzzy controllers, calculates positional error and velocity for all three axes, fuses data calculated or collected from multiple sources, and filters out noise. This process is also responsible for managing the flight path and stage of the testbed.

The navigate process is the only process that requires arguments be given by the user. These arguments correspond to takeoff and landing request and are Boolean variables. If a particular action is to be performed, an integer value of one must be supplied as an argument otherwise a zero must be supplied. If a zero is supplied then the testbed assumes that those operations will be performed by the safety pilot. Make note that assumptions are made based on these arguments. For instance, if the helicopter is told to takeoff then it will assume that it is on the ground and that the engine is powered down. Thus, if the helicopter is in the air when the vehicle's state becomes autonomous, the engine will power down.

Due to its size and complexity this process is broken down into several subsections that as a whole describe the overall process. Figure 33 provides a chart that describes the overall flow of this process.

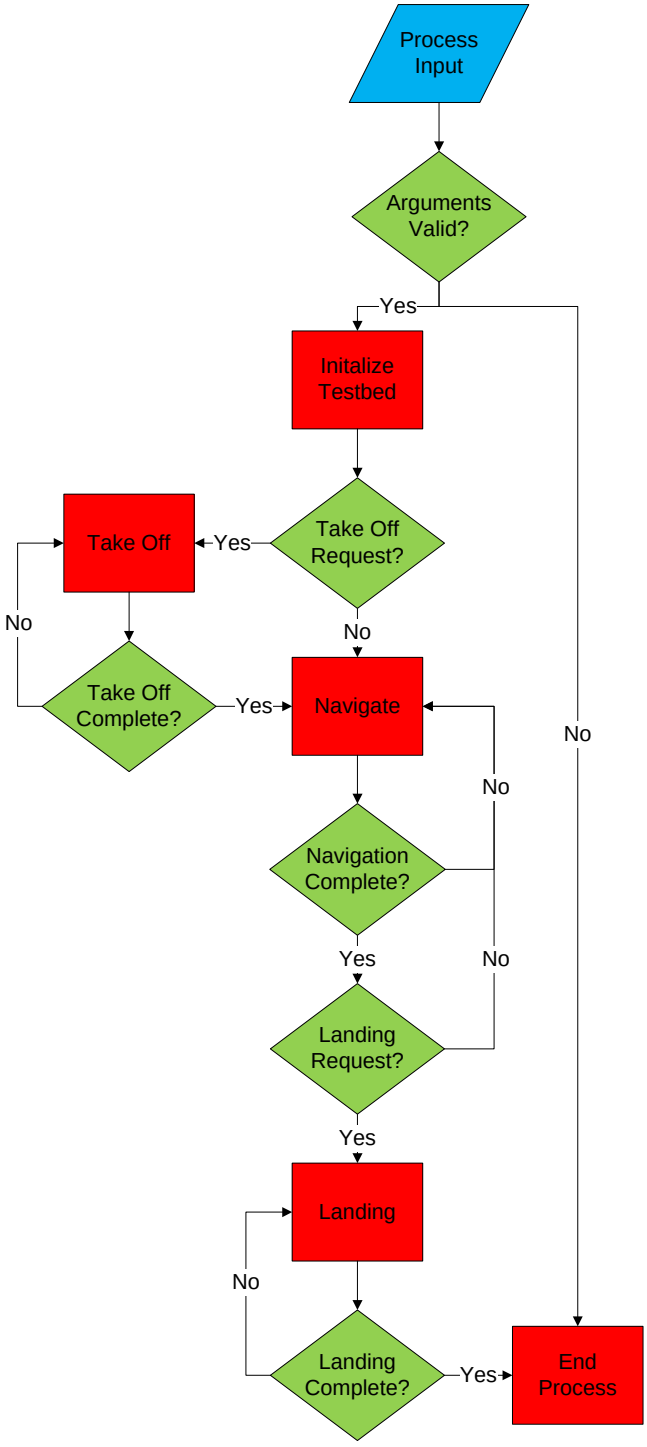

Figure 33: Flow Chart for the Navigation Process

4.10.1 Initialization

Once the navigation process's arguments have been validated the process begins to initialize the system for flight. This is first done by setting the number and locations of waypoints. Next, the process connects to all of the required shared memory and sets up the fuzzy controllers. The process will then install the signal handler designed to free any structures created for accessing the fuzzy controllers. Last, the process attempts to calculate the gravitational force sensed by the IMU. This must be done prior to takeoff to ensure that errors caused by other forces are minimal. This is done by reading four hundred IMU readings, rotating them to the world coordinate frame, and then averaging their values. This averaged vector will be removed from all subsequent acceleration readings. Note that the number of readings averaged is an arbitrary number that attempts to minimize the error in the gravity vector approximation. With respect to the USL testbed, four hundred readings is approximately four seconds worth of IMU data.

4.10.2 Takeoff Procedure

The takeoff procedure is broken down into three phases. These phases are responsible for spinning up the motor, prepping the collective, and lifting off the vehicle. During all phases of this procedure the process will continuously gather laser range data, GPS position, and IMU data. Note that prior to lift off this procedure will lock in the current heading, latitude, and longitude as the desired heading, latitude, and longitude. This method assures that the vehicle will attempt a completely vertical takeoff and will minimize the possibility of extreme maneuvers at low altitudes. Note that this procedure requires that a GPS lock be acquired. If a GPS lock is not present the vehicle will not begin takeoff.

The first phase in the takeoff procedure is to spin up the motor. First the procedure commands the collective and throttle to be low. Low is meant to convey the lower limit of the collective command, negative one, and a throttle value that does not rotate the head, typically a 1000 microsecond PW. Note that the conversion of the collective command to a PW value will be discussed Chapter 5. This procedure will now slowly throttle up the motor by increasing the throttle's PW by ten microseconds per control loop until the throttle neutral value is reached. Throttle neutral is the throttle PW that turns the main rotor at approximately 1600 Revolutions Per Minuet (RPMs). Once the throttle has reached its neutral value, it is held at this value

throughout the remainder of the flight. Once the throttle neutral value has been reached, phase one is deemed complete. Note that in all phases of the takeoff procedure the fuzzy controllers, described in Chapter 6, are controlling the yaw, pitch, and roll as they normally would.

Phase two is used to prepare the collective to be controlled by the fuzzy controller. This is deemed necessary to prevent a sudden vertical acceleration that would cause unnecessary stress on the testbed and may significantly lower the head speed. Preparing the collective is done by slowly increasing the collective command from its minimum value to its neutral value. The neutral value for collective is designed to represent a hovering collective for the testbed and is represented by the value zero. The collective is increased by 0.0025 units until the neutral value has been reached. Once the neutral value has been reached phase two of the takeoff procedure is deemed complete.

The last phase of the takeoff procedure is responsible for lifting off the testbed. This is accomplished by simply giving control of the collective to the fuzzy collective controller. Assuming that the altitude set point is at some distance above the vehicle it will begin to climb. Takeoff will continue monitoring the vehicle until the average laser range value exceeds two meters. Once the vehicle exceeds two meters of altitude the takeoff procedure is marked complete and the navigation procedure can begin. Note that the altitude set point is assumed to be above two meters. If the vehicle reaches the altitude set point during the takeoff procedure the laser is assumed to be malfunctioning. The vehicle will then inform the ground station of the error and will ultimately refuse to land autonomously, discussed further in Section 4.10.4.

4.10.3 Navigation Procedure

The navigation procedure is simply responsible for navigating the testbed to desired locations. Navigation is first is accomplished by collecting GPS and IMU data from shared memory. This data is then used to determine the vehicle's offset from either a waypoint or a desired flight path, discussed in Section 5.2. Next this procedure will calculate the vehicle's current velocity and variant acceleration, discussed in Sections 5.4 & 5.5 respectively. The calculated offsets, velocities, and variant accelerations as well as the orientation are then provided to the controllers. The procedure will then collect the controller outputs and convert them to pulse width values, discussed in Section 5.1. Once a desired location has been reached within a small threshold and maintained for a short period of time the next location is proceeded to. Once

the final position has been reached the process will either initiate the landing procedure or hover at the last location.

4.10.4 Landing Procedure

The landing procedure is broken down into three phases. These phases are responsible for touching down, removing lift, and powering down the vehicle. During all phases this procedure will gather laser range data, GPS position, and IMU data. Also note that during all phases of the landing procedure all negative, or down, collective commands output by the controller are divided by two. This is done to decrease the overall downward speed of the vehicle from that of normal flight. Note that landing requires that the laser be operating correctly. If the laser process has reported an unrecovered failure, the vehicle will refuse to land and will hover indefinitely at its last waypoint.

The first phase of the landing procedure is responsible for lowering the vehicle until a touch down has been achieved. This is done by decreasing the altitude set point in two feet increments. Note that the decrement can only occur if the vehicle stays within a small latitudinal and longitudinal threshold of its last waypoint. This will assist in positional stability and assure that the vehicle stays on target during landing. This decrement is continued until the average laser range value is less than 0.175 meters and at least 90% of the readings are valid (greater than zero). Checking the number of valid range readings is performed to account for erroneous data that is periodically experienced during the landing procedure. Once the laser range average has reached 0.175 meters the skids should be within an inch or two of the ground. At this point the vehicle is considered to have touch downed. This is done so that vehicle can immediately begin to reduce collective, phase 2. If this is not done the vehicle risk catching part of the skids on an uneven surface and tumbling onto its side.

It is noteworthy to mention that as the helicopter enters the final few feet of landing an additional force, known as “ground effects”, begins to influence the helicopter. Ground effects is an additional vertical force that provides the helicopter with extra lift. This is caused by the air flowing through the main rotor head colliding with the ground. This force will typically cause the vehicle’s descent to decrease and may ultimately prevent the vehicle from touching down. To account for this effect an integrator continually reduces the collective trim of the helicopter at any point during landing that the velocity ceases to be downward. This ensures that the vehicle will continue its decent even in the presence of ground effects.

The second phase of the landing procedure is responsible for removing all lift from the vehicle. This is done by continually reducing the previous collective command by a value of 0.02. This is continued until the collective value reaches negative one. This process will remove all vertical force from the vehicle in less than two seconds. This is done to force the vehicle to land rather than risking the vehicle's skids dragging the ground and tipping the vehicle over. Now that the collective is completely lowered the vehicle should be stably landed and phase three can begin.

The final phase of the landing procedure is to power down the motor. The procedure first outputs neutral values to the roll, pitch, and yaw servos. This should level out the swashplate and prevent the vehicle from commanding any movement that may damage the testbed. The throttle is now decremented by values of five microseconds from its neutral value down to 1000. Once the throttle has reached 1000, typically the lower limit for an ESC, the landing procedure is deemed complete.

4.10.5 Controller

Although the specifics of the controllers will be detailed in Chapter 6 it is necessary to mention that they are distinct from the navigation process. The navigation process is simply responsible for collecting and/or calculating state data for the vehicle. This data is then provided to the controllers which will in turn provide the navigation process with control commands. These commands range from [-1,1] and represent the possible range of control. This is done to allow the software to interface with any number of controllers. Controllers are only expected to conform to the expected range of control output. This allows the testbed to easily conform to new controllers with minimal modification to all other software.

4.11 Scripts

The last folder utilized in the USL testbed's software architecture is Scripts. The script folder is designed to store any scripts that may simplify or automate OS processes. At current only one script is being utilized, *prep*. This script is uploaded to the testbed once it has booted. This script is responsible for starting all of the appropriate processes, with the exception of the navigate process, in their correct order. Due to fact that many of the processes share memory

they must be started in an order that assures the memory is created before it is accessed. The specifics contents of this script can be found in Appendix G.

Chapter 5
Algorithms

Although the basic steps and concepts of the software were presented in Chapter 4, the details of the state and PW value calculation algorithms have yet to be detailed. This chapter is dedicated to detailing these algorithms. All of the algorithms presented here reside within the navigation process and function as part of its execution cycle. The detailed algorithms include converting controller commands to PW values, determining positional error on a flight path, determining heading error, calculating GPS, IMU, and fused GPS/IMU velocities, calculating variant accelerations, trim integrators, and GPS antenna translations.

5.1 Servo Cyclic and Collective Pitch Mixing

PW values calculated from roll, pitch, and collective commands are typically governed by the type of swashplate on the vehicle. Most commercial RC helicopters have either a three or four point swashplate. These points represent the number of servo arms that control the position and orientation of the swashplate, see Figure 34.

Four points swashplates are by far the most common and are the easiest to comprehend. Typically, in a four point system there are dedicated servos for lateral commands (roll),

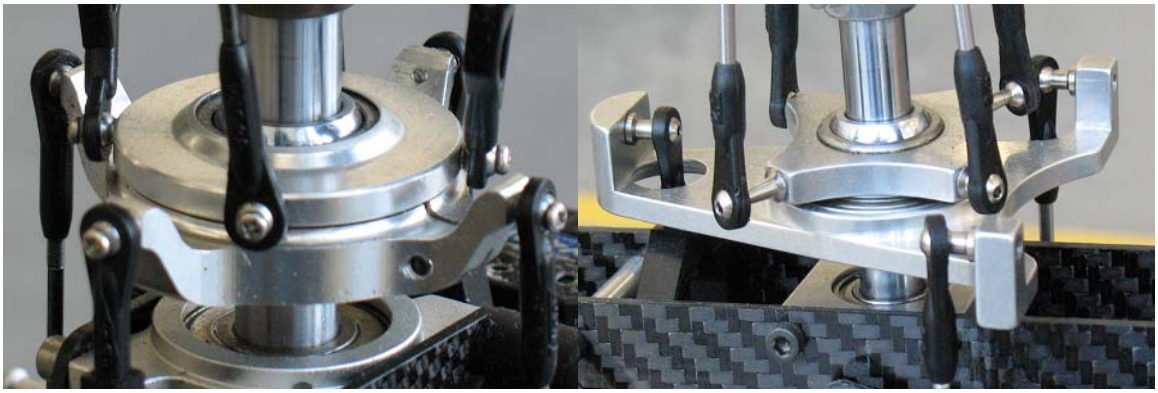

Figure 34: Three Point (left) and Four (right) Point Swashplates

longitudinal commands (pitch), and collective commands. Thus, a lateral movement command from the pilot, human or computer, requires calculating a PW for a single servo. This is also true for lateral and collective commands and greatly simplifies control.

Three point swashplates, although less common, are becoming very popular. Three point systems typically utilize three distinct servos placed at various angles on the swashplate. Typically, one servo arm is connected directly in front of swashplate. The other two are either connected at 90 or 120 degrees from the front servo arm. These systems require that a single command, such as a pitch command, be mixed between the three servos. For example, a pitch forward command will require that the front servo lower the swashplate and the back two servos raise the swashplate. This allows the vehicle to generate forward motion while retaining the collective of the vehicle. The act of mixing servos is commonly referred to as Servo Cyclic and Collective Pitch Mixing (CCPM) but for simplicity will be referred to as servo mixing for the remainder of this work.

The navigate process contains a single function, electricMixing, dedicated to mixing vehicle's commands for the three point swashplate on the Joker Maxi-2. This function mixes the roll, pitch, and collective commands provided by the vehicle's controllers. Commands are provided in values from negative one to one and represent positions from the minimum servo position to the maximum servo position as stated in *Robot_defs.h*. The specific value of a command corresponds to a percent of distance from neutral PW to the maximum or minimum PW. Equation (1) represents the calculation used to determine the PW value corresponding to a command. For Equation (1), Max_{P_i} is the maximum PW value for servo 'i', Min_{P_i} is its minimum PW value, N_{P_i} is its neutral PW value, O_{P_i} is the calculated PW for servo 'i', and α is the control command with a value ranging from [-1,1].

$$O_{Pi} = \begin{cases} (Max_{P_i} - N_{P_i}) * \alpha & for\ \alpha \geq 0 \\ (N_{P_i} - Min_{P_i}) * \alpha & for\ \alpha < 0 \end{cases} \quad (1)$$

The mixing function first mixes the lateral, or roll, command to the vehicle. This is done because roll commands do not require modification to the front servo's position. The function first determines the direction of servo movement from neutral by checking to see if the command is negative or positive. Once this is determined the PW deviation is calculated using Equations (1) for the left servo. This value is then added to both the right and left servo neutral values. This will lower the left servo the exact same amount that it will raise the right servo hence preserving

the current collective value. Note that the deviation is added to both servos because the right servo is mounted inversely to the left and thus an addition to both will lower one and raise the other.

Next the mixing function mixes longitudinal, or pitch, commands. This is also done using Equation (1) for the front servo. Once the deviation is calculated, its value is added to the front servo's neutral position. Due to the configuration of the swashplate there is a 2:1 ratio for front to left and right servo commands. This is due to an unequal ratio of distance between the front control arm and the two side control arms. Thus, a single PW change in the front servo only corresponds to a 0.5 PW changes in the left and right servos. Keeping this in mind, the deviation calculated for the front servo is then divided by two and added to the left servo and subtracted from the right servo. Note that changes to both the left and right servo values are changes made to the values already calculated in the first part of the mixing function.

The last command that must be mixed is the collective command. Collective's min, neutral, and max values are not directly connected to a particular servo. These values describe the amount of change in the right, left, and front servos that is allowed for by the collective commands and for the USL testbed ranges from 0 to 160. Mixing first calculates the PW deviation using Equation (1). This value is then added to the negative of the collective's neutral. This value represents the PW change for the collective input and is added to the front and right servo's value and subtracted from the left servo. It should be mentioned that the Maxi Joker-2 has a "leading edge" rotor. Thus, the swashplate must be lowered to increase the collective. This is why the PW deviation calculated for the collective is added to the negative value of the collective's neutral.

Although PW commands for the yaw are not calculated in this function it is noteworthy to mention that it is also calculated using Equation (1). As yaw commands correspond to a single servo the value calculated using Equation (1) can be added directly to the yaw's neutral value to calculate the desired PW.

5.2 Positional Error Calculations

The calculations described in this section represent the algorithms used to calculate the positional error, or offset, of the testbed. Specifically, position error represents the distance, in feet, from the current position to a desired position.

Positional error is typically calculated by determining the distance between the current GPS coordinate and a desired position on a calculated flight path. Flight paths are represented by straight lines between latitude and longitude waypoints. The desired position on the flight path is determined by finding intersections between a fixed circle around the vehicle and the current flight path. To determine these intersections the software must first calculate the distance from the vehicle to both the previous and current waypoints.

The USL testbed utilizes the Earth model for calculating distances between GPS coordinates, Equations (2)-(10). It should be noted that World model equation requires that GPS coordinates be in decimal degree format. To adhere to this requirement, coordinates are converted from decimal hour format to decimal degree using the function

$$D(c) = [c] + (((c - [c]) * 100) / 60) \tag{2}$$

where c represents a latitude or longitude coordinate in decimal hours format and $[c]$ represents the greatest integer value less than c. The first step in calculating the distance is to obtain the true angles, G_{a1} and G_{a2}, using

$$G_{a1} = C_{R2D}(\arctan(\frac{o^2}{O^2} * \tan(C_{D2R}(D(L_1))))) \tag{3}$$

and

$$G_{a2} = C_{R2D}(\arctan(\frac{o^2}{O^2} * \tan(C_{D2R}(D(L_2))))) \tag{4}$$

where O and o represent the approximated major and minor axis of the earth in meters respectively, C_{D2R} represents a conversion function from degrees to radians, C_{R2D} represents a conversion function from radians to degrees, L_1 represents the GPS latitude for point one, and L_2 represents GPS latitude for point two. Using G_{a1} and $G_{a2,}$ the radiuses, R_1 and R_2, can determined using

$$R_1 = \sqrt{\frac{1}{\frac{\cos^2(C_{D2R}(G_{a1}))}{O^2} + \frac{\sin^2(C_{D2R}(G_{a1}))}{o^2}}} \tag{5}$$

and

$$R_2 = \sqrt{\frac{1}{\frac{\cos^2(C_{D2R}(G_{a2}))}{O^2} + \frac{\sin^2(C_{D2R}(G_{a2}))}{o^2}}}. \tag{6}$$

Utilizing the functions

$$I_c(R,G) = R * \cos(C_{D2R}(G)) \tag{7}$$

and

$$I_s(R,G) = R * \sin(C_{D2R}(G)) \tag{8}$$

in Equations

$$E_x = \begin{cases} -\sqrt{(I_c(R_1,G_1) - I_c(R_2,G_2))^2 + (I_s(R_1,G_1) - I_s(R_2,G_2))^2} & for\ L_2 > L_1 \\ \sqrt{(I_c(R_1,G_1) - I_c(R_2,G_2))^2 + (I_s(R_1,G_1) - I_s(R_2,G_2))^2} & for\ L_2 \le L_1 \end{cases} \tag{9}$$

and

$$E_Y = 2\pi * \frac{(I_c(R_1,G_1) + I_c(R_2,G_2))}{2*360} * (D(l_1) - D(l_2)) \tag{10}$$

are then used to calculate the x and y distance, in meters, between the two GPS coordinates, where l_1 and l_2 represent the GPS longitude for points one and two respectively.

Using Equations (2)-(10) the x and y distances from the vehicle's position to both the previous and current waypoint can be calculated. Note that the vehicle's GPS position is used for variable L_1 to calculate the above mentioned distances. For consistency both x and y distances are then converted from meters to feet.

With the E_X and E_Y distances calculated and converted to feet, the software can now determine the vehicle's relative position to its flight path. This can be done by determining if a secant or tangent line exists. This equation is well defined in [94] and [95] and will not be repeated here. If intersections exist, the testbed will then determine which intersection is closest to its next waypoint, using direct line distance, and then use that point as an intermediary goal. The x and y distances are then calculated to this intermediary goal, also defined in [94] and [95]. If the intersection does not exist, then the vehicle is too far off course to use intermediary goals. In this case, positional x and y error will be calculated by determining the direct line distance from the testbed to the next waypoint.

The last step in determining the positional error is to transform the x and y distances from the world coordinate frame to the local coordinate frame. The local coordinate frame utilizes the heading of the vehicle as the Y axis. These rotated values are provided to the controllers as error in the pitching direction and error in the rolling direction. Figure 35 depicts an example of determining positional error.

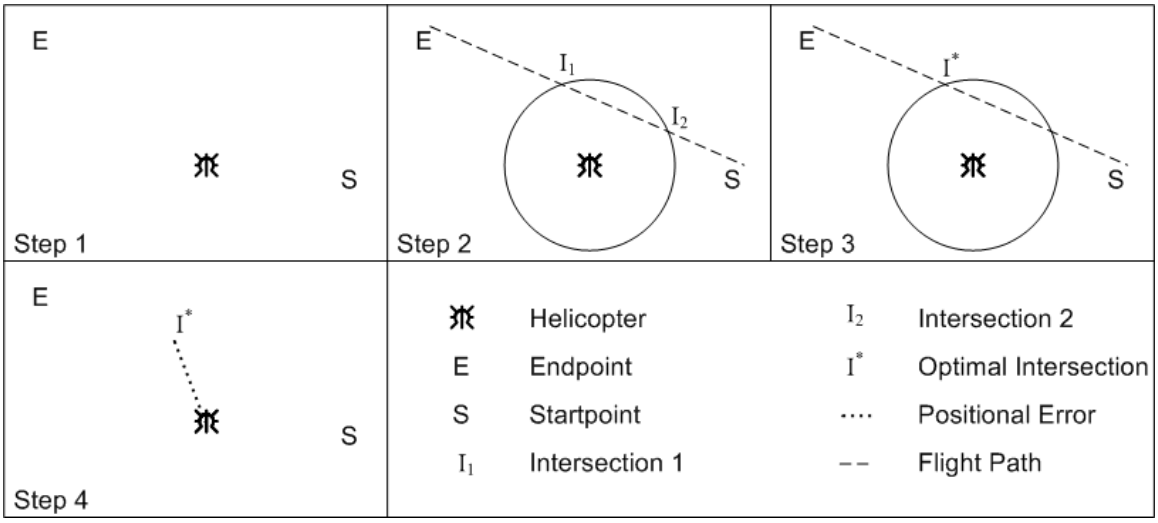

Figure 35: Demonstration of Calculating Positional Error

It should be noted that altitude, or Z axis, positional error is strictly calculated by determining the difference, in feet, between the altitude set point and GPS supplied altitude. This value is supplied to the collective controller as collective positional error.

5.3 Heading Error Calculations

Heading error is the deviation of the current heading from the desired heading. Calculating heading error is done by determining the shortest distance from the current heading to the desired heading. This is typically done by subtracting the current heading from the desired heading. Due to the heading's range of -180 to 180 degrees a check must be performed to assure that movement in the other direction would not be optimal. This is simply done by determining if the difference between the current heading and the desired heading is greater then 180 or less then -180. Once the optimal error is determined it is provided to the yaw controller as heading error.

5.4 Velocity Calculations

Although velocity is typically one of the key elements in controlling a helicopter, it is by far the most difficult to accurately obtain. On the USL testbed, velocity calculations are performed by integrating the accelerations provided by the IMU. The testbed attempts to

compensate for drift, bias, and noise in the velocity calculations using GPS calculated velocities and first order Kalman filters.

Integration of accelerations to acquire velocity first requires the removal of gravity which is calculated during initialization (Section 4.10.1). Thus this procedure must first rotate the gravity vector to the local coordinate frame. This is performed using

$$\begin{bmatrix} g'_y \\ g'_z \\ g'_x \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(-C_{D2R}(\psi)) & -\sin(-C_{D2R}(\psi)) \\ 0 & \sin(-C_{D2R}(\psi)) & \cos(-C_{D2R}(\psi)) \end{bmatrix} * \begin{bmatrix} 0 \\ g_z \\ 0 \end{bmatrix} \tag{11}$$

to rotate the gravity vector g about the X axis and then

$$\begin{bmatrix} g''_y \\ g''_z \\ g''_x \end{bmatrix} = \begin{bmatrix} \cos(-C_{D2R}(\theta)) & -\sin(-C_{D2R}(\theta)) & 0 \\ \sin(-C_{D2R}(\theta)) & \cos(-C_{D2R}(\theta)) & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} g'_y \\ g'_z \\ g'_x \end{bmatrix} \tag{12}$$

to rotate vector g' about the Y axis. ψ and θ represent the Euler angles roll and pitch respectively. Make note that in (11), only gravity readings on the Z axis of the gravity vector are rotated. Since the vehicle is stable when the gravity vector is calculated all accelerations are deemed to be on the Z axis. Gravity accelerations recorded on the X and Y axis are assumed to be erroneous and will be systematically filtered out in the drift calculation. Due to this fact it is unnecessary to rotate the gravity vector about the Z axis.

The rotated gravity vector, g'', can now be subtracted from the IMU provided accelerations. The new acceleration vector, referred to as a, is then partially rotated to the world coordinate frame where an approximated drift, discussed later in this section, is subtracted. For the purpose of this paper a partial rotation simply refers to a rotation procedure that does not rotate about all three axes. The rotation procedure for a is performed using

$$\begin{bmatrix} a'_y \\ a'_z \\ a'_x \end{bmatrix} = \begin{bmatrix} \cos(C_{D2R}(\theta)) & -\sin(C_{D2R}(\theta)) & 0 \\ \sin(C_{D2R}(\theta)) & \cos(C_{D2R}(\theta)) & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} a_y \\ a_z \\ a_x \end{bmatrix} \tag{13}$$

to rotate the vector a about the Y axis and then

$$\begin{bmatrix} a''_y \\ a''_z \\ a''_x \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(C_{D2R}(\psi)) & -\sin(C_{D2R}(\psi)) \\ 0 & \sin(C_{D2R}(\psi)) & \cos(C_{D2R}(\psi)) \end{bmatrix} * \begin{bmatrix} a'_y \\ a'_z \\ a'_x \end{bmatrix} \tag{14}$$

to rotate vector a' about the X axis. This partial rotation is performed as drift calculations are stored in this coordinate system. Once the drift vector has been subtracted from a'' the rotation to the world coordinate frame is completed using

$$\begin{bmatrix} a_y''' \\ a_z''' \\ a_x''' \end{bmatrix} = \begin{bmatrix} \cos(-C_{D2R}(\phi)) & 0 & \sin(-C_{D2R}(\phi)) \\ 0 & 1 & 0 \\ -\sin(-C_{D2R}(\phi)) & 0 & \cos(-C_{D2R}(\phi)) \end{bmatrix} * \begin{bmatrix} a_y'' \\ a_z'' \\ a_x'' \end{bmatrix} \tag{15}$$

where ϕ is the yaw Euler angle.

The next step performed in calculating velocity is to average a''', with the $a'''_{\tau-1}$, defined as the previous control loops a'''. This is done to account for the loss of information due to the use of discrete sensor readings. It is assumed that variations in acceleration are linear for the short period of time between IMU readings. Thus an average of the prior and current accelerations should provide a more accurate value for integration.

The averaged acceleration vector, $\overline{a'''}$, is then integrated using the IMU process's provided timestamps. This calculated velocity vector is then added to two currently stored velocity vectors, V_I and V_F. The vectors V_I and V_F represent velocities calculated for the entire operation of the vehicle using only integrated accelerations and integrated accelerations corrected by GPS respectively. These vectors will be discussed later in this section. Note that all velocity readings are stored in the world coordinate frame and are only temporarily transformed to the local coordinate frame to supply roll, pitch, and collective velocities to the controllers.

When new GPS positions are available, corrections to the current velocity and velocity calculations are performed. This includes calculating an approximation of bias and fusing the GPS and IMU calculated velocities.

Although GPS velocity can simply be calculated by determining the distance traveled divided by the amount of time that has passed, this method typically performs poorly. One reason is that calculating velocity from discrete readings, such as GPS coordinates, introduces an error caused by the resolution of the readings. The value of this error increases significantly as the amount time used for the calculation decreases. This GPS velocity error will now be defined as

$$G_{E_A}(\Delta\tau) = \left| \frac{M_A}{\Delta\tau} \right| \tag{16}$$

where M is the resolution of the sensor readings, in feet, on axis A and $\Delta\tau$ is the amount of time that has passed, in seconds, between distances readings. This is significant when

considering calculating GPS velocity from two consecutive readings. At its lowest resolution the NMEA GPGGA message can only provide position at approximately a six inch resolution. If the GPS is operating at 5 Hz and velocity is calculated using consecutive readings the GPS velocity error could be as large as 2.5 ft/sec. Note that to reduce this error, $\Delta\tau$ must be increased. Simply increasing $\Delta\tau$ assumes that the system is stagnate during that particular time interval. Although this assumption may be valid periodically, it is not valid for every possible time interval. A more optimal algorithm should dynamically increase $\Delta\tau$, as appropriate.

To assure that the USL testbed utilizes a more accurate GPS velocity a new method which dynamically updates $\Delta\tau$ was developed. This method attempts to approximate the current GPS velocity by utilizing past GPS velocities that are deemed plausible. The algorithm first calculates multiple velocity vectors from GPS movements that occurred over the last one second. These vectors are calculated using

$$\begin{bmatrix} T_{i_Y} \\ T_{i_Z} \\ T_{i_X} \end{bmatrix} = \begin{bmatrix} \left(F(E_y(\tau,\tau-i))\right)*\dfrac{Hz_G}{i} \\ \left(F(E_z(\tau,\tau-i))\right)*\dfrac{Hz_G}{i} \\ \left(F(E_x(\tau,\tau-i))\right)*\dfrac{Hz_G}{i} \end{bmatrix}, \tag{17}$$

where T_i represents the temporary GPS velocity vector calculated using 'i+1' GPS readings, F is a function for converting from meters to feet, E is a function for calculating the distance vector between the current GPS position, τ, and the $\tau - i$ GPS position using the World Model equation, and Hz_G is the operating frequency of the GPS. As these calculations are only performed for the past one second's worth of data, 'i' will range from $[1, Hz_G]$ and Hz_G vectors will be calculated.

Initializing the GPS velocity vector V_G to T_1 and beginning with $i = 2$, the elements of vector T_i are compared to calculated error thresholds. This is done using

$$V_{G_A} = \begin{cases} T_{i_A} & for \quad V_{G_A} - G_{E_A}\left(\frac{i-1}{Hz_G}\right) < T_{i_A} < V_{G_A} + G_{E_A}\left(\frac{i-1}{Hz_G}\right) \\ V_{G_A} - G_{E_A}\left(\frac{i-1}{Hz_G}\right) & for \quad T_{i_A} \le V_{G_A} - G_{E_A}\left(\frac{i-1}{Hz_G}\right) \\ V_{G_A} + G_{E_A}\left(\frac{i-1}{Hz_G}\right) & for \quad T_{i_A} \ge V_{G_A} + G_{E_A}\left(\frac{i-1}{Hz_G}\right) \ or \ T_{i_A} * T_{i-1_A} < 0 \end{cases} \tag{18}$$

if T_{i_A} is positive or

$$V_{G_A} = \begin{cases} T_{i_A} & for \quad V_{G_A} + G_{E_A}\left(\frac{i-1}{Hz_G}\right) \le T_{i_A} \le V_{G_A} - G_{E_A}\left(\frac{i-1}{Hz_G}\right) \\ V_{G_A} + G_{E_A}\left(\frac{i-1}{Hz_G}\right) & for \quad T_{i_A} \le V_{G_A} + G_{E_A}\left(\frac{i-1}{Hz_G}\right) \ or \ T_{i_A} * T_{i-1_A} < 0 \\ V_{G_A} - G_{E_A}\left(\frac{i-1}{Hz_G}\right) & for \quad T_{i_A} \ge V_{G_A} - G_{E_A}\left(\frac{i-1}{Hz_G}\right) \end{cases} \tag{19}$$

if T_{i_A} is negative. Note that subscript *A* represent a particular axis (i.e. lateral, longitudinal, or vertical). Updates to V_G continue until either all T_i vectors have been exhausted or until T_i fails to adhere to the boundaries set by V_{G_A} and G_E. Boundary failures are determined by the inequality

$$V_{G_A} - G_{E_A}\left(\frac{i-1}{Hz_G}\right) \le T_{i_A} \le V_{G_A} + G_{E_A}\left(\frac{i-1}{Hz_G}\right) \tag{20}$$

in Equation (18) and

$$V_{G_A} + G_{E_A}\left(\frac{i-1}{Hz_G}\right) \le T_{i_A} \le V_{G_A} - G_{E_A}\left(\frac{i-1}{Hz_G}\right) \tag{21}$$

in Equation (19). If the threshold is broken for a particular element in vector T_i then that axis has used all the data deemed valid for its computation. The failed axis is then finalized by setting the corresponding axis in V_G with the closest valid value in inequality (20) or (21) to the failed T_i element. Although other elements in vector V_G may continue to be updated, axes whose thresholds have been broken cease to be updated. Note that T_i vectors can be individually calculated and compared rather than calculating all possible vectors as described in (17).

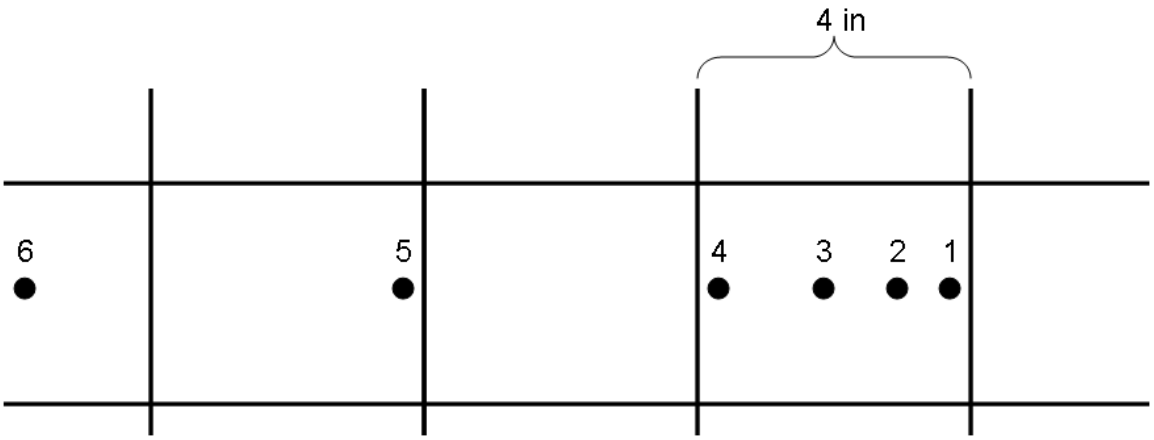

Figure 36: Example of Six GPS Readings Along a Single Axis

Calculating vectors as needed will decrease the computational complexity of the algorithm. The algorithm was described as it is for clarity and ease of understanding.

For clarity, Figure 36 details an example of GPS data from a 5 Hz receiver along a single axis. Note that the position data is labeled 1-6 where 1 is the most current data and 6 is the oldest and the grid represents the lowest resolution of the GPS on that particular axis. Using the above method to calculate the current velocity the algorithm would first calculate the base velocity T_1 using positions 1 and 2. This would provide a velocity of zero ft/sec. Next the algorithm would calculate the velocity using positions 1, 2, and 3. This would also provide a velocity of zero ft/sec which does not violate the inequality (20), i.e. the range (-1.7,1.7). Using positions 1, 2, 3, and 4 again provides a velocity of zero which also does not violate (20), i.e. the range (-0.8,0.8). Using positions 1, 2, 3, 4, and 5 provides a velocity of 0.8 ft/sec. This velocity violates (20), i.e. the range (-0.6,0.6), and thus stops further integration of GPS data on that particular axis. Since the temporary velocity is violated on the positive side of the threshold the final velocity along that axis is set to zero, the last valid velocity, plus the threshold, 0.6 ft/sec.

The method described above allows $\Delta\tau$ in (16) to be dynamically increased by utilizing past data that appears to be valid and thus decreasing the velocity error due to resolution. Figure 37 depicts velocity calculations for a single run of the USL testbed using the standard velocity calculation algorithm and the one used by the USL testbed.

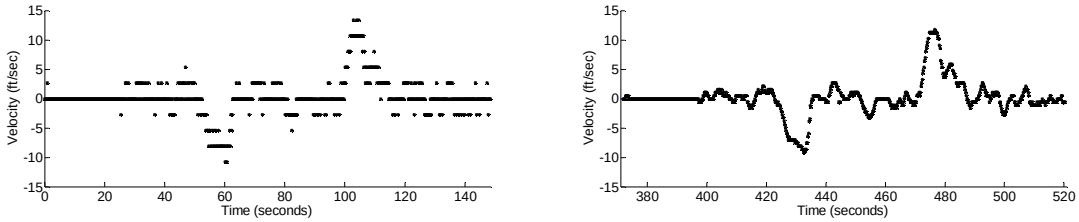

Figure 37: Flight Velocities Calculated Using the Standard Method (left) and USL Method (right)

Velocities calculated from IMU integrated accelerations are typically subject to some level of drift. This drift is typically constant for very short periods of time. Drift can heavily influence velocity calculations and will typically render the velocities useless within a matter of seconds. Figure 38 depicts velocities calculated using IMU supplied accelerations with only the gravity vector removed.

During operation the navigation process continually attempts to calculate drift and remove it from the accelerations readings. This is first done by calculating the difference between a strictly IMU calculated velocity, V_I, and the velocity calculated by GPS, V_G. This is performed using

$$S = (V_{I_\tau} - V_{G_\tau}) - (V_{I_{\tau-H}} - V_{G_{\tau-H}}) \tag{22}$$

where S is the slope vector of the offset, τ is the current time step, and H is the number of time steps in one second. This vector is then rotated about the Z axis using

$$\begin{bmatrix} S'_y \\ S'_z \\ S'_x \end{bmatrix} = \begin{bmatrix} \cos(C_{D2R}(\phi)) & 0 & \sin(C_{D2R}(\phi)) \\ 0 & 1 & 0 \\ -\sin(C_{D2R}(\phi)) & 0 & \cos(C_{D2R}(\phi)) \end{bmatrix} * \begin{bmatrix} S_y \\ S_z \\ S_x \end{bmatrix}. \tag{23}$$

S' is the added to any previously stored bias represented by vector B. Bias, B, is stored in this

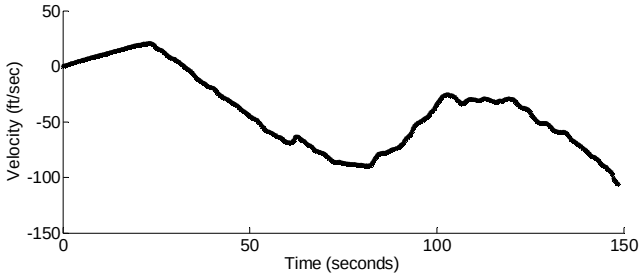

Figure 38: Flight Velocities Calculated Using Integrated Accelerations without Drift Corrections

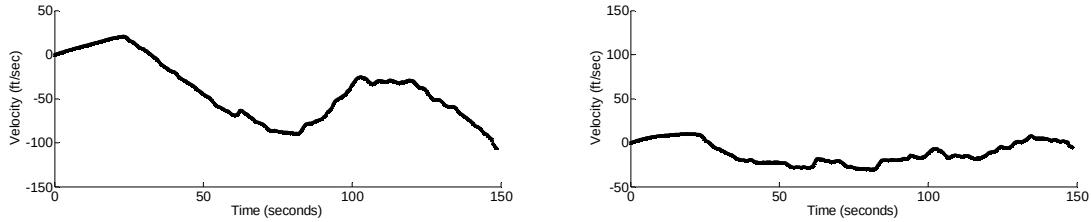

Figure 39: Flight Velocities Calculated with (right) and without (left) Drift Corrections

coordinate frame for two reasons. First, bias is being calculated for specific axis of sensors within the IMU. Thus the bias should be stored utilizing the orientation of the IMU. Second, complete rotation to the local coordinate frame would be redundant. B is only used to subtract bias from the acceleration vector, a''. As this vector is already in the correct coordinated system it is unnecessary to fully rotate B. Figure 39 depicts flight velocities calculated by integrating the accelerations with and without bias removal.

It should be noted that during testing it was determined that an offset between the GPS calculated velocities and the IMU calculated velocities existed. The IMU velocities where preceding the GPS velocities by approximately one second. To account for this the slope calculation described in (22) is offset by one second. Instead of calculating the difference between the current GPS velocities and the current IMU velocity, the slope is calculated by comparing the current GPS velocity with the IMU velocity from one second prior, i.e.

$$S = (V_{I_{\tau-H}} - V_{G_\tau}) - (V_{I_{\tau-2H}} - V_{G_{\tau-H}}). \tag{24}$$

Once drift has been approximated the GPS velocity vector V_G is fused with $V_{F_{\tau-H}}$ using

$$\begin{bmatrix} V_{Fy_\tau} \\ V_{Fz_\tau} \\ V_{Fx_\tau} \end{bmatrix} = \begin{bmatrix} V_{Fy_\tau} \\ V_{Fz_\tau} \\ V_{Fx_\tau} \end{bmatrix} + \begin{bmatrix} K_y \\ K_z \\ K_x \end{bmatrix} * \left(\begin{bmatrix} V_{Gy} \\ V_{Gz} \\ V_{Gx} \end{bmatrix} - \begin{bmatrix} V_{Fy_{\tau-H}} \\ V_{Fz_{\tau-H}} \\ V_{Fx_{\tau-H}} \end{bmatrix} \right), \tag{25}$$

where the vector K represents the standard first order Kalman gain. Equation (25) is performed to remove offset from the velocity calculation that was not previously accounted for by drift. Figure 40 details velocities calculated using only GPS, only bias corrected IMU, and fused IMU and GPS.

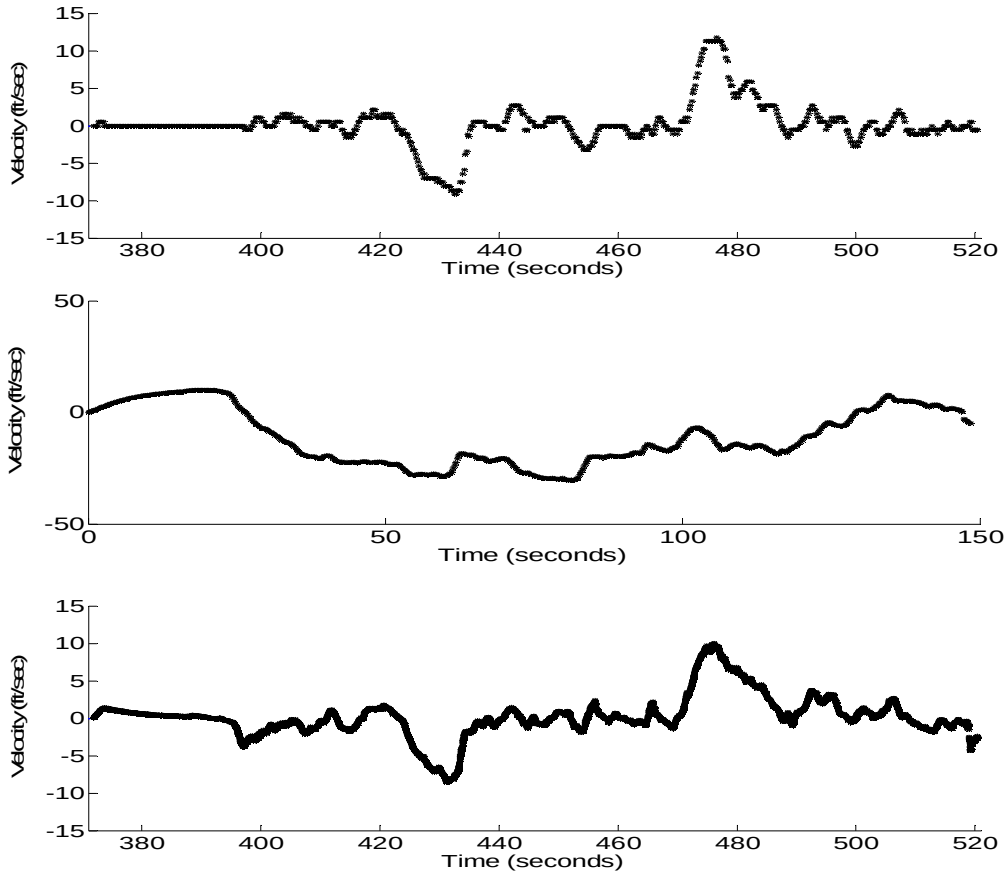

Figure 40: Flight Velocities Calculated Using GPS (top), Bias Corrected IMU (middle), and Fused GPS/IMU (bottom)

5.5 Acceleration Variant Calculation

Controllers for the USL testbed, described in Chapter 6, utilize accelerations to help determine the correct control response. Although accelerations are provided directly by the IMU they are unfiltered and typically do not correspond well with the velocity calculations above. To account for this the accelerations are recalculated, now referred to as the acceleration variant, using the filtered and fused velocity vectors.

