UNIVERSITY OF CALIFORNIA
SANTA CRUZ

DESIGN, IMPLEMENTATION AND FLIGHT VERIFICATION OF A VERSATILE AND RAPIDLY RECONFIGURABLE UAV GNC RESEARCH PLATFORM

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER ENGINEERING

by

Mariano I. Lizarraga Fernandez

December 2009

The Dissertation of Mariano I. Lizarraga Fernandez is approved:

Professor Gabriel H. Elkaim, Chair

Professor Renwick Curry

Professor Vladimir Dobrokhodov

Professor Isaac Kaminer

Tyrus Miller
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

List of Tables

Abstract

Design, Implementation and Flight Verification of a Versatile and Rapidly Reconfigurable UAV GNC Research Platform

by

Mariano I. Lizarraga Fernandez

This work presents the design, development, and flight test results of a rapidly reconfigurable autopilot for small Unmanned Aerial Vehicles, along with the ground station software, and hardware-in-the-loop simulator. The autopilot presented differs from current commercial and open source autopilots mainly as it has been specifically designed to: **(i)** Enable easy modification of all the algorithms supporting the autopilot tasks, including both position and attitude estimation, inner and outer loop control and high-level navigation. This is done by using the advanced capabilities of The Mathwork's Simulink; models are directly transferred to the autopilot through the Real-Time Workshop's code-generation capability. **(ii)** Decouple the traditional tasks of position and attitude estimation, navigation, and flight control by using two Digital Signal Controllers (one for each task) interconnected via a Serial Peripheral Interface; and **(iii)** Interact *directly* with Simulink as a fully capable and versatile Hardware-in-the-Loop simulation engine.

These new capabilities are achieved by offering a seamless workflow of redesign, software simulation, hardware-in-the-loop simulation, and actual flight tests. The

autopilot capabilities are demonstrated by implementing an $\mathcal{L}_1$ output feedback adaptive controller, adopted from the newly developed theory of fast and robust adaptation. Flight test results show significant resilience to severe UAV rudder failures that are consistent with the theoretical claims of the $\mathcal{L}_1$ methodology.

This dissertation is dedicated to my family:

To my wife Erika and my daughters Valeria and Aranza for inspiring me to always try to be a better man.

To my parents for giving me the best a parent can offer: Education and fortitude of spirit.

To my brother and sister for always being there.

Acknowledgments

First and foremost I would like to thank my wife Erika for her unwavering support during all the endless hours she had to be mom and dad while I was working in the lab. And to my daughters Valeria and Aranza, for understanding my absences.

I am grateful to all my advisors for their support, patience and invaluable mentoring. Their advice and friendship has made my stay here unforgettable.

To Dr. Gabriel Elkaim for enabling a great research environment at the Autonomous Systems Laboratory. His style of guiding without imposing and suggesting without commanding has just the right balance of advice and freedom that makes working 14-hours-a-day actually enjoyable. His expertise in embedded systems has been key to the successful fruition of SLUGS. I thank him for all the time invested helping me eliminate the really-hard-to-find bugs in the low-level software drivers.

To Dr. Renwick Curry for all the time he spent explaining many subjects in aeronautics. I am grateful for his patience in explaining aircraft control, applied vector algebra and his particular view of what accelerometers measure. His support and advice during the design of the inner and navigation loops was extremely valuable. His development of the position and attitude estimation complementary filters was instrumental to the SLUGS success during flight tests.

To Dr. Vladimir Dobrokhodov for always, regardless of his schedule, having

time to sit down with me and make complex topics understandable. He introduced me to the Russian-developed Parameter Space Investigation method which played a key role in the development of the six-degree-of-freedom model of the UAV. He was the one who originally encouraged me to pursue the development of SLUGS even when some thought it was not possible.

And, to Dr. Isaac Kaminer for his support throughout my stay. I thank him for introducing me to the fascinating world of unmanned aircraft and spending many hours explaining the complicated topics of path following and $\mathcal{L}_1$ adaptive control.

Thanks are due to all my colleagues at the Autonomous Systems Laboratory (Greg Horn, David Ilstrup, Daniel Homerick, John Burr, Karl Ji-ung Choi, Alana Muldoon, Bryant Mairs, Max Dunne, Craig Moriwaki, and HyukChoong Kang) for contributing with ideas and suggestions to the development of SLUGS. I am honored to have shared this time with them and their contributions to my stay here have by no means been minor.

I would especially like to thank Greg Horn and David Ilstrup. To Greg, currently a graduate student at Stanford University, for his key role in the development of the SLUGS hardware and being the in-house safety pilot. To David, for fostering all sorts of interesting discussions from programming languages to geometry and everything in-between.

I would also like to thank my Mexican Navy colleagues at the Naval Postgraduate School. Particularly Lt. Cmdrs. Juan Sans Aguilar and Felipe Garcia Hernandez for their friendship and encouragement.

Finally, I want to thank the University of California Santa Cruz, the Mexican Navy, and the Mexican National Science and Technology Council for giving me this opportunity and partially funding my research. Especially I would like to express my deepest gratitude to Mexican Navy Admiral (Ret.) Alberto Castro Rosas for offering me the opportunity to embark on a career in scientific research. This changed my life forever and I am grateful for that.