First, the difference between the current velocity, V_{F_Ω}, and the last seven velocities, $V_{F_{\Omega-1}}, V_{F_{\Omega-2}}, \ldots, V_{F_{\Omega-7}}$, is calculated. The use of seven velocities provides adequate smoothing without a large delay in acceleration calculation. This was determined through experimentation with various size sets of velocities. These seven values are then averaged to produce the change

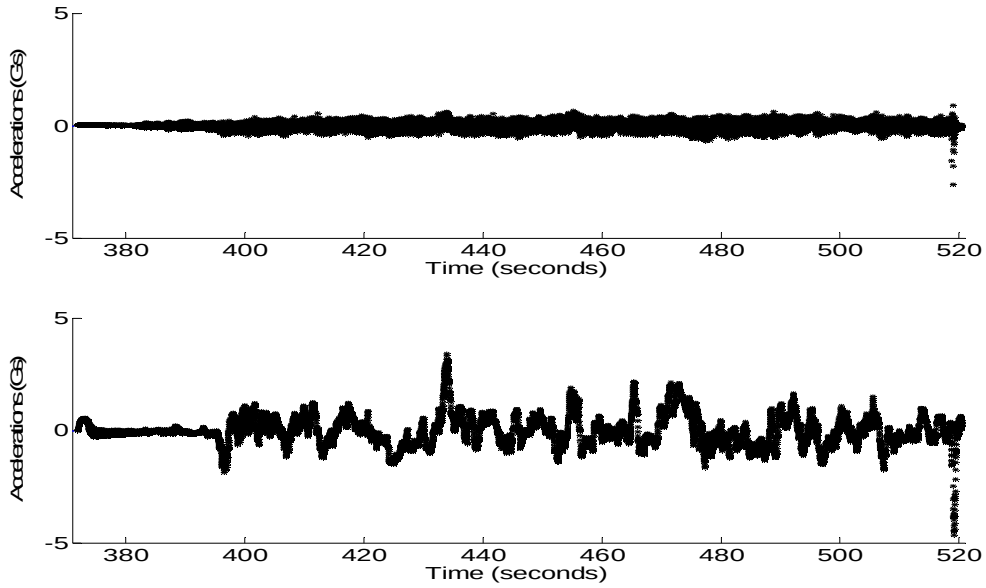

Figure 41: Flight Accelerations from IMU (top) and Variant Calculated (bottom)

in velocity, $\overline{\Delta V_F}$. The acceleration variant, A_V, is then calculated using

$$A_V = \frac{\overline{\Delta V_F}}{\Omega - (\Omega - 7)}, \tag{26}$$

where Ω represents the timestamp for V_{F_Ω} and $(\Omega - 7)$ represents the timestamp for $V_{F_{\Omega-7}}$. The acceleration variant is then passed through a first order Kalman filter and provided to the pitch and roll controllers as accelerations. Figure 41 depicts a comparison of the acceleration variant and raw accelerations provided by the IMU.

5.6 Trim Integrators

On top of its many duties, the navigate function is responsible for determining when progress is not being made by the helicopter. This is done be evaluating the positional error, velocity, and acceleration of the vehicle.

During operation, the navigation function continuously calculates three integrators. These integrators are used as trim values for the roll, pitch and collective and are added to the PW outputs of these axes. Calculations are performed by incrementing or decrementing the integrator by a small constant amount. These calculations are only performed when the vehicle is outside of

a positional threshold of its goal and no progress, be it though acceleration or velocity, is being made. Direction of the integration is based on the direction of progress that needs to be made.

The roll and pitch integrators are also used to alter the neutral orientation of the helicopter. The neutral orientation is considered to be the orientation at which the vehicle will hover. Further discussion of this aspect is held until Chapter 6.

5.7 Antenna Translations

As mentioned in Section 3.3.4 the GPS antenna is mounted towards the end of the tail boom on top of the horizontal fin. This location is utilized to assure that the GPS antenna is not located near any of the vehicle's structures or components that might interfere with its satellite reception. This method of installation induces a constant error into the positional data of the vehicle. Due to the size of the vehicle the offset is fairly small (< 2.5 feet). Although the error is small, it does create issues during heading rotations of the vehicle.

Heading rotations cause the vehicle to rotate about the main shaft of the vehicle. As the GPS antenna is mounted on the tail of the vehicle, a rotation about the main shaft appears as a positional movement around a circle. This positional movement is also calculated as a GPS velocity in Section 5.4. To insure that the USL testbed can safely and efficiently change headings during flight two translations are performed to remove both the positional error and velocities caused by the mounting location of the GPS antenna.

5.7.1 Positional Translation

Positional translations are performed to remove the offset between the GPS antenna and the rotational axis of the helicopter. This translation is only performed in the *gpsRollPitchError* function, described in Appendix G, and does not modify the lateral or longitudinal positions outside of that function.

The positional translation algorithm first determines the antenna's positional error by rotating its constant offset in the local coordinate frame to the world coordinate frame. This is done using

$$\begin{bmatrix} P'_y \\ P'_z \\ P'_x \end{bmatrix} = \begin{bmatrix} \cos(-C_{D2R}(\phi)) & 0 & \sin(-C_{D2R}(\phi)) \\ 0 & 1 & 0 \\ -\sin(-C_{D2R}(\phi)) & 0 & \cos(-C_{D2R}(\phi)) \end{bmatrix} * \begin{bmatrix} P_y \\ P_z \\ P_x \end{bmatrix} \tag{27}$$

where P is the positional offset of the GPS antenna in the world coordinate frame. Note that the x component of the positional offset should always be zero in the local coordinate frame. This is because the antenna is located on the boom of the helicopter and thus located on the y axis of the local coordinate frame.

P' is then converted to offsets in decimal hours format using

$$L_X = \left(P'_X * M_X\right)/1000000 \tag{28}$$

to calculate the longitudinal offset and

$$L_Y = \left(-P'_Y * M_Y\right)/1000000 \tag{29}$$

to calculate the lateral offset. Note that the product of the offset, P', and the GPS resolution, M, is divided by one million to adhere to NMEA positional units.

Although the offsets are now in a form that can be directly removed from the GPS provided positions, to do so may cause unexpected results. As mention in Section 5.4 there is a noticeable time offset between IMU data and GPS data. This offset was approximated to be a one second delay. To account for this, changes to the positional translation, Tp, are delayed for up to one second. This is done by calculating and storing the difference between the corrections calculated in (28) and (29) for the current GPS time step and the previous GPS time step using

$$\begin{bmatrix} Ts_{X_\tau} \\ Ts_{Y_\tau} \end{bmatrix} = \begin{bmatrix} L_{X_\tau} \\ L_{Y_\tau} \end{bmatrix} - \begin{bmatrix} L_{X_{\tau-1}} \\ L_{Y_{\tau-1}} \end{bmatrix}. \tag{30}$$

Note that Ts only stores data for the last one second's worth of GPS data and its size is equal to the frequency of the GPS data. As such, when a new set of data is added to Ts the oldest set of data must be removed. Any non-zero value removed for Ts is added to Tp. This ensures that all translations will be incorporated into the position calculation within one second.

It should be noted that GPS positional data may not be delayed for a full second. Thus this algorithm monitors changes in the positional data and determines if these changes correspond with changes caused by a rotation. This is done by comparing the difference between the current GPS position and the previous GPS position, Ld. If a difference exists on the lateral, Ld_Y, or longitudinal, Ld_X, axis it is compared with that axes corresponding values stored in Ts. These

comparisons are used to make corrections to the position translation and are calculated using

$$Tp_A = \begin{cases} Tp_A + Ts_{A_{\tau-i}} & for\ \ Ld_A = Ts_{A_{\tau-i}} \\ Tp_A + Ts_{A_{\tau-i}} & for\ \ Ld_A > Ts_{A_{\tau-i}} > 0\ or\ Ld_A < Ts_{A_{\tau-i}} < 0 \\ Tp_A + Ld_A & for\ \ Ts_{A_{\tau-i}} > Ld_A > 0\ or\ Ts_{A_{\tau-i}} < Ld_A < 0 \\ Tp_A & otherwise \end{cases}, \tag{31}$$

$$Ts_{A_{\tau-i}} = \begin{cases} 0 & for\ \ Ld_A \geq Ts_{A_{\tau-i}} > 0\ or\ Ld_A \leq Ts_{A_{\tau-i}} < 0 \\ Ts_{A_{\tau-i}} = Ts_{A_{\tau-i}} - Ld_A & for\ \ Ts_{A_{\tau-i}} > Ld_A > 0\ or\ Ts_{A_{\tau-i}} < Ld_A < 0 \\ Ts_{A_{\tau-i}} & otherwise \end{cases}, \tag{32}$$

and

$$Ld_A = \begin{cases} Ld_A = Ld_A - Ts_{A_{\tau-i}} & for\ \ Ld_A > Ts_{A_{\tau-i}} > 0\ or\ Ld_A < Ts_{A_{\tau-i}} < 0 \\ Ld_A = 0 & for\ \ Ts_{A_{\tau-i}} \geq Ld_A > 0\ or\ Ts_{A_{\tau-i}} \leq Ld_A < 0 \\ Ld_A & otherwise \end{cases} \tag{33}$$

where A represents the axis, either longitudinal or lateral, and i represents a variable used to traverse the data stored in Ts. Note that i is initialized to be equal to the frequency of the GPS and is reduced by one after (31), (32), and (33) have been performed on both the lateral and longitudinal axis. The comparison calculations are concluded after the $i = 0$ calculations are performed. It should be mentioned that values stored in Ts are modified in (32). This done to ensure that corrections made during these calculations are not reused in later calculations.

Values Tp_Y, the latitude offset, and Tp_X, the longitude offset, are always removed from the GPS supplied latitude and longitude before positional error calculations are performed. This assures that the positional error calculated in Section 5.2 represents the error from the rotational axis, or main shaft, of the helicopter and not the location of the GPS antenna.

5.7.2 Velocity Translation

Although the positional translation algorithm can effectively translate the GPS location, its translations can not be directly used by the velocity calculation. The positional translation algorithm will only delay a known offset for up to one second. It is feasible that the GPS position take longer than one second to recognize the movement. Thus, velocities calculated using the

translated positions would calculate a velocity from the offset induced by the position translation algorithm and an equal but opposite velocity from the delayed GPS position.

To account for these issues the velocity translation algorithm was developed. This algorithm is designed to remove velocities that would be calculated due to rotations but will never increase or create a velocity. Unlike the positional translation algorithm which attempts to match the GPS antenna's position with the vehicles position, this algorithm only attempts to remove differences between consecutive GPS positions that may have been caused by heading rotations. The velocity translation algorithm is only performed in the *calcVelocity* function, described in Appendix G, and is almost an exact replica of the position translation algorithm.

The velocity translation algorithm, like the positional translation algorithm, first determines the antenna's positional error using (27) and then calculates the lateral and longitudinal offsets using (28) and (29). Ts is then calculated using (30) as described in Section 5.7.1. Note that any unused data that is removed from Ts due to time is simply discarded and is in no way utilized in this algorithm. This method prevents the algorithm from ever creating or increasing a velocity. Unlike the position translation algorithm, the velocity translation algorithm does not use (31), (32), and (33) to calculate modifications to the translation. Instead this algorithm uses

$$Tv_A = \begin{cases} Tv_A + Ts_{A_{\tau-i}} & for \;\; Ld_A = Ts_{A_{\tau-i}} \\ Tv_A + Ts_{A_{\tau-i}} & for \;\; Ld_A > Ts_{A_{\tau-i}} > 0 \; or \; Ld_A < Ts_{A_{\tau-i}} < 0 \\ Tv_A + Ld_A & for \;\; Ts_{A_{\tau-i}} > Ld_A > 0 \; or \; Ts_{A_{\tau-i}} < Ld_A < 0 \\ Tv_A & otherwise \end{cases}, \tag{34}$$

(32), and (33) to directly calculate the velocity translation, Tv, which is always initialized to zeros. As described in Section 5.7.1, i represents a variable used to traverse the data stored in Ts and is initialed to equal to the frequency of the GPS. The variable is reduced by one after (34), (32), and (33) have been performed on both the lateral and longitudinal axis. The comparison calculations are concluded after the $i = 0$ calculations are performed.

Values Tv_Y, the latitude offset, and Tv_X, the longitude offset, are always removed from the GPS supplied latitude and longitude before the GPS velocity calculations are performed. Note that modifications to the GPS position data are not carried outside of the velocity calculation and do not affect any other algorithm or calculation.

Chapter 6
Controller

In autonomous vehicles, the controller is the heart of system. First, a controller must assure that it can efficiently and effectively perform its assigned tasks. This is especially difficult in UAV vehicles which require constant control inputs. Unlike Unmanned Ground Vehicles (UGVs) which can sit ideal for large periods of time while calculations are being performed, a UAV must receive control signals at a reasonably high rate to remain stable. Second, UAVs are typically highly nonlinear. A controller must be able to adhere to the dynamics of the vehicle for which it is designed. Last, a controller must safeguard its vehicle and the surrounding environment from harm. This is especially true for UAV vehicles as they present an enormous risk to onlookers and surrounding property.

6.1 Fuzzy Logic

Fuzzy Logic was chosen as the methodology for developing the controllers for the USL helicopter testbed. Fuzzy logic provides multiple advantages to control development including the use of linguistic variables, functionality using imprecise or contradictory input, ease of modification, and the ability to directly implement knowledge from multiple experts.

6.1.1 Overview

Fuzzy logic is a methodology based on assigning degrees of membership to linguistic terms known as membership functions. As typical linguistic descriptions are loosely defined so are fuzzy sets. For example, an individual that can run two meters per second might be defined by some as being fast while others may define the individual as average. Thus, the man has some degree of membership to both the linguistic terms fast and average. The specifics values of the individual's membership to these terms are defined by the membership functions. Figure 42 details three possible membership functions for the runner.

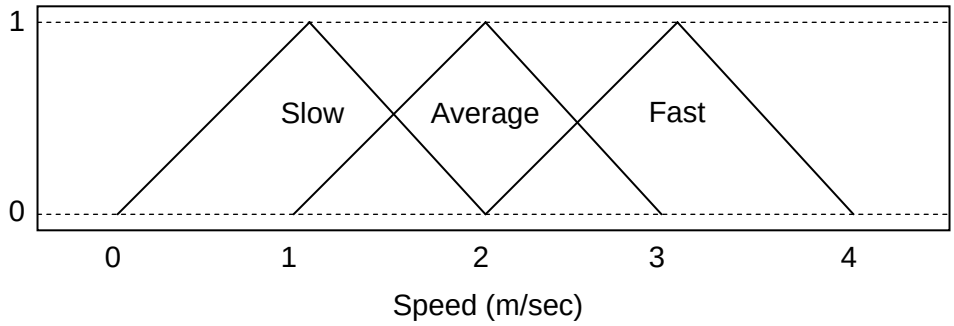

Figure 42: Example of Three Fuzzy Sets for the Speed of a Runner

Note that in Figure 42 each membership function is triangular. Membership functions can take on any shape that accurately depicts the linguistic variable. Although fuzzy sets typically take on values from [0,1], they can take on any range of values.

The act of converting crisp inputs, such as the speed of the runner mentioned above, into memberships of linguistic terms is known as fuzzification. Fuzzification is the first step in using fuzzy logic as a controller.

The second step in fuzzy control is the inference engine. This is the decision making section of fuzzy logic and functions based on linguistic rules. These rules are typically in the form of if-else statements and are developed from information gained from experts. Experts, for the purpose of this research, are individuals who may have no expertise in fuzzy control but are experts in the system being controlled. These experts create rules based on the linguistic sets. An example of a fuzzy rule for controlling the speed of treadmill could be "*If a runner is fast and the speed is slow then increase the speed*". The crisp inputs would be the numerical speed of both the runner and the treadmill.

The strength of an output from a rule is directly affected by each input's degree of membership to that set. Although multiple methods exist for determining the strength of the output, one classical method is the max method. In this method the strength of the output is directly proportional to the degree of membership of the input with the highest degree of membership to its set. Note that if the inputs into the fuzzy controller fell into more than one fuzzy set there would be multiple rules activated. This procedure will most likely produce multiple fuzzy outputs.

The last step in fuzzy control is to convert the fuzzy outputs to crisp outputs. This step is known as defuzzification for which multiple methods exist. One of the more common methods

for defuzzification will assign crisp values to each of the linguistic outputs. This method then multiplies the strength of each fuzzy rule by the value of its linguistic output. These values are then averaged and used as the crisp output.

6.2 Controller Methodology

The USL helicopter controllers were developed with several key aspects in mind. First, the controllers needed to be as robust as physically possible. This was required as the project was originated with the desire to operate a small fleet of autonomous helicopters. Since the dynamics of small RC based UAVs varies greatly even between replicas, any controllers developed for use on multiple UAV's had to be robust. The design of robust controllers coupled with modular software, see Chapter 4, and modular hardware, see Chapter 3, would allow for the fastest implementation of an autonomous UAV fleet.

One important aspect in the design of the controllers is the creation of fuzzy rules based on general RC helicopter flight and not on the Joker Maxi 2 helicopter flight. This allowed focus to be placed on providing control based on the concepts of flight rather than the specifics of one particular helicopter. This is done by utilizing an expert in RC helicopter flight that had little experience with the Joker Maxi 2. This assured that decisions were not based on a single platform.

The second aspect by which the USL helicopter's controllers were designed is the desire to keep the rule base manageable. More precisely, it is desired that modifications to the rule base be quick, simple, and straight forward. This development aspect is introduced as the vehicle is designed to be a testbed. Once developed the testbed could be utilized to create nonlinear coupled controllers but is deemed "overkill" for the original testbed design.

Although helicopter dynamics are heavily coupled, the degree to which they are coupled is heavily influenced by the types of flight being performed. Aggressive maneuvers such as stall and knife edge turns require heavy compensation from all aspects of control. Non-aggressive maneuvers such as hovering and simple waypoint flight can be virtually decoupled. Decoupling of the control is further assisted by the rate and resolution at which newer technology sensors operate. Although the controllers may not immediately compensate for coupling the input from sensors allows compensation to be performed before a noticeable degradation occurs. The work presented here decouples control into four categories: collective control, yawing control, roll control, and pitch control. Note that throttle control is simply held constant during flight.

6.2.1 Generic Vehicle Model

Although the controllers described by this work were designed without a highly developed model they were designed for controlling a specific class of vehicles, i.e. helicopters. This class of vehicle has several fundamental properties that classifies it as a helicopter. Thus, these fundamental properties correspond to a generic model for this class of vehicle.

First, only two controllable surfaces exist on a helicopter: the main rotor and the tail rotor. In the local coordinate frame, the main rotor can directly control a vertical force and two angular forces (latitudinal and longitudinal). The tail rotor can directly control an angular force about the local vertical axis. Thus, there is only a single controllable aspect of the vehicle that causes a non-angular velocity. This non-angular velocity, now referred to as lift, can only create a vertical force in the local coordinate frame. Thus, velocities within the world frame are directly connected to the orientation of the vehicle and can be calculated using

$$V_{TOT} = \int F_{MR} + \grave{F}_G + \grave{F}_D + \grave{F}_E \,, \tag{35}$$

where $\grave{F}_G$ is the force vector produced by gravity in the local frame, F_{MR} is the force vector produced by the main rotor in the local frame, $\grave{F}_D$ is force vector produced by drag in the local frame, and $\grave{F}_E$ is the force vector produced by all other miscellaneous forces in the local frame. Miscellaneous forces encompasses all other forces acting on the helicopter including wind, temperature, weight distribution, etc. Note that $\grave{F}_E$ is assumed to be far smaller than $\grave{F}_G$ or F_{MR}. As $\grave{F}_E$ begins to approach either of these forces the vehicle will most likely become uncontrollable.

It should now be mentioned that V_{TOT} has a natural threshold which prevents the velocity vector from growing without bound. As V_{TOT} increases the drag, $\grave{F}_D$, will increase as an opposing force. Since, $\grave{F}_G$ has a constant strength, F_{MR} is bound by the mechanics of the vehicle, and $\grave{F}_E$ is considered to be small, V_{TOT} will ultimately be constrained by $\grave{F}_D$.

Second, the bulk weight for helicopters is, and should be, centrally located under the main rotor. Thus the center of gravity falls below the vehicle's natural rotational axis located at the center of the main rotor. This design causes a simulated pendulum effect where the bulk weight of the vehicle will naturally swing below the main rotor. This pendulum effect is then

dampened by external forces such as drag. Thus, in the absence of a large angular force the vehicle will naturally stabilize in a near horizontal fashion. This fact allows the controllers to prevent excessive angles with very little control effort.

Using this generic and heavily generalized information a controller was developed to calculate a desired velocity and then achieve that velocity. The controller attempts to maneuver and stabilize the vehicle simply by controlling the velocities of the vehicle.

6.2.2 Control Through Position Error

Position error is the driving force behind maneuvering the vehicle. The controllers described in this work are designed to utilize the position error to determine a desired velocity. Note that desired velocity is strictly a fuzzy variable which describes the desired direction and strength for velocity. This desired velocity is then used by the controllers, along with the state of the vehicle, to determine control output. Desired velocity is calculated for the lateral, longitudinal, and vertical axes as well as the heading orientation.

The heading velocity for the USL testbed is controlled strictly by the heading hold gyro, discussed further in Section 6.3.4. Thus, the desired velocity calculated for the heading is sufficient for controlling the tail rotor.

It should be noted that the desired velocity is never actually calculated. It is described here to show the decision processes that the fuzzy controllers attempt to mimic. This type of description is used for the remainder of this chapter.

6.2.3 Control Through Velocity

As control is based on the desired velocity the first input evaluated by the fuzzy controllers is the velocity of the vehicle. From (35), velocities are directly proportional to the lift and orientation of the helicopter. Assuming that the lift is essentially stagnant, the lateral and longitudinal velocities can be controlled by the orientation of the vehicle. Thus the lateral and longitudinal controllers calculate a desired orientation. This fuzzy variable is calculated by comparing the desired velocity to the actual velocity. If the vehicle's velocity is greater than the desired velocity an angle is selected that should reduce the speed of the vehicle. If the vehicle's velocity is less than the desired velocity an angle is selected that should increase the speed of the vehicle.

Vertical velocities are controlled by calculating a desired change in lift, or collective. This desired change is based on the difference between the current velocity and the desired velocity. Note that during flight the natural lift is assumed to be enough lift to counteract gravity and thus create vertical stability. Desired velocities create the need to reduce or increase the natural lift. If the desired velocity is up the lift is increased. If the desired velocity is down the lift is decreased.

6.2.4 Control Through Acceleration Variant

Section 6.2.3 detailed how the velocity input is used to determine a desired lateral and longitudinal orientation as well as the desired change in lift. Although the controller does its best to determine values that will provide the desired velocities, the specific velocity obtained given an orientation is heavily dependent on the type and configuration of the vehicle. In an attempt to compensate for this type of variation, the desired orientations, calculated in Section 6.2.3, are modified according to the acceleration variant input.

The acceleration variant input determines if the rate at which the velocity is changing is appropriate. For the lateral and longitudinal axes this input attempts to modify the desired orientation to fit the vehicle being used. If the desired angles calculated in 5.2.3 are producing a velocity at too great a rate the desired angles are reduced. If the desired angles are producing a velocity at too slow a rate the desired angles are increased. This concept is also used to modify the change in lift. Once the change in lift has been corrected by the acceleration variant the desired collective is output from the controller. Note that these modifications do not carry from operation to operation and are only calculated using the most current state data.

6.2.5 Control Through Orientation

Now that the desired angles have been calculated the controller must determine an output capable of achieving these angles. This is performed by calculating a desired angular rate which identifies the direction and speed at which the vehicle will rotate. This value is based on the difference between the current and desired angles. If the current angle is too small the controller attempts to create an angular rate that will increase the vehicle's angle. If the current angle is too large the controller attempts to create an angular rate that will decrease the vehicle's angle.

To assure that the vehicle does not obtain an orientation that is difficult to control limitations were designed into the fuzzy rules. These rules allow angles to be achieved within a constrained threshold. The values of these constraints are designed directly into the fuzzy controllers via the two extreme Membership Functions (MFs) of the orientation inputs. As the vehicle begins to approach the assigned thresholds for orientation the desired angular rates become skewed. This skew reduces any desired angular rate that would increase the orientation of the vehicle. Once the vehicle has surpassed the threshold for orientation the controller will only calculate desired angular rates that decrease the angle of the vehicle.

6.3 Implementation

The four fuzzy controllers for the USL testbed were developed in Matlab utilizing Sugeno constant fuzzy logic and a weighted average defuzzification method. All rules for the controllers are based on the 'and' method and use membership products to determine the strength of each rule. Each controller has a single output which ranges from [-1,1] corresponding to the minimum and maximum PW for that particular control respectively. It should be noted that all of the inputs, with the exception of orientation (angle), for the controllers are in various units of feet (i.e. feet, feet/second, feet/second^2). Orientation inputs are based on Euler angles and are in units of degrees.

6.3.1 Assumptions

Several assumptions were made during the design of the USL controllers. First, control for both the roll and pitch is based on the assumption that a level vehicle will create minimal velocity and accelerations. Although this statement is not typically valid, there is an orientation that is optimal for creating minimal velocities. This optimal orientation may or may not be perfectly level and is periodically dynamic. The optimal orientation is based on multiple variables including weight distribution, mechanical setup, and wind to name only a few. To compensate for this the navigation function implements three integrators, discussed in Chapter 5. These integrators are not only used to adjust the vehicles trims but also to modify its internal definition of level. If the vehicle drifts off course or cannot reach a desired position the integrator slowly increases the vehicles trim in that direction. In parallel the vehicles definition of level is rotated in the direction of the integrator. This allows the vehicle to continuously attempt to find

the optimal definition of level which will ultimately increase the accuracy of the controller and validate the afore mentioned assumption.

Second, the collective controller assumes that a neutral, or zero, command will hover the vehicle. This statement is also typically untrue. The hovering collective varies greatly based on the wind, throttle, and battery charge, to name only a few. Thus the collective integrator, discussed in Chapter 5, continuously updates the trim value of the collective. This dynamically increases or decreases the neutral value of the controller to its optimal location thus validating the assumption.

6.3.2 Roll Controller

The roll controller is responsible for all longitudinal movement of the helicopter. Decisions for control are based on four inputs: longitudinal position error, longitudinal velocity, roll angle, and longitudinal acceleration variant (discussed in Chapter 5). Note that all inputs are in the local coordinate frame of the helicopter.

The first three inputs, position error, velocity, and angle, are broken into five fuzzy sets: "bigL" (big left), "left", "small", "right", and "bigR" (big right), see Figure 43. Make note that the "left", "right", and "small" sets for each input are triangular in shape and sets "bigL" and "bigR" are trapezoidal. The "bigL" and "bigR" trapezoidal membership functions extend to $-\infty$ and ∞ respectively. This is to assure that any possible value for these inputs can be appropriately handled by the controller.

The fourth input, acceleration variant, is broken into three fuzzy sets: "left", "small", and "right", see Figure 43. Make note that only the "small" fuzzy set is triangular in shape and sets "left" and "right" are trapezoidal. The "left" and "right" trapezoidal membership functions extend to $-\infty$ and ∞ respectively.

To assure that every possible combination of inputs is accounted for a rule is developed for each. This is foremost done to assure that the fuzzy rule set is complete. This method also assures that every combination of inputs is distinctly evaluated by the expert. This assures that input combinations would not be incorrectly grouped together and assigned a single rule. This method creates 375 distinct rules for the roll controller. Although the specific rules are not presented here, they are provided in Appendix H to assure completeness.

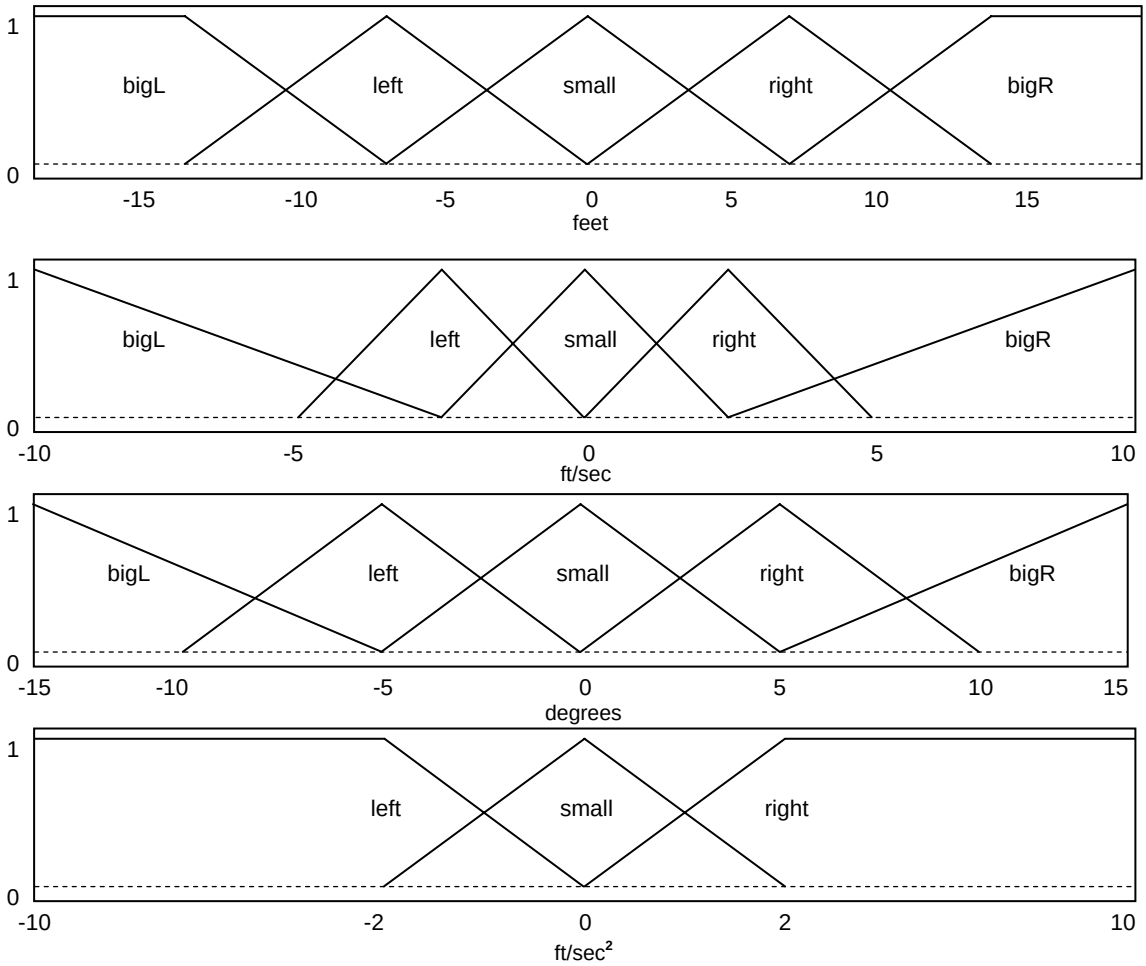

Figure 43: Membership Functions for Positional Error (top), Velocity (second), Angle (third), and Acceleration Variant (bottom) for the Roll Controller

The roll controller has seven possible outputs: “bigL” (big left), “medL” (medium left), “left”, “zero”, “right”, “medR” (medium right), and “bigR” (big right). Each of these outputs is assigned a constant value ranging from [-1,1] and distributed equally around “zero”. The crisp output is determined by averaging the strengths of the fuzzy outputs with their respective constants. Table 7 details the exact values used for outputs for all controllers. Note that all output values were determined by both the expert’s opinion and information gathered during manual flights. The output values were also tuned for performance during testing.

6.3.3 Pitch Controller

The pitch controller is responsible for all lateral movement of the helicopter. Decisions for control are based on four inputs: lateral position error, lateral velocity, pitch angle, and lateral acceleration variant (discussed in Chapter 5). Note that all inputs are in the local coordinate frame of the helicopter.

The first three inputs, position error, velocity, and angle, are broken into five fuzzy sets: "bigF" (big forward), "forward", "small", "backward", and "bigB" (big backward), see Figure 44. Make note that the "forward", "backward", and "small" sets for each input are triangular in shape and sets "bigF" and "bigB" are trapezoidal. The "bigF" and "bigB" trapezoidal membership functions extend to $-\infty$ and ∞. This is to assure that any possible value for these inputs can be appropriately handled by the controller.

The fourth input, acceleration variant, is broken into three fuzzy sets: "forward", "backward", and "small", see Figure 44. Make note that only the "small" fuzzy set is triangular in shape and sets "forward" and "backward" are trapezoidal. The "forward" and "backward" trapezoidal membership functions extend to ∞ and $-\infty$ respectively.

To assure that every possible combination of inputs is accounted for a rule is developed for each. This is foremost done to assure that the fuzzy rule set is complete. This method also assures that every combination of inputs is distinctly evaluated by the expert. This assures that input combinations would not be incorrectly grouped together and assigned a single rule. This method created 375 distinct rules the pitch controller. Although the specific rules are not presented here, they are provided in Appendix H to assure completeness.

It should be noted that rules for both the roll and pitch controllers are identical. This is deemed valid as the tail of an RC helicopter has minimal effect on these two axes. The effect of the tail is further minimized by the heading hold gyro which is responsible for keeping the heading stable.

The pitch controller has seven possible outputs: "bigF" (big forward), "medF" (medium forward), "forward", "zero", "backward", "medB" (medium backward), and "bigB" (big backward). Each of these outputs is assigned a constant value ranging from [-1,1]. The crisp output is determined by averaging the strengths of the fuzzy outputs with their respective constants.

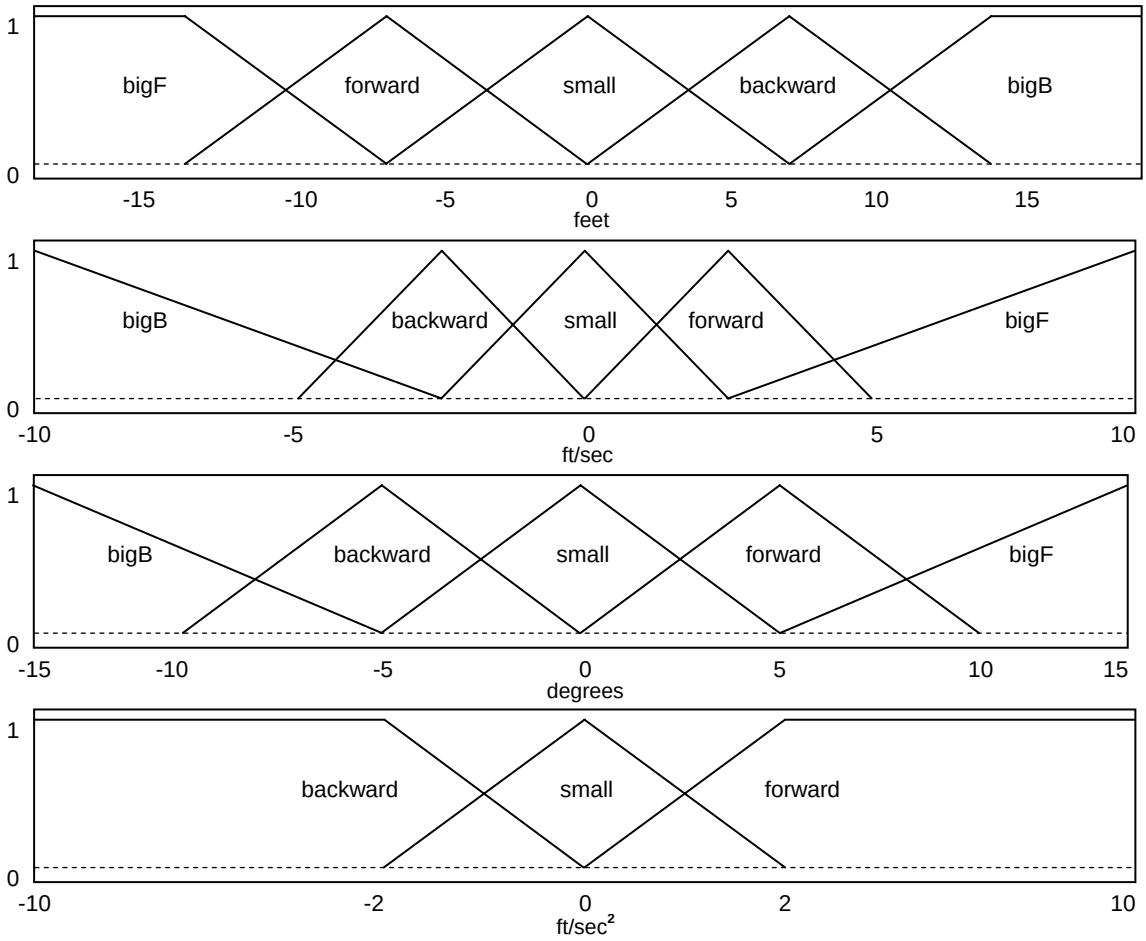

Figure 44: Membership Functions for Positional Error (top), Velocity (second), Angle (third), and Acceleration Variant (bottom) for the Pitch Controller

6.3.4 Collective Controller

The collective controller is responsible for all vertical movement of the helicopter. Decisions for control are based on three inputs: vertical position error, vertical velocity, and vertical acceleration variant. Note that all inputs are in the local coordinate frame of the helicopter.

The first two inputs of the collective controller are broken into five fuzzy sets: “bigD” (big down), “down”, “small”, “up”, and “bigU” (big up), see Figure 45. Make note that the “down”, “up”, and “small” sets for each these inputs are triangular in shape and sets “bigD” and “bigU” are trapezoidal. The “bigD” and “bigU” trapezoidal membership functions extend to -∞

and ∞ respectively. This is to assure that any possible value for these inputs can be appropriately handled by the controller.

The third input, acceleration variant, is broken into three fuzzy sets: “forward”, “backward”, and “small”, see Figure 45. Make note that only the “small” fuzzy set is triangular in shape and sets “forward” and “backward” are trapezoidal. The “forward” and “backward” trapezoidal membership functions extend to ∞ and -∞ respectively.

To assure that every possible combination of inputs is accounted for a rule is developed for each. This is foremost done to assure that the fuzzy rule set is complete. This method also assures that every combination of inputs is distinctly evaluated by the expert. This assures that input combinations would not be incorrectly grouped together and assigned a single rule. This method created 75 distinct rules the collective controller. Although the specific rules are not presented here, they are provided in Appendix H to assure completeness.

The collective controller has seven possible outputs: “bigU” (big up), “medU” (medium up), “up”, “zero”, “down”, “medD” (medium down), and “bigD” (big down). Each of these outputs is assigned a constant value ranging from [-1,1]. The crisp output is determined by averaging the strengths of the fuzzy outputs with their respective constants.

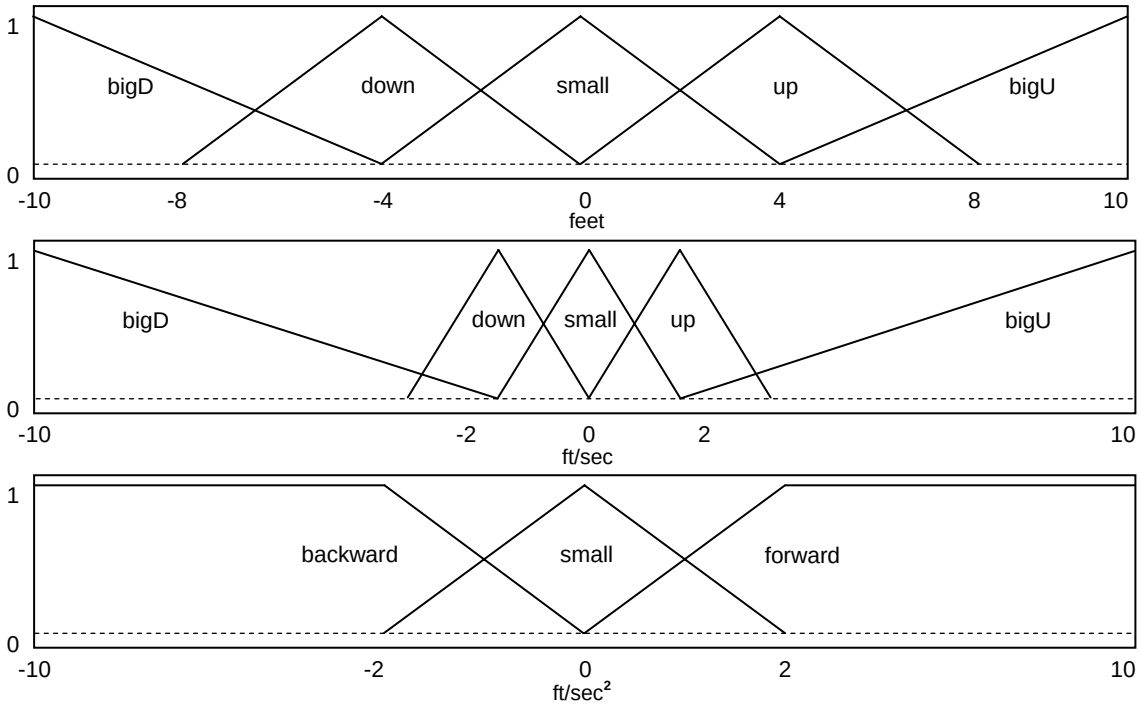

Figure 45: Membership Functions for Vertical Error (top), Velocity (middle), and Variant Acceleration (bottom) for the Collective Controller

6.3.5 Yaw Controller

The yaw controller is responsible for all heading changes of the helicopter. Decisions for control are based on a single input: heading error. Only a single input is required to successfully stabilize and control the heading of the helicopter due to the use of the heading hold gyro. Utilizing of a heading hold gyro converts all control inputs into rate commands. If the control input is zero then the gyro attempts to keep the angular rate at zero. If the control input is not zero then the gyro attempts to create an angular rate proportional to the control.

The yaw controller's input is broken into five fuzzy sets: "bigR" (big right), "right", "small", "left", and "bigL" (big left), see Figure 46. Make note that the "right", "left", and "small" sets for each input are triangular in shape and sets "bigD" and "bigU" are trapezoidal. The "bigD" and "bigU" trapezoidal membership functions extend to -180 and 180 respectively. This is to assure that any possible value for these inputs can be appropriately handled by the controller.

The yaw controller contains only five rules. Although the specific rules are not presented here, they are provided in Appendix H. These rules were designed to simply rotate the vehicle at a rate proportional to the error.

The yaw controller has five possible outputs: "bigR" (big right), "right", "zero", "left", and "bigL" (big left). Each of these outputs is assigned a constant value ranging from [-1,1]. The crisp output is determined by averaging the strengths of the fuzzy outputs with their respective constants.

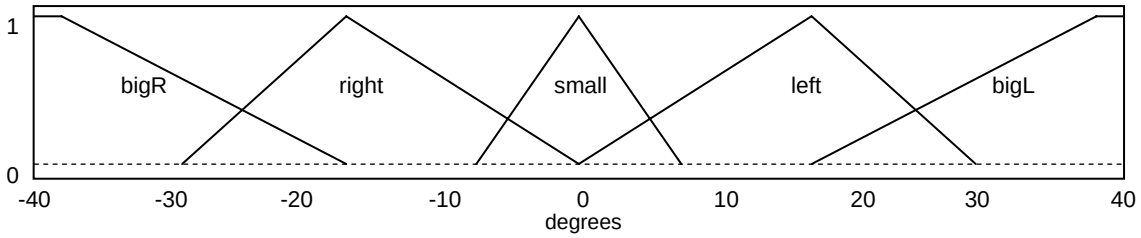

Figure 46: Membership Function for Heading Error for the Yaw Controller

Table 7: Output Values for all Controllers

Controller	Outputs
Roll	bigL = -0.2 medL = -0.15 left = -0.10 small = 0 right = 0.10 medR = 0.15 bigR = 0.2
Pitch	bigF = -0.2 medF = -0.15 forward = -0.10 small = 0 backward = 0.10 medB = 0.15 bigB = 0.2
Collective	bigU = 0.2 medU = 0.15 up = 0.1 small = 0 down = -0.1 medD = -0.15 bigD = -0.2
Yaw	bigL = -0.15 left = -0.06 small = 0 right = 0.06 bigR = 0.15

6.4 Stability

Stability is one of the most difficult aspects of controller design. Proving stability is the act of proving that control input will not amplify the error of the vehicle. This can be done by showing that any set of inputs into the system will not create a positive feedback. Although feasible for systems with a small number of inputs, this method becomes unrealistic as the number of inputs and range of those inputs increase.

Stability can also be proven mathematically by comparing the mathematical model of the controller with the mathematical model of the vehicle. This type of proof assumes that both models are complete and correct. This assumption is typically only valid in completely controlled environments and with linear based systems. The mathematical proof concept has also been extended to non-linear systems by linearizing the system over small sections. This type of proof assumes that variations between the linearized system and the actual non-linear system are

not large enough to cause instability. This is a heavy assumption as the smallest deviation may or may not cause instability.

Stability proofs are typically extremely difficult in fuzzy based controllers. This is due to the fact that fuzzy controllers are prime solutions for systems that cannot be described accurately with a mathematical model. At this time there is no widely accepted mathematical method for proving stability of fuzzy controllers on a non-linear system without an accurate vehicle model. True stability for this type of controller can only be proven by exhausting every possible input into the system. In the case of outdoor flight the number of inputs such as wind, temperature, weight distribution, gravity deviation, and humidity is infinitely large. Thus a stability proof through exhaustive measures is infeasible.

Although it seems hopeless that any type of controller could be considered stable for controlling a helicopter there are acceptable limitations. A good example is the case of a human operator. Humans have shown great ability control and navigate highly complex vehicles utilizing as little as muscle memory. Every day tens of thousands of individuals trust their lives to the stability of these human operators. To date there is no mathematical model for human beings. Thus, there does exist some level of stability assurance, or reliability, which is not a proof but is widely acceptable.

Although this work cannot prove stability of the described controllers, the remainder of this chapter will be dedicated to supplying a level of reliability.

6.4.1 Feedback Instability

Feedback instability is a situation where updates based on sensed errors causes the vehicle to continuously overcorrect thus causing the vehicle to become unstable. At the lowest level the fuzzy controllers tend to have several properties similar to those of open loop controllers. This is due to the fuzzy rules essentially breaking the flight envelope into sections, each with a distinct set of rules. Small alterations during flight will not alter the state of the vehicle significantly enough to move the controllers to a new set of rules. Thus, for short periods of time the system will generally be unaffected by feedback. Even when feedback does affect the vehicle the control response is generally in the form of dampening the nominal control. Nominal control represents the open loop type control dictated by the fuzzy MFs. Utilizing the sensor feedback to dampen nominal control prevents the controllers from overcorrecting. Although this method assist in the reliability of the flight controllers it alone could prevent the vehicle from

have sufficient control to perform a maneuver. This type of issue is compensated for by the trim integrators.

Although sensor feedback within the fuzzy controllers creates minimal reliability issues, a potentially problematic routine external to the fuzzy controllers exists. This routine is the integration function discussed in Section 5.6. This routine contains three variables each of which represents a trim value for the roll, pitch, and collective controllers. These trim values are designed to offset the outputs of the controllers. For the roll and pitch controllers this value also offsets the angles provided to the controllers, discussed in Section 6.3.5.

For the integration routine to operate several key factors must be met. First, the positional error of the vehicle must be outside of some threshold of the vehicle. Second, the velocity of the vehicle must be either zero or in the opposite direction of the error. Third, the variant acceleration must be zero or in the opposite direction of the error. If all of these conditions are met the integration routine will increment the variable for that particular controller. Thus, for the integration routine to cause instability it would first have to integrate out the error causing the routine to run and also integrate in an opposite error that is greater than the original error. This instability integration would have to be completed before the effect of the integration could be sensed by the vehicle. Once the effect is sensed by the vehicle one of the previously mentioned conditions would fail to be met and the integration routine would cease to operate. Thus, given an integration routine that integrates the trim at a very high rate the system will be guaranteed to be unstable. To assure that the vehicle remains stable due to trim integration one would typically consult the vehicle model and adjust the routine accordingly. As this work attempts to generalize the vehicle model we must assume that the utilized vehicle is the worst possible case. Although this case cannot be directly defined it can be reasonably assumed that similar vehicles will respond in a similar way. Experimentation has shown that the integration routine takes less than one second to create a visual effect. Thus, the USL testbed helicopter utilizes an integration routine that takes approximately an order of magnitude greater time to integrate out error. Although this decision ensures that automated trimming of the vehicle may take tens of seconds to complete it does provide a strong assurance that trim integration will not cause the vehicle to become unstable.

It was previously mentioned that the trim values for the lateral and longitudinal axes are used to adjust the sensed Euler angles before they are provided to the controllers. This is done to counteract any external forces on the vehicle. For example, a strong wind will require that the vehicle be slightly leaned to hold a position. Thus the optimal orientation for hover is slightly

offset. This routine attempts to determine the offset and remove it from the roll and pitch angles before they are provided to the controller. This will force the controller to create outputs based on the optimal orientation. It should be noted that this assumes that the vehicle's initial state is correctly trimmed for flight when no external forces are encountered. Thus the neutral values for the roll and pitch servos should provide an optimal orientation for hover.

As this function is part of the feedback routine it is a possible source of instability on the vehicle. The angle adjustments made by this function are a linear function of the trim values. Although a small change in trim values has a minimal effect on the state of the vehicle a small adjustment to the orientation will have a large effect. Thus the stability of the vehicle is directly connected to the slope of the adjustment function. Through experimentation it was discovered that the combination of the trim adjustments and angle adjustments may prevent each from reaching optimality. Although optimality is occasionally not achieved by the individual components, the combination of the two components did provide a semi-optimal orientation. This showed that even with an incorrect angle adjustment function a level of optimality could be accomplished. As such, the stability of the system can be assured by utilizing an overly small slope for the adjustment function.

Chapter 7
Simulation

Although the USL testbed is developed for field testing of UAV technology it is not designed to replace simulation testing. First, field testing requires a large amount of overhead time. The time spent simply moving the equipment to the flight location and prepping the vehicles can take hours. Second, field testing has an inherent level of danger. Any testing performed in the field, no matter what safety precautions are taken, inherently places the vehicle, onlookers, and surrounding environment in a potentially hazardous situation. For these reasons initial testing of controllers and most software algorithms are tested in simulation.

7.1 X-Plane

The USL lab utilizes the X-Plane flight simulator for testing and the initial development of software for the USL testbed helicopter. X-Plane is an FAA certified commercial flight simulator that supports both fixed wing and VTOL vehicles. Besides the standard vehicle models, X-Plane supports customized vehicles that can be modeled directly in X-Plane.

7.1.1 Communication

X-Plane is one of the few commercial simulators available that will export state data from the simulated vehicle and import flight control, both in real-time. This is inherently important as any controller testing would require access to both the state of the vehicle and control inputs. X-Plane is able to import and export data through UDP communication. Update rates for X-Plane are configurable between 1 and 50 Hz. It should be noted that X-Plane will not export or import data from the local host (i.e. to IP 127.0.0.1). Thus, communication and controlling software must reside on a separate machine. For USL, communication and control reside on a laptop. This laptop is connected to a desktop running X-Plane via Ethernet and a single dedicated router.

Communicating with X-Plane first requires configuring the simulator to import and export data. This first requires selecting the state data that should be passed to the controlling machine. This is done by selection "Data Output" from the "Settings" menu in X-Plane. This will create a window that allows the user to specify which variables in X-Plane will be exported. This is simply done by checking the UDP checkbox for any variable desired. Note that X-Plane will output the state data in numerical order, skipping over any unselected variables. As a reference for the list of outputs available from X-Plane could not be found one is supplied in Appendix I. Note that the data provided in Appendix I may or may not vary with X-Plane distributions.

The next step is to select the data rate at which X-Plane will operate. This variable is located in the lower right hand corner of the state data window. Setting this variable will request that X-Plane output a single UDP packet containing all the requested data at approximately the selected data rate. Note that state data will not arrive at the exact data rate selected. The variation in timing is a compilation of network delay and the Windows OS design and is beyond the control of the end user.

The last step in setting up communication is to set the IP and port for exporting and importing data. This is done by selecting the "inet 2" tab from the state data window and setting the data receiver and data sender IPs to that of the controlling machine. Note that X-Plane requires the data receiver port be set to 52000 and the data sender port be set to 49000. It should also be noted that firewalls, on either the host or client machine, may block communication with X-Plane.

7.1.2 USL Model

Although X-Plane provides some RC model vehicles their flight characteristics were not typical to actual RC vehicles. Thus, USL utilizes a third party open source model, the RC3D, available at [96]. This model is designed to represent a 90 size RC vehicle. As the Joker Maxi 2 is slightly larger than a 90 size RC helicopter it is lightly tuned by an expert RC pilot to assure that the flight characteristics were similar to that of the USL testbed. Figure 47 depicts the modified RC3D model used by the USL.

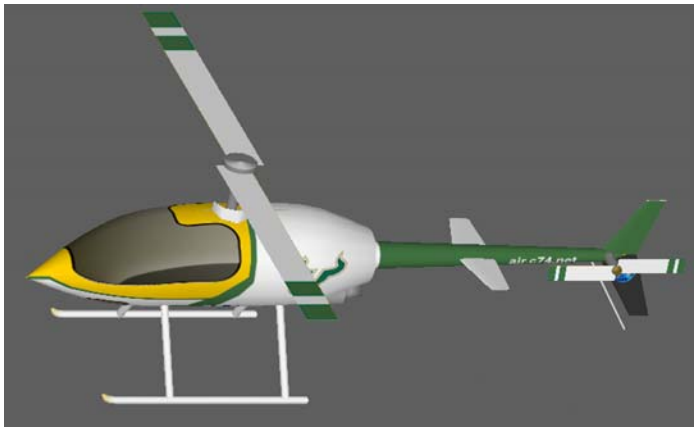

Figure 47: Depiction of the USL X-Plane Model

7.1.3 Tail Stabilization

It should be mentioned that X-Plane attempts to artificially stabilize several components of the vehicle during flight. To assure that the vehicle is being controlled by the USL software all of these stabilization routines were turned off. This presented a somewhat unique issue as the modeled vehicle would not have any heading stabilization. To account for this a small modification is made to the yaw controller mention in Chapter 6. A second input, velocity, is added to the yaw controller. This velocity input is designed to simulate the heading hold gyro by attempting to stabilize the speed at which the heading could be altered. Figure 48 details the membership functions for the modified yaw controller.

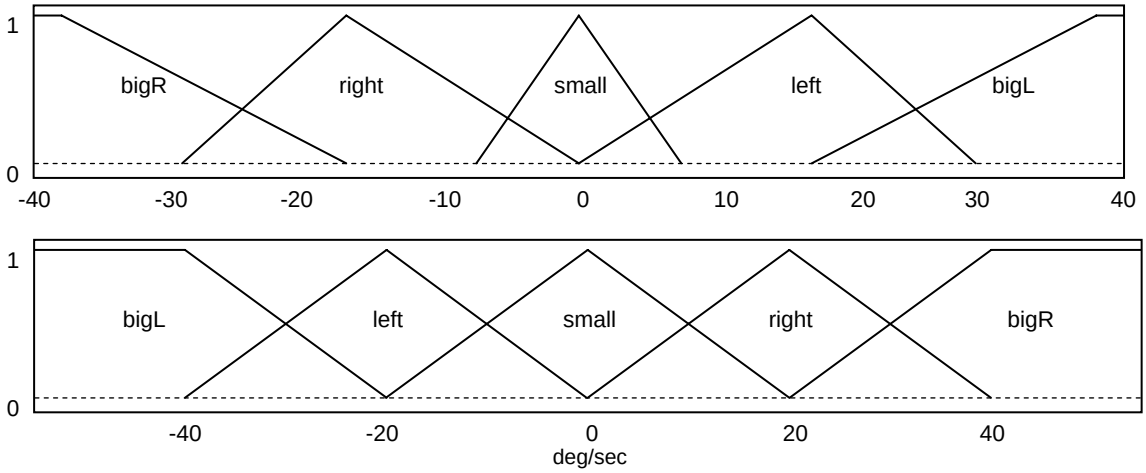

Figure 48: Membership Function for the Heading Error (top) and Angular Rate (bottom) of the Yaw Controller (simulation only)

7.2 Simulink

To interface with X-Plane and provide control a Simulink model was designed using Matlab. This model is used to communicate with X-Plane, parse, extract, and calculate data, and interface with the fuzzy controllers. Figure 49 details the contents of the upper level of the Simulink model.

Although X-Plane could have easily been interfaced with the software described in Chapter 4, key differences with the format and types of data available coupled with the desire to utilize the graphical interface and built-in functionality in Matlab made Simulink a more

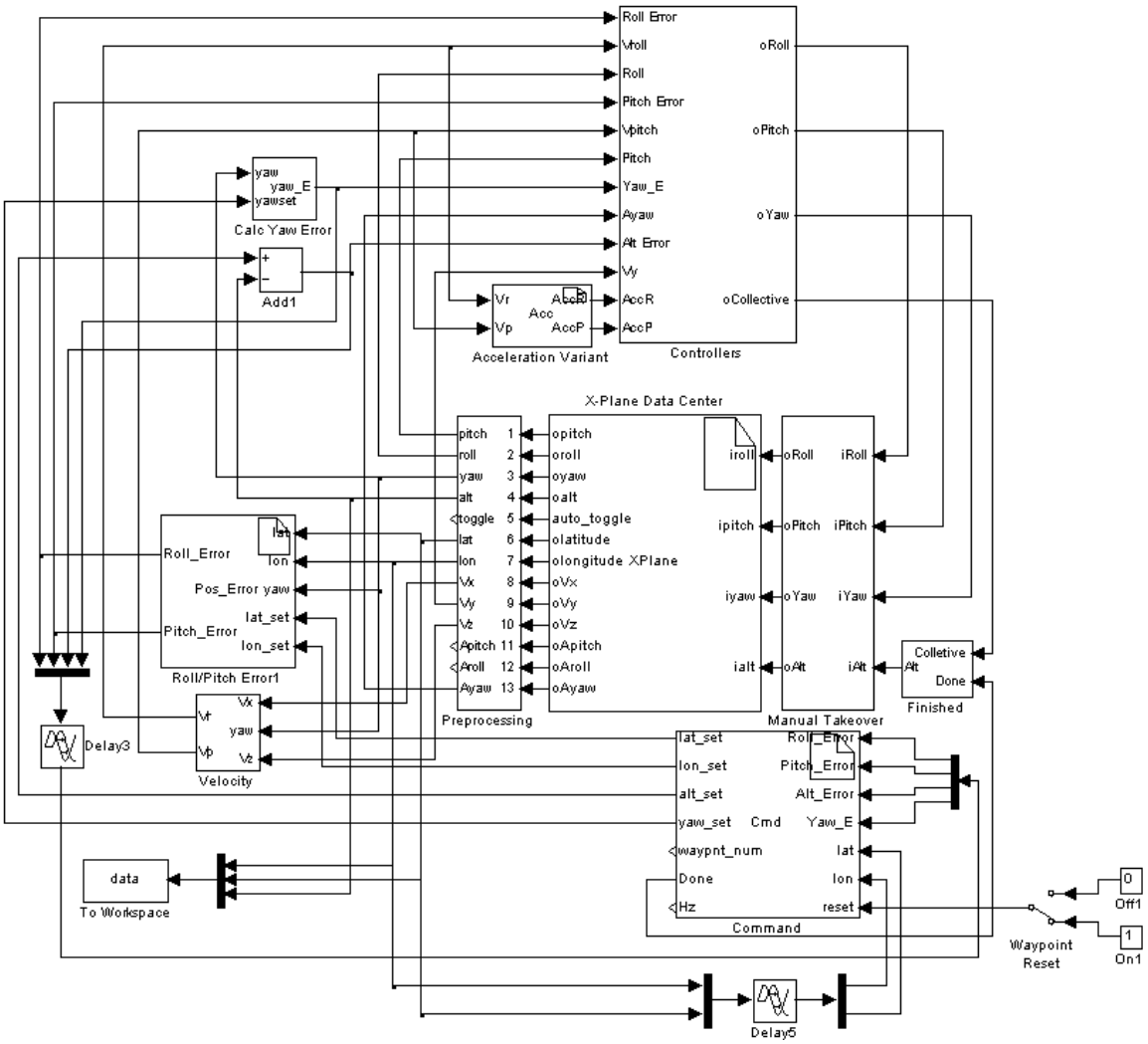

Figure 49: Upper Level of the Simulink Model Used for Simulation Testing

desirable method. Key differences included X-Plane's inability to provide accelerations and the variations in accuracy and data rate from that of the testbed's data.

To assure that algorithms developed in Simulink were easily portable to and from the testbed's software most key functions, or blocks, were written as S-functions. S-functions are C coded Simulink blocks. This type of design allowed for code created and tested in Simulink to be copied and placed directly onto the testbed. This is true for the "Roll/Pitch Error" block, "Variant Accelerations" block, and to a lesser extent the "Command" block. Specifics of these S-function blocks will be discussed later in this chapter.

To store flight data during operation all desired information is multiplexed to the workspace. This provides a multi-dimensional array of data that can be graphed and stored using Matlab's interface.

7.2.1 X-Plane Data Center Block

The "X-Plane Data Center" block (XPDC) is the heart of the Simulink model and is responsible for interfacing with the X-Plane simulator. The XPDC first sets up and binds a socket with the X-Plane machine. This is an initial setup and is only performed once using static variables. Once the socket has been established the XPDC waits for information to arrive from the simulator. Upon receipt, the UDP packet is parsed into a two dimensional array where the row represents the data class as defined by X-Plane and the column represents a specific data reading for that class. An example would be using row one to represent Euler angles and columns one through three of that row to represent the roll, pitch, and yaw.

Once the packet has been correctly parsed the heading input is modified. This is done as the Simulink model expects the yaw to range from (-180, 180] where X-Plane provides a heading ranging from [0, 360). Once this modification has been made the parsed data is assigned to its respective outputs. Make note that data received from X-Plane is in network byte order. Thus, the *ntohl* and *ntohf* commands are utilized to convert the input into host byte order.

Last, the XPDC is responsible for packaging and sending UDP control packets to the simulator. Once control data has been received it is packaged into a "data" type UDP package. This is done by separating the input into bytes of data and placing them in network byte order. The control data is then placed in the UDP packet following the guidelines provided in the *UDP Reference.html* file located in the *X-Plane/Instructions* directory. For clarity Table 8 details all of the input and output variables associated with this block.

Table 8: Variable Names and Descriptions for the "X-Plane Data Center" Block

Variable	Description	Units	Range
Opitch	Euler pitching angle of the vehicle.	degrees	[-180,180]
Oroll	Euler roll angle of the vehicle.	degrees	[-180,180]
Oyaw	Euler yaw angle of the vehicle.	degrees	[-180,180]
Oalt	Altitude of the vehicle above the ground.	feet	[-∞,∞]
auto_toggle	Position of the Auto/Manual toggle switch	integer	1 or 0
Latitude	Latitude position of the vehicle.	degrees	[-90,90]
Longitude	Longitude position of the vehicle.	degrees	[-180,180]
Vx	Velocity down the X, or longitudinal, axis	m/sec	[-∞,∞]
Vy	Velocity down the Y, or vertical, axis	m/sec	[-∞,∞]
Vz	Velocity down the Z, or lateral, axis	m/sec	[-∞,∞]
Apitch	Angular rate of the vehicle in the pitching direction	deg/sec	[-∞,∞]
Aroll	Angular rate of the vehicle in the rolling direction	deg/sec	[-∞,∞]
Ayaw	Angular rate of the vehicle in the yawing direction	deg/sec	[-∞,∞]
Ipitch	Control input for pitch	double	[-1,1]
Iroll	Control input for roll	double	[-1,1]
Iyaw	Control input for yaw	double	[-1,1]
Ialt	Control input for collective	double	[-1,1]

7.2.2 Preprocessing Block

The "Preprocessing" block is simply responsible for assuring that data output from X-Plane is in the same form as the data used by the USL testbed. Although X-Plane provides the data necessary to control the helicopter, this data is sometimes based on axes that are non-standard. To account for this data received following a non-standard reference frame are modified. This is accomplished by inverting the sign of any inputs that are reversed from what would be expected from the USL testbed. Figure 50 details the exact contents of the "Preprocessing" Block.

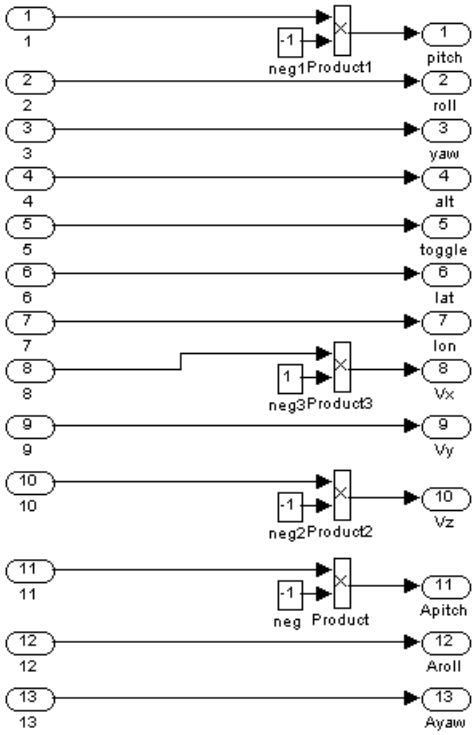

Figure 50: Contents of the "Preprocessing" Simulink Block

7.2.3 Command Block

The "Command" block is a Simulink S-function that partially implements the navigation process described in 4.10. The main difference is that the "Command" block only implements the section of the navigate process responsible for determining the current goal. This includes determining the desired waypoint and heading during all flight maneuvers (i.e. takeoff, waypoint navigation, and landing). The procedures used within this block for determining the desired waypoints and headings follow the description in 4.10. The only modification to the procedures described in 4.10 is that this block does not power up or power down the vehicle. X-Plane initializes the vehicle as powered up and only provides the ability to partially power down the vehicle, both of which were deemed unnecessary for simulation. All waypoints, along with the desired headings are hard coded within this block as a multidimensional array.

Note that the "Command" block is also responsible for outputting a "done" signal to the "Finished" block. This signal represents the lift removal stage of the landing procedure. This signal simply informs the "Finished" block that the vehicle has completed the touch down phase

Table 9: Variable Names and Descriptions for the "Command" Block

Variable	Description	Units	Range
Lat_set	Desired latitude of the vehicle	degrees	[-180,180]
Lon_set	Desired longitude of the vehicle	degrees	[-180,180]
Alt_set	Desired altitude of the vehicle	feet	[-∞,∞]
yaw_set	Desired heading of the vehicle	degrees	[-180,180]
waypnt_num	Identifier for the desired waypoint	integer	[1,∞]
Done	Identifies if vehicle has finished landing sequence	integer	1 or 0
Hz	Data rate from X-Plane	integer	[1,∞]
Roll_Error	Deviation from the desired position on the rolling axis	feet	[-∞,∞]
Pitch_Error	Deviation from the desired position on the pitching axis	feet	[-∞,∞]
Alt_Error	Deviation from the desired position on the altitude axis	feet	[-∞,∞]
Yaw_Error	Deviation from the desired heading	degrees	[-∞,∞]
Reset	Resets the waypoint sequence	integer	1 or 0

of the landing procedure. At this point the "Finished" block will overtake the fuzzy collective control output and output a constant value designed to remove lift from the vehicle. For clarity Table 9 details the input and outputs associated with this block.

7.2.4 Roll/Pitch Error Block

The "Roll/Pitch Error" block is an S-function that directly implements the algorithm discussed in 5.2. This algorithm determines the lateral and longitudinal offset of the vehicle from its goal. The source code located within this S-function is a literal copy-and-paste from the algorithm described in 5.2 and the pseudocode detailed in Appendix G, thus will not be reiterated here.

7.2.5 Velocity Block

The "Velocity" block is designed using the Simulink library. This function is vastly different from the velocity function described in Chapter 5. This is due to the strong variations between the data provided by the simulator and the data provided by the actual helicopter. First, velocity on the testbed is calculated by integrating accelerations and then compensating for drift in the calculation using GPS calculated velocities. As the simulator cannot provide accelerations this method is not feasible. Second, the simulator provides GPS position in floating point

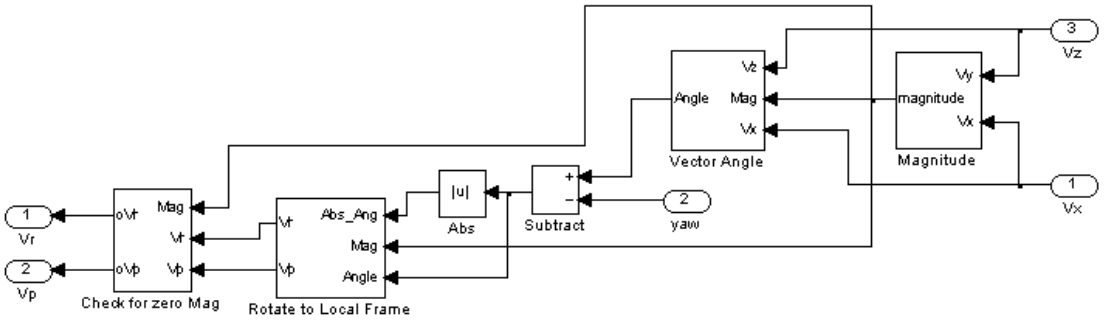

Figure 51: Contents of the "Velocity" Simulink Block

notation. The use of floating point notation significantly reduces the accuracy of the data from that of an NMEA message.

To account for these issues velocity is calculated using X and Y velocities provided by the simulator. This data provides the Simulink model with sub-centimeter velocity in the world coordinate frame at 50 Hz. These components are used to calculate a two dimensional velocity vector. This vector is then transformed to the local coordinate frame which is supplied the roll and pitch velocities for the fuzzy controllers. Figure 51 details the contents of the "Velocity" block.

The "Velocity" block first calculates the magnitude of the two velocity components. This is performed in the "Magnitude" block using

$$Mag = \sqrt{Vx^2 + Vz^2} \tag{36}$$

where Vx is the velocity component of the X, or longitudinal, axis and Vz is the velocity component of the Z, or lateral, axis. Next, the angle formed by the velocity vector is calculated in the "Vector Angle" block using

$$V_{ang} = \begin{cases} C_{R2D}\left(\sin^{-1}\left(\dfrac{Vx}{Mag}\right)\right) & for\ Vz \geq 0 \\ 180 - C_{R2D}\left(\sin^{-1}\left(\dfrac{Vx}{Mag}\right)\right) & for\ Vz < 0\ and\ Vx \geq 0\,. \\ -180 - C_{R2D}\left(\sin^{-1}\left(\dfrac{Vx}{Mag}\right)\right) & for\ Vz < 0\ and\ Vx < 0 \end{cases} \tag{37}$$

Note that V_{ang} is computed by dividing by Mag which may or may not be zero. This anomaly is handled by the "Check for zero Mag" block discussed later in this section. As the components

of the velocity vector are in the world coordinate frame the computed angle is a value, in degrees, from north. This computed angle is then subtracted from the heading of the vehicle producing V_{off} . This offset is then used to rotate the X and Y components of the velocity vector to the local coordinate frame. This is performed in the "Rotate to Local Frame" block using

$$V_{Rtmp} = \begin{cases} Mag * \dfrac{V_{off}}{90} & for \ |V_{off}| \le 90 \\ Mag * \left(\dfrac{180 - V_{off}}{90} \right) & for \ |V_{off}| > 90 \ and \ V_{off} \ge 0 \\ Mag * \left(\dfrac{-180 - V_{off}}{90} \right) & for \ |V_{off}| > 90 \ and \ V_{off} < 0 \end{cases} \quad (38)$$

to calculate a velocity in the roll direction and

$$V_{Ptmp} = \begin{cases} Mag * \left(1 - \left| \dfrac{V_{off}}{90} \right| \right) & for \ |V_{off}| \le 90 \\ -Mag * \left(1 - \left| \dfrac{180 - V_{off}}{90} \right| \right) & for \ |V_{off}| > 90 \ and \ V_{off} \ge 0 \\ -Mag * \left(1 - \left| \dfrac{-180 - V_{off}}{90} \right| \right) & for \ |V_{off}| > 90 \ and \ V_{off} < 0 \end{cases} \quad (39)$$

to calculate a velocity in the pitch direction. Last, a check is performed by the "Check for zero Mag" block to assure that the magnitude is not zero. This block simply assigns V_{Rtmp} to Vr and V_{Ptmp} to Vp if Mag is non-zero. If Mag is zero Vr and Vp are assigned a value of zero.

7.2.6 Calc Yaw Error Block

The "Calc Yaw Error" block is simply used to determine the deviation, in degrees, from the current heading to the local heading. It should be mentioned simple subtraction of the desired heading from the current heading is insufficient. This type of calculation could result in a deviation greater than 180 degrees. Thus a check must be performed to account for this anomaly and appropriate adjustments made. To adhere to these issues the "Calc Yaw Error" block assigns output using

$$H_E = \begin{cases} H_{set} - \phi & for \ -180 \le (H_{set} - \phi) \le 180 \\ (H_{set} - \phi) - 360 & for \ (H_{set} - \phi) > 180 \\ (H_{set} - \phi) + 360 & for \ (H_{set} - \phi) < -180 \end{cases} \quad (40)$$

where H_E is the heading, or yaw, error and H_{set} is the heading set point.

7.2.7 Variant Accelerations Block

The "Variant Accelerations" block is an S-function that attempts to calculate accelerations from velocity. This block is a variation of the algorithm described in 5.5. Due to the precision of the data provided by the simulator filtering is deemed unnecessary. Thus this block simply calculates the change in velocity between consecutive operations. This change is then divided by the amount of time passed between operations. Note that this calculation is performed using velocities in the local coordinate frame. Thus the accelerations generated by this block are also in the local coordinate frame and do not need to be rotated before being supplied to the controllers.

7.2.8 Fuzzy Controllers Block

The "Fuzzy Controllers" block is responsible for supplying the appropriate data to the fuzzy controllers. Figure 52 depicts the contents of the "Controller" block. It should be mentioned that the "Delay" blocks shown throughout the Simulink model are designed to synchronize the model. Without these blocks the model does not know where to pause to wait for new data. All delays within the Simulink model are arbitrarily set to 0.001.

The "Controllers" block is also responsible for adding trim values to the output of the fuzzy controllers. These trim values were determined through trial and error and are simply an attempt to counteract any constant offsets.

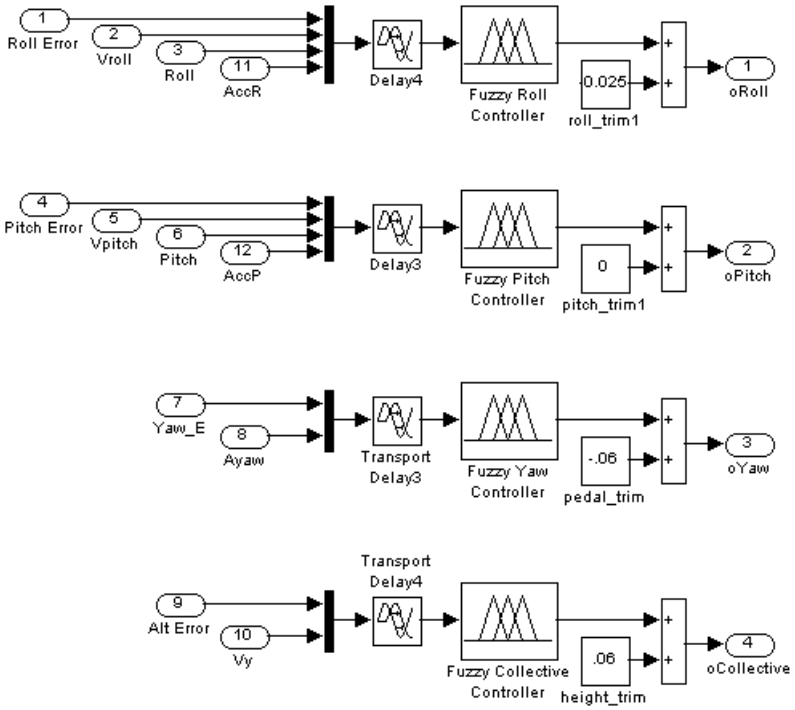

Figure 52: Contents of the "Controllers" Simulink Block

7.2.9 Manual Takeover Block

The "Manual Takeover" block is designed to simulate the ability to remove or grant computer control of the helicopter from a radio controller, see Figure 53. This is done utilizing the USB radio controller supplied with the Realflight G2 simulator. This simulator is not used in the development of the USL helicopter testbed but is utilized to train safety pilots. Utilization of the G2 controller required the Matlab joystick toolbox and minor Windows XP setup.

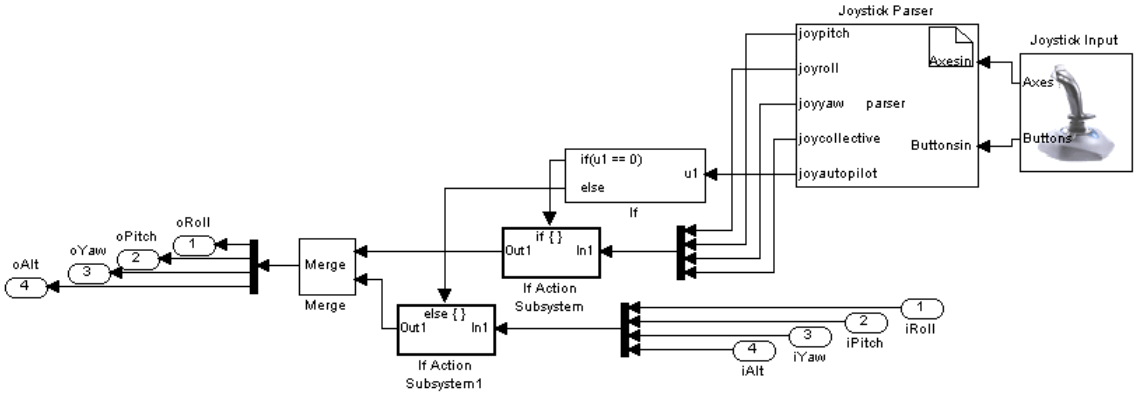

Figure 53: Contents of the "Manual Takeover" Simulink Block

Setting up the joystick for Microsoft Windows simply requires running the joystick calibration wizard. The wizard is simply used to find the center and endpoints of the controller. Once calibrated the controller can be used by the Matlab joystick toolbox.

The "Manual Takeover" block is responsible for gathering and parsing data from both the radio controller and the fuzzy controllers. Parsing of the joystick data is accomplished by using the "Joystick Parser" S-function. This block takes in two one dimensional arrays from the "Joystick Input" block. These arrays correspond to the X and Y positions of the two RC sticks and the positions of all buttons and levers. Through trial and error the data of each individual element is discovered and piped to its appropriate output. The "Joystick Parser" has five outputs. Four of these five outputs are the counterparts for the roll, pitch, yaw, and collective commands output by the fuzzy controllers. The fifth output is the position of one of the levers located on the radio controller. This lever is used to determine which set of outputs will be passed out of the "Manual Takeover" block. If the lever is high, or equal to one, the outputs from the fuzzy controller are passed through the "Manual Takeover" block. If the lever is low, or equal to zero, the outputs of the radio controller are passed through.

Chapter 8
Experiments and Results

Due to the shear breadth of this research the experimentation had to be performed on multiple levels. This included testing and validation of individual sections of the work and testing throughout the integration phase. Experimentations were performed in lab settings, simulations, and outdoors depending on the specific type of experiment. Types of experimentation included endurance, heat, shock, vibration, payload limitations, CPU utilization, and controller validation.

8.1 Payload

Payload experimentations were performed to determine the absolute maximum weight that could be carried by the vehicle. This allowed the USL lab to determine the size and types of sensors and processing equipment that could be utilized by the vehicle as well as the size and types of materials that could be used to design mounts and hardware.

The payload limitation was determined by increasing the weight of the vehicle until the vehicle failed to operate correctly. Increasing the vehicle's weight was done by adding half pound weights to the skids of the vehicle after a successful payload flight. Note that these weights were added in a manner that would assure the weight was equally distributed around the main shaft. Failure was determined by an expert RC pilot and was judged based on the handling characteristics of the vehicle. If at any point the vehicle was unable to maintain altitude or became unstable the payload was considered a failure.

Experimentation showed a maximum payload of approximately 5.5 kg above the stock vehicle's flight weight. This was deemed the maximum payload as the vehicle showed a large degree of degradation in both head speed and response when loaded and flown with a payload of 5.5 kg. Stock weight is defined as the weight of the unmodified Joker Maxi-2 kit with all the necessary equipment to fly, i.e. batteries, servos, radio receiver, etc. For safety and longevity of the equipment the maximum payload for the USL testbed was set at 4.5 kg.

8.2 Endurance

Endurance experiments were performed on all batteries utilized on the testbed. This included the batteries utilized to power the vehicle, servos, SSC, and processing system.

The first battery tested was the processing system's battery. This battery powers the motherboard and all sensors on the vehicle, see Chapter 3. It should be noted that the endurance of this battery is heavily dependent on the sensors and CPU utilization. As such, the overall endurance of this battery will decrease as the use of active sensors or CPU utilization increases. To assure that an operational threshold for this battery was known, several endurance experiments were performed using the best and worst cases scenarios.

The first experiment was designed to represent the best case scenario. This scenario included powering the motherboard and all external sensors and running only the basic operating system. Thus, sensors would be powered but not transmitting data, wireless communication would be idle, and the CPU usage would be regulated simply by the OS's overhead. This was performed by booting the processing system and utilizing its power supply to power all external sensors. Once the system was booted it was left in its idle state. The 11.1V 4.2Ah LiPo battery reached a critical low approximately 1.5 hours after being powered. This value was determined to be the absolute longest the processing system could be expected to perform using this battery.

The second experiment was designed to represent the worst case scenario. This scenario included powering all processing system hardware, requesting continuous updates from all sensors at their maximum throughput, attempting to utilize 100% of the CPU, and attempting to use 100% of the wireless bandwidth. This was accomplished utilizing several startup scripts that were automatically executed once the boot process was complete. These scripts requested continuous outputs from all sensors, setup a continuous broadcast of data to a remote machine, and initialized a simple process targeted at CPU utilization. CPU utilization during the experiments varied between 96% and 100%. The battery reached a critical low 45 minutes after being powered. This value was determined to be the absolute minimum that the processing system could be expected to perform using this battery.

Endurance testing for the safety switch battery was performed simply by powering the SSC with the 11.1V 0.5Ah LiPo. This was deemed sufficient as the SSC will continuously update its outputs regardless of the servo's or computer's state. As the SSC requires a minimum of 10V to operate, the battery was considered expelled when the voltage reached 10V. The battery reached 10V approximately 18 hours after being powered.

Table 10: Maximum Operation Times of Batteries

Battery	Maximum Endurance	USL Set Maximums
11.1V 4.2Ah LiPo	45-90 minutes	40 minutes
11.1V 0.5Ah LiPo	18 hours	10 hours
4.8V 2.0Ah NiMh	45 minutes	30 minutes
37V 10Ah LiPo	16-17 minutes	12 minutes

The servo battery, 4.8V NiMh, was tested by averaging the usage of multiple flights of the vehicle. This was done as the endurance of the battery is dependent on the motion of the servos. In an attempt to simulate the average servo movement the vehicle was flown by the safety pilot. Flights continued until the voltage dropped to 4.4V during servo movements. This voltage was considered the absolute lowest that should be reached while the servos were in motion. Voltage drop was determined by an LED battery indicator on the vehicle which glowed red when the servo battery reached 4.4V. Averaged flight times showed an approximate endurance of 45 minutes.

The last battery tested for endurance was the Joker Maxi 2's main battery. These experiments were performed by flying the testbed, fully equipped, until the 37V 10Ah LiPo reached either a critical low voltage, 33V, or became difficult for the pilot to fly. The average endurance was calculated to be between 16 and 17 minutes.

It should be noted that the endurance of individual batteries is a function of their age and usage. As such the endurance of the equipment they power is a function of the batteries age and usage. To account for this the maximum safe operation times for each battery was reduced. Table 10 details the maximum operation times for each battery as defined in the USL testbed operating protocol.

8.3 Shock & Vibration

Helicopters are naturally subject to high frequency vibrations. These vibrations are caused by the main and tail rotor blades as well as the vehicle's motor. Although RC vehicles are designed to cope with these vibrations the hardware added by USL was not specifically designed for that operating environment. To assure that the USL testbed could operate without serious failure several experiments were performed. These experiments were designed to test the

additional hardware's ability to withstand the rotorcrafts vibrations as well as possible shock due to "hard" landings.

Successful operation during nominal flight was tested by running all of the on-board hardware and software during multiple human controlled flights. The first set of flights was designed to determine if any failures would be experienced during normal flight. These flights consisted of the RC pilot performing non-aggressive maneuvers such as slow (<20degrees/sec) heading changes, point to point flight (< 5 ft/sec), and takeoff and landings. Normal flight produced zero hardware failures.

Next the testbed was subjected to aggressive flights. The first of these flights included obtaining speeds exceeding 35 ft/sec and angles greater than 45 degrees. The testbed was also subjected to heading rotations exceeding 180 degrees per second. These tests were performed to determine if the testbed could withstand the extreme aggressive maneuvers capable of this vehicle. Flight test showed zero hardware failures.

It should be noted that throughout the life of the testbed several extreme flight conditions have been inadvertently tested. These conditions include "hard landings" and violent flight maneuvers. All but one of these test were the result of human error. These "mishaps" were advantageous as it further shows the stability and operational abilities of the testbed. The first of these "mishaps" were two "hard landings".

For the purpose of this work hard landings are defined as landings that physically caused damage to the vehicle. The first of these hard landings was due to the safety pilot inadvertently toggling the motor kill switch. This caused the vehicle's motor to shutdown approximately 100 ft above ground level. Although an autorotation was attempted the main rotor had significantly slowed before the safety pilot was aware the motor had shutdown. The safety pilot was able to orient the vehicle correctly before landing but the shear impact significantly distorted the chassis, broke one weld, and cracked a second weld. Although the shock caused structural damage to the chassis the testbed hardware was not damaged. Repairing the vehicle to optimal performance only required that the chassis be reshaped and welded. Note that this was an early phase of development and that the laser range finder had not yet been mounted to the system.

The second hard landing experienced by the USL testbed was caused by a poorly soldered connector on the main vehicle battery. This solder broke during flight causing the motor to shutdown approximately 100 ft above ground level. This hard landing also caused distortion and cracked welds on the chassis. The chassis was again repaired restoring the testbed to operating condition. It was noted that intermittent failures of the laser were experienced after this

particular hard landing. These failures were difficult to reproduce and were only periodically experienced during takeoff or landing. This laser was eventually replaced.

Multiple "mishaps" have occurred during the software development phase of the vehicle design due to software bugs. Although numerous bugs were created during this phase of development several caused violent flight maneuvers. These are described here only to show that testbed is capable of surviving violent and out of the ordinary conditions. The two main maneuvers of interest were caused by bugs which increased controller outputs by ten fold. As the controller outputs are hard limited this condition caused the testbed to only output the two most extreme output values. This type of bug was experienced by both the pitch and yaw controllers on two separate occasions. This output caused the vehicle to violently flail about on the miss controlled axis. Although control was removed within a few seconds of the failure, this provided a level of confidence about the testbed's ability to operate under the most extreme flight controls of the vehicle.

8.4 Heat

Due to the desire to protect the processing system from environmental hazards as well as electronic interference the enclosure is almost completely sealed. Sealing the enclosure raised issues about the processing system's ability to function without proper heat ventilation. As the enclosure is exclusively made from wood the heat dissipation from within the enclosure is very minimal. To assure that processing system could function without proper ventilation several experiments were performed.

Testing for heat related failures was performed by operating the processing system for long periods of time. These experiments were performed in a lab environment to remove the cooling effect caused by the downwash of the main rotor. These experiments varied from only a few hours up to three days. Even after three days of continuous operation the processing system operated without failure. Internal temperatures during experimentation reached 168 degrees for the system and 150 degrees for the CPU.

8.5 CPU Utilization

As this work describes a helicopter testbed capable of multiple areas of development, the processing system must have free clock cycles available during operation. To determine the amount of CPU available during flight the testbed was initialized and operated as for normal flight. CPU utilization during flight varied between zero and two percent and averaged approximately 0.8 percent. It should be noted that initial controller development was designed using Mamdani fuzzy logic. This method utilized up to 40% of the processor during operation. The controllers were then ported to use Sugeno fuzzy logic which significantly reduced the CPU utilization.

8.6 Controller Validation

To validate the controllers developed for the USL testbed multiple experiments were performed. These experiments included multiple flights using multiple flight patterns under varying environmental conditions. Experiments were performed in both simulation and on the actual hardware in an outdoor environment.

8.6.1 Simulation Experiments

Simulation experiments were performed utilizing the X-Plane simulator and Matlab 2006a, see Chapter 7. Experiments were performed with and without wind. These experiments included hovering and waypoint navigation. Two forms of waypoint navigation were performed including navigation that required the vehicle stop and stabilize at each individual waypoint and navigation that did not allow the vehicle to stabilize at any waypoint. The overall flight envelope tested in simulation is available in Table 11.

Table 11: Flight Envelope Tested in Simulation

Variable	Range
Flight Speed	0-10 mph
Wind	0-23 mph
Wind shear	0-20 degrees
Wind gusts	0-23 mph*

*Total wind speed did not exceed 23 mph

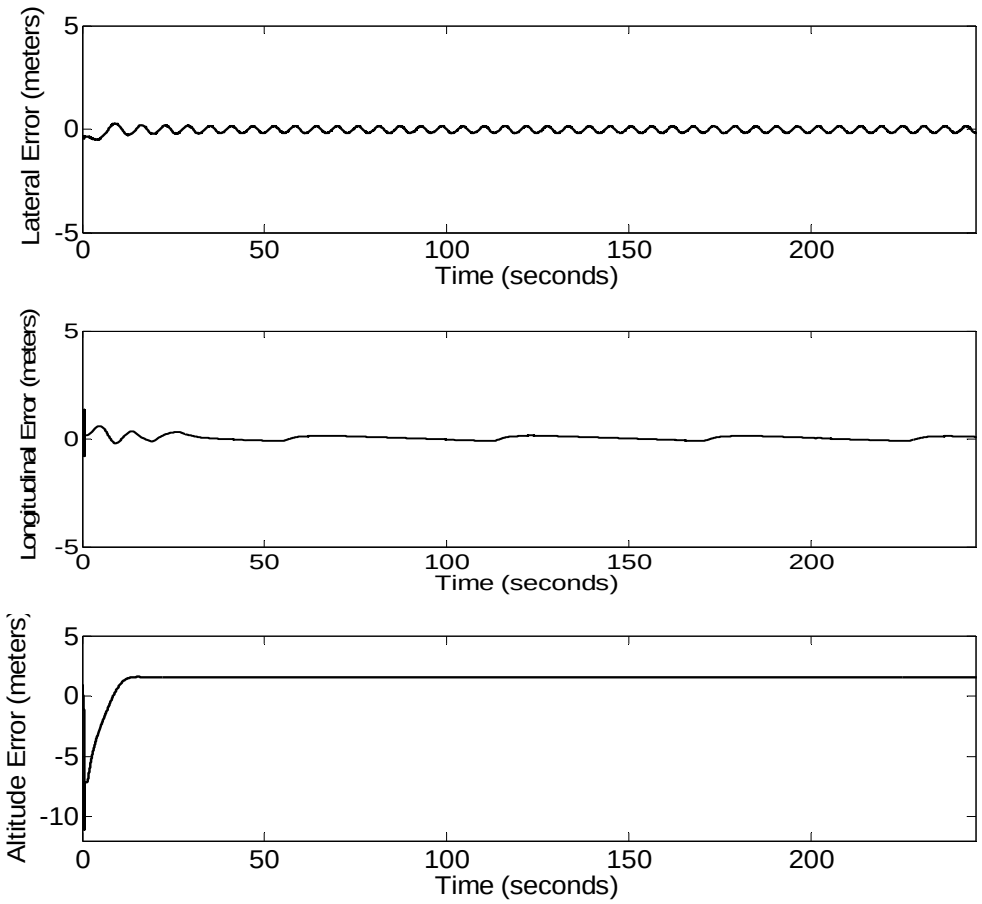

Figure 54: Positional Error for Zero Wind Hovering

Hovering test varied in length of time from several minutes to several hours. These tests were first performed using zero wind. Figure 54 depicts the positional error over time in the lateral, longitudinal, and vertical directions for a single experiment. Note that the vehicle was commanded to a hovering position while landing. This was done by requesting a hover at a predefined point directly above the vehicle and then relinquishing control to the fuzzy controllers. This caused the vehicle to immediately begin increasing in altitude while attempting to hold its position. This is apparent from the altitude error graph in Figure 54.

To assure that vehicle could sustain a hover under varying wind conditions a constant wind with constant direction was added to the simulation. These tests varied in length of time from several minutes to several hours. Figure 55 depicts the positional error over time in the lateral, longitudinal, and vertical directions for a single experiment. The constant velocity utilized during these experiments was 5 kt. These experiments also commanded the vehicle to a predetermined position from a landing.

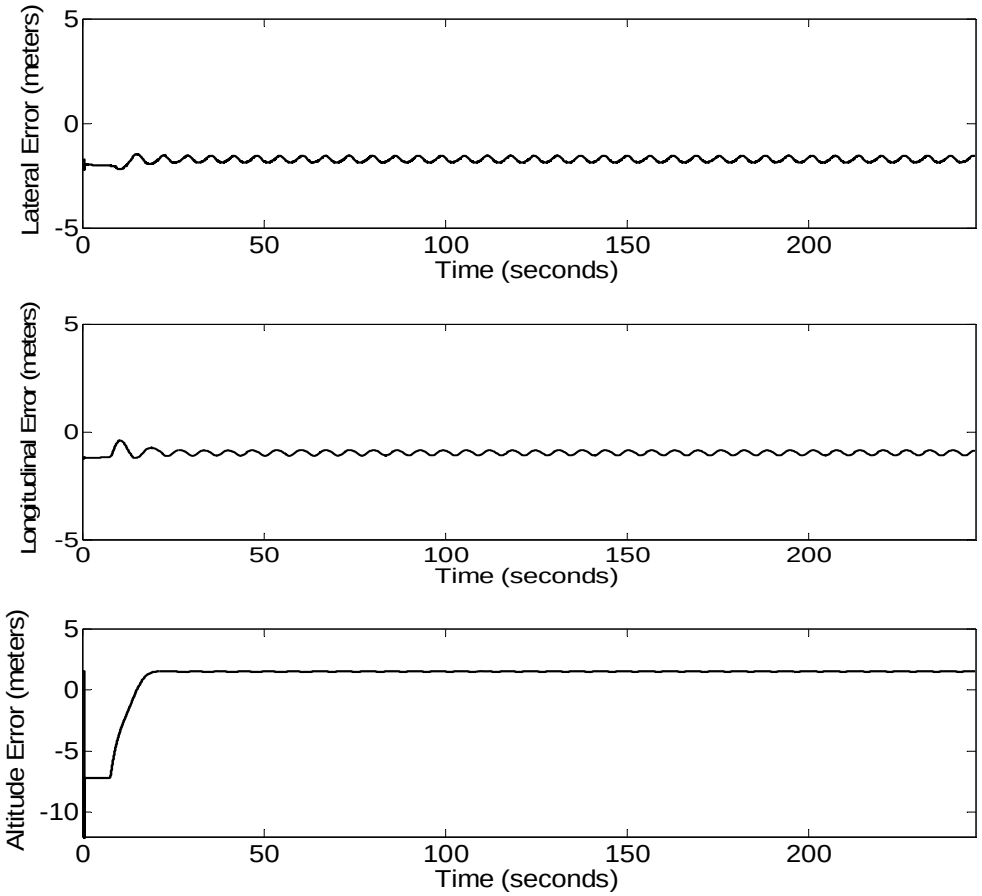

Figure 55: Positional Error for Constant Wind Hovering

Although the previous two sets of experiments provided some level of confidence in the controllers abilities to hover the vehicle they were not realistic for outdoor environments. Outdoor environments typically contain wind gust of varying direction and magnitude. To assure that vehicle could safely operate in an outdoor environment experiments were performed using wind with dynamically changing speeds and direction. Figure 56 depicts the positional error over time in the lateral, longitudinal, and vertical directions for a single experiment. The maximum wind speed obtained during these simulations was 5kt with a wind shear ranging from zero to 20 degrees.