Chapter 1

Introduction

1.1 Overview

This dissertation presents the design, development and flight testing of the **S**anta Cruz **L**ow-cost **U**nmanned Aerial Vehicle (UAV) **G**uidance, Navigation & Control **S**ystem (SLUGS). The system includes an easily reconfigurable UAV autopilot (shown on Figure 1.1), its attendant ground station software to control the UAV while in-flight, and its Hardware-in-the-loop (HIL) simulator. The system has been specifically developed to facilitate UAV research and development. The SLUGS platform successfully streamlines the process of software simulation, HIL simulation and flight test validation. It does so by using The Mathwork's Simulink model-based graphical tool to design the algorithms, simulate the UAV

behavior, and automatically generate the required autopilot firmware. Using The Real-Time Workshop the same Simulink model that is used for software simulation can be compiled and downloaded into the autopilot without the traditional error-prone step of re-coding into a high-level programming language.

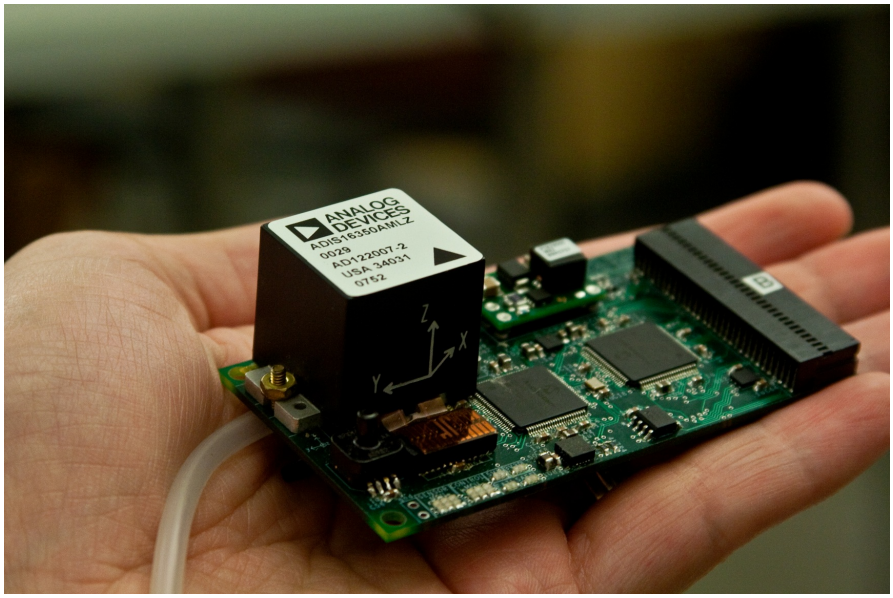


Figure 1.1: SLUGS Autopilot Hardware

Each component is tightly integrated with the rest of the system, which provides a complete solution to using UAVs as a platform for research and development. Both software and hardware have been made available under an MIT open-source license in the hopes of improving the SLUGS code-base and encouraging its adoption by other researchers.

1.2 The Importance of UAVs

Unmanned aircraft have seen a remarkable increase in usage during the last decade, partially fueled by the dramatic increase in performance to cost/size/power ratios of microelectronics that make up the core avionics. Additionally, the appearance of 16-bit and 32-bit microcontroller units (MCUs) and digital signal controllers (DSCs) have made it possible to put sophisticated onboard processing within the small power and payload budgets. This new generation of MCUs and DSCs are not only more computationally efficient, but also consume less power, making them ideal for power limited systems such as small UAVs.

Another key factor in the growth of UAV usage has been the availability of off-the-shelf components that are easily integrated into the on-board avionics. These components have allowed many universities to establish well-equipped UAV laboratories [5] to conduct research in many diverse topics including image processing and controls.

Although UAVs were once thought to be strictly military tools (where they have made impressive inroads [6, 7]), they have also become formidable tools in forest fire monitoring, border surveillance, whale and other marine mammal tracking, power-line verification, and search and rescue missions in disaster areas[8]. These and many other applications have gained the interest of researchers to employ UAVs as research platforms. Currently UAVs are being actively used to

conduct research in fault-tolerance [9, 4], tracking [10, 11], path following [12, 13], and cooperative control [14, 15] to name but a few.

Despite this growth of breadth and depth of research conducted using UAVs as main platforms, very little has really been done to the core UAV autopilot technology. Currently there exists a wide variety of commercial miniature autopilots. Piccolo [16], MicroPilot [17] and Kestrel [18] are the most visible in the US market. And it often seems like each day brings a new design to the market. In spite of this, practically all the autopilots share the same functionality of waypoint navigation and stabilization inherited from decades of manned aircraft operation. Indeed, even though there are no limitations from a human onboard, and in spite of the advances in GNC technologies, the vast bulk of the autopilots implement simple coordinated control of a UAV. Instrumented with a set of sensors all of them deliver a complete ready-to-use (but prohibitive to modify) package.

Similarly there are many in-house and open-source options replicating many if not all of the features available in their commercial counterparts. But even though these options offer full freedom to modify both hardware and software, the learning curve to get started is steep and many of the times the source code is intricate to read and even harder to understand.

1.3 Previous Research

Although much has been published regarding individual components of UAV autopilots, such as position and attitude estimation, trajectory generation, and trajectory tracking, little has been published regarding UAV autopilots as a system.