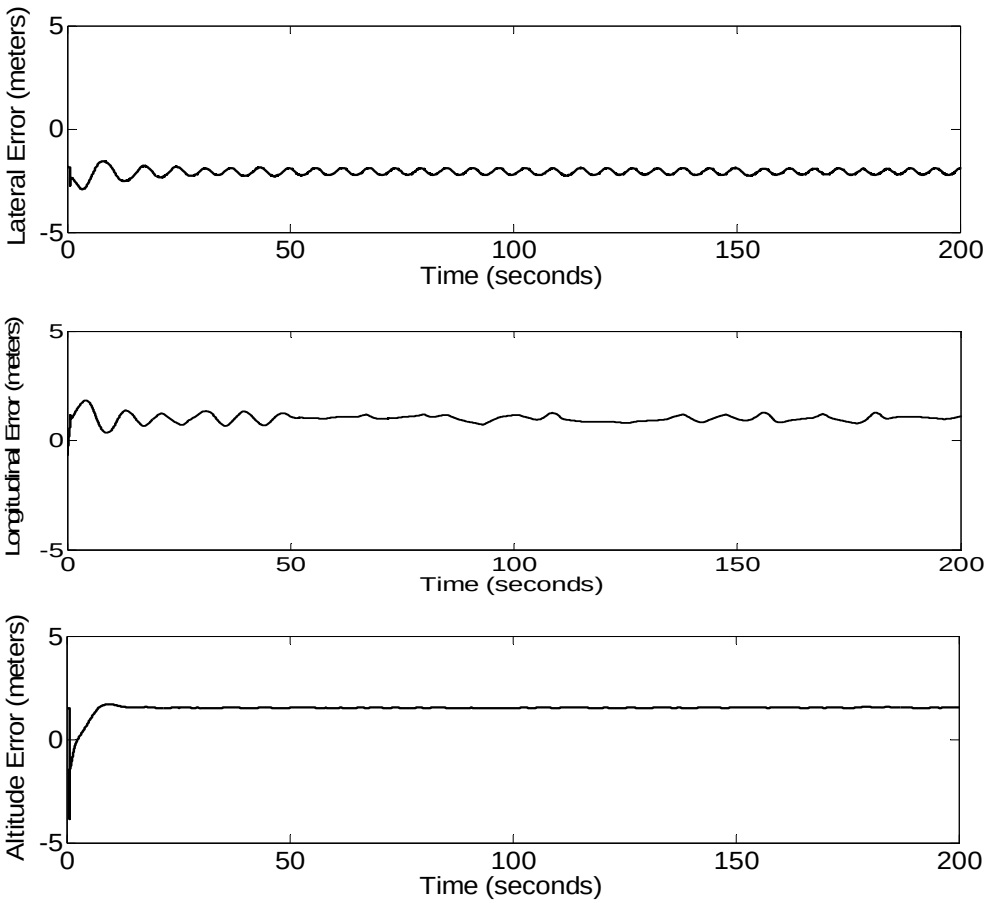

Figure 56: Positional Error for Dynamic Wind Hovering

The first sets of flight paths performed were designed to maneuver the vehicle to several waypoints stopping at each for two seconds. Like hover, these experiments were performed without wind, with constant wind, and with dynamic wind. Figures 57-59 detail the flight paths of several experiments using each of the three types of wind effects. Make note that the desired flight paths for all experiments performed in this section are rectangular. Also note that at each waypoint the vehicle was command to rotate in the direction of the new flight path. Thus, the vehicle is always moving in the forward direction.

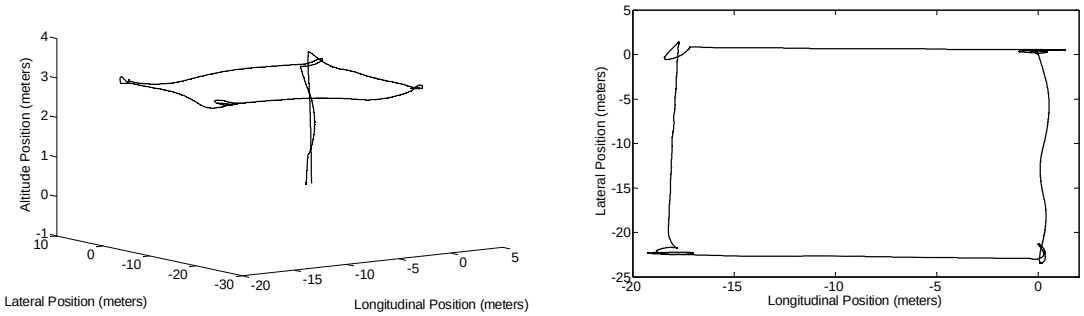

Figure 57: 3D (left) and Birds-Eye-View (right) of a Square Flight Path with No Wind Effects and Stopping at Each Waypoint

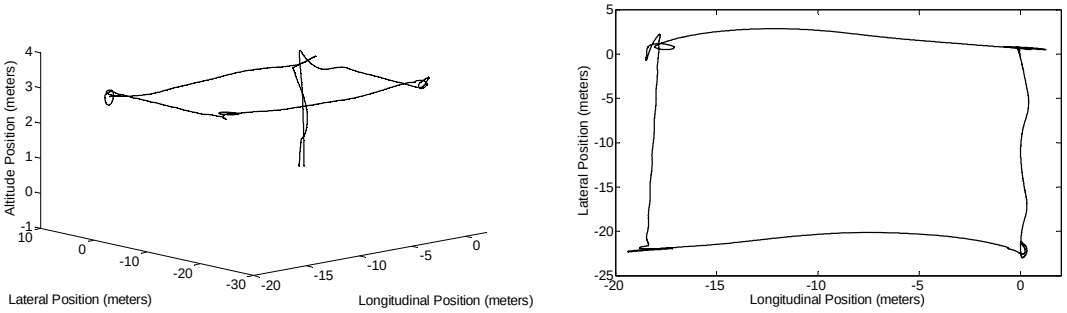

Figure 58: 3D (left) and Birds-Eye-View (right) of a Square Flight Path with a 5kt Wind and Stopping at Each Waypoint

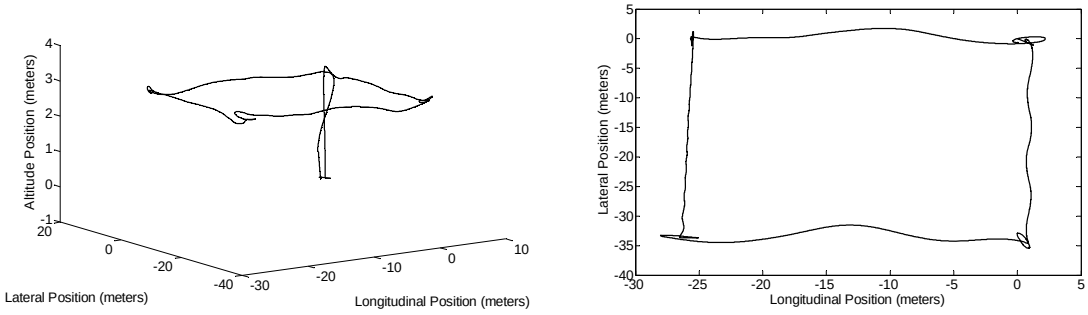

Figure 59: 3D (left) and Birds-Eye-View (right) of a Square Flight Path with a 5kt Wind, 20 Degree Wind Shear, and Stopping at Each Waypoint

The last set of flight paths performed were designed to maneuver the vehicle to several waypoints without allowing the vehicle to stabilize between waypoints. These experiments were designed to discover the controller's reactions to more dynamic flights. Like the previous simulation experiments, these experiments were performed without wind, with constant wind, and with dynamic wind. Figures 60-62 detail the flight paths of several experiments using each of the three types of wind effects.

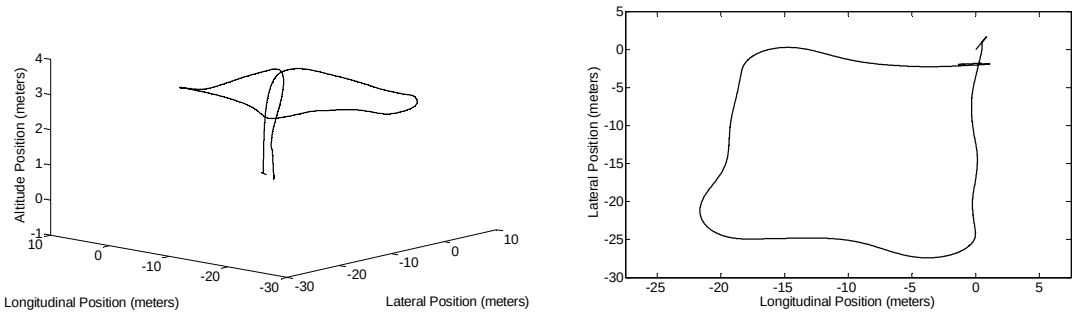

Figure 60: 3D (left) and Birds-Eye-View (right) of a Square Flight Path without Stopping at Waypoints

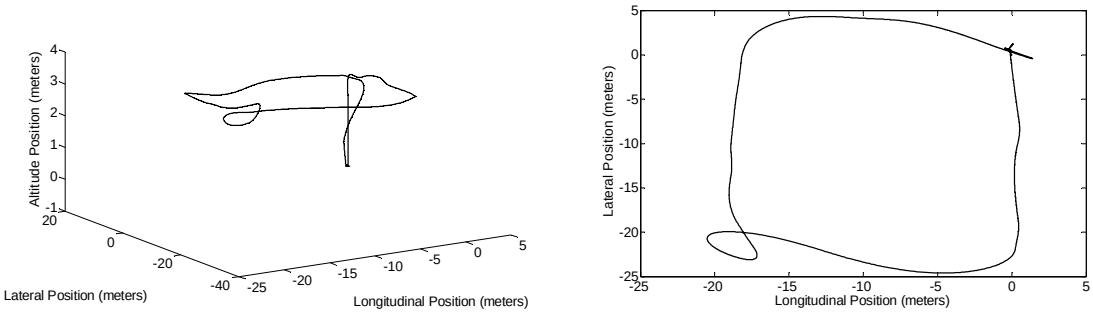

Figure 61: 3D (left) and Birds-Eye-View (right) of a Square Flight Path with a 5kt Wind

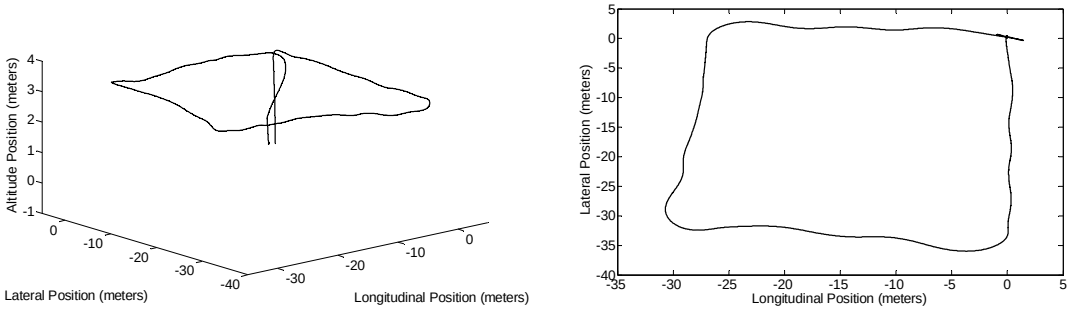

Figure 62: 3D (left) and Birds-Eye-View (right) of a Square Flight Path with a 5kt Wind and 20 Degree Wind Shear

8.6.2 Field Experiments

Although simulation experiments provide a level of assurance in the ability of the controllers they are far from conclusive. The controllers were designed to operate on a helicopter testbed and therefore must be validated on the actual hardware in realistic environments.

The USL helicopter was tested in a non-optimal outdoor environment on the University of South Florida (USF) campus. This test area is approximately 70 meters wide and 100 meters long. The test area is surrounded by multiple buildings including a four story parking garage to the north west, a three story office building to the north, and a four story office building to the south. This environment created varying wind effects and less than optimal GPS reception. Figure 63 visualizes the testing environment.

Field experiments included waypoint navigation, hovering, takeoff, and landing. These experiments were first performed individually and then as an integrated set. The overall flight envelope tested in the field is available in Table 12.

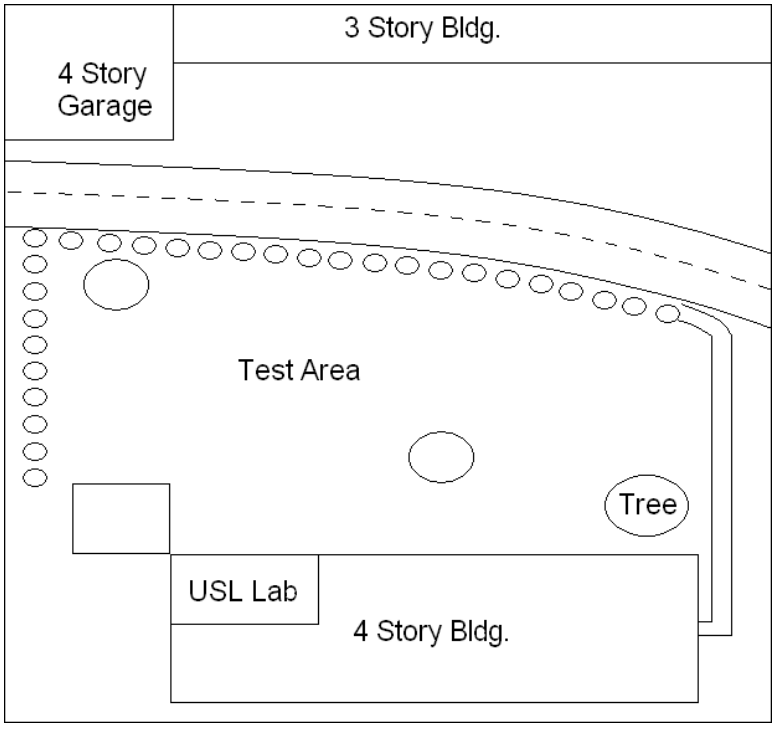

Figure 63: USL Test Area

Table 12: Flight Envelope Tested in the Field

Variable	Range
Flight Speed	0-5 mph
Time	Day & Night
Wind	0-15 mph
Wind shear	0-5 degrees
Wind gusts	0-5 mph
Cloud Cover	Clear-Overcast
Rain	Clear-Misting

Hovering was naturally the first outdoor experiment to be performed on the testbed. These experiments were performed by having the safety pilot takeoff the vehicle and position it in a hover approximately 50 ft off of the ground. Once the vehicle was stable the safety pilot relinquished control to the testbed. The testbed was then responsible for holding the position where it was located when control was relinquished. Figure 64 details the positional error over time that occurs during a continuous hover in a very light wind with very moderate gusts. Note that the first 35 to 40 seconds of flight contained a strictly positive longitudinal error. This error was partially due to the longitudinal wind and vehicle setup. Note that this strictly positive error was consistently reduced and eventually removed from the vehicle. This is a product of the trim adjusters compensating for the lack of progress towards the goal. It should be mentioned that position data used to plot all of the figures in this section were gathered directly from the GPS unit. As such, all of gathered position data are subject to the errors and noise associated with GPS. To compensate, various videos of autonomous flight are available with the original dissertation, on file with the USF library in Tampa, Florida, or available at www.uavrobotics.com.

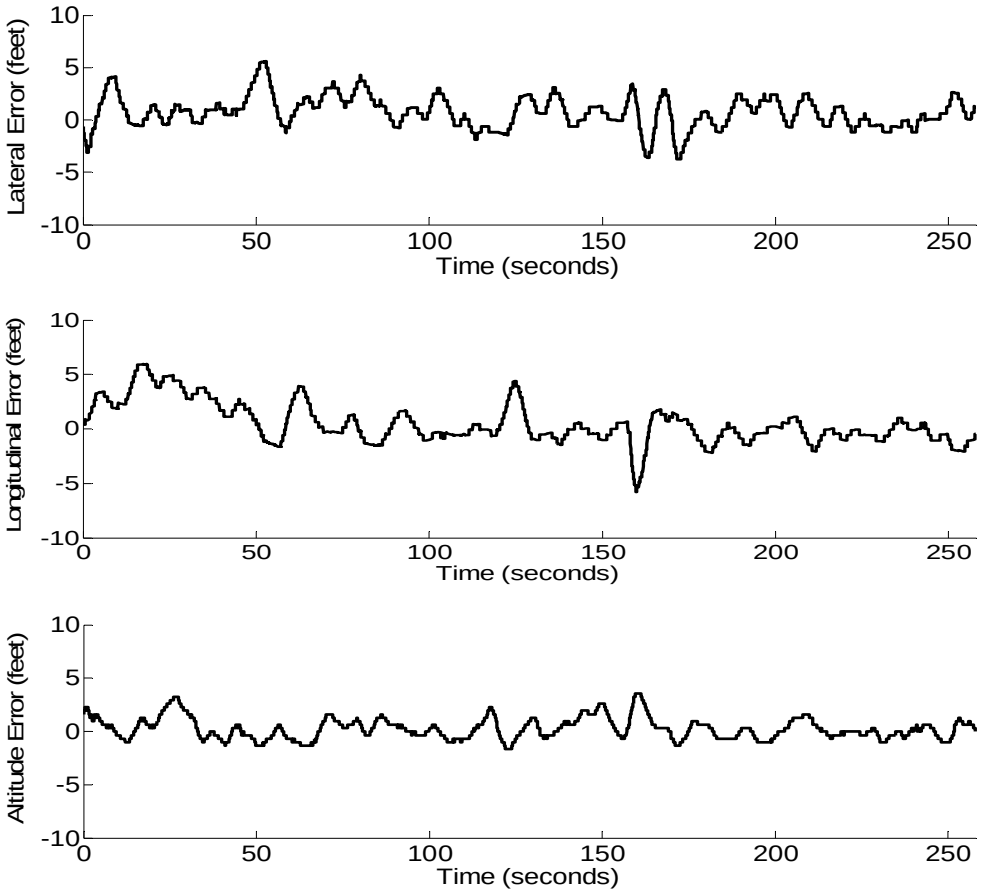

Figure 64: Positional Error During Hover

Once hovering had been sufficiently tested experiments were performed to validate the controller's abilities to maneuver between points. These experiments consisted of the safety pilot taking off the vehicle and positioning it in a hover approximately 50 ft above the ground. The safety pilot would then relinquish control and the vehicle would begin transitioning hard coded waypoints. Once the vehicle had transitioned the entire set of waypoints it would maintain a hover at the last waypoint. Figure 65 details a typical path followed by the testbed while performing a square pattern of four waypoints.

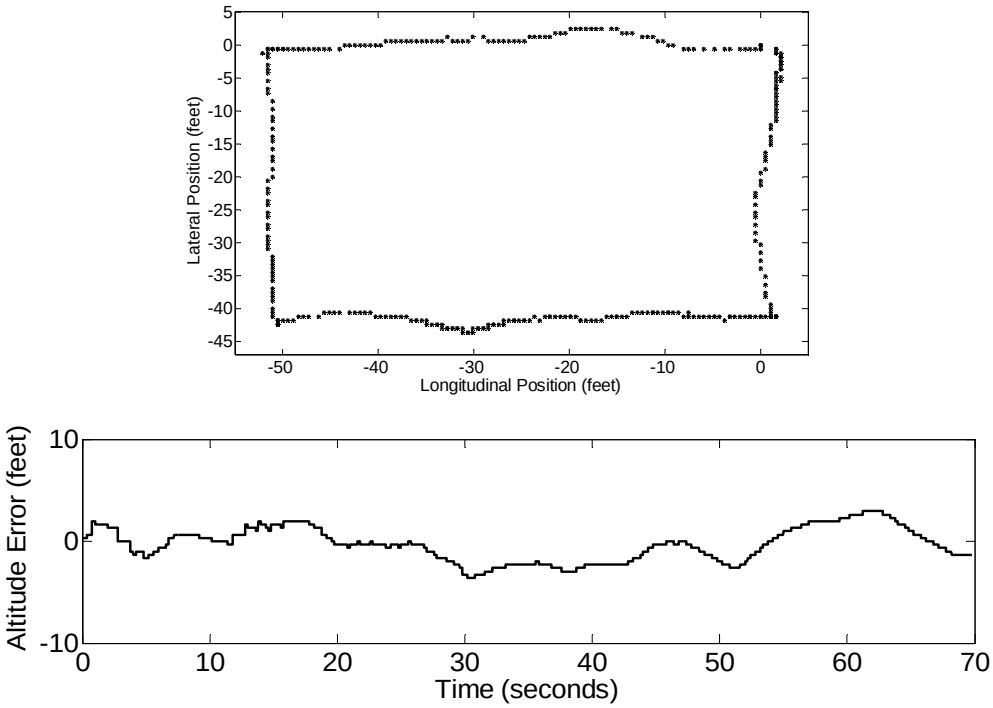

Figure 65: Birds-Eye-View of Square Flight Path (top) and Altitude Error (bottom)

Once the vehicle had shown a consistent ability to successfully perform flight maneuvers, experiments were performed to validate the controller's ability to takeoff and land successfully. These experiments required the safety pilot to relinquish control before the vehicle was powered up. Once control was relinquished the vehicle would power up the main rotor and lift off. After a predetermined altitude had been reached the controller began the landing phase of the experiment. Once the vehicle had successfully landed the main rotor was powered down. Figure 66 details the vertical error as well as the lateral and longitudinal positions during one of these experiments. Note that lateral and longitudinal errors during takeoff are due to the extreme shifts in both the direction and magnitude of accelerometer drift. These errors will eventually be removed by the drift calculation described in Section 5.4.

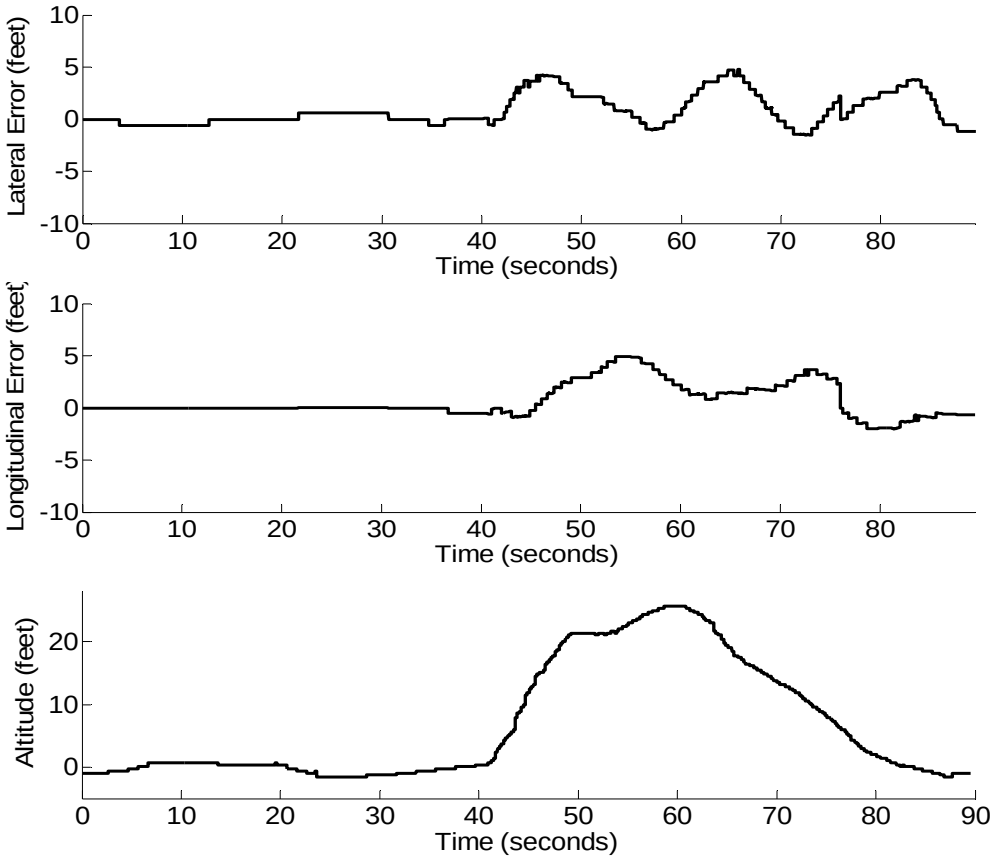

Figure 66: Lateral Error (top), Longitudinal Error (middle), and Altitude (bottom) for a Typical Takeoff and Landing Experiment

The last step in testing the controllers was to perform flights that were integrated sets of all the abilities programmed into the testbed. This included takeoff, landing, hovering, and navigation. These experiments included several flight patterns including the square-s, straight line, and vertical steps. Note that vertical steps are transitions in the longitudinal and vertical axes. Figures 67-69 detail the flight paths of the testbed for all three flight patterns.

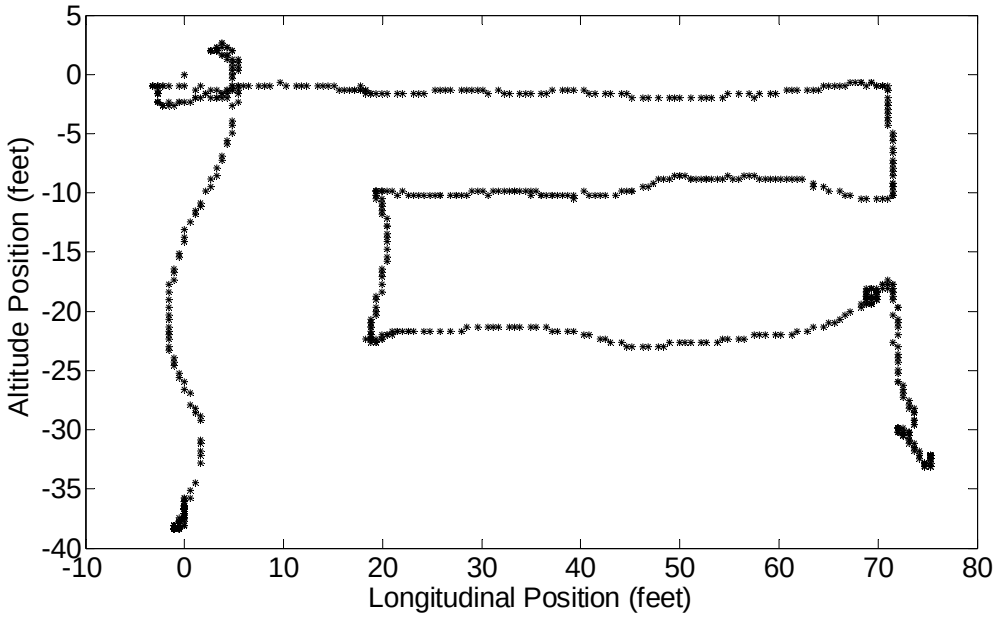

Figure 67: Vertical Steps Flight Path

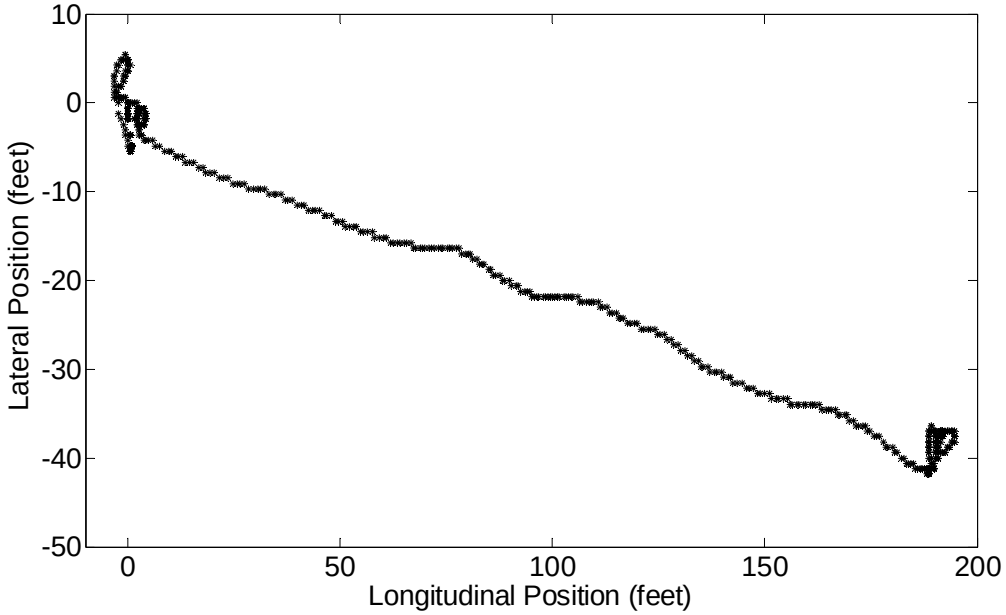

Figure 68: Straight Line Flight Path

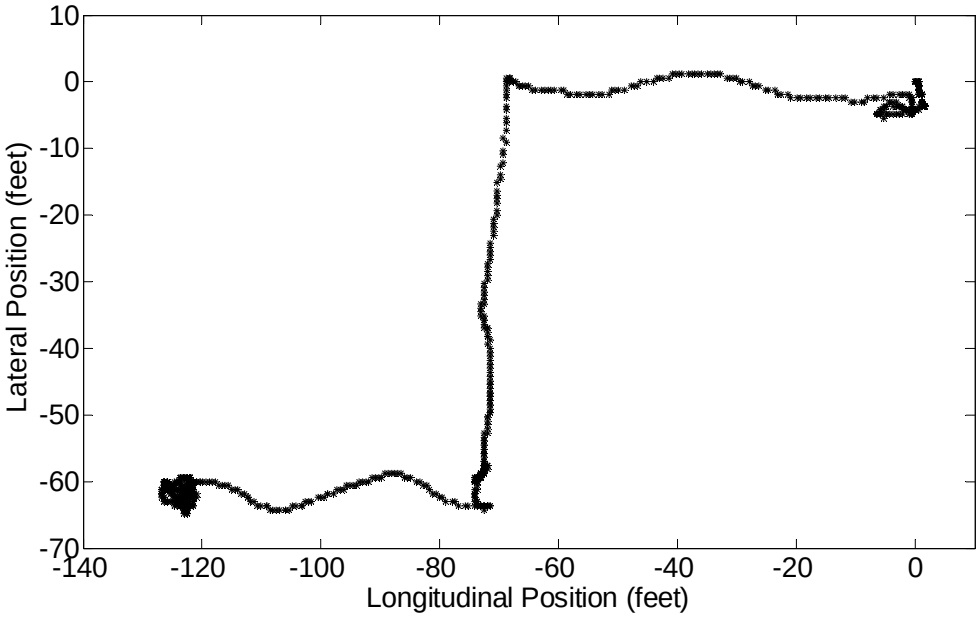

Figure 69: Square-S Flight Path

Chapter 9
Conclusions & Future Work

9.1 Conclusion

This research presented the full design and implementation of an unmanned helicopter testbed designed by USL. This vehicle was specifically designed to support removal, modification, and replacement of both hardware and software as deemed necessary by the end user. This is made possible through the design and implementation of modular hardware and software as well as the use of COTS components. This vehicle has performed over two hundred fully autonomous flights and represents one of only a handful of fully autonomous helicopters in the world. It is distinct due to its ground up design as a testbed vehicle and furthers the UAV area of research by fully disclosing the design and implementation of the system. It is this researcher's hope that this work will provide the opportunity and motivation for others to join and help further this technology.

9.2 Future Work

The USL helicopter testbed has shown the ability to autonomously navigate waypoints, hover, takeoff, and land as well as the ability to filter and fuse data without relying on a vehicle specific model. The possible future work that can be accomplished from this point is almost limitless. The vehicle is designed for modification and testing and as such lends itself to a multitude of areas. This includes vision processing, controller design, software architecture design, hardware design, filtering and fusion, and mechanical design to name only a few.

Foreseeable work in the immediate future would most likely include controller updates that attempt to separate uncontrollable forces into external and internal categories and correctly compensate for these forces. Controllers could also be updated or developed to utilize state feedback to permanently modify the controller's outputs. One specific feedback update would be

to allow the acceleration variant to permanently update the desired angle calculated by the controller.

Another foreseeable update includes mounting and testing the chassis and processing system on other platforms. This could include variations of types and sizes of helicopters as well as other types of unmanned vehicles including fixed wing aircraft and ground vehicles. Note that ground vehicle usage has been tested to a limited degree in [97] and [98] where the processing system, sensors, and software described in this work were ported to a four wheel drive RC truck. This UGV has been utilized in UAV/UGV coordination, swarm control, and pattern formation.

Vision, although only briefly mention in this work, could also provide a large amount of information about the state of the vehicle as well as provide the vehicle the ability to accomplish many more tasks. Types of information could include state data regarding the velocity, relative position to an object, or even failure detection for the vehicle. Vision processing algorithms could also provide the vehicle the ability to identify and track objects, perform statistical calculations, and be readily fault tolerant.

Although it is not detailed in this work the use of a Graphical User Interface (GUI) would greatly improve the appeal of this testbed. This interface could easily be designed to allow users to dynamically change flight paths and gather information in a more user friendly environment.

References

1. Unknown. *Helicopter Development in the Early Twentieth Century*. [cited 2007 September 18]; Available from: http://www.centennialofflight.gov/essay/Rotary/early_20th_century/HE2.htm.
2. Unknown. *Rotary Wing History*. 2007 [cited 2007 September 18]; Available from: http://helicopter-history.org/.
3. Fay, J. *Helicopter Pioneers - Evolution of Rotary Wing Aircraft*. 1997 [cited 2007 September 18]; Available from: http://www.helis.com/pioneers/1.php.
4. McKee, M.W., *VTOL UAVs Come of Age: US Navy Begins Development of VTUAV* in *Vertiflite*. 2000, American Helicopter Society.
5. Shim, D.H., H.J. Kim, and S. Sastry, *A Flight Control System for Aerial Robots: Algorithms and Experiments.* IFAC Control Engineering Practice, 2003: p. 1389–1400.
6. Wills, L., et al., *An Open Platform for Reconfigurable Control.* Control Systems Magazine, IEEE, 2001. 21(3): p. 49-64.
7. Rojas, I., et al., *On-line Adaptive Fuzzy Controller: Application of Helicopter Stabilization of the Altitude of a Helicopter.* Computational Intelligence for Measurement Systems and Applications, 2003. Proceedings. CIMSA '03. IEEE International Symposium on, 2003: p. 119-123.
8. Kanade, T., B. Mettler, and M.B. Tischler, *System Identification of Small-size Unmanned Helicopter Dynamics.* American Helicopter Society 55th Forum, 1999.
9. Abbeel, P., et al., *An Application of Reinforcement Learning to Aerobatic Helicopter Flight*, in *Neural Information Processing Systems (NIPS)*. 2007: Vancouver, B.C., Canada.
10. Day, D.A. *Introduction to Rotary-Wing Flight*. [cited 2007 September 18]; Available from: http://www.centennialofflight.gov/essay/Rotary/HE-OV.htm.
11. Altuğ, E., *Vision Based Control of Unmanned Aerial Vehicles with Applications to an Autonomous Four Rotor Helicopter, Quadrotor*, in *Department of Mechanical Engineering and Applied Mechanics*. 2003, University of Pennsylvania.
12. Schrage, D.P., et al., *Instrumentation of the Yamaha R-50/RMAX Helicopter Testbeds for Airloads Identification and Follow-on Research.* European Rotorcraft Forum, 25th, 1999.
13. Garcia, R.D., *A Modular Onboard Processing System for Small Unmanned Vehicles* in *Department of Computer Science and Engineering*. 2006, University of South Florida: Tampa. p. 53.
14. Richardson, T., *Micro UAVs*. 2007, University of Bristol: Birmingham, UK.
15. Raicu, I., *MicroElectroMechanical Systems: MEMS Technology Overview and Limitations*. 2004, Wayne State University: Detroit.
16. Unknown, *TUAC/EI Statement to the 2006 OECD meeting of ministers of education*, in *Higher Education: Quality, Equity, and Efficiency*. 2006: Athens.
17. Quigley, M., M.A. Goodrich, and R.W. Beard, *Semi-autonomous Human-UAV Interfaces for Fixed-wing Mini-UAVs.* Intelligent Robots and Systems, 2004. Proceedings of the 2004 IEEE/RSJ International Conference on.

18. Ryan, A. and J.K. Hedrick, *A Mode-switching Path Planner for UAV-assisted Search and Rescue.* Decision and Control, 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on, 2005: p. 1471-1476.
19. Ryan, A., et al., *An Overview of Emerging Results in Cooperative UAV Control.* Decision and Control, 2004 European Control Conference. CDC-ECC'04. 43th IEEE Conference on, 2004.
20. Hrabar, S., et al., *Combined Optic-flow and Stereo-based Navigation of Urban Canyons for a UAV.* Submitted to IEEE/RSJ International Conference on Intelligent Robots and Systems, 2005.
21. Loyall, J., et al., *Model-Based Design of End-to-End Quality of Service in a Multi-UAV Surveillance and Target Tracking Application.* 2 ndRTAS Workshop on Model-Driven Embedded Systems (MoDES), Toronto, Canada, May, 2004.
22. Quigley, M., et al., *Target Acquisition, Localization, and Surveillance Using a Fixed-Wing Mini-UAV and Gimbaled Camera.* Robotics and Automation, 2005. Proceedings of the 2005 IEEE International Conference on, 2005: p. 2600-2605.
23. Freed, M., R. Harris, and M. Shafto, *Human-interaction Challenges in UAV-based Autonomous Surveillance.* Proceedings of the 2004 Spring Symposium on Interactions Between Humans and Autonomous Systems Over Extended Operations, 2004.
24. Freed, M., R. Harris, and M.G. Shafto, *Human vs. Autonomous Control of UAV Surveillance.* AIAA 1 st Intelligent Systems Technical Conference, 2004: p. 1-7.
25. Nygaards, J., et al., *Navigation Aided Image Processing in UAV Surveillance: Preliminary Results and Design of an Airborne Experimental System.* Journal of Robotic Systems, 2004. 21(2): p. 63-72.
26. Kingston, D., R. Beard, and D. Casbeer, *Decentralized Perimeter Surveillance Using a Team of UAVs.* Proceedings of the AIAA Conference on Guidance, Navigation, and Control.
27. Dovis, F., et al., *HeliNet: A Traffic Monitoring Cost-effective Solution Integratedwith the UMTS System.* Vehicular Technology Conference Proceedings, 2000. VTC 2000-Spring Tokyo. 2000 IEEE 51st, 2000.
28. Coifman, B., et al., *Surface Transportation Surveillance from Unmanned Aerial Vehicles.* Procs. of the 83rd Annual Meeting of the Transportation Research Board, 2004: p. 11-20.
29. Srinivasan, S., et al., *Airborne Traffic Surveillance Systems: Video Surveillance of Highway Traffic.* Proceedings of the ACM 2nd international workshop on Video surveillance & sensor networks, 2004: p. 131-135.
30. Coifman, B., et al., *Roadway Traffic Monitoring from an Unmanned Aerial Vehicle.* IEE Proc. Intell. Transp. Syst. Vol, 2006. 153(1): p. 11.
31. Casbeer, D.W., et al., *Cooperative Forest Fire Surveillance using a Team of Small Unmanned Air Vehicles.* International Journal of Systems Science, 2006. 37(6): p. 351-360.
32. Rufino, G. and A. Moccia, *Integrated VIS-NIR Hyperspectral/Thermal-IR Electro-Optical Payload System for a Mini-UAV.* Infotech@ Aerospace, 2005: p. 1-9.
33. Martínez-de Dios, J.R., L. Merino, and A. Ollero, *Fire Detection Using Autonomous Aerial Vehicles with Infrared and Visual Cameras.* Proceedings of the 16th IFAC World Congress., 2005.
34. Ollero, A., et al., *Motion Compensation and Object Detection for Autonomous Helicopter Visual Navigation in the COMETS System.* Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on, 2004. 1: p. 19-24 Vol.1.

35. Hausamann, D., et al., *Monitoring of Gas Pipelines–A Civil UAV Application.* Aircraft Engineering and Aerospace Technology: An International Journal, 2005. 77(5): p. 352-360.
36. Ostendorp, M., *Innovative Airborne Inventory and Inspection Technology for Electric Power Line Condition Assessments and Defect Reporting.* Transmission and Distribution Construction, Operation and Live-Line Maintenance Proceedings. 2000 IEEE ESMO-2000 IEEE 9th International Conference on, 2000: p. 123-128.
37. Morris, J., *DHS Border Patrol Flights with Hunter UAV Under Way.* Business Aviation, 2004: p. 33.
38. Girard, A.R., A.S. Howell, and J.K. Hedrick, *Border Patrol and Surveillance Missions using Multiple Unmanned Air Vehicles.* Decision and Control, 2004. CDC. 43rd IEEE Conference on, 2004.
39. Garcia, R.D., K.P. Valavanis, and A. Kandel, *Fuzzy Logic Based Autonomous Unmanned Helicopter Navigation With A Tail Rotor Failure.* CR-ROM Proceedings, 15th Mediterranean Conference on Control and Automation, Athens, Greece, June 2007.
40. Gilmore, J.M., *CBO Testimony: The Army's Future Combat Systems Program.* 2006.
41. Unknown. *UAV Helicopter Controller*. 2007 [cited 2007 September 20]; Available from: http://rotomotion.com/prd_UAV_CTLR.html.
42. Wyatt, D., ed. *Eagle Eye Pocket Guide*. 2005, Bell Helicopter Textron Inc. 51.
43. Goebel, G., *Unmanned Aerial Vehicles*. 2006.
44. Unknown, *Sikorsky Cypher II - Dragon Warrior*. 2005.
45. Freeland, R., *MQ-8B Fire Scout: Facts Sheet*. 2007, Northrop Grumman Corporation: Los Angeles, CA.
46. Unknown. *What is Vigilante*. 2007 [cited 2007 September 20]; Available from: http://www.saic.com/products/aviation/vigilante/vigilante.pdf.
47. Unknown. *CL-227 / CL-327*. 2000 January 8, 2000 [cited 2007 September 20]; Available from: http://www.fas.org/man/dod-101/sys/ac/row/cl-327.htm.
48. Parsch, A. *Boeing A160 Hummingbird*. 2007 July 1, 2007 [cited 2007 September 20]; Available from: http://www.designation-systems.net/dusrm/app4/hummingbird.html.
49. Unknown. *Boeing: Integrated Defense Systems - A160 Hummingbird*. [cited 2007 September 20]; Available from: http://www.boeing.com/ids/advanced_systems/hummingbird.html.
50. Unknown. *SR200 VTOL UAV*. 2005 [cited 2007 September 23]; Datasheet for the SR200 VTOL]. Available from: http://rotomotion.com/datasheets/sr200_uav_sheet.pdf.
51. Unknown. *SR100 VTOL UAV*. 2005 [cited 2007 September 23]; Datasheet for the SR100 VTOL]. Available from: http://rotomotion.com/datasheets/sr100_uav_sheet.pdf.
52. Unknown. *SR20 VTOL UAV*. 2005 [cited 2007 September 23]; Datasheet for the SR20 VTOL]. Available from: http://rotomotion.com/datasheets/SR20_uav_sheet.pdf.
53. Crane, D. *Unmanned Mini-Helicopter Gets 'Weaponized' with AA-12 Shotgun*. 2007 September 20, 2007 [cited; Available from: http://www.defensereview.com/article846.html.
54. Unknown. *About Neural Robotics*. 2007 [cited 2007 September 20]; Available from: http://www.neural-robotics.com/About%20NRI/About.html.
55. Abbeel, P., V. Ganapathi, and A.Y. Ng, *Learning Vehicular Dynamics, with Application to Modeling Helicopters.* Neural Information Processing Systems (NIPS), 2006.
56. Buskey, G., G. Wyeth, and J. Roberts, *Autonomous Helicopter Hover using an Artificial Neural Network.* Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on, 2001.

57. Corke, P., et al., *Autonomous Deployment and Repair of a Sensor Network using an Unmanned Aerial Vehicle.* Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on, 2004. 4: p. 3602- 3608.
58. Dittrich, J.S. and E.N. Johnson, *Multi-sensor Navigation System for an Autonomous Helicopter.* Digital Avionics Systems Conference, 2002. Proceedings. The 21st, 2002. 2: p. 8C1-1-8C1-19 vol.2.
59. Sinopoli, B., et al., *Vision Based Navigation for an Unmanned Aerial Vehicle.* IEEE International Conference on Robotics and Automation, 2001.
60. Driankov, D. and A. Saffiotti, *Fuzzy Logic Techniques for Autonomous Vehicle Navigation*. 2001: Physica Verlag.
61. Miller, R., O. Amidi, and M. Delouis, *Arctic Test Flights of the CMU Autonomous Helicopter.* Proceedings of the Association for Unmanned Vehicle Systems International 1999, 26th Annual Symposium, 1999.
62. Saripalli, S., J.F. Montgomery, and G.S. Sukhatme, *Visually Guided Landing of an Unmanned Aerial Vehicle.* IEEE Transactions on Robotics and Automation, 2003. 19(3): p. 371-380.
63. Saripalli, S. and G.S. Sukhatme, *Landing on a Moving Target using an Autonomous Helicopter.* Proceedings of the International Conference on Field and Service Robotics, 2003.
64. Garcia-Pardo, P.J., G.S. Sukhatme, and J.F. Montgomery, *Towards vision-based safe landing for an autonomous helicopter.* Robotics and Autonomous Systems, 2002. 38(1): p. 19-29.
65. Mejias, L., et al., *Detection and Tracking of External Features in an Urban Environment using an Autonomous Helicopter.* Robotics and Automation, 2005. Proceedings of the 2005 IEEE International Conference on, 2005: p. 3972-3977.
66. Nordberg, K., et al., *Vision for a UAV Helicopter.* Proceedings of IROS, 2002. 2.
67. Hrabar, S. and G.S. Sukhatme, *Omnidirectional Vision for an Autonomous Helicopter.* Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on, 2003.
68. Ng, A.Y., et al., *Autonomous Inverted Helicopter Flight Via Reinforcement Learning.* International Symposium on Experimental Robotics, 2004.
69. Piedmonte, M. and E. Feron, *Aggressive Maneuvering of Autonomous Aerial Vehicles: A Human-Centered Approach.* International Symposium on Robotics Research, 1999.
70. Gavrilets, V., et al., *Control Logic for Automated Aerobatic Flight of Miniature Helicopter.* AIAA Guidance, Navigation and Control Conference, 2002.
71. Ng, A.Y., et al., *Autonomous Helicopter Flight via Reinforcement Learning.* Advances in Neural Information Processing Systems, 2004.
72. Johnson, E.N. and D.P. Schrage, *The Georgia Tech Unmanned Aerial Research Vehicle: GTMax.* Proceedings of the AIAA Guidance, Navigation, and Control Conference, 2003.
73. Johnson, E.N. and S. Mishra, *Flight Simulation for the Development of an Experimental UAV.* Proceedings of the AIAA Modeling and Simulation Technologies Conference, 2002.
74. Unknown. *UAV Research Facility*. 2008 [cited 2008 February 18]; Available from: http://uav.ae.gatech.edu/pics/.
75. Gavrilets, V., et al., *Avionics System for Aggressive Maneuvers.* Aerospace and Electronic Systems Magazine, IEEE, 2001. 16(9): p. 38-43.
76. Gavrilets, V., et al., *Avionics system for a small unmanned helicopter performingaggressive maneuvers.* Digital Avionics Systems Conferences, 2000. Proceedings. DASC. The 19th, 2000.

77. Saripalli, S., et al., *A Tale of Two Helicopters.* IEEE/RSJ International Conference on Intelligent Robots and Systems, 2003. 1: p. 805-810 vol.1.
78. Kanade, T., O. Amidi, and Q. Ke, *Real-time and 3D Vision for Autonomous Small and Micro Air Vehicles.* Decision and Control, 2004. CDC. 43rd IEEE Conference on, 2004.
79. Mettler, B., M.B. Tischler, and T. Kanade, *System Identification of Small-size Unmanned Helicopter Dynamics.* Annual Forum Proceedings- American Helicopter Society, 1999. 2: p. 1706-1717.
80. Mettler, B., et al., *Attitude Control Optimization for a Small-Scale Unmanned Helicopter.* AIAA Guidance, Navigation and Control Conference, 2000: p. 40–59.
81. Johnson, E.N., *UAV Research at Georgia Tech*. 2002: Presentation at TU Delft.
82. Meingast, M., C. Geyer, and S. Sastry, *Vision Based Terrain Recovery for Landing Unmanned Aerial Vehicles.* Decision and Control, 2004. CDC. 43rd IEEE Conference on, 2004. 2: p. 1670-1675 Vol.2.
83. Mejias, L., et al., *Visual Servoing of an Autonomous Helicopter in Urban Areas using Feature Tracking.* Journal of Field Robotics, 2006. 23(3): p. 185-199.
84. Kelly, J., S. Saripalli, and G.S. Sukhatme, *Combined Visual and Inertial Navigation for an Unmanned Aerial Vehicle.* International Conference on Field and Service Robotics, 2007.
85. Roberts, J.M., P.I. Corke, and G. Buskey, *Low-cost Flight Control System for a Small Autonomous Helicopter.* Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on, 2003. 1: p. 546-551 vol.1.
86. Montgomery, J.F., et al., *The JPL Autonomous Helicopter Testbed: A Platform for Planetary Exploration Technology Research and Development.* Journal of Field Robotics, 2006.
87. Unknown. *The Autonomous Helicopter Testbed*. 2004 [cited 2008 January 28]; Available from: http://www-robotics.jpl.nasa.gov/systems/system.cfm?System=13.
88. Unknown. *Autonomous Helicopter: Stanford University AI Lab*. 2008 [cited 2008 February 18]; Available from: http://heli.stanford.edu/.
89. Mejias, L., et al., *A Visual Servoing Approach for Tracking Features in Urban Areas using an Autonomous Helicopter.* Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on, 2006: p. 2503-2508.
90. Sukhatme, G., J. Montgomery, and R. Vaughan, *Experiments with Cooperative Aerial-Ground Robots*, in *Robot Teams: From Diversity to Polymorphism*, T. Balch and L. Parker, Editors. 2001, A.K. Peters.
91. Unknown. *Flying Lessons*. 2001 [cited 2007 September 22]; Lists and Describes how to perform multiple UAV stunts.]. Available from: http://www.littlerotors.com/flyinglessons/index.aspx.
92. Ireland, S., et al. *Measurement of Altitude At Hot Air Balloon Competitions*. 2007 February 10 [cited 2007 November 23]; Available from: ftp://www.fai.org/ballooning/meetings/pc/2007/altitude%20measurement.doc.
93. Management, O.o.S., *United States Frequency Allocations: The Radio Spectrum*, in *Adobe Acrobat*, allochrt.pdf, Editor. 2003, U.S. DEPARTMENT OF COMMERCE. p. U.S.A. Frequency Allocation Table.
94. Weisstein, E.W. *Circle-Line Intersection*. [cited 2007 October 19]; Available from: http://mathworld.wolfram.com/Circle-LineIntersection.html.
95. Rhoad, R., G. Milauskas, and R. Whipple, *Geometry for Enjoyment and Challenge, rev. ed.* 1984, Evanston, IL: McDougal, Littell & Company.
96. Chandler, J. *Helicopters*. [cited 2007 November, 06]; Open source models of vehicles for XPlane]. Available from: http://c74.net/xplane/_helicopters.html.

97. Barnes, L. and W. Alvis, *Heterogeneous Swarm Formation Control Using Bivariate Normal Functions to Generate Potential Fields.* Proceedings of the IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications (DIS'06)-Volume 00, 2006: p. 85-94.
98. Barnes, L., M. Fields, and K. Valavanis. *Unmanned ground vehicle swarm formation control using potential fields*. in *Control & Automation, 2007. MED '07. Mediterranean Conference on*. 2007.
99. Unknown. *X-Plane UDP Data Outputs*. 2003 [cited 2006 July 11]; Available from: http://www.x-plane.info/udp/data_outputs.html.

Appendices

Appendix A Parts List

Table 13: COTS Parts List for the USL Helicopter Testbed

Hardware	Distributor	Model/Part Information	Quantity
Processing System			
Frame Grabber	Vox Technologies	VTC 4749 Model IVC-200G	1
Flexible PCI Extender	Adex Electronics, Inc.	PCIRX4-FLEX-A5	1
Wireless Card	XP Passport	Intel Pro 2200 B/G	1
Motherboard	Mini-itx.com	G5M100-N	1
CPU	Newegg.com	Pentium M 755	1
Fan/Heat Sink	Eaglebit.com	CoolJag SEA-A2	1
RAM	Crucial Technology	CT2KIT12872Y335 (2 Gig kit) 184-Pin	1
Power Supply	Mini-itx.com	picoPSU-120	1
2.4GHz antenna	Fab-Corp	5.5 dBi R-SMA Rubber Duck Antenna	1
2.4GHz cable	Fab-Corp	6 in Hirose U.FL to R-SMA Female	1
Phono Jacks	Radio Shack	Board with Four Standard-Type Phono Jacks (274-322)	1
Power switch	Digi-key Corp.	EG1512-ND	1
GPS			
GPS	Navtech GPS	SSII-5-5Hz (Superstar 2) PVT	1
GPS Extension Cable	Navtech GPS	700016 RF cable MCX to SMA Female Bulkhead, 6 in	1
GPS Antenna	Navtech GPS	GAACZ-A41 13DB 5V SMA	1
TTL to USB Adapter	Mouser Electronics Inc.	DLP-TXRX-G	1
USB Cable	Provantage	USB Type A to Mini-B Cable	1
Wire Mesh	TWP Inc.	TWP Part #: 022X022C0150W48T	< 1 sqft
Heat Shrink	Batteryspace.com	PVC-28 (1 1/8 inch PVC)	< 1 ft
GPS Connector	Samtec	TCSD-10-S-06.00-01	1
Voltage Regulator	Radio Shack	5V 1A 7805A Regulator	1
Double Sided Foam Tape	Amazon.com	4008 (3M 1" Double-Sided Foam Tape)	1
Enclosure			
EMI Foil	3M	1345 Tape (23 inch width)	~ 3 sqft
Basswood	Hobby Town USA	3mm Thick Basswood	< 2 sqft
Wood Glue	Lowes	Elmer's Wood Glue	< 8 oz

Table 13 (Continued)

Double Sided Servo Tape	Tower Hobbies	DTXR1215 (DuraTrax Servo Tape 1x36")	1
Masking Tape	Lowes	00-05130-01 (Clean Release Tape)	< 1 ft
Pan/Tilt			
Servos	Tower Hobbies	Hobbico CS-59 Lo Profile	2
Small Gears	Qtcgears.comm	KPS1-45 (45 Teeth Plastic Spur Gear)	2
Large Gears	Qtcgears.comm	KPS1-28 (28 Teeth Plastic Spur Gear)	2
Chassis			
Rubber Isolation Mounts	SDP-SI	A10Z 2-302A	4
Rubber Isolation Mounts	SDP-SI	A10Z 2-301B	4
Camera			
Camera	Aegis Electronic Group Inc.	FCB-EX980S	1
Camera Interface Board	Aegis Electronic Group Inc.	IFB-EX232	1
Video Transmitter	Eyespyvideo.com	THX-9100	1
D-sub 9 Serial Connectors	Allelectronics.com	9-PIN Female D-sub Connector, IDC Style	1
Heat Shrink	Batteryspace.com	PVC-126 (1 1/8 inch PVC)	< 1ft
Transmitter Power Conn.	Radioshack	274-1569 (Size M Coaxial DC Pwr Plug)	1
Futaba Connector	Tower Hobbies	EMS Female Connector J	1
Futaba Connector	Tower Hobbies	EMS Male Connector J	1
IMU			
IMU	Microstrain	3DMG-X1	1
Voltage Regulator	Medusa Research	6V 1.5A MR-BEC-35015-6	1
Futaba Connector	Tower Hobbies	EMS Male Connector J	3
Futaba Connector	Tower Hobbies	EMS Female Connector J	2
Laser			
Laser	Acroname	Hokuyo URG-04LX	1

Table 13 (Continued)

Laser Connector (Shroud)	Digi-key Corp.	PHR- 8	1
Laser Connector (Pins)	Digi-key Corp.	SPH-002T-P0.5S	1
Futaba Connector	Tower Hobbies	EMS Male Connector J	1
Electrical Tape	RadioShack	64-2375 (3/4" Electrical Tape)	< 5 ft
SSC Interface Board			
SSC Header	Digi-key Corp.	8844FE-ND (DB44 Female HD DIP)	1
Headers (Double Sided)	Sametc	TSW-137-16-T-S	1
Headers (Single Sided)	Digi-key Corp.	WM6436-ND	1
SSC	Microbotics	Servo Switch/Controller	1
RS232 to USB Adapter	Mouser	UC232R	1
Battery	Thunder Power	TP730-3SJPL	1
Futaba Connector	Tower Hobbies	EMS Male Connector J	2
Power Input Connector		Ultra Dean Male Connector	1
Helicopter			
Helicopter Kit	Joker USA	Joker Maxi 2 Kit	1
Main Rotor Blades	Joker USA	Joker Maxi 2 Symmetrical Blades	1
Tail Rotor Blades	Joker USA	Joker Maxi 2 Tail Blades	1
Motor	Joker USA	Plettenberg HP 370/40/A2 Heli	1
Speed Controller	Joker USA	Schulze Future 40/160H	1
Servos	Tower Hobbies	Futaba S9250 Digital Servos	3
Gyro	Tower Hobbies	Futaba GY401 w/ 9254 Digital Servo	1
Receiver	Tower Hobbies	R319DPS 9 Channel Receiver	1
Battery	Tower Hobbies	HydriMax 4.8 2 Ah NiMh	1
Battery Switch	Tower Hobbies	HCAM2761 HD Switch Harness	1
Battery	Austin Else LLC	Joker Maxi 2 37V 10Ah LiPo	1
Power Cable			
Low Voltage Alarm	Heliproz.com	HRPoly X Low Voltage Warning Device	1
Ultra Dean (Male)	Tower Hobbies	W.S. Deans Male Ultra Plug	1
Ultra Dean (Female)	Tower Hobbies	W.S. Deans Female Ultra Plug	1
Power Cable	RadioShack	278-567 (18 Gauge Speaker Wire)	< 1 ft
ESC Power Adapter Cable			
Ultra Dean (Male)	Tower Hobbies	W.S. Deans Male Ultra Plug	2

Table 13 (Continued)

Ultra Dean (Female)	Tower Hobbies	W.S. Deans Female Ultra Plug	1
Power Cable	RadioShack	278-567 (18 Gauge Speaker Wire)	< 1 ft
Misc			
Servo Safety Clips	Tower Hobbies	LXDT85	4
Battery	Thunder Power	THP 4200 3S2P PL	1
Servo Wire	Tower Hobbies	Hitec HD Servo Wire 20GA	< 25 ft
Spiral Cable Wrap	RadioShack	278-1638 (3/8" Spiral Cable Wrap)	< 10 ft
J Male Connectors	Tower Hobbies	EMS Unassembled Male Connector J	30
J Female Connectors	Tower Hobbies	EMS Unassembled Female Connector J	10

Appendix B Servo/Safety Controller Interface Schematic

This appendix details the schematic for the wiring of the SSC interface board. "In" and "out" components are for servo connections where pin 1 represents the PW signal, pin 2 represents the 5V power, and pin 3 represents the respective ground. For the "battery" component pin 2 represents the power input for the SSC and pin 3 represents the respective ground. For the RS-232 component pin 1 is the TX wire, pin 2 is the RX wire, and pin 3 is the communication ground. DB 44 HD represents the 44 pin high density connector used to mate with the SSC.

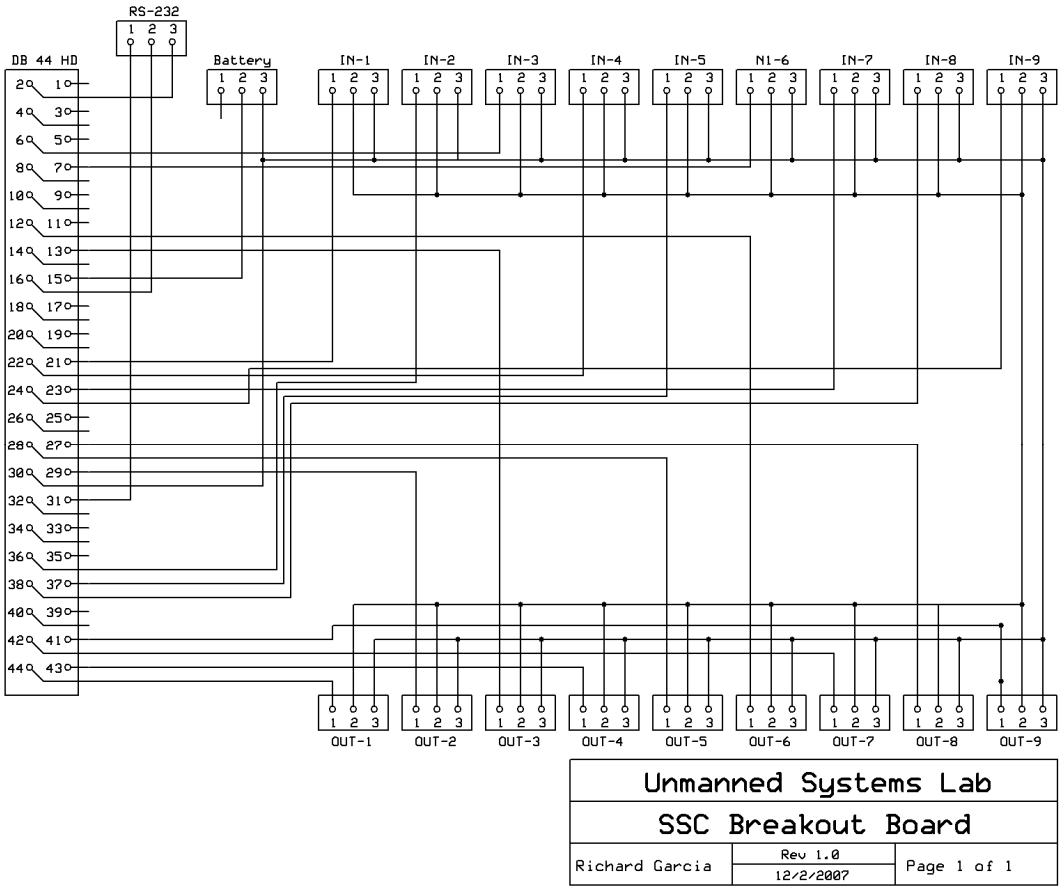

Figure 70: SSC Connections Schematic

Appendix C Chassis Schematics

The Chassis is the largest of all the custom made components of the USL testbed and encompasses the Chassis frame and the enclosure mounting plate. Make note that all measurements are in millimeters.

C.1 Chassis

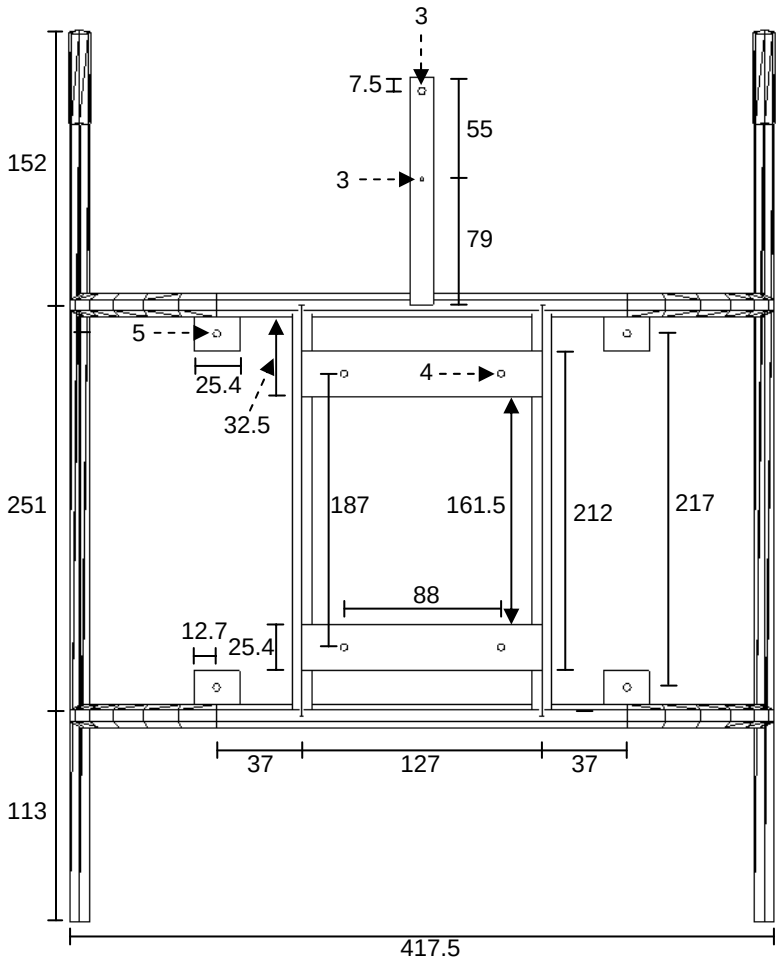

Figure 71: Top View of the USL Chassis

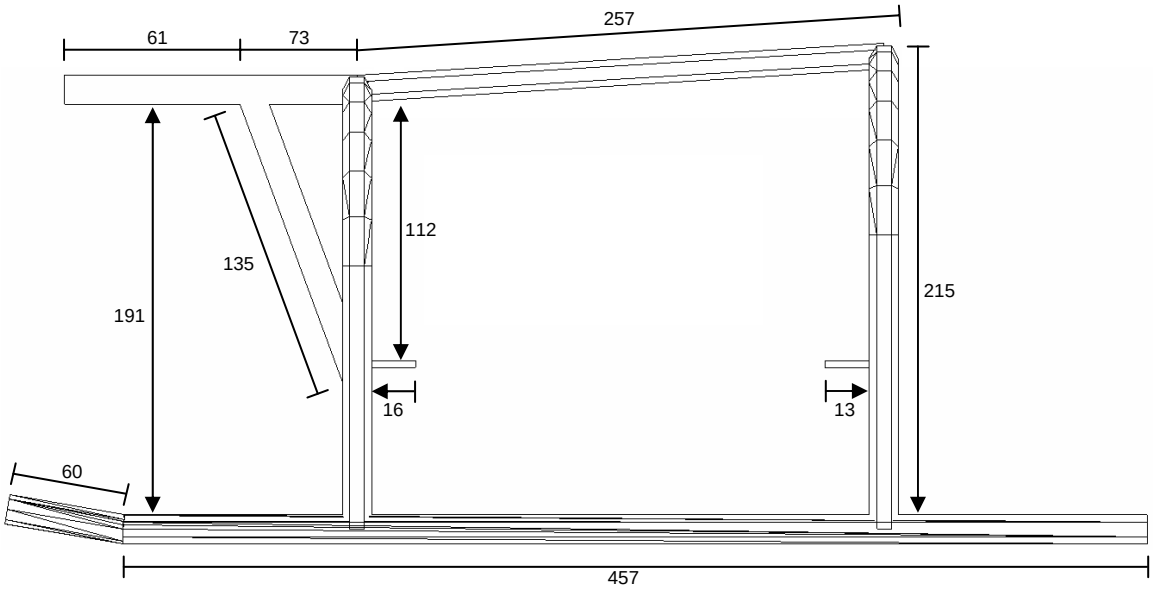

Figure 72: Side View of the USL Chassis

Note that in Figure 73 is only designed to detail the arc of the bends in the chassis and is not a rear view of the chassis. Both arcs on the USL chassis are identical and only vary based on their height above the bottom of the chassis, see Figure 73.

Appendix C (Continued)

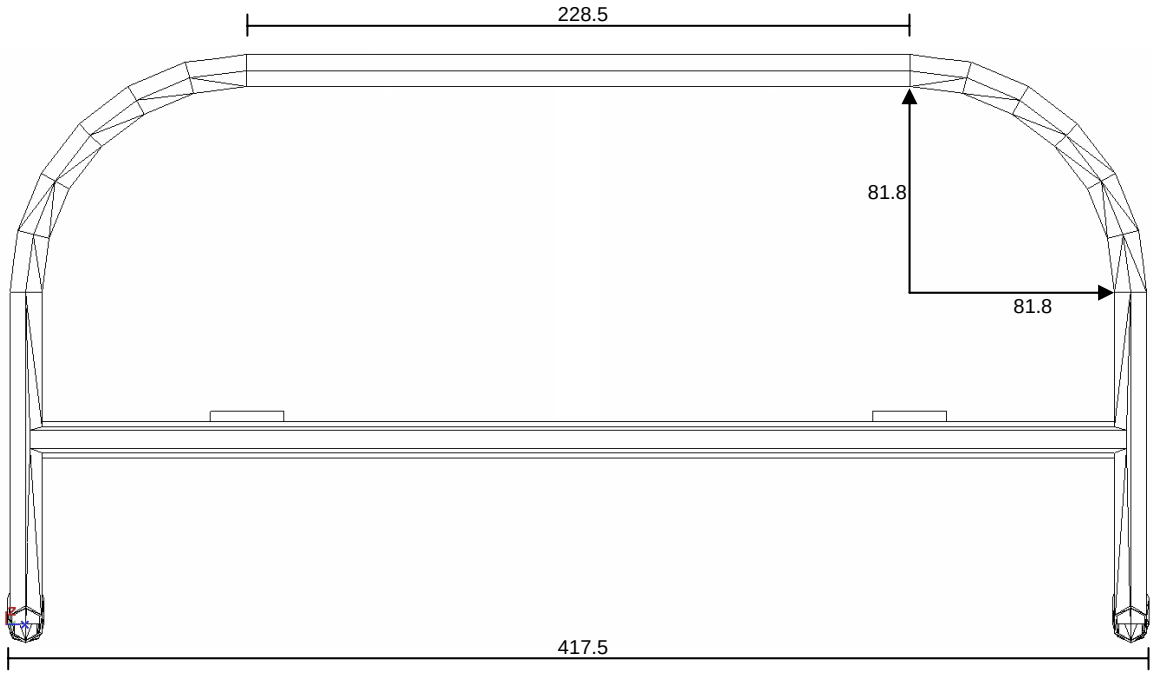

Figure 73: Schematic of the Arcs on the Chassis

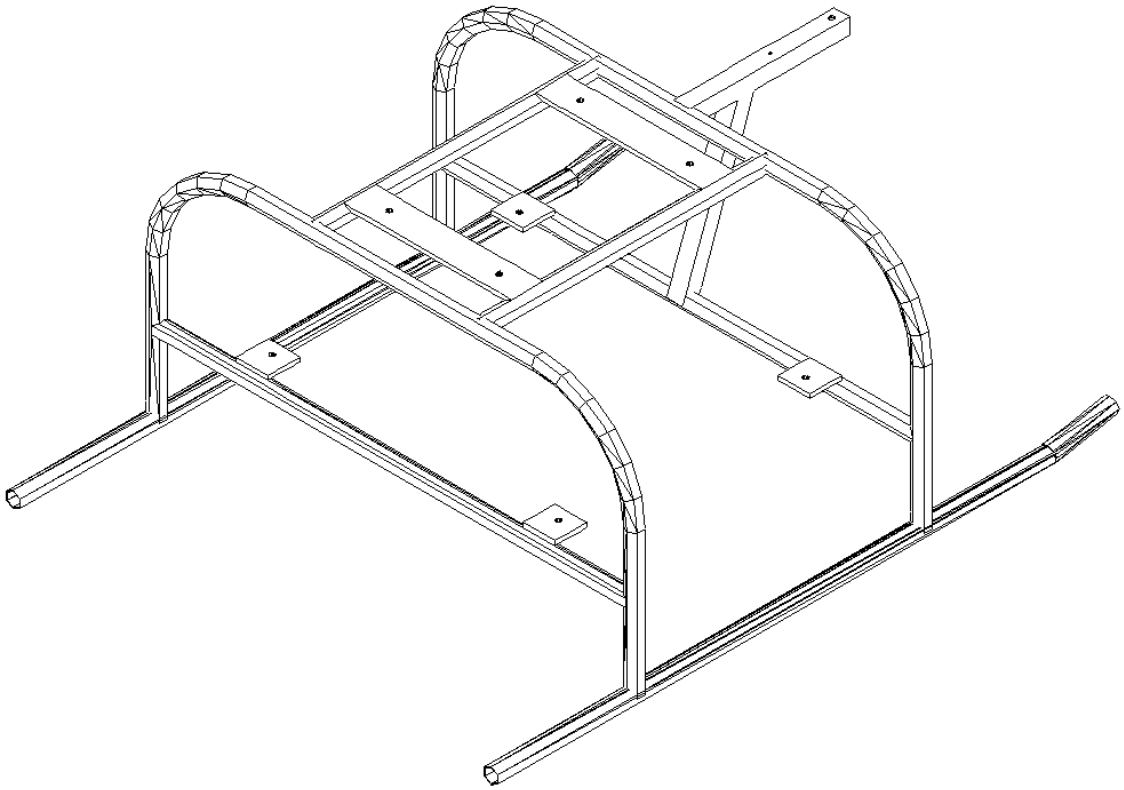

Figure 74: 3D View of the Assembled Chassis

C.2 Enclosure Mounting Plate

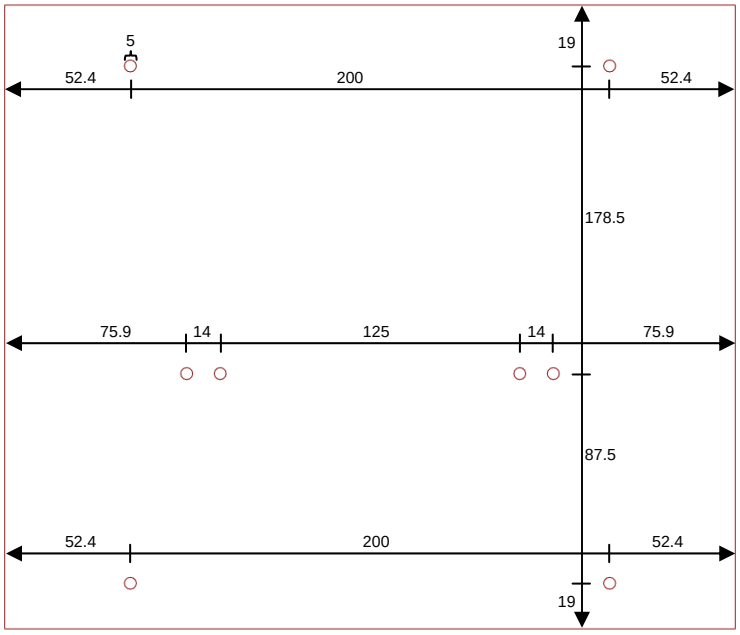

Figure 75: Top View of the Enclosure Mounting Plate

C.3 Chassis Mounting Adapter

The chassis mounting adapter is a 3mm aluminum plate used to mount the custom aluminum chassis to the stock skid mounts on the Joker Maxi-2 helicopter. Note that two identical adapters are needed to fully install the chassis.

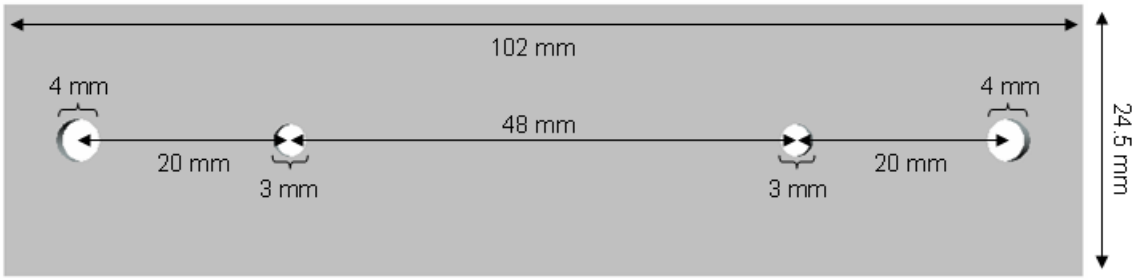

Figure 76: Top View of Chassis/Helicopter Mounting Adapter

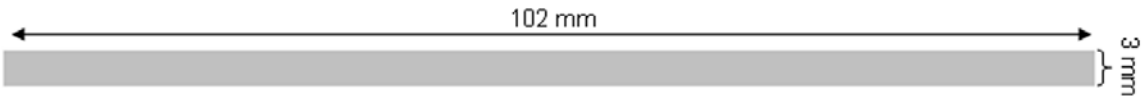

Figure 77: Side View of Chassis/Helicopter Mounting Adapter

Appendix D Pan/Tilt Schematics

The pan/tilt schematics consist of the camera's upper and lower brackets and the servos upper and left brackets. Note that all the hardware described in this section is formed from 3mm thick aluminum. Also note that measurements marked with an '*' represent measurements that are from the center of the hole. This notation is only used where the location of measurements may be confusing.

D.1 Camera Upper Bracket

Figure 78: 3D Perspective of the Upper Camera Bracket

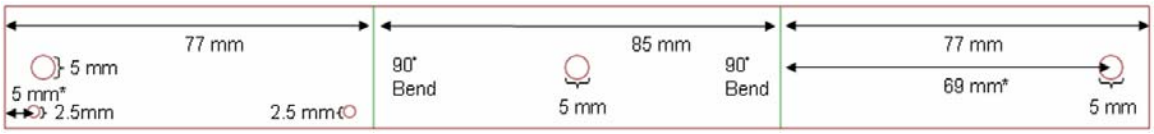

Figure 79: Flattened View of the Upper Camera Bracket

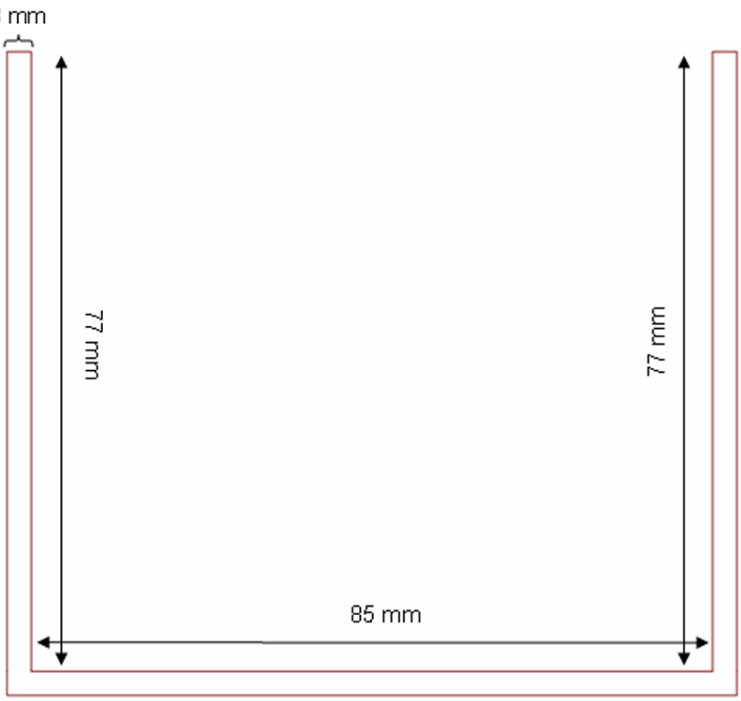

Figure 80: Side View of the Upper Camera Bracket

Appendix D (Continued)

D.2 Camera Lower Bracket

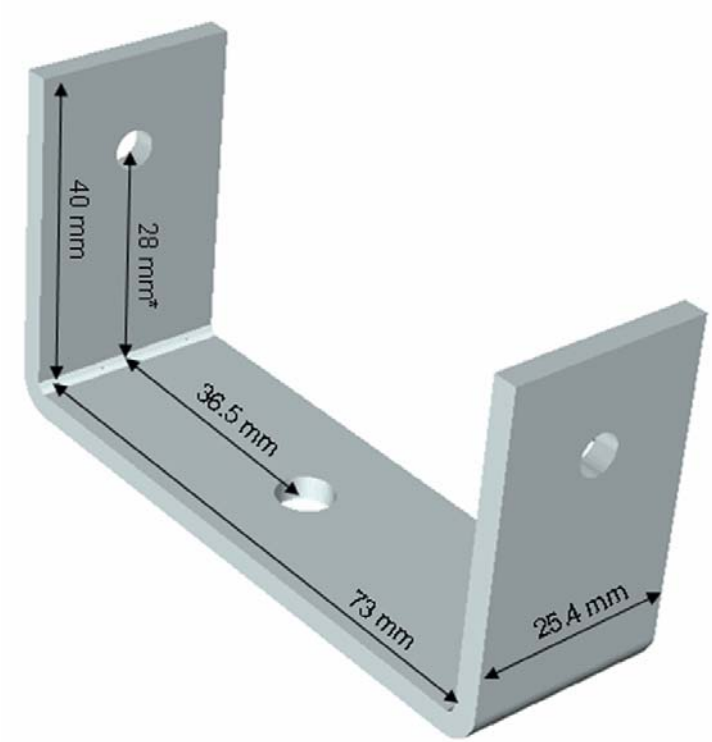

Figure 81: 3D Perspective of the Lower Camera Bracket.

Figure 82: Flattened View of the Lower Camera Bracket.

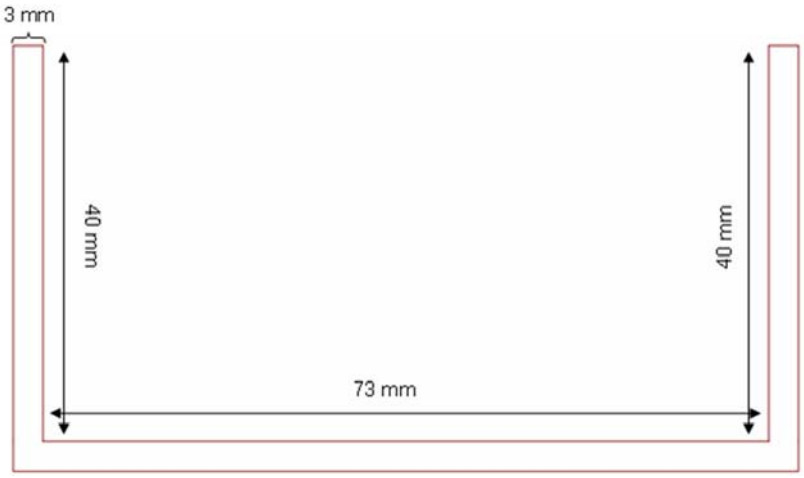

Figure 83: Side View of the Lower Camera Bracket

D.3 Servo Upper Bracket

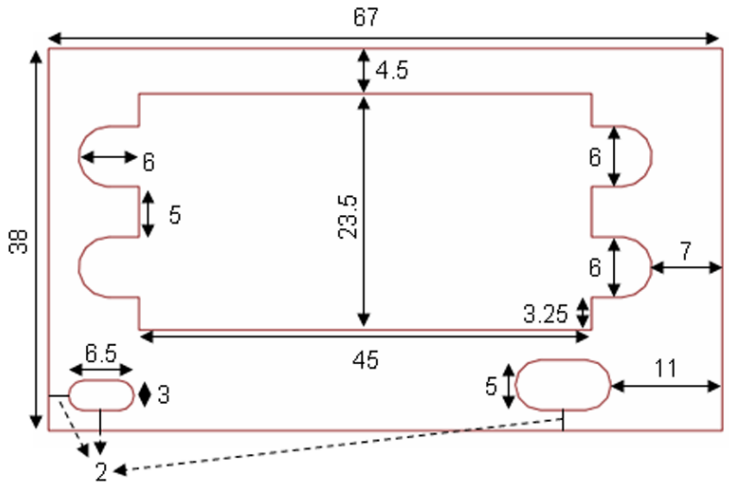

Figure 84: Top View of the Upper Servo Bracket

Figure 85: 3D Perspective of the Upper Servo Bracket

D.4 Servo Side Bracket

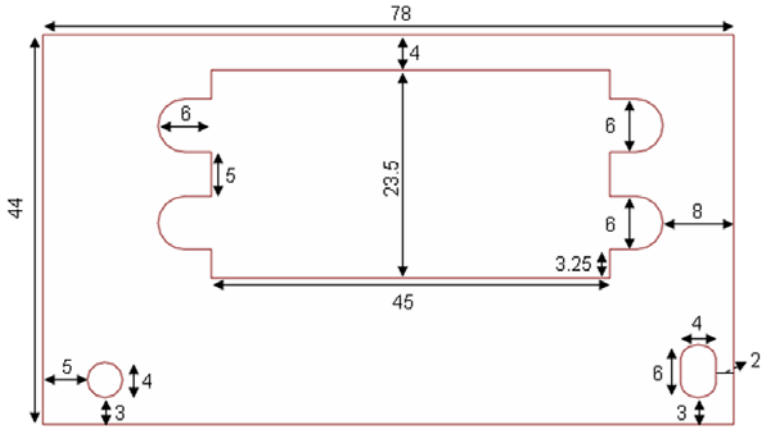

Figure 86: Top View of the Side Servo Bracket

Figure 87: 3D Perspective of the Side Servo Bracket

Appendix E Enclosure Schematics

This appendix presents all of the schematic designs necessary to replicate the enclosure used on the USL testbed. Make note that all measurements are in millimeters.

E.1 Enclosure Faceplate

Make note that in Figure 88 the schematic does not show the bottom plate of the enclosure. This is because the bottom plate is mounted to the back of the faceplate. This designed allows for more surface area near the edge of the faceplate which makes manufacturing somewhat simpler.

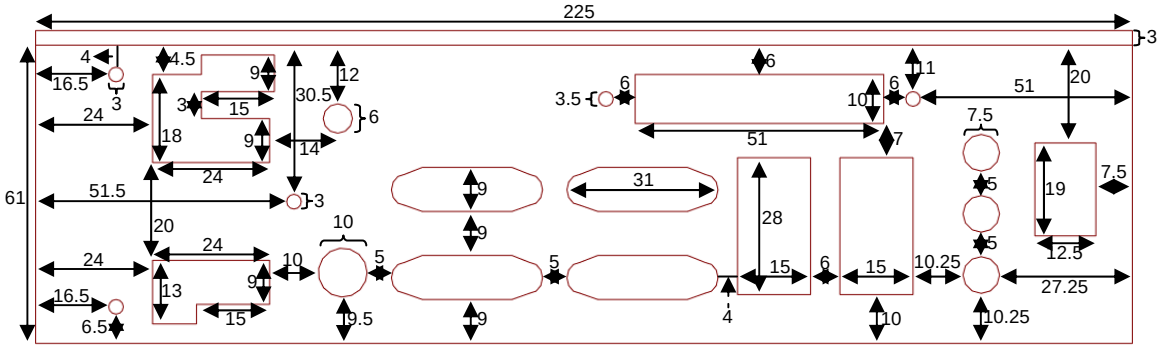

Figure 88: Enclosure's Faceplate Schematic

E.2 Enclosure Left Side

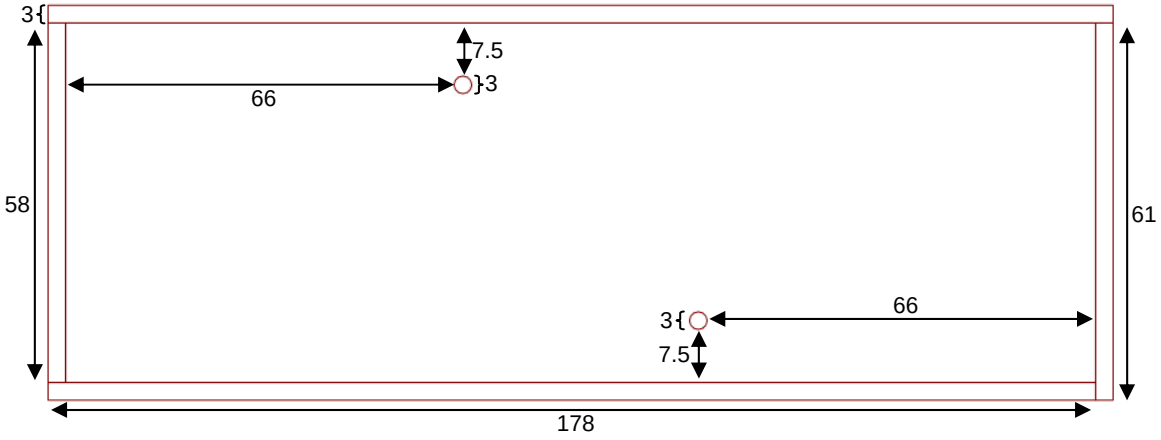

Figure 89: Enclosure's Left Side Schematic

E.3 Enclosure Right Side

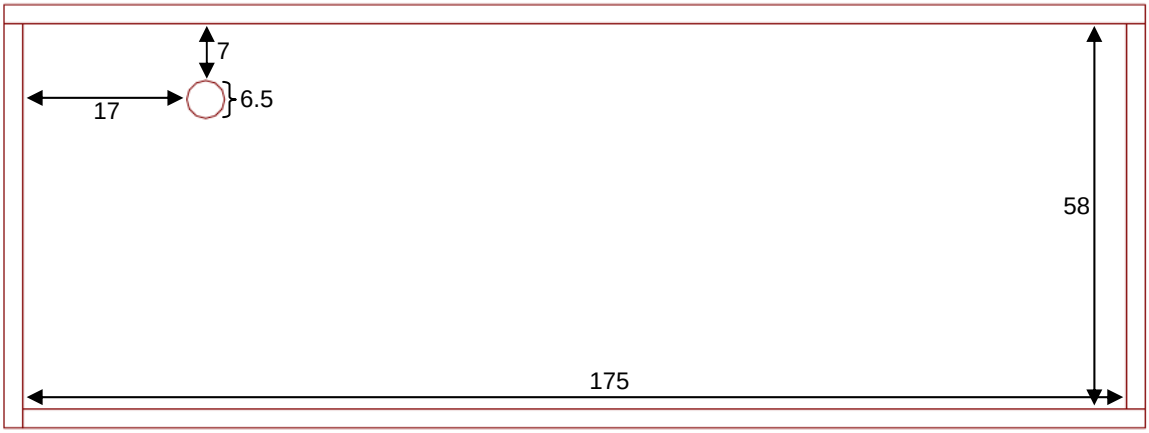

Figure 90: Enclosure's Right Side Schematic

E.4 Enclosure Box

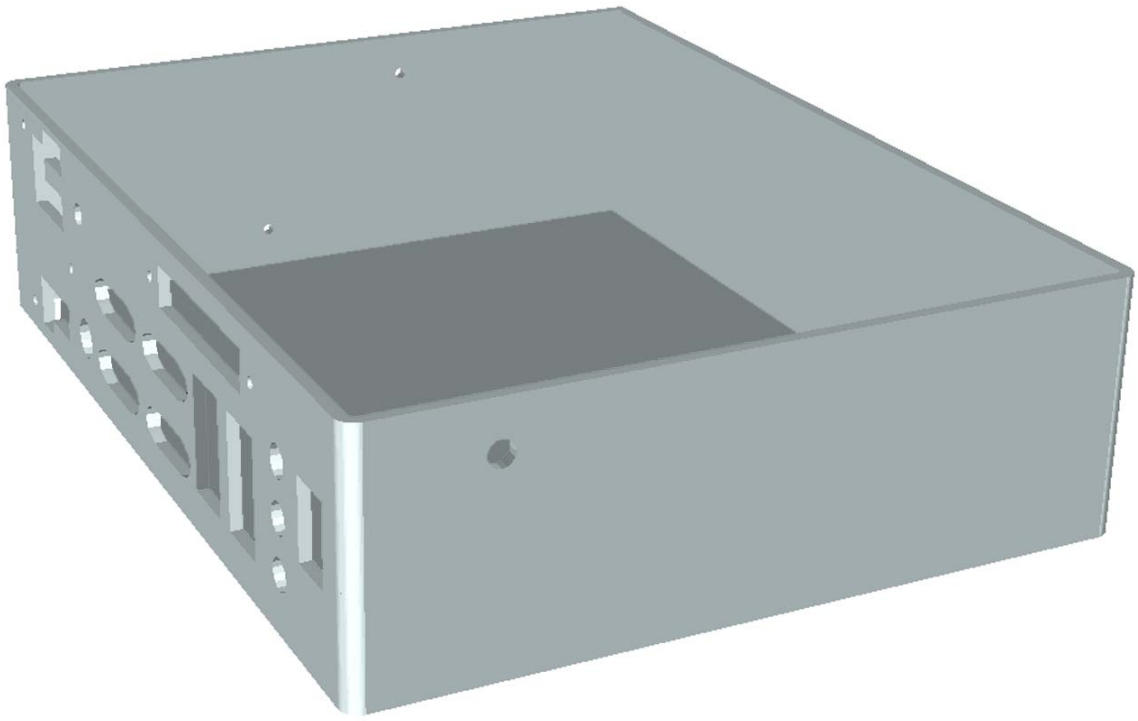

Figure 91: 3D Representation of the Enclosure (without lid)

Appendix F Operating System Package Deletion List

This appendix is dedicated to listing the packages removed during the installation of the OS. This appendix specifically refers to the installation that will eventually be copied onto the USB and utilized on the testbed. This appendix is broken down into two sections representing the two groups of packages that were selected during the installation. Each section list the packages that were deselected for installation from that of the default selections.

F.1 Base Linux System

Packages deselected: acpid, apmd, cpio, cryptsetup, cupps, eject, elvis ,gawk, gpm, jfsutils, kernel generic, kernel generic smp, kernel huge smp, mdadm, mt-st, mtx, ncompress, pcmciautils, quota, reiserfsprogs, rpm2tgz, splitvt, sysklogd, tree, unarj, which, xfsprogs, zoo.