At the beginning of the decade Jang and Tomlin reported about the autopilot for Stanford's Dragonfly UAV[19, 20]. This autopilot had a modular architecture and employed a complete PC104 computer stack running the QNX real-time operating system. In 2005 Reference [21] presented a low-cost UAV test-bed for educational purposes developed at Georgia Tech. This autopilot employed a Rabbit 3000 off-the-shelf development board as its avionics processing unit, extended with an in-house developed expansion-board. This development also included a ground control station software. A year later Klenke proposed[22] a two-processor UAV autopilot that divided the processor tasks into flight control and mission application. For flight control, a single 32-bit *soft-core* microcontroller implemented in a Xilinx Field Programmable Gate Array (FPGA) was used. The same year Brisset et. al. presented one of the key players today in UAV autopilots – the Paparazzi open-source UAV autopilot[23]. This project gained a wide following for two main reasons. First it was the first complete open-source autopilot, including hardware, software and firmware; second, instead of using a full Inertial

Measurement Unit (IMU) for attitude estimation it proposed a novel thermopile-based method of computing the UAV's attitude. These thermopiles measure the temperature difference between the ground and the sky, thus greatly simplifying the attitude estimation problem, albeit sacrificing robustness. The same year, another open-source alternative was presented[24] employing Crossbow Technologies' MNAV sensor board and Stargate autopilot for waypoint navigation of small UAVs. This had the disadvantage that when compared to Paparazzi, the hardware was proprietary and thus it was not as widely adopted.

Chao et. al. [25] recently presented a survey of the key commercial UAV autopilots available in the U.S. market. His work presented a series of tables comparing some key features such as size, weight, power consumption, cost, and operating temperature among many others.

1.4 Motivation for an R&D Autopilot

While these autopilots reported in the literature and those systems available today (commercial and open-source) are very capable and provide a useful platform for UAV flights, they are all difficult to modify or extend. For instance, reconfiguring the control loops or add a feedforward term to the available control loops is a very complicated endeavor, if even possible at all. This is the main driving force behind this autopilot design (and the genesis for its development).

Research and Development (R&D) laboratories conducting GNC research require an autopilot that is easy to modify to suit their research objectives. Currently, these labs face two main options: **(i)** Augment the fixed set of autopilot commands available to the end user with their proposed control system [26] or **(ii)** invest a considerable amount of time and resources in modifying the autopilot's source code, hardware, or both (if available) to suit their needs[27].

Though option **(i)** has been shown to work in some cases, it is not ideal since it forces the control engineer to design and implement an algorithm as an *outer* (and thus slower) loop instead of an integral part of the autopilot control system; and there are certain cases, such as multiple failure tolerance, where outer loop *augmentation* is not feasible. Option **(ii)** is more promising in the long run[1], but the learning curve is steep and requires in-house expertise in either hardware design (if extending the hardware) or in a high level C/C++ programming (if modifying the firmware), or unfortunately often both.

To overcome these limitations this dissertation presents an autopilot that has been designed from the outset to be easily modifiable. On the hardware side, it not only contains a rich sensor suite, but it is also fully extendable by means of daughterboard cards connected to the autopilot via a Controller Area Network (CAN).

[1] This is debatable in academic research laboratories, where most of the student members stay for short spans of time. The expertise gained by a student is *lost* by the time he graduates and leaves the lab.

On the software side it has been designed to be reprogrammable using Simulink's Real Time Workshop (RTW) code generation tool. By harnessing the power of automatic code generation[2], anything from a simple change in controller architecture to a complete redesign can be achieved without the need of directly writing source code[3].

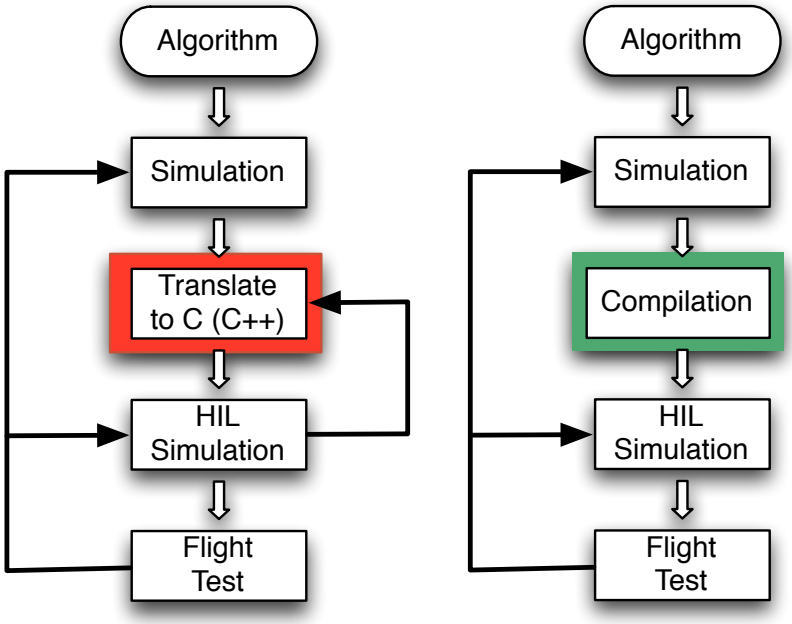
Traditionally the UAV research development cycle is as follows (see Figure 1.2a): The researcher designs the algorithm, tests it in simulation and then the algorithm is translated into compilable source-code for the avionics[4]. Once the algorithm is embedded into the avionics the correctness of the algorithm and its implementation is verified and validated in a HIL simulator. If it fails then the code translation process is revisited. If it passes, the researcher moves on to flight tests which could include further modifications to the simulation (and its subsequent C translation) or the HIL simulation.

By using Simulink and its automatic code generation tool, the translation process is replaced by a simple direct-compilation process (see Figure 1.2b) drastically

[2]Although it is widely believed that automatically generated code is neither as reliable nor as efficient as handwritten code, these beliefs are outdated. Automotive companies have used code generation to reduce development and deployment time by up to 75% [28, 29]. In 1998 Toyota presented results at the Global Automotive Engineering Seminar showing that code generated by the Real-Time Workshop is only 5% larger and 15% slower than handwritten code. And there are reports of code generation being employed in safety-critical systems such as industrial PLCs [30], navigation and flight control systems [31] and experimental hypersonic scramjets [32].