F.2 Networking

Packages deselected: biff+comsat, bitchx, bluez-firmware, bluez-hcidump, bluez-libs, bluez-utils, bridge-utils, cyrus-sasl, elm, epic4, fetchmail, getmail, imapd, iptables, irssi, links, lynx, mailx, mcabber, metamail, mutt, nettalk, netkit-bootparamd, netwrite, nn, popa3d, procmail, samba, sendmail, sendmail-cf, slrn, tin, trn, ytalk

Appendix G Pseudocode

G.1 Inc.c

G.1.1 getCurrentTime(…)

```
/*Function:     Captures the current system time which is the number of seconds since   */
/*              midnight January 1st 1970                                               */
/* Inputs:      N/A                                                                     */
/* Outputs:     1) seconds                                                              */
/*              2) microseconds                                                         */
void getCurrentTime(…)
{
        //gettimeofday(…,NULL);
        //set seconds output
        //set microseconds output
}
```

G.1.2 timeDifference(…)

```
/*Function:     Returns the time difference between two supplied times                  */
/*Inputs:       1) old time (seconds)                                                   */
/*              2) old time (microseconds)                                              */
/*              3) new time (seconds)                                                   */
/*              4) new time (microseconds)                                              */
/*Outputs:      1) difference (seconds.microseconds)                                    */
double timeDifference(…)
{
        //calculate the difference between the old and new seconds (secDiff)
        //calculate the difference between the old and new microseconds (microDiff)

        //if microDiff < zero
        {
                //return secDiff - microDiff/1000000
        }
        //else
        {
                //return secDiff + microDiff/1000000
        }
}
```

Appendix G (Continued)

G.2 Inc.h

```
/*Structure: Shared memory structure for the laser range data                */
//struct laser_struct
{
        //laser array
};

/*Structure: Shared memory structure for the state of control                */
//struct state_struct
{
        //state of the control (human or computer)
 };

/*Structure: Shared memory structure for data collecting                     */
//struct store_data
{
        // fuzzy controllers output array
        // fuzzy controllers input array (collective)
        // fuzzy controllers input array (yaw)
        // fuzzy controllers input array (roll)
        // fuzzy controllers input array (pitch)
        // position array (latitude, longitude, altitude)
};

/*Structure: Shared memory structure for servo control                       */
//struct servo_control
{
        //left servo pulse width
        //right servo pulse width
        //forward servo pulse width
        //tail servo pulse width
        //gyro pulse width
        //throttle pulse width
};

/*Structure: Shared memory structure for pan/tilt control                    */
//struct PnT
{
        //panning servo pulse width
        //tilt servo pulse width
};

/*Structure: Shared memory structure for GPS data                            */
//struct gps
{
```

Appendix G (Continued)

```
        //latitude
        //longitude
        //latitude direction
        //longitude direction
        //type of gps lock
        //number of satellites in lock
        //altitude
        //gps position incrementer
};
/*Union: Union for semaphores                                   */
//union semun
{
        //int val;
        //struct semid_ds *buf;
        //ushort *array;
};

/*Structure: Shared memory structure for IMU data               */
//struct imu
{
        //orientation array (roll, pitch, yaw)
        //accelerations array
        //angular rate array
        //timestamp (seconds)
        //timestamp (microseconds)
};

/*Structure: Shared memory structure for data passed between vehicles   */
//struct vehicledata
{
        //latitude
        //longitude
        //heading
};

//function declaration for getCurrentTime(…);
//function declaration for timeDifference(…);
```

G.3 Robot_defs.h

```
//define number of channels for SSC communication
//define front servo's neutral, maximum, and minimum pulse width values
//define left servo's neutral, maximum, and minimum pulse width values
//define right servo's neutral, maximum, and minimum pulse width values
//define throttle's neutral, maximum, and minimum pulse width values
```

Appendix G (Continued)

```
//define tail servo's neutral, maximum, and minimum pulse width values
//define gyro's neutral, maximum, and minimum pulse width values
//define collective neutral, maximum, and minimum pulse width offsets
//define pan servo's neutral pulse width value
//define tilt servo's neutral pulse width value
//function declaration for connectHeliServo();
//function declaration for disconnectRobotServo(…);
//function declaration for heliDriveCommand(…);
//function declaration for CalcCheckSum(…);
```

G.4 Robot_motor.c

```
//include "Robot_defs.h"
```

G.4.1 connectHeliServo( )

```
/*Function:     Setup comms and connect to SSC device                    */
/*Inputs:       N/A                                                      */
/*Outputs:      1) file descriptor pointing to SSC device                */
int connectHeliServo()
{
        //open gps device with the O_RDWR | O_NOCTTY | O_NDELAY options
        // if open fails
        {
                //print error
                //return -1;
        }

        /*SETUP COMMUNICATION OPTIONS*/
        // no parity
        // one stopbit
        // No hardware flow control.
        // 8N1
        // set min read characters to 0
        // VTIME takes over (i.e. = 0)
        // make output raw (i.e. cfmakeraw())
        // set input/output baud rate

        // flush input buffer
        // set attributes

        // return file descriptor
}
```

G.4.2 disconnectRobotServo(…)

```
/*Function:     Stop comms with the SSC                                  */
/*Inputs:       1) file descriptor pointing to the SSC device            */
/*Outputs:      N/A                                                      */
void disconnectRobotServo(…)
{
        //close file descriptor
}
```

G.4.3 calcCheckSum(…)

```
/*Function:     Calculates a two byte checksum for the SSC data packet   */
/*Inputs:       1) data packet                                           */
/*Outputs:      1) checksum byte one                                     */
/*              2) checksum byte two                                     */
void calcCheckSum(…)
{
        //Calculate a Fletcher checksum as defined in RFC 1145
}
```

G.4.4 heliDriveCommand(…)

```
/*Function:     Creates and sends a packet to the SSC requesting servo movement  */
/*Inputs:       1) file descriptor pointing to the SSC device            */
/*              2) left servo pulse width                                */
/*              3) throttle pulse width                                  */
/*              4) tail servo pulse width                                */
/*              5) gyro pulse width                                      */
/*              6) right servo pulse width                               */
/*              7) pan servo pulse width                                 */
/*              8) tilt servo pulse width                                */
/*Outputs:      N/A                                                      */
void heliDriveCommand(…)
{
        //put header bytes into data packet
        //put message type byte into data packet
        //put data byte count into data packet

        //break individual inputs into two bytes each and add to data packet

        //calculate the checksum for data packet (i.e. CalcCheckSum(…))
```

Appendix G (Continued)

```
	//add checksum bytes to data packet

	//send data packet to SSC
}
```

G.5 Gps.c

```
#include Inc.h
```

G.5.1 connectRobotGPS( )

```
/*Function:	Setup comms and connect to GPS device		*/
/*Inputs:	N/A				*/
/*Outputs:	1) global file descriptor pointing to GPS device	*/
int connectRobotGPS( )
{
	//open gps device with the O_RDWR | O_NOCTTY options
	// if open fails
	{
		//print error
		//return -1;
	}

	/*SETUP COMMUNICATION OPTIONS*/
	// no parity
	// one stopbit
	// No hardware flow control.
	// 8N1
	// set min read characters to 0
	// VTIME takes over (i.e. = 0)
	// set input/output baud rate

	// flush input buffer
	// set attributes

	//return(0);
}
```

G.5.2 getGPS( )

```
/*Function:	Receive and parse data packet from the GPS	*/
/*Inputs:	N/A				*/
```

Appendix G (Continued)

```
/*Outputs:      1) latitude                                                    */
/*              2) longitude                                                   */
/*              3) direction of latitude                                       */
/*              4) direction of longitude                                      */
/*              5) type of GPS lock                                            */
/*              6) number of satellites tracking                               */
/*              7) altitude                                                    */
int getGPS(double *lat, double *lon, char *dirLat, char *dirLon, int *lock, int *sats, float
*altitude)
{
        //while not done
        {
                //while end of line has not been reached
                {
                        //read a single character
                        //if read is successful
                        {
                                //add character to input string
                        }
                        //else
                                //sleep for maximum transition time of one character
                }
        }

        //if input string looks incorrect
                //return(-1)

        //if message type is GPGGA
        {
                //parse string for latitude, longitude, latitude direction, longitude direction, lock,
                //number of satellites, and altitude
        }

        //convert latitude to degrees.hours
        //convert longitude to degrees.hours
        //return 0;
}
```

G.5.3 signalHandler(…)

```
/*Function:     Closes devices if a terminate signal is received               */
/*Inputs:       1) Signal that signals this function should be performed       */
/*Outputs:      N/A                                                            */
static void signalHandler(int signal)
{
        //close file descriptor for GPS device
```

```
	//exit(0);
}
```

G.5.4 main( )

```
/*Function:	Main for the GPS process	*/
/*Inputs:	N/A	*/
/*Outputs:	N/A	*/
int main( )
{
	//connect to the gps
	//if (connectRobotGPS() < 0)
		//exit(-1);

	// install signal handler for manual break
	//signal(SIGINT, signalHandler);

	//create shared memory of appropriate size
	//attach shared memory to this process

	//create a single semaphore with no semaphore flags

	//decrement semaphore
	//initialize GPS shared memory to default values
	//increment semaphore;

	//getGPS(…);

	//while (1)
	//{
		//getGPS(…);

		//decrement semaphore
		//update shared memory
		//increment semaphore;

		//if type of lock changes output status to display

		//sleep for 90% of the time between data packets
	}
	//exit(0);
}
```

Appendix G (Continued)

G.6 Laser.c

```
#include Inc.h
```

G.6.1 connectLaser( )

```
/*Function:     Setup comms and connect to Laser device                    */
/*Inputs:       N/A                                                        */
/*Outputs:      N/A                                                        */
int connectLaser( )
{
        //open laser device (ttyACM?) with the O_RDWR | O_NOCTTY | O_NDELAY options
        // if open fails
        {
                //print error
                //return -1;
        }

        /*SETUP COMMUNICATION OPTIONS*/
        // no parity
        // one stopbit
        // No hardware flow control.
        // 8N1
        // set min read characters to 0
        // VTIME takes over (i.e. = 0)
        // c_lflag = ICANON
        // set input/output baud rate

        // set attributes
        // flush input/output buffers

        //return(0);
}
```

G.6.2 fillBuffer(…)

```
/*Function:     Fills a string with data until an end of line character is received  */
/*Inputs:       1) file descriptor for device                                        */
/*Outputs:      1) string of input data                                              */
fillBuffer(…)
{
        //while end of line has not been reached
```

```
        {
                //read a single character
                //if read is successful
                {
                        //add character to input string
                }
                //if read fails
                        //sleep for maximum transition time of one character
        }
}
```

G.6.3 getVersion( )

```
/*Function:     Attempts to gather version data for the laser device        */
/*Inputs:       N/A                                                         */
/*Outputs:      N/A                                                         */
int getVersion( )
{
        //send version request command (VV) to laser
        //if write to device fails
                //return -1;

        //read laser's echo of data request
        //fillBuffer(…)

        //read laser response
        //fillBuffer(…)

        //if error packet was not received (i.e. "00P" msg)
        {
                //fillBuffer(…)
                //output version information to display
        }
        //if error packet was received
        {
                //output error
                //fflush(stdout);
        }
        //return 1;
}
```

G.6.4 getSpecifications( )

```
/*Function:     Attempts to gather the specifications for the laser device      */
/*Inputs:       N/A                                                             */
/*Outputs:      N/A                                                             */
getSpecifications( )
{
        //send specification request command (PP) to laser
        //if write to device fails
                //return -1;

        //read laser's echo of data request
        //fillBuffer(…)

        //read laser response
        //fillBuffer(…)

        //if error packet was not received (i.e. "00P" msg)
        {
                //fillBuffer(…)
                //output specification information to display
        }
        //if error packet was received
        {
                //output error
                //fflush(stdout);
        }
        //return 1;
}
```

G.6.5 getState( )

```
/*Function:     Attempts to gather information regarding the current state of the laser */
/*              device                                                          */
/*Inputs:       N/A                                                             */
/*Outputs:      N/A                                                             */
getState( )
{
        //send state request command (II) to laser
        //if write to device fails
                //return -1;

        //read laser's echo of data request
        //fillBuffer(…)
```

Appendix G (Continued)

```
        //read laser response
        //fillBuffer(…)

        //if error packet was not received (i.e. "00P" msg)
        {
                //fillBuffer(…)
                //output state information to display
        }
        //if error packet was received
        {
                //output error
                //fflush(stdout);
        }
        //return 1;
}
```

G.6.6 setVersion( )

```
/*Function:     Sets the communication version of the laser to version 2.0      */
/*Inputs:       N/A                                                             */
/*Outputs:      N/A                                                             */

int setVersion( )
{
        //send version set request command (SCIP2.0) to laser
        //if write to device fails
                //return -1;

        //read laser's echo of data request
        //fillBuffer(…)

        //read laser response
        //fillBuffer(…)

        //if first returned character is zero
        {
                //display "Version change successful"
                //fflush(stdout);
        }
        //if first returned character is not zero
        {
                //output error
                //fflush(stdout);
        }

        //flush I/O buffers
```

Appendix G (Continued)

```
        return 1;
}
```

G.6.7 turnOnLaser( )

```
/*Function:     Commands the laser to the "On" state and sets the operation mode    */
/*              measurement                                                         */
/*Inputs:       N/A                                                                 */
/*Outputs:      N/A                                                                 */
int turnOnLaser( )
{
        //send on request command (BM) to laser
        //if write to device fails
                //return -1;

        //read laser's echo of data request
        //fillBuffer(…)

        //read laser response
        //fillBuffer(…)

        //if the first two returned characters are zero
        {
                //display "Laser is ON"
                //fflush(stdout);
        }
        //else if the first returned character is zero and the second is two
        {
                //display "Laser was already ON"
                //fflush(stdout);
        }
        //else
        {
                //display error message
                //fflush(stdout);
        }

        //flush I/O buffers
        //return 1;
}
```

G.6.8 turnOffLaser( )

```
/*Function:     Commands the laser to the "Off" state                    */
/*Inputs:       N/A                                                      */
/*Outputs:      N/A                                                      */
int turnOffLaser( )
{
        //send turn off request command (QT) to laser
        //if write to device fails
                //return -1;

        //read laser's echo of data request
        //fillBuffer(…)

        //read laser response
        //fillBuffer(…)

        //if error packet was not received (i.e. "00P" msg)
        {
                //display "Laser is off"
                //fflush(stdout);
        }
        //if error packet was received
        {
                //output error
                //fflush(stdout);
        }

        //flush I/O buffers
        //return 1;
}
```

G.6.9 resetLaser( )

```
/*Function:     Commands the laser to reset                              */
/*Inputs:       N/A                                                      */
/*Outputs:      N/A                                                      */
int resetLaser( )
{
        //flush I/O buffers

        //send turn off request command (RS) to laser
        //if write to device fails
                //return -1;

        //read laser's echo of data request
```

Appendix G (Continued)

```
        //fillBuffer(…)
        //if echo does not equal request
                //return(-1);

        //read laser response
        //fillBuffer(…)

        //if error packet was not received (i.e. "00P" msg)
        {
                //display "Reset Successful"
                //fflush(stdout);
        }
        //if error packet was received
        {
                //output error
                //fflush(stdout);
        }

        //flush I/O buffers
        //return 1;
}
```

G.6.10 startMS( )

```
/*Function:     Request laser send data continuously without requests    */
/*Inputs:       N/A                                                      */
/*Outputs:      N/A                                                      */
int startMS( )
{

        //create MS request packet (i.e. MS0360041001000)
        //send range data request command (MS…) to laser
        //if write to device fails
                //return -1;
        //return 0;
}
```

G.6.11 getLaser(…)

```
/*Function:     Parses data sent by the laser                            */
/*Inputs:       N/A                                                      */
/*Outputs:      1) array of laser range readings                         */
int getLaser(…)
{
```

Appendix G (Continued)

```
        //read echo of laser command being performed
        //fillBuffer(…)
        //if echo does not equal request (MS)
                return(-1);

        //read status
        //fillBuffer(…)
        //if laser is in a failure state (i.e. data packet is NOT "99b…" or "00P"
        {
                //if attempting to recover from error (i.e. data packet is "2…")
                {
                        //display "Laser: Possible Error Detected...checking\n";
                        //return -2;
                }
                //if recovered from previous error state (i.e. data packet is "98…"
                {
                        // display "Laser Diagnosis Complete\n";
                        //return -3;
                }

                //display error message
                //fflush(stdout);
                //return -1;
        }

        //read timestamp and sum
        //fillBuffer(…)

        //while (1)
        {

                //read a single line of range data
                //fillBuffer(…)
                //if first character in range data packet is an end of line character
                        //return(1);
                //else
                        // Parse data packet for laser range readings and place in output array
        }
}
```

G.6.12 signalHandler(…)

```
/*Function: Closes devices if a terminate signal is received                    */
/*Inputs:   1) Signal that signals this function should be performed            */
/*Outputs:  N/A                                                                 */
static void signalHandler(…)
```

Appendix G (Continued)

```
{
        //reset_laser();
        //close file descriptor for GPS device
        //exit(0);
}
```

G.6.13 main( )

```
/*Function: Main function for the laser process                    */
/*Inputs:       N/A                                                */
/*Outputs:      N/A                                                */
int main( )
{
        //initialize laser array to zeros

        //if (connectLaser() < 0)
                //return (-1);

        //create shared memory
        //attach shared memory to this process

        //create a single semaphore with no semaphore flags

        // install signal handler for manual break

        //decrement semaphore
        //fill shared memory with laser array values
        //increment semaphore;

        //resetLaser();
        //sleep(2);
        //turnOnLaser();

        //startMS();

        //while (1)
        {
                //getLaser(…);

                //if return value > 0
                {
                        //decrement semaphore
                        //fill shared memory with laser array values
                        //increment semaphore;
                //else if return value is -1
                {
```

Appendix G (Continued)

```
                        //display "Attempting to reset laser\n";
                        //if (reset_laser() == -1)
                        {
                                //sleep(1);
                                //if (reset_laser() == -1)
                                {
                                        //get_version();
                                        //get_specifications();
                                        //get_state();

                                        //decrement semaphore
                                        //fill shared memory with zero values
                                        //increment semaphore;

                                        //display "Laser appears to be dead...reattempt in 5
                                        //seconds\n";

                                        //turn_off_laser();

                                        //sleep(3);
                                }
                        }
                        //sleep(2);
                        //turn_on_laser();
                        //sleep(1);
                }
                //sleep for 90% of time between data outputs
        }
        exit(0);
}
```

G.7 Imu.c

```
//include m3dmgErrors.h
//include m3dmgSerial.h
//include m3dmgAdapter.h
//include m3dmgUtils.h

//include src/Inc.h
//include src/Inc.c
```

Appendix G (Continued)

G.7.1 signalHandler(…)

```
/*Function: Closes devices if a terminate signal is received                    */
/*Inputs:       1) Signal to be used to cause this function to run               */
/*Outputs:      N/A                                                              */
static void signalHandler(int signal)
{
        //m3dmg_closeDevice(…);
        //exit(0);
}
```

G.7.2 main(…)

```
/*Function:     Main function for the IMU process                                */
/*Inputs:       N/A                                                              */
/*Outputs:      N/A                                                              */
int main()
{

        /* open device */
        //portNum = m3dmg_openPort(…);
        //if open fails
        {
                //display error
                //return -1;
        }

        /*map device*/
        //m3dmg_mapDevice(…);
        //if mapping fails
        {
                //display error
                //return -1;
        }

        /* install signal handler */
        //signal(…);

        //create shared memory
        //attach shared memory to this process

        //create a single semaphore with no flags

        //getCurrentTime(…);
        //decrement semaphore
```

Appendix G (Continued)

```
	//initialize imu data to default values
	//increment semaphore;

	//while (1)
	{
		//m3dmgGetEverything(…);
		//if error
			//display error message

		//getCurrentTime(…);

		//decrement semaphore
		//update shared memory
		//increment semaphore;
	}
	//exit(0);
}
```

G.8 m3dmgAdapter.h

```
/*add a function declaration to this manufacture supplied source code file*/
//function declaration for m3dmgGetEverything(…)
```

G.9 m3dmgAdapter.c

G.9.1 m3dmgGetEverything(…)

```
/*Function:	Request Euler angles, angular rates, and accelerations	*/
/*Inputs:	1) IMU file descriptor	*/
/*		2) Euler angles array	*/
/*		3) Accelerations array	*/
/*		4) Angular rates array	*/
/*Outputs:	1) Error Code	*/
int m3dmgGetEverything(…)
{

	/*request MSG 49 from IMU*/
	//m3dmg_sendCommand(…);
	//if error code returned
		//return status;
	//else
	{
```

Appendix G (Continued)

```
                //fill Euler angles array with request response//
                //fill Euler angles array with request response//
                //fill Euler angles array with request response//
                //return M3D_OK;
        }
}
```

G.10 Servo.c

```
//include " Robot_defs.h"
//include "Inc.h"
```

G.10.1 sigintHandler(…)

```
/*Function:     Closes devices if a terminate signal is received                */
/*Inputs:       1) Signal to be used to cause this function to run              */
/*Outputs:      N/A                                                             */
void sigintHandler(…)
{
        //close all open file descriptors
        //exit(0);
}
```

G.10.2 createSharedMem( )

```
/* Function:    Connects to shared memory create for the control, PnT, and stat_struct  */
/*              structures                                                              */
/* Inputs:      N/A                                                                     */
/* Outputs:     N/A                                                                     */
int createSharedMem( )
{

        //create control structure shared memory
        //create PnT structure shared memory
        //create stat_struct shared memory

        //attach shared memory to this process

        //create control structure semaphore
        //create PnT structure semaphore
        //create stat_struct semaphore
```

```
        //return 0;
}
```

G.10.3 getControllServos(…)

```
/* Function:    Retrieves the most up-to-date position request for the helicopter's servos */
/* Inputs:      N/A                                                                        */
/* Outputs:     1) left servo pulse width                                                  */
/*              2) right servo pulse width                                                 */
/*              3) front servo pulse width                                                 */
/*              4) tail servo pulse width                                                  */
/*              5) gyro pulse width                                                        */
/*              6) throttle pulse width                                                    */
void getControllServos()
{
        //decrement control semaphore
        //fill output data with shared memory data
        //increment control semaphore
}
```

G.10.4 getPanTiltServos(…)

```
/* Function:    Retrieves the most up-to-date position request for the pan/tilt servos    */
/* Inputs:      N/A                                                                        */
/* Outputs:     1) panning servo pulse width                                               */
/*              2) tilting servo pulse width                                               */
void getPanTiltServos(…)
{
        //decrement PnT semaphore
        //fill output data with shared memory data
        //increment PnT semaphore
}
```

G.10.5 putStatus(…)

```
/* Function:    Retrieves the most up-to-date position request for the pan/tilt servos    */
/* Inputs:      1) indicator of the safety switch position (human/computer control)        */
/* Outputs:     N/A                                                                        */
void putStatus(…)
{
        //decrement stat_struct semaphore
        //update shared memory with input data
```

Appendix G (Continued)

```
	//increment stat_struct semaphore;
}
```

G.10.6 statusMessage(…)

```
/* Function:	Partially parses status messages received from the SSC		*/
/* Inputs:	1) data string received from SSC				*/
/* Outputs:	N/A								*/
void statusMessage(…)
{

	/* Use bits 1&2 to determine state*/
	//switch (…)
	{
		//case 0:
			//signal for switch position is not available
			//break;
		//case 1:
			//switch is in position one
			//break;
		//case 2:
			//switch is in position 1.5
			//break;
		//case 3:
			//switch is in position 2
			//break;
	}
}
```

G.10.7 messageID(…)

```
/* Function:	Determines type of message received from the SSC		*/
/* Inputs:	1) message ID					*/
/*		2) data string					*/
/* Outputs:	N/A						*/
void messageID(…)
{
	/*determine type of message*/
	//switch (…)
	{
		//case 10:
			//statusMessage(…);
			//break;
		//default:
```

Appendix G (Continued)

```
                        //display "Message type not supported"
                        //break;
        }
}
```

G.10.8 main( )

```
/* Function:    Main function for the servo process                    */
/* Inputs:      N/A                                                    */
/* Outputs:     N/A                                                    */
int main()
{

        /*create the shared memory*/
        //createSharedMem()
        //if shared memory fails
        {
                //display error message
                //return -1;
        }

        /*setup connection to SSC*/
        //connectHeliServo();

        /*install signal handler*/
        //signal(…);

        //decrement control semaphore
        //fill shared memory with default data
        //increment control semaphore;

        //decrement PnT semaphore
        //fill shared memory with default data
        //increment PnT semaphore;

        //decrement stat_struct semaphore
        //fill shared memory with default data
        //increment stat_struct semaphore;

        //getCurrentTime(…);

        //while(1)
        {
                //do
                {
                        //usleep(100);
```

```
                        //getCurrentTime(…);
                        //timeDifference(…);
                }//while time passed since last update is not approximately 100Hz

                //getControllServos(…);
                //getPanTiltServos(…);

                //heliDriveCommand(…);
                //getCurrentTime(…);

                //check for input from SSC
                //if input is available
                {
                        //read input data
                        //parse out message ID
                        //parse out data string
                        //messageID(…);
                }
        }

        //return 0;
}
```

G.11 Collect.c

```
//include "src/Inc.h"
```

G.11.1 sigintHandler(…)

```
/*Function:     Closes devices if a terminate signal is received        */
/*Inputs:       1) Signal to be used to cause this function to run      */
/*Outputs:      N/A                                                     */
void sigintHandler(int sig_num)
{
        //close all open file descriptors
        //exit(0);
}
```

G.11.2 createSharedMem( )

```
/*Function:     Creates shared memory locations for the store_data and state_struct  */
/*              structures                                                          */
```

Appendix G (Continued)

```
/*Inputs:       N/A                                                    */
/*Outputs:      N/A                                                    */
int createSharedMem ( )
{
        // create shared memory for store_data structure
        // create shared memory for state_struct structure

        //attach shared memory to this process

        //create a single semaphore with no flags for store_data
        //create a single semaphore with no flags for state_struct

        //return 0;
}
```

G.11.3 getData(…)

```
/*Function:     Retrieves information stored in the store_data shared memory structure  */
/*Inputs:       N/A                                                                     */
/*Outputs:      1) Array of fuzzy controller output                                     */
/*              2) Array of fuzzy controller input (collective controller)              */
/*              3) Array of fuzzy controller input (yaw controller)                     */
/*              4) Array of fuzzy controller input (roll controller)                    */
/*              5) Array of fuzzy controller input (pitch controller)                   */
/*              6) Array of position data (latitude, longitude, altitude)               */
void getData(…)
{
        //decrement semaphore
        //place data in shared memory into appropriate array
        //increment semaphore;
}
```

G.11.4 getStat(…)

```
/*Function:     Retrieves information stored in the stat_struct shared memory structure  */
/*Inputs:       N/A                                                                      */
/*Outputs:      1) variable describing position of safety switch (computer/human control) */
void getStat(…)
{
        //decrement semaphore
        //place data in shared memory into output variable
        //increment semaphore;
}
```

Appendix G (Continued)

G.11.5 main( )

```
/*Function:     Main function for the collect data process                    */
/*Inputs:       N/A                                                           */
/*Outputs:      N/A                                                           */
int main()
{
        /*install signal handler for manual break*/
        //signal(SIGINT, sigintHandler);

        //createSharedMem()
        //if shared memory fails
        {
                //display error message
                //return -1;
        }

        //open file to store data

        //while(1)
        {
                //getData(…);
                //getStat(…);

                //output collected data to file

                /*force a maximum output rate*/
                //usleep(…);
        }
}
```

G.12 Pan_tilt.c

```
//include "Robot_defs.h"
```

G.12.1 sigintHandler(…)

```
/* Function:     Closes devices if a terminate signal is received              */
/* Inputs:       1) Signal to be used to cause this function to run            */
/* Outputs:      N/A                                                           */
void sigintHandler(int sig_num)
{
        //close all open file descriptors
```

```
        //exit(0);
}
```

G.12.2 main( )

```
/* Function:    Main function for the pan/tilt process                    */
/* Inputs:      N/A                                                       */
/* Outputs:     N/A                                                       */
int main( )
{

        //connect to shared memory for PnT structure
        //connect to semaphore for PnT structure

        /*install signal handler*/
        //signal(…);

        //while(1)
        {
                //calculate pan position
                //calculate tilt position

                //decrement semaphore
                //update shared memory
                //increment semaphore

                /*sleep for desired update rate to pan/tilt*/
                //usleep(…);
        }
}
```

G.13 Data_test.c

```
//include Inc.h
```

G.13.1 createSharedMem( )

```
/* Function:    connects to shared memory for the gps, imu, and laser structures  */
/* Inputs:      N/A                                                               */
/* Outputs:     N/A                                                               */
int createSharedMem( )
{
```

```
        //connect to shared memory for gps
        //connect to shared memory for imu
        //connect to shared memory for laser
        //attach shared memory to this process

        //connect to semaphore for gps
        //connect to semaphore for imu
        //connect to semaphore for laser

        //return (0);
}
```

G.13.2 getLaser(…)

```
/* Function:    Retrieves laser data from shared memory                    */
/* Inputs:      N/A                                                        */
/* Outputs:     1) Array of laser range data                               */
void getLaser(…)
{
        //decrement laser semaphore
        //fill output array with data from shared memory
        //increment laser semaphore;
}
```

G.13.3 getGPS(…)

```
/* Function:    Retrieves gps data from shared memory                      */
/* Inputs:      N/A                                                        */
/* Outputs:     1) Array of position data (latitude, longitude, latitude direction, longitude*/
/*                 direction, type of lock, number of satellites being tracked, and altitude)*/
void getGPS(…)
{
        //decrement gps semaphore
        //fill output array with data from shared memory
        //increment gps semaphore;
}
```

G.13.4 getIMU(…)

```
/* Function:    Retrieves imu data from shared memory                      */
/* Inputs:      N/A                                                        */
/* Outputs:     1) Array of Euler angles                                   */
/*              2) Array of accelerations                                  */
```

Appendix G (Continued)

```
/*              3) Array of angular rate                                    */
void getIMU(…)
{
        //decrement imu semaphore
        //fill output arrays with data from shared memory
        //increment imu semaphore;
}
```

G.13.5 main( )

```
/* Function:    Main function for the data_test process                    */
/* Inputs:      N/A                                                        */
/* Outputs:     N/A                                                        */
int main( )
{

        //createSharedMem()
        //if shared memory fails
        {
                //display error message
                //return -1;
        }

        //while(1)
        {
                //getGPS(…);
                //getIMU(…);
                //getLaser(…);

                //average laser readings requested

                //display laser average
                //display position information
                //display imu information

                //sleep for desired output rate
                //usleep(…);
        }
}
```

G.14 Navigate.c

```
//include " Robot_defs.h"
//include " Inc.h"
```

Appendix G (Continued)

```
//include " fis.c"
//include " navigate.h"
```

G.14.1 sigintHandler(…)

```
/* Function:     Attempts to shutdown devices properly if process termination is   */
/*               requested                                                         */
/* Inputs:       1) signal that tells this function to operate                     */
/* Outputs:      N/A                                                               */
void sigintHandler(…)
{
        /*free the four fis nodes created for the four fuzzy controllers*/
        //fisFreeFisNode(..);
        //fisFreeFisNode(..);
        //fisFreeFisNode(..);
        //fisFreeFisNode(..);

        /*free the four fis matrices created for the four fuzzy controllers*/

        //fisFreeMatrix(…);
        //fisFreeMatrix(…);
        //fisFreeMatrix(…);
        //fisFreeMatrix(…);

        //close all open sockets
        //exit(0);
}
```

G.14.2 createSharedMem( )

```
/* Function:     Connects to shared memory locations for gps, imu, control, store_data,  */
/*               stat_struct, and laser structures                                       */
/* Inputs:       N/A                                                                     */
/* Outputs:      N/A                                                                     */
int createSharedMem( )
{
        //connect to gps shared memory
        //connect to imu shared memory
        //connect to control shared memory
        //connect to store_data shared memory
        //connect to stat_struct shared memory
        //connect to laser shared memory

        //attach all shared memory to this process
```

```
        //connect to gps semaphore
        //connect to imu semaphore
        //connect to control semaphore
        //connect to store_data semaphore
        //connect to stat_struct semaphore
        //connect to laser semaphore

        //return (0);
}
```

G.14.3 getGPS(…)

```
/* Function:    Retrieves gps data from shared memory                                    */
/* Inputs:      N/A                                                                      */
/* Outputs:     1) Array of position data (latitude, longitude, latitude direction, longitude */
/*                 direction, type of lock, number of satellites being tracked, and altitude)*/
/*              2) variable describing if this is new gps data                           */
void getGPS(…)
{
        //decrement gps semaphore
        //fill output array with data from shared memory
        //increment gps semaphore;

        //if count variable in shared memory has changed since last function call
                //set new gps variable to true
        //else
                //set new gps variable to false
}
```

G.14.4 getIMU(…)

```
/* Function:    Retrieves imu data from shared memory                                    */
/* Inputs:      N/A                                                                      */
/* Outputs:     1) Array of Euler angles                                                 */
/*              2) Array of accelerations                                                */
/*              3) Array of angular rate                                                 */
void getIMU(…)
{
        //decrement imu semaphore
        //fill output arrays with data from shared memory
        //increment imu semaphore;
}
```

G.14.5 getStat(…)

```
/* Function:    Retrieves position of the safety switch (human/computer control)   */
/* Inputs:      N/A                                                                 */
/* Outputs:     1) variable describing the location of the safety switch            */
void getStat(int *stat)
{
        //decrement semaphore
        //fill output variable with data from shared memory
        //increment semaphore
}
```

G.14.6 getLaser(…)

```
/* Function:    Retrieves laser data from shared memory                             */
/* Inputs:      N/A                                                                 */
/* Outputs:     1) Array of laser range data                                        */
void getLaser(…)
{
        //decrement laser semaphore
        //fill output array with data from shared memory
        //increment laser semaphore;
}
```

G.14.7 putControllServos(…)

```
/* Function:    Updates control shared memory with new servo pulse width values     */
/* Inputs:      1) left servo's pulse width                                         */
/*              2) right servo's pulse width                                        */
/*              3) front servo's pulse width                                        */
/*              4) tail servo's pulse width                                         */
/*              5) gyro's pulse width                                               */
/*              6) throttle's pulse width                                           */
/* Outputs:     N/A                                                                 */
void putControllServos(…)
{
        //decrement control semaphore
        //update control shared memory with new pulse width values
        //increment control semaphore;
}
```

G.14.8 storeData(…)

```
/* Function:    Updates store_data shared memory with new data                */
/* Inputs:      1) Array of fuzzy controller outputs                          */
/*              2) Array of fuzzy controller inputs (collective controller)   */
/*              3) Array of fuzzy controller inputs (yaw controller)          */
/*              4) Array of fuzzy controller inputs (roll controller)         */
/*              5) Array of fuzzy controller inputs (pitch controller)        */
/*              6) Array of position data (latitude, longitude, altitude)     */
/* Outputs:     N/A                                                           */
void storeData(…)
{
        //decrement store_data semaphore
        //update store_data shared memory with new values
        //increment store_data semaphore;
}
```

G.14.9 setWorldRotationMatrices(…)

```
/* Function:    Sets up three matrices to rotate data from the local coordinate frame to */
*/              the world coordinate frame                                    */
/* Inputs:      1) Array of angles which will be rotated by                   */
/* Outputs:     1) 2D Array for rotation about X axis                         */
/*              2) 2D Array for rotation about Z axis                         */
/*              3) 2D Array for rotation about Y axis                         */
void setWorldRotationMatrices(…)
{
        /*setup rotation matrix for X axis*/
        /*setup rotation matrix for Z axis*/
        /*setup rotation matrix for Y axis*/
}
```

G.14.10 setLocalRotationMatrices(…)

```
/* Function:    Sets up three matrices to rotate data from the world coordinate frame to */
*/              the local coordinate frame                                    */
/* Inputs:      1) Array of angles which will be rotated by                   */
/* Outputs:     1) 2D Array for rotation about X axis                         */
/*              2) 2D Array for rotation about Z axis                         */
/*              3) 2D Array for rotation about Y axis                         */
void setLocalRotationMatrices(…)
{
        /*setup rotation matrix for X axis*/
```

Appendix G (Continued)

```
        /*setup rotation matrix for Z axis*/
        /*setup rotation matrix for Y axis*/
}
```

G.14.11 multMatrices(…)

```
/* Function:    Multiplies a 3x3 matrix with a 3x1 matrix                    */
/* Inputs:      1) 2D Array used for 3x3 matrix                              */
/*              2) Array used for 3x1 matrix                                 */
/* Outputs:     1) Solution array for multiplying inputs 1 and 2             */
void multMatrices(…)
{
        //multiply inputs one and two together
        //set output array to solution
}
```

G.14.12 translatePosition(…)

```
/* Function:    Translates the latitude and longitude position of the GPS antenna to the  */
/*              center (main shaft) of the helicopter.  This is done as the GPS antenna is */
/*              mounted on the tail boom of the helicopter.  Translation vaules are stored*/
/*              external to this function and after being initialized are only modified by */
/*              this function.                                                             */
/* Inputs:      1) Latitude position provided by the GPS process                           */
/*              2) Longitude position provided by the GPS process                          */
/*              3) Array of Euler angles provided by the IMU process                       */
/*              4) New GPS variable (is the position data being processed for the first    */
/*                 time                                                                    */
/* Outputs:     1) N/A                                                                     */
void translatePosition(…)
{
        /*rotate the position offset of GPS antenna from the local frame to the world frame*/
        /*offset should only have a value on the Y axis…i.e. [2.4, 0, 0] for a 2.4 ft offset  */
        //setWorldRotationMatrices(...);
        //multMatrices(…);

        //negate the lateral offset

        //divide lateral offset by the lateral resolution and round to nearest int value
        //divide longitudinal offset by the longitudinal resolution and round to nearest int value

        //divide both the lateral and longitudinal offset by one million

        //if this is the intinal run of this function
```

Appendix G (Continued)

```
{
        //initialize a FIFO of lateral and longitudinal GPS offsets differences to zero
        //FIFO size should be equal to the rate of the GPS

        //set the old lateral offset = lateral offset
        //set the old longitudinal offset = longitudinal offset

        //set global lateral change = lateral offset
        //set global longitudinal change = longitudinal offset

        //set old latitude =  current latitude
        //set old longitude = current longitude

        //set current latitude = current latitude – global latitude change
        //set current longitude = current longitude – global longitude change

        //return;
}

//if we are not using new GPS data
{
        //set current latitude = current latitude – global latitude change
        //set current longitude = current longitude – global longitude change
        //return;
}

//pop lateral offset difference from FIFO and add to global lateral change
//pop longitudinal offset difference from FIFO and add to global longitudinal change

//set current latidude offset difference = old lateral offset – lateral offset
//set current lontitudinal offset difference = old longitude offset – longitude offset

//push current lateral offset difference onto FIFO
//push current longitudinal offset difference onto FIFO

//set latitude point difference = current latitude – old latitude
//set longitude point difference = current longitude – old longitude

//if current latitude > old latitude
{
        //for each element in the latitude FIFO
        {
                //if (FIFO value > 0)
                {
                        //if (FIFO value == latitude point difference)
                        {
                                //global latitude change+= FIFO value;
```

Appendix G (Continued)

```
                                        //FIFO value = 0
                                        //break out of loop
                                }
                                //else if (FIFO value < latitude point difference)
                                {
                                        //latitude point difference-=FIFO value
                                        //global latitude change += FIFO value
                                        //FIFO value = 0
                                }
                                //else
                                {
                                        //global latitude change+= latitude point difference
                                        //FIFO value = FIFO value-latitude difference
                                        //break out of loop
                                }
                        }
                }
        }
        //else if current latitude < old latitude
        {
                //for each element in the latitude FIFO
                {
                        //if (FIFO value < 0)
                        {
                                //if (FIFO value == latitude point difference)
                                {
                                        //global latitude change+= FIFO value;
                                        //FIFO value = 0
                                        //break out of loop
                                }
                                //else if (FIFO value > latitude point difference)
                                {
                                        //latitude point difference-=FIFO value
                                        //global latitude change += FIFO value
                                        //FIFO value = 0
                                }
                                //else
                                {
                                        //global latitude change+= latitude point difference
                                        //FIFO value = FIFO value-latitude difference
                                        //break out of loop
                                }
                        }
                }
        }

        //if current longitude < old longitude
```

Appendix G (Continued)

```
{
        //for each element in the longitude FIFO
        {
                //if (FIFO value < 0)
                {
                        //if (FIFO value == longitude point difference)
                        {
                                //global longitude change+= FIFO value;
                                //FIFO value = 0
                                //break out of loop
                        }
                        //else if (FIFO value > longitude point difference)
                        {
                                //longitude point difference-=FIFO value
                                //global longitude change += FIFO value
                                //FIFO value = 0
                        }
                        //else
                        {
                                //global longitude change+= longitude point difference
                                //FIFO value = FIFO value-longitude difference
                                //break out of loop
                        }
                }
        }
}
//else if current longitude > old longitude
{
        //for each element in the longitude FIFO
        {
                //if (FIFO value > 0)
                {
                        //if (FIFO value == longitude point difference)
                        {
                                //global longitude change+= FIFO value;
                                //FIFO value = 0
                                //break out of loop
                        }
                        //else if (FIFO value < longitude point difference)
                        {
                                //longitude point difference-=FIFO value
                                //global longitude change += FIFO value
                                //FIFO value = 0
                        }
                        //else
                        {
                                //global longitude change+= longitude point difference
```

Appendix G (Continued)

```
                                        //FIFO value = FIFO value-longitude difference
                                        //break out of loop
                                }
                        }
                }
        }

        //set the old lateral offset = the lateral offset
        //set the old longitudinal offset = the longitudinal offset

        //set old latitude =  current latitude
        //set old longitude = current longitude

        //set current latitude = current latitude – global latitude change
        //set current longitude = current longitude – global longitude change
}
```

G.14.13 translateVelocity(…)

```
/* Function:    Modifies the GPS latitude and longitude to account for movments that   */
/*              may have been caused by heading changes.  The changes are not stored   */
/*              and are recalculated every time.                                       */
/* Inputs:      1) Latitude position provided by the GPS process                       */
/*              2) Longitude position provided by the GPS process                      */
/*              3) Array of Euler angles provided by the IMU process                   */
/*              4) New GPS variable (is the position data being processed for the first */
/*                 time                                                                */
/* Outputs:     1) N/A                                                                 */
void translateVelocity(…)
{
        /*rotate the position offset of GPS antenna from the local frame to the world frame*/
        /*offset should only have a value on the Y axis…i.e. [2.4, 0, 0] for a 2.4 ft offset  */
        //setWorldRotationMatrices(...);
        //multMatrices(…);
        //negate the lateral offset
        //divide lateral offset by the lateral resolution and round to nearest int value
        //divide longitudinal offset by the longitudinal resolution and round to nearest int value

        //divide both the lateral and longitudinal offset by one million

        //if this is the intinal run of this function
        {
                //initialize a FIFO of lateral and longitudinal GPS offsets differences to zero
                //FIFO size should be equal to the rate of the GPS
```

Appendix G (Continued)

```
                //set the old lateral offset = the lateral offset
                //set the old longitudinal offset = the longitudinal offset

                //set old latitude =  current latitude
                //set old longitude = current longitude

                //return;
        }

        //if we are not using new GPS data
        {
                //return;
        }

        //pop lateral offset difference from FIFO
        //pop longitudinal offset difference from FIFO

        //set current latidude offset difference = old lateral offset – lateral offset
        //set current lontitudinal offset difference = old longitude offset – longitude offset

        //push current lateral offset difference onto FIFO
        //push current longitudinal offset difference onto FIFO

        //set latitude point difference = current latitude – old latitude
        //set longitude point difference = current longitude – old longitude

        //if current latitude > old latitude
        {
                //set latitude change to zero
                //for each element in the latitude FIFO
                {
                        //if (FIFO value > 0)
                        {
                                //if (FIFO value == latitude point difference)
                                {
                                        //latitude change+= FIFO value;
                                        //FIFO value = 0
                                        //break out of loop
                                }
                                //else if (FIFO value < latitude point difference)
                                {
                                        //latitude point difference-=FIFO value
                                        //latitude change += FIFO value
                                        //FIFO value = 0
                                }
                                //else
                                {
```

```
                                        //latitude change+= latitude point difference
                                        //FIFO value = FIFO value-latitude difference
                                        //break out of loop
                                }
                        }
                }
        }
        //else if current latitude < old latitude
        {
                //set latitude change to zero
                //for each element in the latitude FIFO
                {
                        //if (FIFO value < 0)
                        {
                                //if (FIFO value == latitude point difference)
                                {
                                        //latitude change+= FIFO value;
                                        //FIFO value = 0
                                        //break out of loop
                                }
                                //else if (FIFO value > latitude point difference)
                                {
                                        //latitude point difference-=FIFO value
                                        //latitude change += FIFO value
                                        //FIFO value = 0
                                }
                                //else
                                {
                                        //latitude change+= latitude point difference
                                        //FIFO value = FIFO value-latitude difference
                                        //break out of loop
                                }
                        }
                }
        }

        //if current longitude < old longitude
        {
                //set longitude change to zero
                //for each element in the longitude FIFO
                {
                        //if (FIFO value < 0)
                        {
                                //if (FIFO value == longitude point difference)
                                {
                                        //longitude change+= FIFO value;
                                        //FIFO value = 0
```

```
                                        //break out of loop
                                }
                                //else if (FIFO value > longitude point difference)
                                {
                                        //longitude point difference-=FIFO value
                                        //longitude change += FIFO value
                                        //FIFO value = 0
                                }
                                //else
                                {
                                        //longitude change+= longitude point difference
                                        //FIFO value = FIFO value-longitude difference
                                        //break out of loop
                                }
                        }
                }
        }
        //else if current longitude > old longitude
        {
                //set longitude change to zero
                //for each element in the longitude FIFO
                {
                        //if (FIFO value > 0)
                        {
                                //if (FIFO value == longitude point difference)
                                {
                                        //longitude change+= FIFO value;
                                        //FIFO value = 0
                                        //break out of loop
                                }
                                //else if (FIFO value < longitude point difference)
                                {
                                        //longitude point difference-=FIFO value
                                        //longitude change += FIFO value
                                        //FIFO value = 0
                                }
                                //else
                                {
                                        //longitude change+= longitude point difference
                                        //FIFO value = FIFO value-longitude difference
                                        //break out of loop
                                }
                        }
                }
        }
```

Appendix G (Continued)

```
        //set the old lateral offset = the lateral offset
        //set the old longitudinal offset = the longitudinal offset

        //set old latitude =  current latitude
        //set old longitude = current longitude

        //set current latitude = current latitude – latitude change
        //set current longitude = current longitude – longitude change
}
```

G.14.14 gpsRollPitchError(…)

```
/* Function:    Calculates the lateral and longitudinal offset from a desired path in the   */
/*              local coordinate frame.                                                      */
/* Inputs:      1) current latitude (value not reference)                                    */
/*              2) current longitude (value not reference)                                   */
/*              3) current goal's latitude                                                   */
/*              4) current goal's longitude                                                  */
/*              5) previous goal's latitude                                                  */
/*              6) previous goal's longitude                                                 */
/*              7) Array of Euler angles                                                     */
/*              8) current altitude                                                          */
/* Outputs:     1) lateral offset                                                            */
/*              2) longitudinal offset                                                       */
void gpsRollPitchError(…)
{

        //translatePosition(…)

        //Use the world model calculation to determine the distance, in feet, between the
          current lat, lon and the previous goal's lat, lon//

        //Use the world model calculation to determine the distance, in feet, between the
          current lat, lon and the current goal's lat, lon//

        //if the straight line path between the current and previous goal intersects with a
          predetermined circle around the current lat,lon//
        {
                //calculate the intersection points of the path with the circle
                //determine which intersection point is closer to goal
                //calculate distance vector to intersection point
                //rotate distance vector to local coordinate frame
                //set components of rotated distance vector to outputs
        }
        else
        {
```

Appendix G (Continued)

```
                //calculate distance vector from current lat, lon to current goal's lat, lon
                //rotate distance vector to local coordinate frame
                //set components of rotated distance vector to outputs
        }
}
```

G.14.15 calcYawError(…)

```
/* Function:    Calculates the heading offset                                   */
/* Inputs:      1) current heading                                              */
/*              2) desired heading                                              */
/* Outputs:     1) heading offset                                               */
float calcYawError(…)
{
        /*calculate offset*/
        //subtract desired heading from current heading

        //if offset is > 180
                //subtract 360 from offset
        //else if offset is < -180
                //add 360 to offset

        //set offset to output
}
```