[3]Admittedly, the graphical based Simulink is its own programming language, with its own quirks and difficulties, but the speed in which reasonable proficiency can be gained is, in the author's experience, much faster than with a traditional high-level language such as C

[4]This translation process is error-prone and can insert software bugs that could potentially be discovered much later in the workflow.



(a) (b)

Figure 1.2: UAV Research workflow comparison. (a) Traditional UAV research workflow includes the error-prone process (shown in red) of converting the algorithms implemented in a simulation package (like Matlab, Simulink or LabView) to embeddable C/C++ code. (b) By eliminating the translation process and going into direct compilation (shown in green) all the tranlation errors are eliminated.

reducing the time required to go from simulation to flight tests. This also significantly reduces the potential for *bug introduction* intrinsic to the C translation process.

From the previous discussion, it is clear that the HIL simulator is a key component of any UAV research process. In general, it allows the end user to conduct overall system check, training, and, in the R&D case, algorithm verification and validation without the need to fly. Although the simulators shipped with cur-

rently available autopilots work well for operator training, they are quite limited once one tries to simulate complicated environmental phenomena or use a different aircraft model. For example, simulating a component failure, a combination of failures or employing a different plant model from the ones pre-programmed is difficult (and depending on the specific case, often not possible).

To overcome this, the autopilot presented in this dissertation uses Simulink as the main simulation engine for its HIL simulator. A Simulink model is used to send synthetic sensor data to the autopilot and receive back control commands. This setup offers flexibility not previously available and, provided that a plant model obtained by system identification is accurate, reduces the amount of parameter tuning when transitioning from HIL to real flight implementation.

1.5 Contributions

The primary objective of this dissertation is to design, implement and flight-test a low-power, low-cost, easily reconfigurable UAV autopilot, its ground station software, and HIL simulator. A secondary objective is to implement a fault-tolerant controller and flight-test it under control surface failures to showcase the overall development process. In particular, the original contributions of this dissertation are:

1. Design and implement a novel two-processor hardware architecture for a

UAV autopilot that decouples the tasks of position and attitude estimation from those related to control of the UAV.

2. Design, develop and flight test a low-cost, low-power, light-weight UAV autopilot system that is easily extendible, alterable and reprogrammable.

3. Design and develop a new architecture for a HIL simulator that is easily modifiable to allow the simulation of complex situations or environmental phenomena.

4. Design, develop and flight test a sideslip compensator to reduce the effect of rudder failures in the UAV flight performance.

5. Implement and flight test a fault-tolerant adaptive controller as an integral part of the autopilot.

1.6 Dissertation Organization

The rest of this dissertation is structured to describe in detail each of the components that comprise the system along with the overall design trade-offs that led this specific implementation.

Chapter 2 presents an overall view of the three main components of the system. It starts by describing the autopilot, its hardware, and software architectures.

It then presents the ground control station architecture and the ground station software feature set.

Chapter 3 presents the design and development of the HIL simulator. It briefly presents the methodology employed in obtaining an accurate six-degree-of-freedom (6-DOF) model used in all the software and HIL simulations. It then presents the new architecture and its advantages over traditional UAV HIL simulators.

Chapter 4 discusses the guidance, navigation and control components of the autopilot. The inner (stabilization) and outer (control) loops are presented. This chapter also presents the sideslip compensator designed to reduce the effect of rudder failures on the UAVs performance. It concludes by presenting the waypoint following (navigation) implementation, some measures taken to reduce the effect of wind on navigation, and software simulation results.

Chapter 5 discusses the relevance of UAV control surface failure-tolerance. It also introduces the adaptive controller employed, the key factors in deciding to employ that particular controller, and its implementation. The chapter concludes with software simulation results.

Finally, Chapter 6 presents flight test results (divided into four flight tests) and discusses each set of results. The dissertation's conclusions and suggestions for future research are presented in Chapter 7.

Chapter 2

Overall System Design

2.1 Introduction

The majority of UAVs share the same operational architecture regardless of size (see Figure 2.1). Onboard the aircraft an autopilot stabilizes the aircraft and takes care of (typically waypoint-driven) navigation[1]. The autopilot has at least two operating modes: autonomous and manual. When in autonomous mode, the autopilot directly commands the UAV's control surfaces and is thus fully responsible for the aircraft. When in manual mode, a trained pilot takes control from the ground (via a *pilot console*) and it is the pilot who becomes responsible for the aircraft[2].

[1] In some cases, although not very common, the autopilot is also tightly integrated with the mission sensors and exercises control over these too.

[2] Another mode, common in large UAVs is the pilot-augmentation mode, were the pilot, through a control joystick controls turn-rate and climb-rate as opposed to directly commanding

Most UAVs are also equipped with additional sensors (dictated by the UAV's mission), for example visual or Infra-Red (IR) cameras, that are commonly referred to as the UAV's *payload*.