G.14.16 gpsVelocity(…)

```
/* Function:    Attempts to use the last one seconds worth of GPS data to increase the */
/*              accuracy of the velocity calculated only using two consecutive GPS     */
/*              readings                                                               */
/* Inputs:      1) Array representing the current velocity vector                      */
/* Outputs:     1) Updated velocity vector                                             */
void gpsVelocity(...)
{

        //if this is the first run of this function
        {
                //initialize old gps velocities to zero
        }


        /*Attempt to update the lateral component of the velocity vector.*/
        /*Use old velocity calculations to reduce the error due to resolution until*/
        /*all stored data has been used or the calculation violates the threshold of error*/
```

Appendix G (Continued)

```
/*Start with the most recent data and work backwards*/
//for up to one seconds worth of velocity inputs and while range is not broken
{
        //calculate the possible lateral error due to GPS resolution
        //calculate a valid range for the velocity element (current velocity±error)

        //using one extra reading calculate a temp velocity for the lateral element

        //if the temp velocity element is not in the valid range
        {
                //range is broken
                //if temp velocity element < current velocity element
                        //set current velocity element to low end of valid range
                //else
                        //set current velocity element to high end of valid range
        }
        //else
        {
                //set current velocity element to temp velocity element
        }
}

/*Attempt to update the longitudinal component of the velocity vector*/
/*use old velocity calculations to reduce the error due to resolution until*/
/*all stored data has been used or the calculation violates the threshold of error*/
/*Start with the most recent data and work backwards*/
//for up to one seconds worth of velocity inputs and while range is not broken
{
        //calculate the possible longitudinal error due GPS to resolution
        //calculate a valid range for the velocity element (current velocity±error)

        //using one extra reading calculate a temp velocity for the longitudinal element

        //if the temp velocity element is not in the valid range
        {
                //range is broken
                //if temp velocity < current velocity
                        //set current velocity element to low end of valid range
                //else
                        //set current velocity element to high end of valid range
        }
        //else
        {
                //set current velocity element to temp velocity element
        }
}
```

Appendix G (Continued)

```
        /*Attempt to update the vertical component of the velocity vector*/
        /*use old velocity calculations to reduce the error due to resolution until*/
        /*all stored data has been used or the calculation violates the threshold of error*/
        /*Start with the most recent data and work backwards*/
        //for up to one seconds worth of velocity inputs and while range is not broken
        {
                //calculate the possible vertical error due to GPS resolution
                //calculate a valid range for the velocity element (current velocity±error)

                //using one extra reading calculate a temp velocity for the vertical element

                //if the temp velocity element is not in the valid range
                {
                        //range is broken
                        //if temp velocity < current velocity
                                //set current velocity element to low end of valid range
                        //else
                                //set current velocity element to high end of valid range
                }
                //else
                {
                        //set current velocity element to temp velocity element
                }
        }
}
```

G.14.17 calcSlope(…)

```
/* Function:    Attempts to estimate the drift, or slope of the error, in the integrated   */
/*              velocity calculations                                                       */
/* Inputs:      1) Array representing the GPS velocity vector                               */
/*              2) Array representing the current IMU calculated velocity vector            */
/*              3) Array representing the previously calculate slope vector                 */
/*              4) Array representing the Euler angles                                      */
/* Outputs:     1) Updated slope vector                                                     */
void calcSlope(…)
{
        /*Once the slope has converged it should be relatively constant…thus update the*/
        /*Kalman filter to allow for slower change after takeoff*/
        //if takeoff is complete and all filters are not finished updating
        {
                //set updating to finished
                //if Kalman for element one is less then desired
                {
                        //increment Kalman one (R[1] + a constant)
```

Appendix G (Continued)

```
                        //set updating to not finished
                }

                //if Kalman for element two is less then desired
                {
                        //increment Kalman two (R[2] + a constant)
                        //set updating to not finished
                }

                //if Kalman for element three is less then desired
                {
                        //increment Kalman three (R[3] + a constant)
                        //set updating to not finished
                }
        }

        //if this is the first run of this function
        {
                /*setup array to store one seconds worth of integrated velocities*/
                //initialize array of old integrated velocities to zero

                /*setup array to store one seconds worth of offsets between velocities*/
                //initialize array of old offsets to zero
        }

        //store the integrated velocity input into old velocity array

        /*calculate offset of the two velocities for each component…note that GPS is*/
        /*assumed to have a one second delay*/
        //GPS velocity minus integrated velocity from one second ago (element one)
        //GPS velocity minus integrated velocity from one second ago (element two)
        //GPS velocity minus integrated velocity from one second ago (element three)

        /*calculate the slope of error based on the current offset and the offset from one*/
        /*second ago*/
        //current offset minus offset from one second ago (element one)
        //current offset minus offset from one second ago (element two)
        //current offset minus offset from one second ago (element three)

        /*rotate elements one and three to the local coordinate frame about the vertical axis*/
        /*…use zero value for element two*/
        //setLocalRotationMatrices(...);

        /*only multiply the matrix to rotate about the vertical axis*/
        //multMatrices(…);

        /*update slope values*/
```

Appendix G (Continued)

```
        //slope element one = input slope element one + rotated element one
        //slope element two = input slope element two + slope of error for element two
        //slope element one = input slope element three + rotated element three

        //store calculated offset into old offset array

       //Combine input slope with newly calculated slope using Kalman filter (element one)
       //Combine input slope with newly calculated slope using Kalman filter (element two)
       //Combine input slope with newly calculated slope using Kalman filter (element three)

        //set output array to values of Kalman filtered slopes
}
```

G.14.18 calcVelocity(…)

```
/* Function:    Calculates the velocity for the current time step                    */
/* Inputs:      1) current latitude (value not reference)                            */
/*              2) current longitude (value not reference)                           */
/*              3) previous latitude                                                 */
/*              4) previous longitude                                                */
/*              5) current altitude                                                  */
/*              6) previous altitude                                                 */
/*              7) Array of Euler angles                                             */
/*              8) IMU timestamp                                                     */
/*              9) new gps variable (first time this data has been processed by this */
/*              function                                                             */
/*              10) IMU timestamp last time this function was called                 */
/* Outputs:     1) Array representing the velocity vector                            */
void calcVelocity(…)
{
        /*calc time difference between current and old IMU timestamps*/
        //timeDifference(…);

        //if this is the first time this function has been called
        {
                //set old_gps velocity to zero
                //create array to store one second of integrated velocities
                //initialize array of velocities to zero

                //set slope vector to zero
                //set output vector to zero
                //return;
        }

        /*Kalman filters initialized to allow velocity to change quickly (i.e. low 'R' value)*/
```

Appendix G (Continued)

```
/*to allow quick compensation while the bias is converging.  After convergence */
/*the Kalman filters are updated to prevent quick velocity changes*/
//if takeoff is complete and all filters are not finished updating
{
        //set updating to finished
        //if Kalman for element one is less then desired
        {
                //increment Kalman one (R[1] + a constant)
                //set updating to not finished
        }

        //if Kalman for element two is less then desired
        {
                //increment Kalman two (R[2] + a constant)
                //set updating to not finished
        }

        //if Kalman for element three is less then desired
        {
                //increment Kalman three (R[3] + a constant)
                //set updating to not finished
        }
}

//if the gps data is new
{
        //translateVelocity(…)

        //use the world model to calculate the X, Y distances in feet
        //set vertical distance to current altitude minus old altitude

        /*create a velocity vector based on distances times the GPS frequency*/
        //velocity element one = Y distance *GPS frequency
        //velocity element two = vertical distance *GPS frequency
        //velocity element three = X distance *GPS frequency

        // gpsVelocity(…);
        // calcSlope(…);

        /*use the difference between the current GPS velocity and the velocity from*/
        /*one second ago to update to update the current integrated velocity*/
        //integrated velocity element one += Kalman gain*velocity one diff
        //integrated velocity element two += Kalman gain*velocity two diff
        //integrated velocity element three += Kalman gain*velocity three diff

        //store integrated velocity vector in old vector array
```

Appendix G (Continued)

```
		//set old gps velocity to current gps velocity for next function call
	}

	//if time difference between the old and new IMU timestamps is > zero
	{
		/*setup local rotation matrices*/
		//setLocalRotationMatrices(…);

		/*rotate global gravity vector about the lateral axis*/
		//multMatrices(…);
		/*rotate global gravity vector about longitudinal axis*/
		//multMatrices(…);
		/*setup the acceleration matrix*/
		//accelerations element one = acceleration on pitch axis;
		//accelerations element two = acceleration on vertical axis;
		//accelerations element three = acceleration on roll axis;

		//subtract gravity vector from acceleration vector

		/*setup world rotation matrices*/
		//setWorldRotationMatrices(…);

		/*rotate accelerations about the longitudinal axis*/
		//multMatrices(…);
		/*rotate accelerations about the lateral axis*/
		//multMatrices(…);

		//subtract the drift vector from the accelerations vector

		/*rotate accelerations vector about the vertical axis*/
		//multMatrices(…);

		//average current acceleration vector with old acceleration vector

		/*calculate velocity change since last IMU reading*/
		/*9.800722 = conversion from g-force to meters per second*/
		/*3.2808399 = conversion from meters to feet*/
		//change vector= averaged vector*9.800722*3.2808399*IMU time diff

		//add change vector to the integrated velocity vector
		//add change vector to IMU calculated velocity vector

		//set old accel vector to current accel vector for next function call
	}

	/*setup local rotation matrices*/
	//setLocalRotationMatrices(...);
```

Appendix G (Continued)

```
	//rotate integrated velocity vector about the vertical axis
	//multMatrices(…);

	//set output vector to rotated velocity vector
}
```

G.14.19 electricMixing(…)

```
/* Function:	Converts controller outputs to pulse width values for a three point	*/
/*		swashplate	*/
/* Inputs:	1) roll command from controller	*/
/*		2) pitch command from controller	*/
/*		3) collective command from controller	*/
/* Outputs:	1) left servo pulse width	*/
/*		2) right servo pulse width	*/
/*		3) front servo pulse width	*/
void electricMixing(…)
{

	/*setup local rotation matrices*/
	//setLocalRotationMatrices(…);

	/*rotate global integration vector about the vertical axis*/
	//multMatrices(…);

	/*compute pulse widths for the roll command*/
	//if roll command > zero
		//roll PW = (left servo's max PW – its neutral PW) * roll command
	//else
		//roll PW = (left servo's neutral PW – its minimum PW) * roll command

	//if (left's neutral PW+roll PW+roll trim+roll integration element) > left's max PW
	{
		//display "Warning: PW value exceeded Left Max\n");
		//set left servo's PW to the left servo's max PW
		//set right servo's PW = right's neutral PW+left's max PW-left's neutral PW
	}
	//else if (left's neutral PW+roll PW+roll trim+roll integration element) < left's min PW
	{
		//display "Warning: PW value exceeded Left Min\n");
		//set left servo's PW to the left servo's min PW
		//set right servo's PW = right's neutral PW-(left's neutral PW-left's min PW)
	}
	else
	{
```

Appendix G (Continued)

```
            //set left's PW = left's neutral PW+roll PW+roll trim+roll integration element
            //set right's PW = right's neutral PW+roll PW+roll trim+roll integration element
}

/*compute pulse widths for the pitch command*/
//if pitch command > zero
            //pitch PW = (front servo's max PW – its neutral PW) * pitch command
//else
            //pitch PW = (front servo's neutral PW – its minimum PW) * pitch command

//if (front's neutral PW+pitch PW+pitch trim+pitch integration elem) > front's max PW
{
            //display "Warning: PW value exceeded Front Max\n");
            //set front servo's PW to the front servo's max PW
            //set left servo's PW += (front's max PW-front's neutral PW)/2.0
            //set right servo's PW -= (front's max PW-front's neutral PW)/2.0
}
//elseif (front neutral PW+pitch PW+pitch trim+pitch integration elem) < front's min PW
{
            //set front servo's PW to the front servo's min PW
            //set left servo's PW += (front's neutral PW-front's min PW)/2.0
            //set right servo's PW -= (front's neutral PW-front's min PW)/2.0
}
//else
{
            //set front servo's PW =front's neutral+pitch PW+pitch trim+pitch integrate elem
            //set left servo's PW += (pitch PW+pitch trim+pitch integrate elem)/2.0
            //set right servo's PW -= (pitch PW+pitch trim+pitch integrate elem)/2.0
}

/*compute pulse widths for the pitch command*/
//if collective (coll) command > zero
            //coll PW = (coll's max PW – its neutral PW) * coll command
//else
            //coll PW = (coll's neutral PW – its minimum PW) * coll command

//if (coll's neutral PW+coll PW+coll trim+coll integration elem) > coll's max PW
{
            //display "Warning: PW value exceeded Collective Max\n");
            //set front servo's PW += coll's max PW- coll neutral PW
            //set left servo's PW -= coll's max PW- coll neutral PW
            //set right servo's PW += coll's max PW- coll neutral PW
}
//elseif (coll's neutral PW+coll PW+coll trim+coll integration elem) < coll's min PW
{
            //display "Warning: PW value exceeded Collective Min\n");
            //set front servo's PW -= coll's neutral PW+(coll's neutral PW- coll's min PW)
```

Appendix G (Continued)

```
                //set left servo's PW += coll's neutral PW+(coll's neutral PW- coll's min PW)
                //set right servo's PW -= coll's neutral PW+(coll's neutral PW- coll's min PW)
        }
        //else
        {
                //set front servo's PW += coll PW+coll trim+coll integration elem-coll's neutral
                //set left servo's PW -= coll PW+coll trim+coll integration elem-coll's neutral
                //set right servo's PW += coll PW+coll trim+coll integration elem-coll's neutral
        }
}
```

G.14.20 initalizeSystem( )

```
/* Function:    Sets the waypoints, sets up the fuzzy controllers, calculates global    */
/*              gravity                                                                  */
/*              vector, and initializes several process variables                        */
/* Inputs:      N/A                                                                      */
/* Outputs:     N/A                                                                      */
int initalizeSystem( )
{
        //setup global array of latitude waypoints
        //setup global array of longitude waypoints
        //setup global array of altitude waypoints
        //setup global array of heading waypoints

        //set a variable for the number of waypoints to attempt
        //set starting waypoints to first elements in waypoint arrays

        //set acceleration variants velocity array to zero

        //createSharedMem()
        //if shared memory fails
        {
                //display error message
                //return -1;
        }

        /*obtain data matrix and FIS matrix for roll, pitch, yaw, and collective controllers*/
        //roll matrix = returnFismatrix(…);
        //pitch matrix = returnFismatrix(…);
        //yaw matrix = returnFismatrix(…);
        //coll matrix = returnFismatrix(…);

        /* build FIS data structure for roll, pitch, yaw, and collective*/
        //roll fis = (FIS *)fisCalloc(…);
        // fisBuildFisNode(…);
```

Appendix G (Continued)

```
//pitch fis = (FIS *)fisCalloc(…);
// fisBuildFisNode(…);
//yaw fis = (FIS *)fisCalloc(…);
// fisBuildFisNode(…);
//coll fis = (FIS *)fisCalloc(…);
// fisBuildFisNode(…);

/*install signal handler for manual break*/
//signal(…);

//getIMU();

//set old imu timestamp to current imu timestamp
//set global gravity vector to zero

/*attempt to calculate the total gravity vector*/
//while 5 seconds of data has not been gathered
{
        /*wait for new imu data*/
        //while old imu timestamp = = new imu timestamp
        {
                usleep(…);
                getIMU(…);
        }

        //set old timestamp to new timestamp

        /*setup world rotation matrices*/
        //setWorldRotationMatrices(…);

        /*setup acceleration vector*/
        //set acceleration element one to pitch axis acceleration
        //set acceleration element two to vertical axis acceleration
        //set acceleration element three to roll axis acceleration

        /*rotate acceleration vector about longitudinal, lateral, then vertical axes*/
        //multMatrices(…);
        //multMatrices(...);
        //multMatrices(...);

        //add the rotated vector to the global gravity vector
        //increment a counter
}

/*average gravity readings*/
//divide gravity vector by counter
```

Appendix G (Continued)

```
        //getGPS(…)
        //set old latitude to current latitude
        //set old longitude to current longitude
        //set old altitude to current altitude (in feet)

        /*set old imu timestamp*/
        //getCurrentTime(…);
}
```

G.14.21 takeOff( )

```
/* Function:    This function manages the takeoff procedure by collecting and      */
/*              calculating the appropriate data for the controllers, passing the data to */
/*              the controllers, converting controller output to PW signals, and sending */
/*              the requested servo positions to the SSC.  This procedure is complete */
/*              when the vehicle has successfully lifted off to a preset altitude.  */
/* Inputs:      N/A                                                                */
/* Outputs:     N/A                                                                */
int takeOff( )
{
        //setup local variables (including static variables)

        //getLaser(…);

        //getGPS(…);
        //convert altitude measurement from meters to feet

        //getIMU(…);

        //if there is no new GPS or IMU data
        {
                usleep(100);
                return 1;
        }

        //if we do not have GPS lock (i.e. lock == 0)
        {
                /*set a reset variable to assure velocities are not calculated due to loss*/
                reset=1;
                //if we have never had a GPS lock
                {
                        //if a new gps message was just received
                        {
                                //output to screen "GPS: No Lock!  Waiting for lock."
                        }
```

Appendix G (Continued)

```
                        /*output neutral values and low throttle value*/
                        //electricMixing(…);
                        //putControllServos(…);

                        return 1;
                }
                //else if a new gps message was just received
                        //output to screen "GPS: No Lock!"
        }
        //else if gps lock is poor (lock == 1) and a new message was received
        {
                //output to screen "WARNING: Differential Lock NOT Available"
        }

        //if we just acquired a gps lock (i.e. reset == 1 && lock > 0)
        {
                //set old latitude to current latitude
                //set old longitude to current longitude
                //set old altitude to current altitude
                reset = 0;
        }

        //getStat(…);

        //if we are in computer control and the takeoff heading has not been set
        {
                //set takeoff heading to current heading
        }

        //if computer control and takeoff position has not been set and we are in stage 1
        {
                //set the last last waypoint and current waypoint equal to current latitude and
                longitude//
        }
        else if not in computer control
        {
                //set the last waypoint to the current waypoint
                //unset takeoff position
        }

        //average all laser readings requested

        //if the laser average is zero and we are in stage 1 or 2
        {
                //display to screen "Warning: Laser may not be active"
                fflush(stdout);
        }
```

Appendix G (Continued)

```
/*determine if takeoff altitude has been reached)
//if laser average is 1.5 meters or zero and we are in stage 3
{
        //return 0;
}
//else if we have reached the desired altitude for the first waypoint (assumed to be above
the takeoff altitude)//
{
        //display to screen "Error: Altitude reached...assuming laser is malfunctioning"
        //display to screen "Error: Vehicle will not land automatically"
        fflush(stdout);
        //set landing request to zero
        return 0;
}

//if we have a gps lock
{
        //calcVelocity(…);
        //gpsRollPitchError(…);

        /*setup rotation matrices*/
        //setLocalRotationMatrices(…);

        /*rotate lateral and longitudinal error to local coordinate frame*/
        //multMatrices(…);
        //if abs(pitching error) is > 50 and the abs (pitching error) > abs(rolling error)
        {
                //set pitching error to 50
                //proportionally reduce the rolling error
        }
        //if abs(rolling error) is > 50 and the abs (rolling error) > abs(pitching error)
        {
                //set rolling error to 50
                //proportionally reduce the pitching error
        }
}
//else
{
        //Set the position error and velocities of roll, pitch, and collective to zero
}

//if computer is in control and we are in stage 3
{

        //if the collective error is not small && the collective velocity is not in the
        direction of the error//
        {
```

Appendix G (Continued)

```
                //set a temporary collective trim variable
        }

        //if the pitching error is not small && the pitching velocity is not in the direction
        of the error && the pitching acceleration variant is not in the direction of error//
        {
                //set a temporary pitching trim variable
        }

        //if the rolling error is not small && the rolling velocity is not in the direction of
        the error && the rolling acceleration variant is not in the direction of error//
        {
                //set a temporary rolling trim variable
        }

        /*setup rotation matrices*/
        //setWorldRotationMatrices(…);

        /*rotate the temporary trim variables*/
        //multMatrices(...);

        //add rotated variables to the global trim variables
}

/*rotate global trim variables to the local coordinate frame*/
//setLocalRotationMatrices(...);
//multMatrices(...);
//set roll angle equal to the IMU roll angle – rotated global trim value*a small constant
(i.e. 0.1)//
//set pitch angle equal to the IMU pitch angle – rotated global trim value*a small constant
(i.e. 0.1)//
//set heading error = calcYawError(…);

//if we have a gps lock
        //set collective error to the altitude set point – current altitude

//sum the last seven velocity calculations for the roll, pitch, and collective
//divide each sum by the time that passed over those last seven readings (acc_change)

//update stored velocities and times with the current velocity

//if we have at least seven velocities stored
{
        /*use a first order kalman filter to update the variant acceleration for the roll*/
        //variant_accel = variant_accel + gain * (acc_change - variant_accel)

        /*use a first order kalman filter to update the variant acceleration for the pitch*/
```

Appendix G (Continued)

```
		//variant_accel = variant_accel + gain * (acc_change - variant_accel)

		/*use a first order kalman filter to update the variant acceleration for collective*/
		//variant_accel = variant_accel + gain * (acc_change - variant_accel)
	}

	/*get control output for the roll*/
	//getFisOutput(…);
	/*allow other processes to operate*/
	usleep(1);
	/*get control output for the pitch*/
	//getFisOutput(…);
	/*get control output for the yaw*/
	//getFisOutput(…);
	/*get control output for the collective*/
	//getFisOutput(…);

	/*calculate PW for the yaw*/
	//if yaw control is positive
		//yaw PW = yaw control *(yaw max – yaw neutral)+yaw neutral;
	//else
		//yaw PW = yaw neutral + yaw control * ( yaw neutral – yaw minimum);

	//if we are in stage zero
	{
		//if we are using a new gps message
		{
			//display to screen "Waiting for convergence"
			fflush(stdout);
		}

		//if we have been in this stage for approximately 15 seconds
			//set stage = 1

		//electricMixing(…)
		//putControllServos(…)
	}
	//else if we are in stage one
	{
		//if we are in computer control
		{
			/*throttle up slowly at first*/
			//if throttle PW > 1200
			{
				//if we are using a new gps message
				{
					//increase throttle PW by 5
```

Appendix G (Continued)

```
                        }
                }
                //else if we are using a new gps message and throttle PW < throttle
                neutral//
                {
                        /* ESC requires a small pause*/
                        //if this statement has been ran > 100 times
                                //increase throttle PW by 10
                }

                //if throttle PW >= THROTTLE_N
                        //set stage = 2

                //electricMixing(…)
                //putControllServos(…)
        }
        //else
        {
                //if we are using a new gps message
                {
                        //display to screen "Takeoff Restart"
                        fflush(stdout);
                }
                //set throttle PW to throttle minimum

                //electricMixing(…)
                //putControllServos(…);
        }
}
//else if we are in computer control and we are in stage two
{
        //if the collective command is less than the collective's neutral
        {
                //increase the collective command by a small amount (i.e. 0.0025)
                //electricMixing(…)
        }
        else
        {
                //electricMixing(…)
                //set stage to three
        }

        //putControllServos(…);
}
//else if we are in computer control
{
        //electricMixing(…)
```

Appendix G (Continued)

```
                //putControllServos(…)
        }
        //else
        {
                //electricMixing(…)
                //putControllServos(…)
        }

        //if we are using a new gps
        {
                //set old latitude equal to current latitude
                //set old longitude equal to current longitude
                //set old altitude equal to current altitude
        }

        //set old imu time to current imu time

        //storeData(…)

        //getCurrentTime(…);

        /*calculate the time passed since the last time this function completed
        //diff = timeDifference(…);

        //if diff is too large (i.e. diff>0.015)
        {
                //display to screen "Navigation Alarm: Time between control calculations low"
                fflush(stdout);
        }

        /*set the end time for the last time this function ran*/
        //getCurrentTime(…)

        return 1;
}
```

G.14.22 landing( )

```
/* Function:    This function manages the landing procedure by collecting and   */
/*              calculating the appropriate data for the controllers, passing the data to   */
/*              the controllers, converting controller output to PW signals, and sending   */
/*              the requested servo positions to the SSC.  This procedure is complete   */
/*              when the vehicle has successfully landed and the motor has powered   */
/*              down   */
/* Inputs:      N/A   */
```

Appendix G (Continued)

```
/* Outputs:      N/A                                                    */
int landing( )
{

        //setup local variables (including static variables)

        //getLaser(…);

        //getGPS(…);
        //convert altitude measurement from meters to feet

        //getIMU(…);

        //if there is no new GPS or IMU data
        {
                usleep(100);
                return 1;
        }

        //if we do not have GPS lock (i.e. lock == 0)
        {
                /*set a reset variable to assure velocities are not calculated due to loss*/
                reset=1;
                //output to screen "GPS: No Lock!”
        }
        //else if gps lock is poor (lock == 1) and a new message was received
        {
                //output to screen "WARNING: Differential Lock NOT Available”
        }

        //if we just acquired a gps lock (i.e. reset == 1 && lock > 0)
        {
                //set old latitude to current latitude
                //set old longitude to current longitude
                //set old altitude to current altitude
                reset = 0;
        }

        //if we have a gps lock
        {
                //calculate offset between current latitude and the waypoint latitude
                //calculate offset between current longitude and the waypoint longitude
        }
        //else
        {
                //set latitude and longitude offsets to zero
        }
```

Appendix G (Continued)

```
//getStat(…);


//if we just entered stage 2
{
        //set the waypoint latitude to the current latitude
        //set the waypoint longitude to the current longitude
}

//for all laser data
{
        //if laser data is > 100 (i.e. a valid value)
        {
                //total = total + laser data
        }
}

//if the number of valid laser readings is greater than zero
        //set laser average = laser_average / number of valid readings;
//else
        //set laser_average = 0;

//if the laser average > 100 and < 2500 and at least 75% of the laser readings were valid
        //set stage to two
//else if the laser average < 175 and at least 90% of the laser readings were valid and we
are in stage two//
        //set stage to three
//else if our stage is > 2 and the laser average > 500)
{
        //display to screen "LANDING FAILURE: Final Stage entered incorrectly"
        //display to screen "LANDING FAILURE: Attempting to recover"
        fflush(stdout);
        //set stage to two
}
//else if the laser average is zero and our stage is greater than one)
{
        //display to screen "Warning: Laser reporting average of zero"
        fflush(stdout);
}

/*have we reached our waypoint*/
//if we have a gps lock and abs(lateral offset)<0.00001) and abs(longitudinal
offset )<0.000010//
{
        //if we are in stage one or two
        {
                //if collective error is <= 2.0
```

Appendix G (Continued)

```
                                        //decrement altitude
                }
        }

        /*assure the altitude set point is not above the current position*/
        //if we just started the landing procedure
        {
                //set the desired altitude to the current altitude
        }

        /*calculate the positional error and velocities*/
        //if we have a gps lock
        {
                //calcVelocity(…)
                //gpsRollPitchError(…)

                /*rotate positional offset to the local coordinate frame*/
                //setLocalRotationMatrices(...)
                //multMatrices(…)

                //if abs(pitching error) is > 50 and the abs (pitching error) > abs(rolling error)
                {
                        //set pitching error to 50
                        //proportionally reduce the rolling error
                }
                //if abs(rolling error) is > 50 and the abs (rolling error) > abs(pitching error)
                {
                        //set rolling error to 50
                        //proportionally reduce the pitching error
                }
        }
        //else
        {
                //Set the position error and velocities of roll, pitch, and collective to zero
        }

        //if the computer is in control
        {
                //if our stage is <= 2
                {
                        //if the collective error is negative and it is not small && the collective
                        velocity is not down//
                        {
                                //set a negative temporary collective trim variable
                        }
                }
```

Appendix G (Continued)

```
		//if the pitching error is not small && the pitching velocity is not in the direction
		of the error && the pitching acceleration variant is not in the direction of error//
		{
			//set a temporary pitching trim variable
		}

		//if the rolling error is not small && the rolling velocity is not in the direction of
		the error && the rolling acceleration variant is not in the direction of error//
		{
			//set a temporary rolling trim variable
		}

		/*setup rotation matrices*/
		//setWorldRotationMatrices(…);

		/*rotate the temporary trim variables*/
		//multMatrices(...);

		//add rotated variables to the global trim variables
	}

	/*rotate global trim variables to the local coordinate frame*/
	//setLocalRotationMatrices(...);
	//multMatrices(...);

	//set roll angle equal to the IMU roll angle – rotated global trim value*a small constant
	(i.e. 0.1)//
	//set pitch angle equal to the IMU pitch angle – rotated global trim value*a small constant
	(i.e. 0.1)//
	//set heading error = calcYawError(…);

	//if we have a gps lock
		//set collective error to the altitude set point – current altitude

	//sum the last seven velocity calculations for the roll, pitch, and collective
	//divide each sum by the time that passed over those last seven readings (acc_change)

	//update stored velocities and times with the current velocity

	//if we have at least seven velocities stored
	{
		/*use a first order kalman filter to update the variant acceleration for the roll*/
		//variant_accel = variant_accel + gain * (acc_change - variant_accel)

		/*use a first order kalman filter to update the variant acceleration for the pitch*/
		//variant_accel = variant_accel + gain * (acc_change - variant_accel)
```

Appendix G (Continued)

```
		/*use a first order kalman filter to update the variant acceleration for collective*/
		//variant_accel = variant_accel + gain * (acc_change - variant_accel)
	}

	/*get control output for the roll*/
	//getFisOutput(…);
	/*allow other processes to operate*/
	usleep(1);
	/*get control output for the pitch*/
	//getFisOutput(…);
	/*get control output for the yaw*/
	//getFisOutput(…);
	/*get control output for the collective*/
	//getFisOutput(…);

	/*calculate PW for the yaw*/
	//if yaw control is positive
		//yaw PW = yaw control *(yaw max – yaw neutral)+yaw neutral;
	//else
		//yaw PW = yaw neutral + yaw control * ( yaw neutral – yaw minimum);

	//if we are in stage three
	{
		//if this is the first run of stage 3
		{
			//if the fuzzy collective output is less than zero
				//set the collective command = collective output / 2.0
			//else
				//set collective command = 0.0
		}
		//else
		{
			//if collective command > -1.0
				//decrement collective command by a small constant (i.e. 0.02)
			//else
				//set stage = 4
		}

		//electricMixing(…)
	}
	//else if we are in stage four
	{
		//if the throttle command > 1000
			// set the throttle command = throttle command - 5

		//electricMixing(…)
	}
```

Appendix G (Continued)

```
        else
                //electricMixing(…)

        //if our stage is < 4
        {
                /*output PW request using the neutral throttle PW*/
                putControllServos(…)
        }
        //else
        {
                /*output PW request using the PW calculated*/
                putControllServos(…)
        }

        //if we are using a new gps
        {
                //set old latitude equal to current latitude
                //set old longitude equal to current longitude
                //set old altitude equal to current altitude
        }

        //set old imu time to current imu time

        //storeData(…)

        //getCurrentTime(…);

        /*calculate the time passed since the last time this function completed
        //diff = timeDifference(…);

        //if diff is too large (i.e. diff>0.015)
        {
                //display to screen "Navigation Alarm: Time between control calculations low"
                fflush(stdout);
        }

        /*set the end time for the last time this function ran*/
        //getCurrentTime(…)
        return 1;
}
```

G.14.23 navigate(…)

```
/* Function:    This function manages the waypoint navigation procedure by collecting  */
/*              and calculating the appropriate data for the controllers, passing the data  */
/*              to the controllers, converting controller output to PW signals, and  */
```

Appendix G (Continued)

```
/*              sending the requested servo positions to the SSC.  This procedure is   */
/*              complete only if an autonomous landing has been requested and only     */
/*              when all of the available waypoints have been transitioned.            */
/* Inputs:      1)  variable identifying if an autonomous landing was requested        */
/* Outputs:     N/A                                                                    */
int navigate(…)
{
        //setup local variables (including static variables)

        //getLaser(…);

        //getGPS(…);
        //convert altitude measurement from meters to feet

        //getIMU(…);

        //if there is no new GPS or IMU data
        {
                usleep(100);
                return 1;
        }

        //if we do not have GPS lock (i.e. lock == 0)
        {
                /*set a reset variable to assure velocities are not calculated due to loss*/
                reset=1;
                //output to screen "GPS: No Lock!"
        }
        //else if gps lock is poor (lock == 1) and a new message was received
        {
                //output to screen "WARNING: Differential Lock NOT Available"
        }

        //if we just acquired a gps lock (i.e. reset == 1 && lock > 0)
        {
                //set old latitude to current latitude
                //set old longitude to current longitude
                //set old altitude to current altitude
                reset = 0;
        }

        //if we have a gps lock
        {
                //calculate offset between current latitude and the waypoint latitude
                //calculate offset between current longitude and the waypoint longitude
        }
        //else
```

Appendix G (Continued)

```
{
        //set latitude and longitude offsets to zero
}

//getStat(…);

//if our operating mode is “hover only”
{
        //if we are in computer control
        {
                //if computer control was just now given
                {
                        //set waypoint latitude to current latitude
                        //set waypoint longitude to current longitude
                        //set waypoint altitude to current altitude
                }
        }
        //else
        {
                //set waypoint latitude to current latitude
                //set waypoint longitude to current longitude
        }
}

/*has the waypoint been reached*/
//if we have gps lock && abs(lateral offset)<0.000005 && abs(longitudinal
offset )<0.000005 && abs(altitude offset)<=2.5 && abs(heading offset)<=5//
{
        //if the vehicle has held this point for at least one second
        {
                //display "Waypoint Reached"
                fflush(stdout);

                //if there are more waypoints to go to
                {
                        //set last waypoint to current waypoint
                        //set current waypoint to the next waypoint
                }
                //else if we are supposed to land
                        return 0;
        }
        //else
                //display "Waiting for system to stabilize"
}

/*calculate position offsets and velocities*/
//if we have a gps lock
```

```
{
        //calcVelocity(…)
        //gpsRollPitchError(…)

        /*rotate positional offsets to the local coordinate frame*/
        //setLocalRotationMatrices(...)
        //multMatrices(…)

        //if abs(pitching error) is > 25 and the abs (pitching error) > abs(rolling error)
        {
                //set pitching error to 25
                //proportionally reduce the rolling error
        }
        //if abs(rolling error) is > 25 and the abs (rolling error) > abs(pitching error)
        {
                //set rolling error to 25
                //proportionally reduce the pitching error
        }
}
//else
{
        //Set the position error and velocities of roll, pitch, and collective to zero
}

//if we are in computer control
{

        /*don't make adjustments when transitioning to a new waypoint*/
        //if we have recently changed waypoints (within one second)
        {
                //do nothing
        }
        //else
        {

                //if the collective error is not small && the collective velocity is not in
                the direction of the error//
                {
                        //set a temporary collective trim variable
                }

                //if the pitching error is not small && the pitching velocity is not in the
                direction of the error && the pitching acceleration variant is not in the
                direction of error//
                {
                        //set a temporary pitching trim variable
                }
```

Appendix G (Continued)

```
                        //if the rolling error is not small && the rolling velocity is not in the
                        direction of the error && the rolling acceleration variant is not in the
                        direction of error//
                        {
                                //set a temporary rolling trim variable
                        }

                        /*setup rotation matrices*/
                        //setWorldRotationMatrices(…);

                        /*rotate the temporary trim variables*/
                        //multMatrices(...);

                        //add rotated variables to the global trim variables
                }
        }

        /*rotate global trim variables to the local coordinate frame*/
        //setLocalRotationMatrices(...);
        //multMatrices(...);

        //set roll angle equal to the IMU roll angle – rotated global trim value*a small constant
        (i.e. 0.1)//
        //set pitch angle equal to the IMU pitch angle – rotated global trim value*a small constant
        (i.e. 0.1)//
        //set heading error = calcYawError(…);

        //if we have a gps lock
                //set collective error to the altitude set point – current altitude

        //sum the last seven velocity calculations for the roll, pitch, and collective
        //divide each sum by the time that passed over those last seven readings (acc_change)

        //update stored velocities and times with the current velocity and time

        //if we have at least seven velocities stored
        {
                /*use a first order kalman filter to update the variant acceleration for the roll*/
                //variant_accel = variant_accel + gain * (acc_change - variant_accel)

                /*use a first order kalman filter to update the variant acceleration for the pitch*/
                //variant_accel = variant_accel + gain * (acc_change - variant_accel)

                /*use a first order kalman filter to update the variant acceleration for collective*/
                //variant_accel = variant_accel + gain * (acc_change - variant_accel)
        }
```

Appendix G (Continued)

```
        /*get control output for the roll*/
        //getFisOutput(…);
        /*allow other processes to operate*/
        usleep(1);
        /*get control output for the pitch*/
        //getFisOutput(…);
        /*get control output for the yaw*/
        //getFisOutput(…);
        /*get control output for the collective*/
        //getFisOutput(…);

        /*calculate PW for the yaw*/
        //if yaw control is positive
                //yaw PW = yaw control *(yaw max – yaw neutral)+yaw neutral;
        //else
                //yaw PW = yaw neutral + yaw control * ( yaw neutral – yaw minimum);

        //electricMixing(…)
        //putControllServos(…)

        //if we are using a new gps
        {
                //set old latitude equal to current latitude
                //set old longitude equal to current longitude
                //set old altitude equal to current altitude
        }

        //set old imu time to current imu time

        //storeData(…)

        //getCurrentTime(…);

        /*calculate the time passed since the last time this function completed
        //diff = timeDifference(…);

        //if diff is too large (i.e. diff>0.015)
        {
                //display to screen "Navigation Alarm: Time between control calculations low"
                fflush(stdout);
        }

        /*set the end time for the last time this function ran*/
        //getCurrentTime(…)
        return 1;
}
```

Appendix G (Continued)

G.14.24 main(…)

```
/* Function:    This is the main function for the navigation process and is responsible for*/
/*              the flow of all navigation procedures                                      */
/* Inputs:      1)  variable identifying if an autonomous takeoff is being requested      */
/*              2)  variable identifying if an autonomous landing is being requested       */
/* Outputs:     N/A                                                                        */
int main(…)
{

        //if the correct arguments are not passed to this process
        {
                //display "Usage Error"
                return;
        }

        // initalizeSystem()

        //if there is only one waypoint and neither a takeoff or landing was requested
                //set operation to "hover_only"

        /*takeoff*/
        //if takeoff was requested
        {
                //display "Takeoff Initiated"
                fflush(stdout);

                //while the takeoff procedure has not returned a zero value
                        // takeOff()

                //display "Takeoff Complete"
                fflush(stdout);
        }

        //while the navigate procedure has not returned a zero value
                //navigate(…)

        /*land*/
        //display "Landing Initiated"
        fflush(stdout);

        //while the landing procedure has not returned a zero value
                //landing()
        //display "Landing Complete"
        fflush(stdout);
```

```
        //while(1)
        {
                /*send neutral values for roll, pitch, and yaw and minimum values for*/
                /*throttle and collective*/
                //electricMixing(…)
                //putControllServos(…)
                //sleep for 1/100th of a second
        }
}
```

G.15 Scripts

G.15.1 Prep

```
echo 'Starting GPS'
./gps &
sleep 1
echo 'Starting IMU'
./imu &
sleep 1
echo 'Starting Servo Controller'
./servo &
sleep 1
echo 'Starting Data Collection'
./collect &
sleep 1
echo 'Starting Laser Collection'
./laser &
sleep 2
```

G.15.2 Transfer

```
make
#note that 15.10.10.109 is the IP for the testbed
scp gps imu data_test servo collect navigate baro laser src/conv/*.fis src/scripts/prep
15.10.10.109:.
```

G.15.3 Syslinux.cfg

```
DEFAULT Lite
PROMPT 1
IMPLICIT 0
```

Appendix G (Continued)

```
TIMEOUT 1

LABEL Lite
        KERNEL vmlinuz
        APPEND initrd=lite.gz ramdisk=200000 rw root=/dev/ram
```

G.15.4 Makefile

```
SRC      = src
SRC1     = src/gps
SRC2     = src/conv
SRC3     = src/imu
SRC4     = src/test
SRC5     = src/collect
SRC6     = src/servo
SRC7     = src/baro
SRC8     = src/laser
INC     = -I. -lm
CC       = gcc

ifeq ($(HOSTTYPE),x86_64)
        LDLIBS  = -L. -L../lib -L/lib64 -L/usr/lib64 -L/usr/local/lib64
else
        LDLIBS  = -L. -L../lib -L/lib -L/usr/lib -L/usr/local/lib
endif

DBG     = -g
CFLAGS  = $(DBG) $(INC) $(LDLIBS)

TARGET1 = gps
TARGET2 = navigate
TARGET3 = imu
TARGET4 = data_test
TARGET5 = collect
TARGET6 = servo
TARGET7 = baro
TARGET8 = laser

FILES1  = $(SRC1)/$(TARGET1).c
FILES2  = $(SRC2)/$(TARGET2).c $(SRC)/*.c #$(SRC2)/robotControls/*.c $(SRC2)/fis.c
FILES3  = $(SRC3)/$(TARGET3).c $(SRC3)/m3dmgAdapter.c $(SRC3)/m3dmgUtils.c
$(SRC3)/m3dmgSerialLinux.c
FILES4  = $(SRC4)/$(TARGET4).c
FILES5  = $(SRC5)/$(TARGET5).c
FILES6  = $(SRC6)/$(TARGET6).c $(SRC)/*.c
```

Appendix G (Continued)

```
FILES7  = $(SRC7)/$(TARGET7).c
FILES8  = $(SRC8)/$(TARGET8).c

all:    $(TARGET1) $(TARGET2) $(TARGET3) $(TARGET4) $(TARGET5) $(TARGET6)
$(TARGET7) $(TARGET8)

$(TARGET1): $(FILES1)
        $(CC) $(FILES1) $(CFLAGS) -o $(TARGET1)
$(TARGET2): $(FILES2)
        $(CC) $(FILES2) $(CFLAGS) -o $(TARGET2) -lm
$(TARGET3): $(FILES3)
        $(CC) -DLINUX_OS $(FILES3) $(CFLAGS) -o $(TARGET3) -lm
$(TARGET4): $(FILES4)
        $(CC) $(FILES4) $(CFLAGS) -o $(TARGET4)
$(TARGET5): $(FILES5)
        $(CC) $(FILES5) $(CFLAGS) -o $(TARGET5)
$(TARGET6): $(FILES6)
        $(CC) $(FILES6) $(CFLAGS) -o $(TARGET6)
$(TARGET7): $(FILES7)
        $(CC) $(FILES7) $(CFLAGS) -o $(TARGET7)
$(TARGET8): $(FILES8)
        $(CC) $(FILES8) $(CFLAGS) -o $(TARGET8)

clean:
        rm -f $(SRC)/*~ $(SRC1)/*~ $(SRC2)/*~ $(SRC2)/*.o $(SRC3)/*~ $(SRC3)/*.o
$(SRC4)/*~ $(SRC5)/*~ $(SRC6)/*~ $(SRC7)/*~ $(SRC8)/*~ *~ $(TARGET2) $(TARGET1)
$(TARGET3) $(TARGET4) $(TARGET5) $(TARGET6) $(TARGET7) $(TARGET8)
```

Appendix H Fuzzy Controller Rules

H.1 Roll Controller Rules

Table 14: Fuzzy Rules for the Roll Controller

If	Error is:	&	Velocity is:	&	Angle is:	&	Acceleration is:	then	Roll is:
	small		small		small		small		zero
	small		small		small		left		right
	small		small		small		right		left
	small		small		left		small		right
	small		small		left		left		right
	small		small		left		right		zero
	small		small		right		small		left
	small		small		right		left		zero
	small		small		right		right		left
	small		small		bigL		small		medR
	small		small		bigL		left		bigR
	small		small		bigL		right		right
	small		small		bigR		small		medL
	small		small		bigR		left		left
	small		small		bigR		right		bigL
	small		left		small		small		right
	small		left		small		left		right
	small		left		small		right		right
	small		left		left		small		medR
	small		left		left		left		bigR
	small		left		left		right		right
	small		left		right		small		right
	small		left		right		left		medR
	small		left		right		right		right
	small		left		bigL		small		bigR
	small		left		bigL		left		bigR
	small		left		bigL		right		right
	small		left		bigR		small		zero
	small		left		bigR		left		zero
	small		left		bigR		right		left
	small		right		small		small		left

Table 14 (Continued)

	Small		right		small		left		left
	Small		right		small		right		left
	Small		right		left		small		left
	Small		right		left		left		left
	Small		right		left		right		medL
	small		right		right		small		medL
	small		right		right		left		left
	small		right		right		right		bigL
	small		right		bigL		small		zero
	small		right		bigL		left		right
	small		right		bigL		right		zero
	small		right		bigR		small		bigL
	small		right		bigR		left		left
	small		right		bigR		right		bigL
	small		bigL		small		small		bigR
	small		bigL		small		left		bigR
	small		bigL		small		right		medR
	small		bigL		left		small		bigR
	small		bigL		left		left		bigR
	small		bigL		left		right		bigR
	small		bigL		right		small		medR
	small		bigL		right		left		medR
	small		bigL		right		right		right
	small		bigL		bigL		small		bigR
	small		bigL		bigL		left		bigR
	small		bigL		bigL		right		bigR
	small		bigL		bigR		small		zero
	small		bigL		bigR		left		zero
	small		bigL		bigR		right		zero
	small		bigR		small		small		bigL
	small		bigR		small		left		medL
	small		bigR		small		right		bigL
	small		bigR		left		small		medL
	small		bigR		left		left		left
	small		bigR		left		right		medL
	small		bigR		right		small		bigL

Table 14 (Continued)

	small		bigR		right		left		bigL
	small		bigR		right		right		bigL
	small		bigR		bigL		small		zero
	small		bigR		bigL		left		zero
	small		bigR		bigL		right		zero
	small		bigR		bigR		small		bigL
	small		bigR		bigR		left		bigL
	small		bigR		bigR		right		bigL
	left		small		small		small		left
	left		small		small		left		zero
	left		small		small		right		left
	left		small		left		small		zero
	left		small		left		left		zero
	left		small		left		right		left
	left		small		right		small		left
	left		small		right		left		left
	left		small		right		right		medL
	left		small		bigL		small		right
	left		small		bigL		left		right
	left		small		bigL		right		zero
	left		small		bigR		small		medL
	left		small		bigR		left		left
	left		small		bigR		right		bigL
	left		left		small		small		zero
	left		left		small		left		zero
	left		left		small		right		left
	left		left		left		small		zero
	left		left		left		left		right
	left		left		left		right		zero
	left		left		right		small		zero
	left		left		right		left		zero
	left		left		right		right		left
	left		left		bigL		small		medR
	left		left		bigL		left		bigR
	left		left		bigL		right		right
	left		left		bigR		small		medL

Table 14 (Continued)

	left		left		bigR		left		left
	left		left		bigR		right		medL
	left		right		small		small		medL
	left		right		small		left		medL
	left		right		small		right		bigL
	left		right		left		small		medL
	left		right		left		left		left
	left		right		left		right		medL
	left		right		right		small		medL
	left		right		right		left		medL
	left		right		right		right		bigL
	left		right		bigL		small		zero
	left		right		bigL		left		zero
	left		right		bigL		right		zero
	left		right		bigR		small		bigL
	left		right		bigR		left		bigL
	left		right		bigR		right		bigL
	left		bigL		small		small		bigR
	left		bigL		small		left		bigR
	left		bigL		small		right		medR
	left		bigL		left		small		bigR
	left		bigL		left		left		bigR
	left		bigL		left		right		medR
	left		bigL		right		small		medR
	left		bigL		right		left		bigR
	left		bigL		right		right		right
	left		bigL		bigL		small		bigR
	left		bigL		bigL		left		bigR
	left		bigL		bigL		right		bigR
	left		bigL		bigR		small		zero
	left		bigL		bigR		left		zero
	left		bigL		bigR		right		zero
	left		bigR		small		small		bigL
	left		bigR		small		left		bigL
	left		bigR		small		right		bigL
	left		bigR		left		small		bigL