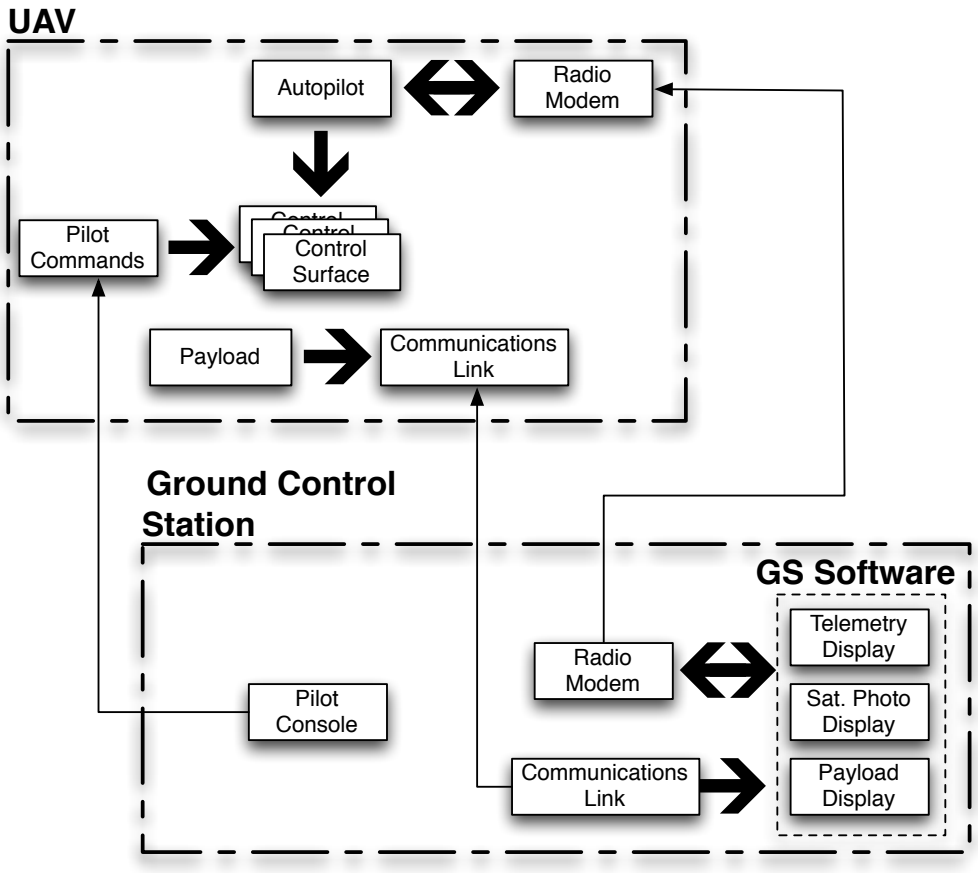


Figure 2.1: General UAV Architecture

Control and monitoring of the UAV is conducted from a *ground control station* (GCS) that receives, via a radio link, the UAV's sensor telemetry, controls its navigation, and displays the payload data. The GCS contains also the pilot's

the control surfaces.

console which is used as a safety measure in case of an autopilot malfunction. For those UAVs that do not have autonomous launch or landing, the pilot's console is also used during takeoff and landing. All of the UAV mission planning and control is exercised from the GCS. The core component of the GCS is the ground station software. This is the component where the operator interacts with the UAV and it typically allows him to modify the UAV's mission while still airborne. It also displays the aircraft's position in a geo-referenced satellite photo or chart.

A third, and optional, component of the system (not shown in Figure 2.1) is the HIL simulator. Although end users mostly use it for operator training, it is a vital component for a research facility. The HIL simulator places the autopilot in a mode where its sensor data is sent by an external computer as opposed to the internal sensor suite. This allows the autopilot to operate using simulated data and thus "fly" the aircraft without ever leaving the ground. This allows end-users and researchers alike, to safely test the system and verify its correct operation.

The UAV system presented in this dissertation also follows the general architecture previously discussed. But even though the overall architecture is similar to almost all other UAV systems, this and the next chapters will show that it differs significantly in the details of the key components, namely, the autopilot, the ground station software, and the HIL simulator. The presented architecture concentrates on providing a rapidly modifiable UAV platform for R&D.

2.2 Autopilot

Most UAV autopilots available today have somewhat similar architecture: on the firmware side they provide stability augmentation, and simple waypoint navigation, as previously discussed; on the hardware side, most of them use the classical sensor suite: IMU, GPS, pressure sensors, and a radio downlink for a GCS. The architecture implemented in the SLUGS autopilot differs from existing architectures by dividing the regular tasks performed by the autopilot among two independent processing units. This provides a higher processing power that allows for implementation of computationally complex control algorithms. By having two processing units, one for position and attitude estimation and the other for navigation and control, researchers can focus on either one of these topics while leaving the other unchanged; significantly reducing the risk of mutual disruption. These two processing units are referred as the sensor and the control DSCs respectively. By having two processing units the autopilot has twice the processing power with negligible impact on the power and size restrictions.

The autopilot operation is as follows(see Figure 2.2): The sensor DSC reads the data from each of the sensors in the sensor suite. This data is provided to a sensor fusion algorithm that computes the aircraft attitude and position. The attitude and position information is packed in a predefined communication proto-

col (described in Section 2.2.3) and sent[3] via SPI to the control DSC. The control DSC, based on the data received from the sensor DSC and the current commands received from the GCS, generates the commands for the control surfaces and schedules the data to be sent to the ground for telemetry reports.

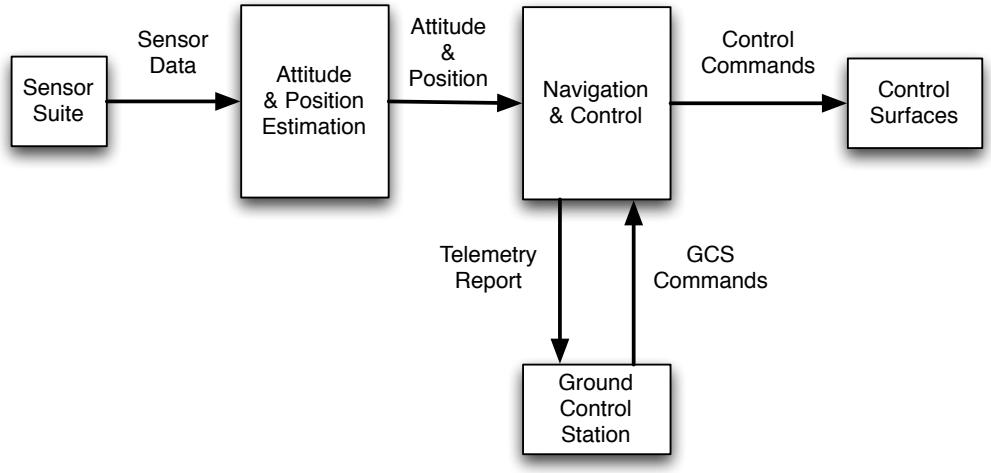


Figure 2.2: High level autopilot data flow.

2.2.1 Hardware Architecture

The hardware architecture (shown in Figure 2.3) was designed to support a rich sensor suite, provide enough processing power for attitude estimation and control tasks, and be reprogrammable using Simulink's graphical programming language.

[3]In reality much more information than just attitude and position data is sent to the control DSC. This will be described in detail in Section 2.2.3

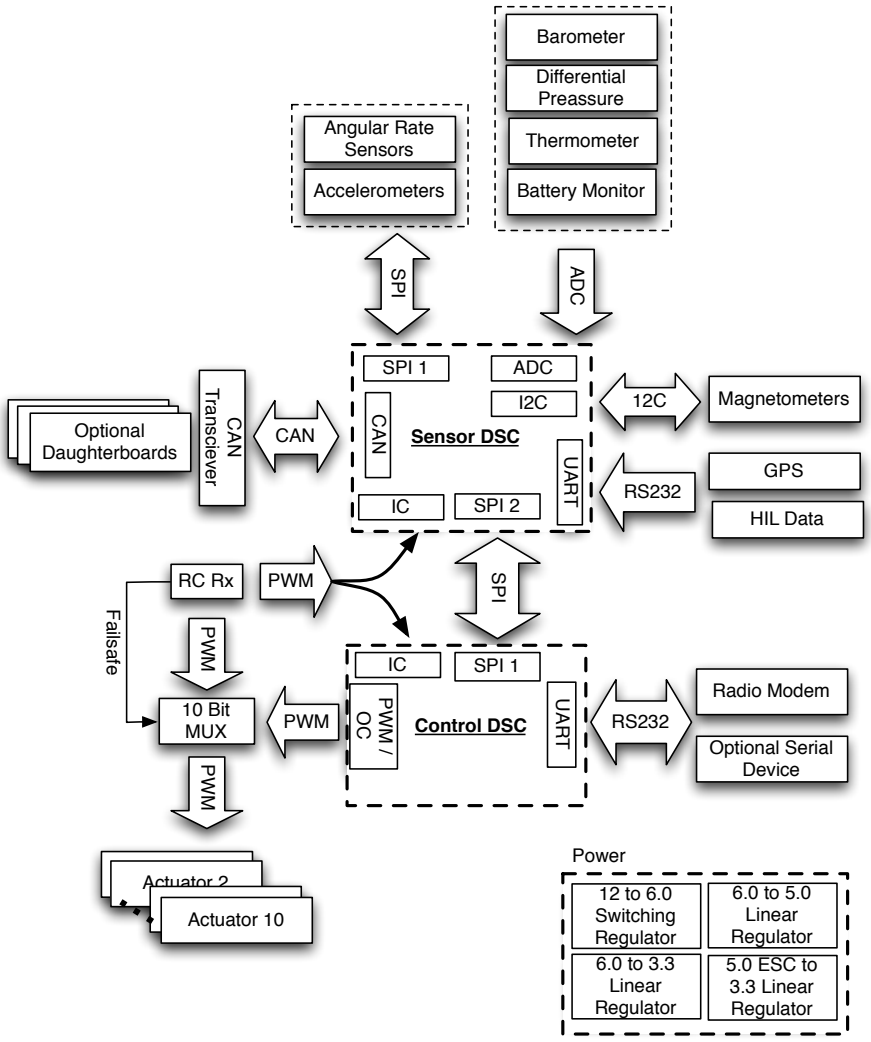


Figure 2.3: Block Diagram of the Autopilot Hardware Architecture.

At the core of the autopilot sit two Microchip dsPIC33FJ256MC710 DSCs [33] interconnected via a high-speed Serial Peripheral Interface (SPI) bus. These DSCs, further referred as the *sensor* and the *control* DSCs, are configured to run at 80 Mhz and are able to execute up to 40 Million of Instructions per Second

(MIPS). The main sample period for their allotted tasks is 10 mS, thus they execute their task at 100 Hz sampling frequency.

The individual integrated circuits that comprise the autopilot[4] were selected based on three key deciding factors: **(i) Functionality** – that the component included features that reduced either the required external components (such as capacitors, inductors, etc.), the required software for utilizing its data (such as factory calibration or temperature compensation), and the interface to communicate with the device (where digital was always preferred over analog for simplicity). **(ii) Size** – great care was taken to make the autopilot as small as possible without sacrificing functionality, and; **(iii) Availability** – all the components selected were chosen so that they were easily available in the market with no restriction to its purchase in countries different than the U.S; this was to attempt to make the SLUGS platform available to the widest possible group of research labs.