Table 14 (Continued)

	left		bigR		left		left		medL
	left		bigR		left		right		bigL
	left		bigR		right		small		bigL
	left		bigR		right		left		bigL
	left		bigR		right		right		bigL
	left		bigR		bigL		small		zero
	left		bigR		bigL		left		zero
	left		bigR		bigL		right		zero
	left		bigR		bigR		small		bigL
	left		bigR		bigR		left		bigL
	left		bigR		bigR		right		bigL
	right		small		small		small		right
	right		small		small		left		right
	right		small		small		right		zero
	right		small		left		small		right
	right		small		left		left		medR
	right		small		left		right		right
	right		small		right		small		zero
	right		small		right		left		right
	right		small		right		right		zero
	right		small		bigL		small		medR
	right		small		bigL		left		bigR
	right		small		bigL		right		right
	right		small		bigR		small		left
	right		small		bigR		left		zero
	right		small		bigR		right		left
	right		left		small		small		medR
	right		left		small		left		bigR
	right		left		small		right		medR
	right		left		left		small		medR
	right		left		left		left		bigR
	right		left		left		right		medR
	right		left		right		small		medR
	right		left		right		left		medR
	right		left		right		right		right
	right		left		bigL		small		bigR

Table 14 (Continued)

	right		left		bigL		left		bigR
	right		left		bigL		right		bigR
	right		left		bigR		small		zero
	right		left		bigR		left		zero
	right		left		bigR		right		zero
	right		right		small		small		zero
	right		right		small		left		right
	right		right		small		right		zero
	right		right		left		small		zero
	right		right		left		left		right
	right		right		left		right		zero
	right		right		right		small		zero
	right		right		right		left		zero
	right		right		right		right		left
	right		right		bigL		small		medR
	right		right		bigL		left		medR
	right		right		bigL		right		right
	right		right		bigR		small		medL
	right		right		bigR		left		left
	right		right		bigR		right		bigL
	right		bigL		small		small		bigR
	right		bigL		small		left		bigR
	right		bigL		small		right		bigR
	right		bigL		left		small		bigR
	right		bigL		left		left		bigR
	right		bigL		left		right		bigR
	right		bigL		right		small		bigR
	right		bigL		right		left		bigR
	right		bigL		right		right		medR
	right		bigL		bigL		small		bigR
	right		bigL		bigL		left		bigR
	right		bigL		bigL		right		bigR
	right		bigL		bigR		small		zero
	right		bigL		bigR		left		zero
	right		bigL		bigR		right		zero
	right		bigR		small		small		bigL

Table 14 (Continued)

	right		bigR		small		left		medL
	right		bigR		small		right		bigL
	right		bigR		left		small		medL
	right		bigR		left		left		left
	right		bigR		left		right		bigL
	right		bigR		right		small		bigL
	right		bigR		right		left		medL
	right		bigR		right		right		bigL
	right		bigR		bigL		small		zero
	right		bigR		bigL		left		zero
	right		bigR		bigL		right		zero
	right		bigR		bigR		small		bigL
	right		bigR		bigR		left		bigL
	right		bigR		bigR		right		bigL
	bigL		small		small		small		left
	bigL		small		small		left		left
	bigL		small		small		right		left
	bigL		small		left		small		left
	bigL		small		left		left		zero
	bigL		small		left		right		left
	bigL		small		right		small		medL
	bigL		small		right		left		left
	bigL		small		right		right		medL
	bigL		small		bigL		small		zero
	bigL		small		bigL		left		zero
	bigL		small		bigL		right		zero
	bigL		small		bigR		small		bigL
	bigL		small		bigR		left		medL
	bigL		small		bigR		right		bigL
	bigL		left		small		small		zero
	bigL		left		small		left		zero
	bigL		left		small		right		left
	bigL		left		left		small		zero
	bigL		left		left		left		zero
	bigL		left		left		right		left
	bigL		left		right		small		left

Table 14 (Continued)

	bigL		left		right		left		zero
	bigL		left		right		right		medL
	bigL		left		bigL		small		right
	bigL		left		bigL		left		medR
	bigL		left		bigL		right		zero
	bigL		left		bigR		small		bigL
	bigL		left		bigR		left		medL
	bigL		left		bigR		right		bigL
	bigL		right		small		small		medL
	bigL		right		small		left		medL
	bigL		right		small		right		bigL
	bigL		right		left		small		medL
	bigL		right		left		left		left
	bigL		right		left		right		medL
	bigL		right		right		small		medL
	bigL		right		right		left		medL
	bigL		right		right		right		bigL
	bigL		right		bigL		small		zero
	bigL		right		bigL		left		zero
	bigL		right		bigL		right		zero
	bigL		right		bigR		small		bigL
	bigL		right		bigR		left		bigL
	bigL		right		bigR		right		bigL
	bigL		bigL		small		small		right
	bigL		bigL		small		left		medR
	bigL		bigL		small		right		right
	bigL		bigL		left		small		medR
	bigL		bigL		left		left		bigR
	bigL		bigL		left		right		medR
	bigL		bigL		right		small		right
	bigL		bigL		right		left		right
	bigL		bigL		right		right		zero
	bigL		bigL		bigL		small		bigR
	bigL		bigL		bigL		left		bigR
	bigL		bigL		bigL		right		bigR
	bigL		bigL		bigR		small		zero

Table 14 (Continued)

	bigL		bigL		bigR		left		zero
	bigL		bigL		bigR		right		zero
	bigL		bigR		small		small		bigL
	bigL		bigR		small		left		bigL
	bigL		bigR		small		right		bigL
	bigL		bigR		left		small		bigL
	bigL		bigR		left		left		medL
	bigL		bigR		left		right		bigL
	bigL		bigR		right		small		bigL
	bigL		bigR		right		left		bigL
	bigL		bigR		right		right		bigL
	bigL		bigR		bigL		small		zero
	bigL		bigR		bigL		left		zero
	bigL		bigR		bigL		right		zero
	bigL		bigR		bigR		small		bigL
	bigL		bigR		bigR		left		bigL
	bigL		bigR		bigR		right		bigL
	bigR		small		small		small		right
	bigR		small		small		left		right
	bigR		small		small		right		right
	bigR		small		left		small		medR
	bigR		small		left		left		medR
	bigR		small		left		right		right
	bigR		small		right		small		right
	bigR		small		right		left		right
	bigR		small		right		right		zero
	bigR		small		bigL		small		bigR
	bigR		small		bigL		left		bigR
	bigR		small		bigL		right		medR
	bigR		small		bigR		small		zero
	bigR		small		bigR		left		zero
	bigR		small		bigR		right		zero
	bigR		left		small		small		medR
	bigR		left		small		left		bigR
	bigR		left		small		right		medR
	bigR		left		left		small		medR

Table 14 (Continued)

	bigR		left		left		left		bigR
	bigR		left		left		right		medR
	bigR		left		right		small		medR
	bigR		left		right		left		medR
	bigR		left		right		right		right
	bigR		left		bigL		small		bigR
	bigR		left		bigL		left		bigR
	bigR		left		bigL		right		bigR
	bigR		left		bigR		small		zero
	bigR		left		bigR		left		zero
	bigR		left		bigR		right		zero
	bigR		right		small		small		zero
	bigR		right		small		left		right
	bigR		right		small		right		zero
	bigR		right		left		small		right
	bigR		right		left		left		medR
	bigR		right		Left		right		zero
	bigR		right		Right		small		zero
	bigR		right		Right		left		right
	bigR		right		Right		right		zero
	bigR		right		bigL		small		bigR
	bigR		right		bigL		left		bigR
	bigR		right		bigL		right		medR
	bigR		right		bigR		small		left
	bigR		right		bigR		left		zero
	bigR		right		bigR		right		medL
	bigR		bigL		Small		small		bigR
	bigR		bigL		Small		left		bigR
	bigR		bigL		Small		right		bigR
	bigR		bigL		Left		small		bigR
	bigR		bigL		Left		left		bigR
	bigR		bigL		Left		right		bigR
	bigR		bigL		Right		small		bigR
	bigR		bigL		Right		left		bigR
	bigR		bigL		Right		right		medR
	bigR		bigL		bigL		small		bigR

Table 14 (Continued)

	bigR		bigL		bigL		left		bigR
	bigR		bigL		bigL		right		bigR
	bigR		bigL		bigR		small		zero
	bigR		bigL		bigR		left		zero
	bigR		bigL		bigR		right		zero
	bigR		bigR		small		small		left
	bigR		bigR		small		left		left
	bigR		bigR		small		right		medL
	bigR		bigR		left		small		left
	bigR		bigR		left		left		zero
	bigR		bigR		left		right		left
	bigR		bigR		right		small		medL
	bigR		bigR		right		left		medL
	bigR		bigR		right		right		bigL
	bigR		bigR		bigL		small		zero
	bigR		bigR		bigL		left		zero
	bigR		bigR		bigL		Right		Zero
	bigR		bigR		bigR		Small		bigL
	bigR		bigR		bigR		Left		bigL
	bigR		bigR		bigR		Right		bigL

H.2 Pitch Controller Rules

Table 15: Fuzzy Rules for the Pitch Controller

If	Error is:	&	Velocity is:	&	Angle is:	&	Acceleration is:	then	Pitch is:
	small		small		small		small		zero
	small		small		small		backward		forward
	small		small		small		forward		backward
	small		small		backward		small		forward
	small		small		backward		backward		forward
	small		small		backward		forward		zero
	small		small		forward		small		backward
	small		small		forward		backward		zero
	small		small		forward		forward		backward

Table 15 (Continued)

	small		small		bigB		small		medF
	small		small		bigB		backward		bigF
	small		small		bigB		forward		forward
	small		small		bigF		small		medB
	small		small		bigF		backward		backward
	small		small		bigF		forward		bigB
	small		backward		small		small		forward
	small		backward		small		backward		forward
	small		backward		small		forward		forward
	small		backward		backward		small		medF
	small		backward		backward		backward		bigF
	small		backward		backward		forward		forward
	small		backward		forward		small		forward
	small		backward		forward		backward		medF
	small		backward		forward		forward		forward
	small		backward		bigB		small		bigF
	small		backward		bigB		backward		bigF
	small		backward		bigB		forward		forward
	small		backward		bigF		small		zero
	small		backward		bigF		backward		zero
	small		backward		bigF		forward		backward
	small		forward		small		small		backward
	small		forward		small		backward		backward
	small		forward		small		forward		backward
	small		forward		backward		small		backward
	small		forward		backward		backward		backward
	small		forward		backward		forward		medB
	small		forward		forward		small		medB
	small		forward		forward		backward		backward
	small		forward		forward		forward		bigB
	small		forward		bigB		small		zero
	small		forward		bigB		backward		forward
	small		forward		bigB		forward		zero
	small		forward		bigF		small		bigB
	small		forward		bigF		backward		backward
	small		forward		bigF		forward		bigB

Table 15 (Continued)

	small		bigB		small		small		bigF
	small		bigB		small		backward		bigF
	small		bigB		small		forward		medF
	small		bigB		backward		small		bigF
	small		bigB		backward		backward		bigF
	small		bigB		backward		forward		bigF
	small		bigB		forward		small		medF
	small		bigB		forward		backward		medF
	small		bigB		forward		forward		forward
	small		bigB		bigB		small		bigF
	small		bigB		bigB		backward		bigF
	small		bigB		bigB		forward		bigF
	small		bigB		bigF		small		zero
	small		bigB		bigF		backward		zero
	small		bigB		bigF		forward		zero
	small		bigF		small		small		bigB
	small		bigF		small		backward		medB
	small		bigF		small		forward		bigB
	small		bigF		backward		small		medB
	small		bigF		backward		backward		backward
	small		bigF		backward		forward		medB
	small		bigF		forward		small		bigB
	small		bigF		forward		backward		bigB
	small		bigF		forward		forward		bigB
	small		bigF		bigB		small		zero
	small		bigF		bigB		backward		zero
	small		bigF		bigB		forward		zero
	small		bigF		bigF		small		bigB
	small		bigF		bigF		backward		bigB
	small		bigF		bigF		forward		bigB
	backward		small		small		small		backward
	backward		small		small		backward		zero
	backward		small		small		forward		backward
	backward		small		backward		small		zero
	backward		small		backward		backward		zero
	backward		small		backward		forward		backward

Table 15 (Continued)

	backward		small		forward		small		backward
	backward		small		forward		backward		backward
	backward		small		forward		forward		medB
	backward		small		bigB		small		forward
	backward		small		bigB		backward		forward
	backward		small		bigB		forward		zero
	backward		small		bigF		small		medB
	backward		small		bigF		backward		backward
	backward		small		bigF		forward		bigB
	backward		backward		small		small		zero
	backward		backward		small		backward		zero
	backward		backward		small		forward		backward
	backward		backward		backward		small		zero
	backward		backward		backward		backward		forward
	backward		backward		backward		forward		zero
	backward		backward		forward		small		zero
	backward		backward		forward		backward		zero
	backward		backward		forward		forward		backward
	backward		backward		bigB		small		medF
	backward		backward		bigB		backward		bigF
	backward		backward		bigB		forward		forward
	backward		backward		bigF		small		medB
	backward		backward		bigF		backward		backward
	backward		backward		bigF		forward		medB
	backward		forward		small		small		medB
	backward		forward		small		backward		medB
	backward		forward		small		forward		bigB
	backward		forward		backward		small		medB
	backward		forward		backward		backward		backward
	backward		forward		backward		forward		medB
	backward		forward		forward		small		medB
	backward		forward		forward		backward		medB
	backward		forward		forward		forward		bigB
	backward		forward		bigB		small		zero
	backward		forward		bigB		backward		zero
	backward		forward		bigB		forward		zero

Table 15 (Continued)

	backward		forward		bigF		small		bigB
	backward		forward		bigF		backward		bigB
	backward		forward		bigF		forward		bigB
	backward		bigB		small		small		bigF
	backward		bigB		small		backward		bigF
	backward		bigB		small		forward		medF
	backward		bigB		backward		small		bigF
	backward		bigB		backward		backward		bigF
	backward		bigB		backward		forward		medF
	backward		bigB		forward		small		medF
	backward		bigB		forward		backward		bigF
	backward		bigB		forward		forward		forward
	backward		bigB		bigB		small		bigF
	backward		bigB		bigB		backward		bigF
	backward		bigB		bigB		forward		bigF
	backward		bigB		bigF		small		zero
	backward		bigB		bigF		backward		zero
	backward		bigB		bigF		forward		zero
	backward		bigF		small		small		bigB
	backward		bigF		small		backward		bigB
	backward		bigF		small		forward		bigB
	backward		bigF		backward		small		bigB
	backward		bigF		backward		backward		medB
	backward		bigF		backward		forward		bigB
	backward		bigF		forward		small		bigB
	backward		bigF		forward		backward		bigB
	backward		bigF		forward		forward		bigB
	backward		bigF		bigB		small		zero
	backward		bigF		bigB		backward		zero
	backward		bigF		bigB		forward		zero
	backward		bigF		bigF		small		bigB
	backward		bigF		bigF		backward		bigB
	backward		bigF		bigF		forward		bigB
	forward		small		small		small		forward
	forward		small		small		backward		forward
	forward		small		small		forward		zero

Table 15 (Continued)

	forward		small		backward		small		forward
	forward		small		backward		backward		medF
	forward		small		backward		forward		forward
	forward		small		forward		small		zero
	forward		small		forward		backward		forward
	forward		small		forward		forward		zero
	forward		small		bigB		small		medF
	forward		small		bigB		backward		bigF
	forward		small		bigB		forward		forward
	forward		small		bigF		small		backward
	forward		small		bigF		backward		zero
	forward		small		bigF		forward		backward
	forward		backward		small		small		medF
	forward		backward		small		backward		bigF
	forward		backward		small		forward		medF
	forward		backward		backward		small		medF
	forward		backward		backward		backward		bigF
	forward		backward		backward		forward		medF
	forward		backward		forward		small		medF
	forward		backward		forward		backward		medF
	forward		backward		forward		forward		forward
	forward		backward		bigB		small		bigF
	forward		backward		bigB		backward		bigF
	forward		backward		bigB		forward		bigF
	forward		backward		bigF		small		zero
	forward		backward		bigF		backward		zero
	forward		backward		bigF		forward		zero
	forward		forward		small		small		zero
	forward		forward		small		backward		forward
	forward		forward		small		forward		zero
	forward		forward		backward		small		zero
	forward		forward		backward		backward		forward
	forward		forward		backward		forward		zero
	forward		forward		forward		small		zero
	forward		forward		forward		backward		zero
	forward		forward		forward		forward		backward

Table 15 (Continued)

	forward		forward		bigB		small		medF
	forward		forward		bigB		backward		medF
	forward		forward		bigB		forward		forward
	forward		forward		bigF		small		medB
	forward		forward		bigF		backward		backward
	forward		forward		bigF		forward		bigB
	forward		bigB		small		small		bigF
	forward		bigB		small		backward		bigF
	forward		bigB		small		forward		bigF
	forward		bigB		backward		small		bigF
	forward		bigB		backward		backward		bigF
	forward		bigB		backward		forward		bigF
	forward		bigB		forward		small		bigF
	forward		bigB		forward		backward		bigF
	forward		bigB		forward		forward		medF
	forward		bigB		bigB		small		bigF
	forward		bigB		bigB		backward		bigF
	forward		bigB		bigB		forward		bigF
	forward		bigB		bigF		small		zero
	forward		bigB		bigF		backward		zero
	forward		bigB		bigF		forward		zero
	forward		bigF		small		small		bigB
	forward		bigF		small		backward		medB
	forward		bigF		small		forward		bigB
	forward		bigF		backward		small		medB
	forward		bigF		backward		backward		backward
	forward		bigF		backward		forward		bigB
	forward		bigF		forward		small		bigB
	forward		bigF		forward		backward		medB
	forward		bigF		forward		forward		bigB
	forward		bigF		bigB		small		zero
	forward		bigF		bigB		backward		zero
	forward		bigF		bigB		forward		zero
	forward		bigF		bigF		small		bigB
	forward		bigF		bigF		backward		bigB
	forward		bigF		bigF		forward		bigB

Table 15 (Continued)

	bigB		small		small		small		backward
	bigB		small		small		backward		backward
	bigB		small		small		forward		backward
	bigB		small		backward		small		backward
	bigB		small		backward		backward		zero
	bigB		small		backward		forward		backward
	bigB		small		forward		small		medB
	bigB		small		forward		backward		backward
	bigB		small		forward		forward		medB
	bigB		small		bigB		small		zero
	bigB		small		bigB		backward		zero
	bigB		small		bigB		forward		zero
	bigB		small		bigF		small		bigB
	bigB		small		bigF		backward		medB
	bigB		small		bigF		forward		bigB
	bigB		backward		small		small		zero
	bigB		backward		small		backward		zero
	bigB		backward		small		forward		backward
	bigB		backward		backward		small		zero
	bigB		backward		backward		backward		zero
	bigB		backward		backward		forward		backward
	bigB		backward		forward		small		backward
	bigB		backward		forward		backward		zero
	bigB		backward		forward		forward		medB
	bigB		backward		bigB		small		forward
	bigB		backward		bigB		backward		medF
	bigB		backward		bigB		forward		zero
	bigB		backward		bigF		small		bigB
	bigB		backward		bigF		backward		medB
	bigB		backward		bigF		forward		bigB
	bigB		forward		small		small		medB
	bigB		forward		small		backward		medB
	bigB		forward		small		forward		bigB
	bigB		forward		backward		small		medB
	bigB		forward		backward		backward		backward
	bigB		forward		backward		forward		medB

Table 15 (Continued)

	bigB		forward		forward		small		medB
	bigB		forward		forward		backward		medB
	bigB		forward		forward		forward		bigB
	bigB		forward		bigB		small		zero
	bigB		forward		bigB		backward		zero
	bigB		forward		bigB		forward		zero
	bigB		forward		bigF		small		bigB
	bigB		forward		bigF		backward		bigB
	bigB		forward		bigF		forward		bigB
	bigB		bigB		small		small		forward
	bigB		bigB		small		backward		medF
	bigB		bigB		small		forward		forward
	bigB		bigB		backward		small		medF
	bigB		bigB		backward		backward		bigF
	bigB		bigB		backward		forward		medF
	bigB		bigB		forward		small		forward
	bigB		bigB		forward		backward		forward
	bigB		bigB		forward		forward		zero
	bigB		bigB		bigB		small		bigF
	bigB		bigB		bigB		backward		bigF
	bigB		bigB		bigB		forward		bigF
	bigB		bigB		bigF		small		zero
	bigB		bigB		bigF		backward		zero
	bigB		bigB		bigF		forward		zero
	bigB		bigF		small		small		bigB
	bigB		bigF		small		backward		bigB
	bigB		bigF		small		forward		bigB
	bigB		bigF		backward		small		bigB
	bigB		bigF		backward		backward		medB
	bigB		bigF		backward		forward		bigB
	bigB		bigF		forward		small		bigB
	bigB		bigF		forward		backward		bigB
	bigB		bigF		forward		forward		bigB
	bigB		bigF		bigB		small		zero
	bigB		bigF		bigB		backward		zero
	bigB		bigF		bigB		forward		zero

Table 15 (Continued)

	bigB		bigF		bigF		small		bigB
	bigB		bigF		bigF		backward		bigB
	bigB		bigF		bigF		forward		bigB
	bigF		small		small		small		forward
	bigF		small		small		backward		forward
	bigF		small		small		forward		forward
	bigF		small		backward		small		medF
	bigF		small		backward		backward		medF
	bigF		small		backward		forward		forward
	bigF		small		forward		small		forward
	bigF		small		forward		backward		forward
	bigF		small		forward		forward		zero
	bigF		small		bigB		small		bigF
	bigF		small		bigB		backward		bigF
	bigF		small		bigB		forward		medF
	bigF		small		bigF		small		zero
	bigF		small		bigF		backward		zero
	bigF		small		bigF		forward		zero
	bigF		backward		small		small		medF
	bigF		backward		small		backward		bigF
	bigF		backward		small		forward		medF
	bigF		backward		backward		small		medF
	bigF		backward		backward		backward		bigF
	bigF		backward		backward		forward		medF
	bigF		backward		forward		small		medF
	bigF		backward		forward		backward		medF
	bigF		backward		forward		forward		forward
	bigF		backward		bigB		small		bigF
	bigF		backward		bigB		backward		bigF
	bigF		backward		bigB		forward		bigF
	bigF		backward		bigF		small		zero
	bigF		backward		bigF		backward		zero
	bigF		backward		bigF		forward		zero
	bigF		forward		small		small		zero
	bigF		forward		small		backward		forward
	bigF		forward		small		forward		zero

Table 15 (Continued)

	bigF		forward		backward		small		forward
	bigF		forward		backward		backward		medF
	bigF		forward		backward		forward		zero
	bigF		forward		forward		small		zero
	bigF		forward		forward		backward		forward
	bigF		forward		forward		forward		zero
	bigF		forward		bigB		small		bigF
	bigF		forward		bigB		backward		bigF
	bigF		forward		bigB		forward		medF
	bigF		forward		bigF		small		backward
	bigF		forward		bigF		backward		zero
	bigF		forward		bigF		forward		medB
	bigF		bigB		small		small		bigF
	bigF		bigB		small		backward		bigF
	bigF		bigB		small		forward		bigF
	bigF		bigB		backward		small		bigF
	bigF		bigB		backward		backward		bigF
	bigF		bigB		backward		forward		bigF
	bigF		bigB		forward		small		bigF
	bigF		bigB		forward		backward		bigF
	bigF		bigB		forward		forward		medF
	bigF		bigB		bigB		small		bigF
	bigF		bigB		bigB		backward		bigF
	bigF		bigB		bigB		forward		bigF
	bigF		bigB		bigF		small		zero
	bigF		bigB		bigF		backward		zero
	bigF		bigB		bigF		forward		zero
	bigF		bigF		small		small		backward
	bigF		bigF		small		backward		backward
	bigF		bigF		small		forward		medB
	bigF		bigF		backward		small		backward
	bigF		bigF		backward		backward		zero
	bigF		bigF		backward		forward		backward
	bigF		bigF		forward		small		medB
	bigF		bigF		forward		backward		medB
	bigF		bigF		forward		forward		bigB

Table 15 (Continued)

	bigF		bigF		bigB		small		zero
	bigF		bigF		bigB		backward		zero
	bigF		bigF		bigB		forward		zero
	bigF		bigF		bigF		small		bigB
	bigF		bigF		bigF		backward		bigB
	bigF		bigF		bigF		forward		bigB

H.3 Collective Controller Rules

Table 16: Fuzzy Rules for the Collective Controller

If	Error is:	&	Velocity is:	&	Acceleration is:	then	Collective is:
	down		down		Down		up
	down		down		small		neutral
	down		down		up		neutral
	down		small		down		neutral
	down		small		small		down
	down		small		up		down
	down		Up		down		Down
	down		Up		small		medDown
	down		up		up		medDown
	down		bigD		down		bigUp
	down		bigD		small		medUp
	down		bigD		up		medUp
	down		bigU		down		bigDown
	down		bigU		small		bigDown
	down		bigU		up		bigDown
	up		down		down		medUp
	up		down		small		medUp
	up		down		up		Up
	up		small		down		medUp
	up		small		small		up
	up		small		up		up
	up		up		down		neutral
	up		up		small		neutral

Table 16 (Continued)

	up		up		up		down
	up		bigD		down		bigUp
	up		bigD		small		bigUp
	up		bigD		up		bigUp
	up		bigU		down		medDown
	up		bigU		small		medDown
	up		bigU		up		bigDown
	bigD		down		down		up
	bigD		down		small		neutral
	bigD		down		up		neutral
	bigD		small		down		down
	bigD		small		small		down
	bigD		small		up		medDown
	bigD		up		down		medDown
	bigD		up		small		medDown
	bigD		up		up		medDown
	bigD		bigD		down		bigUp
	bigD		bigD		small		medUp
	bigD		bigD		up		medUp
	bigD		bigU		down		bigDown
	bigD		bigU		small		bigDown
	bigD		bigU		up		bigDown
	bigU		down		down		bigUp
	bigU		down		small		medUp
	bigU		down		up		medUp
	bigU		small		down		medUp
	bigU		small		small		up
	bigU		small		up		up
	bigU		up		down		up
	bigU		up		small		up
	bigU		up		up		neutral
	bigU		bigD		down		bigUp
	bigU		bigD		small		bigUp
	bigU		bigD		up		bigUp
	bigU		bigU		down		down
	bigU		bigU		small		down

Table 16 (Continued)

	bigU		bigU		up		medDown
	small		down		down		medUp
	small		down		small		up
	small		down		up		up
	small		small		down		up
	small		small		small		neutral
	small		small		up		down
	small		up		down		down
	small		up		small		down
	small		up		up		medDown
	small		bigD		down		bigUp
	small		bigD		small		bigUp
	small		bigD		up		bigUp
	small		bigU		Down		bigDown
	small		bigU		small		bigDown
	small		bigU		Up		bigDown

H.4 Yaw Controller Rules

Table 17: Fuzzy Rules for the Yaw Controller

If	Error is:	then	Yaw is:
	right		right
	left		left
	bigR		bigR
	bigL		bigLeft
	small		small

Appendix I UDP Packages Available from X-Plane

Table 18: UDP Data Packages Available from X-Plane [99]

UDP 00	Times.
V0	Frame Rate (frames/sec) - Number of Frames rendered per second. [float]
V1	Time Ratio (ratio) This is how close XP time matches real time. Ideal ratio is 1. [float]
UDP 01	Time Elapsed, cockpit Timer...
V0	Time Elapsed - Seconds. [float]
V1	Cockpit Timer - Seconds. [float]
V2	Local Time (hours expressed digitally) [float]
V3	Zulu Time (hours) [float]
UDP 02	Speed, Vertical Speed
V0	True Speed in Kts. [float]
V1	Indicated Speed in kts. [float]
V2	True Speed in mph [float]
V3	Indicated Speed in mph [float]
V4	Vertical Speed in feet/minute. [float]
UDP 03	Mach, G-Loads
V0	Mach Ratio [float]
V1	G-Load Normal
V2	G-Load Axial
V3	G-Load Side
UDP 04	Alpha, Beta (Attack Angles)
V0	Alpha (angle of attack) - in degrees. [float]
V1	Beta (Sideslip/Yaw) - in degrees. [float]
UDP 05	Atmosphere - sea level
V0	SLprs inHG
V1	SLtmp degC
V2	Wind Speed
V3	Wind Direction
UDP 06	Atmosphere – ambient
V0	AMprs inHG
V1	AMtmp degC
V2	LEtmp degC
V3	gravi fts2

Appendix I (Continued)

Table 18 (Continued)

V4	dens part
V5	Q psf
V6	A ktas
UDP 07	Atmosphere – systems
V0	ALprs inHG
V1	edens part
V2	vacuum ratio
UDP 08	Joystick ail/elev/rudd/swe/vec
V0	Joystick Elevator Input (0-1). (These are the 'user' control inputs) [float]
V1	Joystick Aileron Input (0-1) [float]
V2	Joystick Rudder Input (0-1) [float]
V3	Joystick Sweep Request (Degrees) [float]
V4	Joystick Vector Request (Degrees) [float]
UDP 09	Artificial Stability Values ail/elv/rud
V0	Elevator. (These are the calculated Artificial Stability Values) [float]
V1	Aileron. [float]
V2	Rudder. [float]
UDP 10	Flight Condition Values ail/elv/rud
V0	Elevator. (Surface) (These are the sum of UDP #8 and #9) [float]
V1	Aileron (Surface) [float]
V2	Rudder (Surface) [float]
V3	Nosewheel Steering (degrees) [float]
UDP 11	Wing Sweep, Thrust Vector
V0	Sweep - Ratio [float]
V1	Sweep 1 (deg) [float]
V2	Sweep 2 (deg) [float]
V3	Sweep 3 (deg) [float]
V4	Thrust Vector (deg) [float]
UDP 12	Trim, Flap, Slat, Speedbrakes
V0	Elevator Trim [float]
V1	Aileron Trim [float]
V2	Rudder Trim [float]
V3	Rotor Trim [float]

Table 18 (Continued)

V4	Flap Request
V5	Flap Position
V6	Slat (ratio) [float]
V7	Speedbrake (ratio) [float]
UDP 13	Gear & Brakes
V0	Gear, 1=Down, 0=Up [integer]
V1	Wheelbrake. Part (1=on)
V2	Left Brake (0-1) [float]
V3	Right Brake (0-1) [float]
UDP 14	Angular Moments
V0	M (ft/lb) [float]
V1	L (ft/lb) [float]
V2	N (ft/lb) [float]
UDP 15	Angular Accelerations
V0	Qdot - d/ss [float]
V1	Pdot - d/ss [float]
V2	Rdot - d/ss [float]
UDP 16	Angular Velocities
V0	Q - d/s [float]
V1	P - d/s [float]
V2	R - d/s [float]
UDP 17	Pitch, Roll, Headings
V0	Pitch - degrees [float]
V1	Roll - degrees [float]
V2	True Heading - degrees [float]
V3	Magnetic Heading - degrees [float]
V4	Mag Var – degrees [float]
V5	Heading Bug - degrees (true) [float]
UDP 18	Lat, Lon, Altitude
V0	Latitude – degrees (Origin Reference Point) [float]
V1	Longitude - degrees [float]
V2	Altitude - fmsl [float]
V3	Altitude - fagl [float]

Table 18 (Continued)

V4	
V5	Altitude – indicated
V6	Latitude - South [float]
V7	Longitude - West [float]
UDP 19	X, Y, Z, distance traveled
V0	X - m [float]
V1	Y - m [float]
V2	Z - m [float]
V3	Velocity in X – m [float]
V4	Velocity in Y – m [float]
V5	Velocity in Z – m [float]
V6	Distance - feet [float]
V7	Distance - nm [float]
UDP 20	All Planes: X
V0	X - m [float]
V1	Repeated for each craft...
UDP 21	All Planes: Y
V0	Y - m [float]
V1	Repeated for each craft...
UDP 22	All Planes: Z
V0	Z - m [float]
V1	Repeated for each craft...
UDP 23	Throttle Settings
V0	Throttle Setting (Part). Negative value indicates reverse thrust. [float]
V1	This repeats for each engine...
UDP 24	Reverse Settings
V0	Reverse Thrust Setting (0/1). [integer]
V1	This repeats for each engine...
UDP 25	Propellor Settings
V0	Propellor Setting (rpm). [float]
V1	This repeats for each propellor...
UDP 26	Mixture Settings
V0	Mixture Setting (Ratio). [float]

Table 18 (Continued)

V1	This repeats for each Engine...
UDP 27	Carb Heat Settings
V0	Carb Heat Setting (Ratio). [float]
V1	This repeats for each Carb...
UDP 28	Ignition Settings
V0	Ignition Switch Position (0,1,2 or 3). [integer]
V1	This repeats for each Ignition...
UDP 29	Cowl Flap Settings
V0	Cowl Flap Setting (Set). [float]
V1	This repeats for each Cowl...
UDP 30	Engine Power
V0	Engine 1 - hp [float]
V1	Repeated for each engine...
UDP 31	Engine Thrust
V0	Engine 1 - lb [float]
V1	Repeated for each engine...
UDP 32	Engine Torque
V0	Engine 1 - ft/lb [float]
V1	Repeated for each engine...
UDP 33	Engine RPM
V0	Engine 1 - rpm [float]
V1	Repeats for each engine...
UDP 34	Prop RPM
V0	Prop 1 - rpm [float]
V1	Repeated for each prop...
UDP 35	Prop Pitch
V0	Prop 1 - degrees [float]
V1	Repeated for each prop...
UDP 36	Propwash/Jetwash
V0	Prop 1 - knots [float]
V1	Repeated for each prop...
UDP 37	Manifold Pressures
V0	Engine 1 - inhg [float]

Table 18 (Continued)

V1	Repeated for each engine...
UDP 38	N1
V0	Engine 1 - Percent [float]
V1	Repeated for each engine...
UDP 39	EPR
V0	Engine 1 - Part [float]
V1	Repeated for each engine...
UDP 40	Fuel Flow
V0	Engine 1 - lb/hour [float]
V1	Repeated for each engine...
UDP 41	EGT
V0	Engine 1 - Ratio [float]
V1	Repeated for each engine...
UDP 42	ITT
V0	ITT1 (deg) [float]
V1	Repeats for each engine...
UDP43	Oil Pressures
V0	Pressure 1 (ratio) [float]
V1	Repeats for each engine...
UDP44	Oil Temperatures
V0	Temp 1 (ratio) [float]
V1	Repeats for each engine...
UDP45	Alternator Amps
V0	Amps 1 (ratio) [float]
V1	Repeats for each engine...
UDP46	Battery Amps
V0	Battery 1 (ratio) [float]
V1	Repeats for each...
UDP47	Battery Voltage
V0	Volts 1 (ratio) [float]
V1	Repeats for each...
UDP48	Fuel Pumps
V0	Pump 1 (0/1)

Table 18 (Continued)

V1	Repeats for each...
UDP49	Generators
V0	Generator 1 (0/1)
V1	Repeats for each...
UDP50	Inverters
V0	Inverter 1 (0/1)
V1	Repeats for each...
UDP51	Starter Timeout
V0	Starter 1 (secs/1)
V1	Repeats for each...
UDP 52	Aero Forces
V0	Lift (lb) [float]
V1	Drag (lb) [float]
V2	Side (lb) [float]
UDP 53	Engine Forces
V0	Normal (lb) [float]
V1	Axial (lb) [float]
V2	Side (lb) [float]
UDP 54	Landing Gear - Vertical Forces
V0	Gear 1 (lb) [float]
V1	Gear 2 (lb) [float]
V2	Gear 3 (lb) [float]
UDP 55	Landing Gear - Vertical Deflections
V0	Gear 1 (ft) [float]
V1	Gear 2 (ft) [float]
V2	Gear 3 (ft) [float]
UDP 56	Lift over Drag Ratio
V0	L/D (Ratio) [float]
V1	
UDP 57	Prop Efficiency
V0	Prop 1 (ratio) [float]
V1	Repeats for each Prop...
UDP 58	Aileron Deflections

Appendix I (Continued)

Table 18 (Continued)

V0	Left Aileron 2 (degrees) [float]
V1	Left Aileron 1 (degrees) [float]
V2	Right Aileron 1 (degrees) [float]
V3	Right Aileron 2 (degrees) [float]
UDP 59	Roll Spoiler Deflections
V0	Spoiler 1 (degrees) [float]
V1	Spoiler 2 (degrees) [float]
UDP 60	Elevator Deflections
V0	Elevator 1 (degrees) [float]
V1	Elevator 2 (degrees) [float]
UDP 61	Rudder Deflections
V0	Rudder (degrees) [float]
UDP 62	Yaw-Brake Deflections
V0	Drudd (degrees) [float]
V1	Drudd (degrees) [float]
UDP 63	TOTAL vertical thrust vectors
V0	Vert1 (vector) [float]
V1	vert2 (vector) [float]
UDP 64	TOTAL lateral thrust vectors
V0	Lateral1 (vector) [float]
V1	Lateral2 (vector) [float]
UDP 65	Pitch cyclic Disc Tilts
V0	cyclic [float]
V1	cyclic [float]
UDP 66	Roll Cyclic Disc Tilts
V0	cyclic [float]
V1	cyclic [float]
UDP 67	pitch Cyclic Flapping
V0	flap [float]
V1	flap [float]
UDP 68	Roll Cyclic Flapping
V0	flap [float]
V1	flap [float]

Appendix I (Continued)

Table 18 (Continued)

UDP 69	Payload Weights
V0	Payload (lb) [float]
V1	Fuel1 (lb) [float]
V2	Fuel2 (lb) [float]
V3	Fuel3 (lb) [float]
V4	Jettison (lb) [float]
V5	Fuel Tank 1 % Full [float]
V6	Fuel Tank 2 % Full [float]
V7	Fuel Tank 3 % Full [float]
UDP 70	Total Weights & CG
V0	Empty (lb) [float]
V1	Current (lb) [float]
V2	Maximum (lb) [float]
V3	
V4	CofG (ftref) [float]
UDP 66	Ground effect on lift, wings
V0	Wing1 (L cl*) [float]
V1	Wing1 (R cl*) [float]
V2	Wing2 (L cl*) [float]
V3	Wing2 (R cl*) [float]
V4	Wing3 (L cl*) [float]
V5	Wing3 (R cl*) [float]
UDP 67	Ground effect on drag, wings
V0	Wing1 (L cd*) [float]
V1	Wing1 (R cd*) [float]
V2	Wing2 (L cd*) [float]
V3	Wing2 (R cd*) [float]
V4	Wing3 (L cd*) [float]
V5	Wing3 (R cd*) [float]
UDP 68	Ground effect on lift, stabs
V0	Hstab (L cl*) [float]
V1	Hstab (R cl*) [float]
V2	Vstab1 (cl*) [float]

Table 18 (Continued)

V3	Vstab2 (cl*) [float]
UDP 69	Ground effect on drag, stabs
V0	Hstab (L cd*) [float]
V1	Hstab (R cd*) [float]
V2	Vstab1 (cd*) [float]
V3	Vstab2 (cd*) [float]
UDP 70	Ground effect on lift, props
V0	Prop1 (cl*) [float]
V1	Prop2 (cl*) etc [float]
UDP 71	Ground effect on drag, props
V0	Prop1 (cd*) [float]
V1	Prop2 (cd*) etc [float]
UDP 77	Wing Lift
V0	Wing1 (lift) [float]
V1	Wing1 (lift) [float]
V2	Wing2 (lift) [float]
V3	Wing2 (lift) [float]
V4	Wing3 (lift) [float]
V5	Wing3 (lift) [float]
UDP 78	Wing Drag
V0	Wing1 (drag) [float]
V1	Wing1 (drag) [float]
V2	Wing2 (drag) [float]
V3	Wing2 (drag) [float]
V4	Wing3 (drag) [float]
V5	Wing3 (drag) [float]
UDP 79	Stab Lift
V0	Hstab (lift) [float]
V1	Hstab (lift) [float]
V2	Vstab1 (lift) [float]
V3	Vstab2 (lift) [float]
UDP 80	Stab Drag
V0	Hstab (drag) [float]

Appendix I (Continued)

Table 18 (Continued)

V1	Hstab (drag) [float]
V2	Vstab1 (drag) [float]
V3	Vstab2 (drag) [float]
UDP 81	Com 1/2 Frequency
V0	Com 1 Frequency [integer]
V1	Com 1 Standby Frequency [integer]
V2	Com 2 Frequency [integer]
V3	Com 2 Standby Frequency [integer]
UDP 82	NAV 1/2 Frequency
V0	NAV 1 Frequency [integer]
V1	NAV 1 Standby Frequency [integer]
V2	NAV 2 Frequency [integer]
V3	NAV 2 Standby Frequency [integer]
V4	NAV 1 Type
V5	NAV 2 Type
UDP 83	NAV 1/2 OBS
V0	NAV 1 OBS [float]
V1	NAV 1 flag (0=off, 1=To, 2=From) [integer]
V2	NAV 2 OBS [float]
V3	NAV 2 flag (0=off, 1=To, 2=From) [integer]
UDP 84	NAV 1/2 Deflections
V0	NAV 1 hdef (degrees) [float]
V1	NAV 1 vdef (degrees) [float]
V2	NAV 2 hdef (degrees) [float]
V3	NAV 2 vdef (degrees) [float]
UDP 85	ADF 1/2 Status
V0	ADF 1 Frequency [integer]
V1	ADF 1 card (degrees) [float]
V2	ADF 1 Relative Bearing (degrees) [float]
V3	
V4	ADF 2 Frequency [integer]
V5	ADF 2 card (degrees) [float]
V6	ADF 2 Relative Bearing (degrees) [float]

Table 18 (Continued)

V7	
UDP 86	DME Status
V0	Select [integer]
V1	Distance [float]
V2	Speed [float]
V3	Time [float]
V4	Found
V5	
V6	Freq
V7	Found
UDP 87	GPS Status
V0	Mode [integer] (1=APT 2=NDB 3=VOR 11=INT)
V1	Index [integer]
V2	Distance nm [float]
V3	OBS mag [float]
V4	Course - true [float]
V5	Href dots [integer]
V6	Flag (to/from)
UDP 88	XPNDR Status
V0	Frequency [integer]
V1	ID [integer]
V2	Inter (0/1) [integer]
UDP 89	Autopilot Status
V0	Speed [integer]
V1	Heading [integer]
V2	Altitude [integer]
V3	Back Course [integer]
V4	Speed [float]
V5	Heading [float]
V6	Altitude [float]
V7	VVI [float]
UDP 90	FS/HSI/RMI/Audio/Map

Table 18 (Continued)

V0	Mode [integer]
V1	Pitch (degrees) [float]
V2	Roll (degrees) [float]
V3	HSI (select) [integer]
V4	RMI (select) [integer]
V5	Audio Select. [integer] 0=NAV1 1=NAV2 2=ADF1 3=ADF2 5=DME 10=COM1 11=COM2
V6	Map Range Select [integer] (This is the map range switch position, 0-6)
UDP 91	Marker & Audio Morse
V0	Outer Marker Morse (0/1) [integer]
V1	Middle Marker Morse (0/1) [integer]
V2	Inner Marker Morse (0/1) [integer]
V3	Audio (active) [integer]
V4	Gear Audio (active) [integer]
UDP 92	Switches 1
V0	Battery (0/1) [integer]
V1	Avionics (0/1) [integer]
V2	Nav Lights (0/1) [integer]
V3	Beacon (0/1) [integer]
V4	Strobe (0/1) [integer]
V5	Landing Lights (0/1) [integer]
V6	HUD Power (0-1) [float]
V7	HUD Brightness (0-1) [float]
UDP 93	Switches 2
V0	Pitot Heat (0/1) [integer]
V1	Anti-Ice (0/1) [integer]
V2	Auto-Brake (0/1) [integer]
V3	Auto-Feather (0/1) [integer]

Table 18 (Continued)

V4	Yaw Damper (0/1) [integer]
V5	Art Stability (0/1) [integer]
V6	FADEC Power (0-1) [integer]
V7	FADEC Power (0-1) [integer]
UDP 94	Switches 3
V0	Prop Sync (0/1) [integer]
V1	Arrestor (0/1) [integer]
V2	ECAM (Mode) [integer] 0=Engines 1=Fuel 2=Stat (control positions) 3=Hydraulics 4=Failures
V3	Radal (Bug) [float]
V4	P-Rot (Power) [integer]
V5	P-Rot (Level) [integer]
V6	Fuel Selector [integer]
V7	Auto Feather
UDP95	Pressurisation
V0	Alt (set)
V1	vvi (set)
V2	alt (actual)
V3	vvi (actual)
V4	test (time)
V5	bleed (air)
UDP96	Weapon Stats
V0	Heading (to target)
V1	ptch (to target)
V2	R (deg/sec)
V3	Q (deg/sec)
V4	rudder (ratio)
V5	elev (ratio)
V6	V (knots)
V7	Distance (feet)

Table 18 (Continued)

UDP97	Camera Location
V0	X [float]
V1	Y [float]
V2	Z [float]
V3	Pitch (degrees) [float]
V4	Heading (degrees) [float]
V5	Roll (degrees) [float]
V6	Zoom [float]
UDP98	Warnings
V0	Annunciator (Test) [integer]
V1	Annunciator (Master Accept) [integer]
V2	Master Caution [integer]
V3	Stall Warning (0/1) [integer]
V4	Ice Warning (0/1) [float]
V5	Paused (0/1) [integer]
UDP99	Anunciators – General
V0	Master Caution
V1	Master Accept
V2	Auto Disconnect
V3	Low Vacuum
V4	Low Voltage
V5	Fuel Quantity
V6	Hydraulic Pressure
UDP100	Anunciators – Engine
V0	Fuel Pressure
V1	Oil Pressure
V2	Oil Temperature
V3	Inverter Warn
V4	Generator Warn
V5	Chip Detect
V6	Engine Fire
V6	Auto Ignition
UDP101	Vspeeds

Table 18 (Continued)

V0	Vso (ktas) Stall Speed in Landing Configuration [float]
V1	Vs (ktas) Stall Speed in Cruise Configuration [float]
V2	Vfe (ktas) Max Velocity with Flaps Extended [float]
V3	Vle (ktas) Max Velocity with Gear Extended [float]
V4	Vno (ktas) Max Sturctural Cruising Speed [float]
V5	Vne (ktas) Velocity Never Exceed [float]
V6	Mmo (Mach)
UDP102	Hardware Options
V0	Pedal - nobrk [integer]
V1	Pedal - nobrk [integer]
V2	PFC - function [integer]
V3	PFC - Cert [integer]
V4	PFC - Pedal [integer]
V5	PFC - Cirrs [integer]

About the Author

Richard Garcia received a Bachelors Degree in Computer Science from Texas Tech University in 2003 and a M.S. in Computer Science from the University of South Florida in 2006. During his undergraduate program he worked three co-op tours with Lockheed Martin Missile and Fire Control and one internship with IBM. During his time as a Master's student he worked two internships with the Army Research Lab at Aberdeen Proving Ground in Maryland and worked as a graduate researcher for both the Unmanned Systems Lab and the Center for Robot Assisted Search and Rescue. Upon entering the Ph.D. program at the University of South Florida he received a fellowship from the Oak Ridge Institute for Science and Education (ORISE) as well as continued his work with the Unmanned Systems Lab and worked a third internship with the Army Research Lab.

While in the Ph.D. program at the University of South Florida, Mr. Garcia received several flight certifications for radio controlled vehicles (fixed and rotary wing). Mr. Garcia also published nine papers including a book chapter and was awarded first (2006) and second (2005) place in the Outstanding Graduate Research Poster Competition during his time at the University of South Florida.