Conceptually, there are four main blocks in the autopilot: The sensor DSC, the control DSC, the pilot command's multiplexor, and the power supply. Each of these is described in more detail in the following subsections.

2.2.1.1 Sensor DSC

The sensor DSC, as its name implies, reads data from all the sensors available to the autopilot, performs the low level transformations to generate engineering

[4]For a detailed list of the actual electronic components used in the autopilot see Referece [34]

units, and provides them to the position and attitude estimation algorithm. This data is also packed in several communications protocol sentences and delivered to the control DSC. It makes use of most of the peripherals available in the DSC as described below:

- **Serial Peripheral Interface (SPI)**: Both SPI ports available in the DSC are used. SPI1 is used to query and configure a tri-axis inertial sensor which includes the angular rate sensors (rate gyros) and the accelerometers. As previously described, data is sent to the control DSC via the SPI2 port.

- **Analog to Digital Converter(ADC)**: Four of the 32 ADC channels available are used to query the analog sensors. The barometer and the differential pressure sensors are used to compute altimetry and airspeed respectively. The thermometer is used to temperature-compensate all the sensor data and provide general information about the autopilot's temperature to the GCS. The battery monitor reports the current voltage supplied to the autopilot and is used to monitor battery life.

- **Inter-Integrated Circuit (I2C)**: One of the two I2C ports available is used to query and configure the magnetometer triad.

- **Serial (UART)**: Both UART ports available are used. Port one is used to configure the GPS and read off its NMEA 0183 serial stream. Port two is

used when in HIL mode to read simulated data from the simulation PC to replace the data provided by the onboard sensors (see Chapter 3).

- **Input Capture(IC)**: All of the eight IC module channels are used. Six are employed to read the Pulse Width Modulation (PWM) commands sent from the pilot's console to the Commands Multiplexor (see subsection 2.2.1.3). One is used to monitor the pilot's console commanded mode; either manual or automatic mode. The last one is built in is a feature for UAV engines that contain a tachometer to read the actual motor speed.

- **Controller Area Network (CAN)**: The CAN1 port is connected to an in-board, properly terminated CAN bus, through a transceiver. This allows the autopilot to connect to external daughterboards without the need of an actual hardware modification, thus providing great expandability with little overhead.

2.2.1.2 Control DSC

The control DSC receives commands from the ground station, and attitude and position computations from the sensor DSC. As its name implies, it executes the control commands that actuate the UAV's control surfaces.

Although not as peripheral-intensive as the sensor DSC its peripherals are used as follows:

- **Serial Peripheral Interface (SPI)**: The SPI1 port is used to read the computed attitude and position as well as the sensor data sent from the sensor DSC.

- **Input Capture (IC)**: In the same way as in the sensor DSC, all of the eight input capture module channels are used. In this case, the read PWM commands are used as starting trim conditions for the UAV's control surfaces.

- **Pulse Width Modulation (PWM/OC)**: The autopilot can drive up to ten PWM channels for servos. Four of them are obtained from the motor control module available in the DSC. The remaining six are obtained from the Output Compare (OC) module[5].

- **Serial (UART)**: It is through this serial port that the autopilot reports all the telemetry sentences to the GCS and receives commands from it. It is directly connected to the UAV's radio modem.

2.2.1.3 Commands Multiplexor

A fundamental safety requirement is that in the case of an autopilot malfunction and within visual range, the safety pilot can take control of the aircraft from

[5]The motor control module has the advantage over the OC module that the former does not use any extra resources from the DSC, while the latter requires the configuration of a device timer to properly generate the PWM signals

the ground through the pilot's console. This backup system not only has to be reliable but also completely independent (power, signal and radio) from the autopilot. The pilot's console commands are received onboard the aircraft by an off-the-shelf Radio Control Receiver (RCRx). This receiver produces independent PWM outputs for each channel. Five channels are used in total; four contain the pilot's control surface commands (ailerons, rudder, elevator and throttle) and the fifth is the commanded autopilot mode selector (autonomous or manual).

The commands multiplexor architecture(shown in Figure 2.4) is comprised of an an 8-bit PIC12F683 MCU and a 10-bit digital multiplexor. The MCU's IC module reads the RCRx's mode selector PWM command and configures the 10-bit multiplexor with two digital input/output (DIO) lines. The multiplexor, based on its configuration, routes the PWM signals from either the autopilot or the RCRx, to the control surfaces' servos.

2.2.1.4 Power Supply

The power supply architecture (shown in Figure 2.5) has been designed to provide power to the avionics independently of the airborne-critical systems. For this purpose two Lithium-Polymer (LiPo) batteries are employed. A 1325 mAHr to power the avionics and radio modem, and a 5000 mAHr to power the airborne-critical systems.

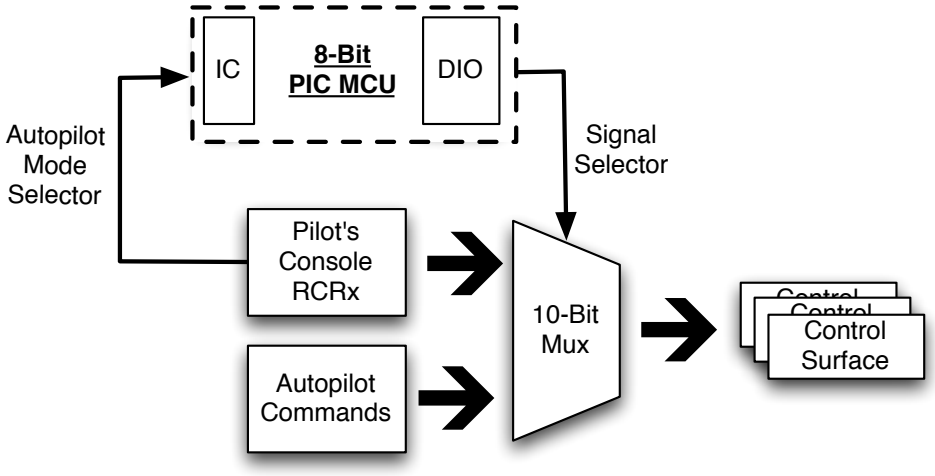


Figure 2.4: Autopilot – Pilot Commands Multiplexor Subsystem.

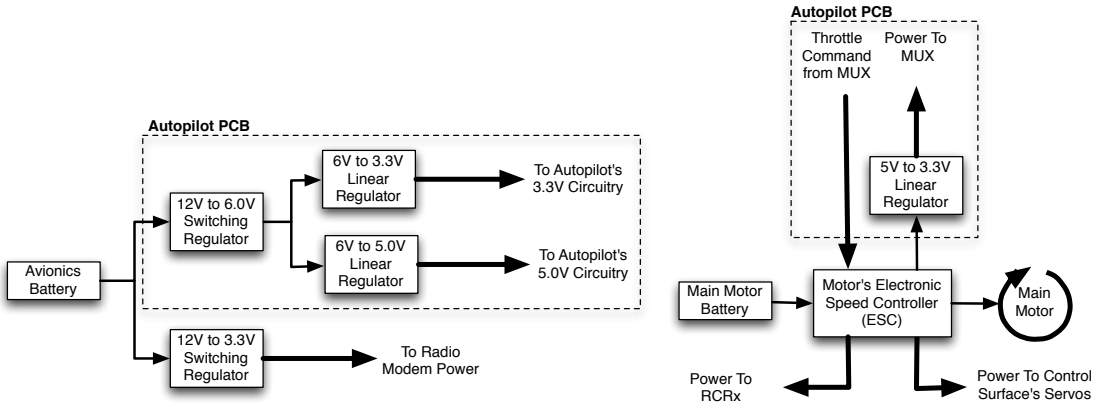


Figure 2.5: Power supply and distribution on the UAV. (a) The avionics battery provides power to the autopilot and the radio modem. (b) The main motor battery provides power to the air-critical systems such as the servos, the commands multiplexor, the RCRx, and the main motor.

The avionics battery output is connected to a switching regulator (in the autopilot PCB) that drops the voltage from 12 to 6 volts. Further down the power

line, two low-dropout (LDO) linear regulators lower the voltage to 5.0 and 3.3 Volts for the circuitry in the autopilot[6]. An independent switching regulator board drops the voltage from 12 to 3.3 volts to power the UAV's radio modem.

The main motor battery powers an off-the-shelf 60 Ampers electric motor Electronic Speed Controller (ESC). The ESC performs the following tasks: **(i)** Provide 5.0 V power to the RCRx; **(ii)** Provide power to the control surfaces' servos; **(iii)** Provide power to a 5 to 3.3V LDO linear regulator (in the autopilot PCB) which in turn powers the commands multiplexor described in Section 2.2.1.3, and; **(iv)** Receive throttle commands from the commands multiplexor and generate the necessary signals for the UAV's main motor. This architecture guarantees that in the case of an autopilot power loss or malfunction it does not impact the pilot's capability to take control of the UAV.

The total power consumption of the autopilot and the commands multiplexor, in normal operation is of 3.1 Watts at 12 Volts.

2.2.2 Low-level Software Architecture

To achieve the goal of providing a fully Simulink-reprogrammable autopilot there was a need to develop a set of Simulink sources and sinks[7] to interact

[6]These regulators and its accompanying capacitors also serve the purpose of reducing some of the noise intrinsic to the output of switching regulators.

[7]In Simulink, sources are those blocks that *source* (i.e. produce) a signal and sinks are those where a signal is terminated or *sinked*.

with the autopilot hardware. Some of them where directly available through a dsPIC Embedded Target (ET) Simulink blockset. This ET makes available, via a Simulink library, a set of blocks to either facilitate access to the processor's peripherals or write and configure these directly in C[8]. The ET employed by the autopilot presented here is that of Reference [35]. Although the selected ET offered ready-to-use blocks for most of the DSC's peripherals it was decided early in the design process that these would be rewritten in C. This aimed to provide full control of the implementation and at the same time offer greater flexibility. For instance, to use less *processor time* while sending data through the serial port it was necessary to use the Direct Memory Access (DMA) feature of the DSC. The ET's UART block did not offer this capability, and thus it was necessary to rewrite it in C and provide it as a C-function call block in Simulink.

The low-level software architecture of both DSC's can be thought of in three main sections: the sources (either written in C or provided by the ET) that provide data to the main algorithm, the algorithm itself(which is implemented exclusively in Simulink), and the sinks which send out the data to the next recipient. The sources and sinks have been designed in such a way that when reconfiguring the autopilot, only the algorithm itself is modified leaving the rest unchanged.

[8]It also facilitates the compiling and linking process by providing hooks to the compiler from within Simulink. One can create the Simulink model, generate code, and compile directly within Simulink.

The following subsections present a brief description of the low-level software architecture and implementation in the sensor and control DSCs.

2.2.2.1 Sensor DSC

The sensor DSC's low-level software architecture (shown in figure 2.6) is a combination of blocks provided by the ET (shown in blue), blocks implemented in C (shown in yellow), and basic Simulink blocks (shown in white).

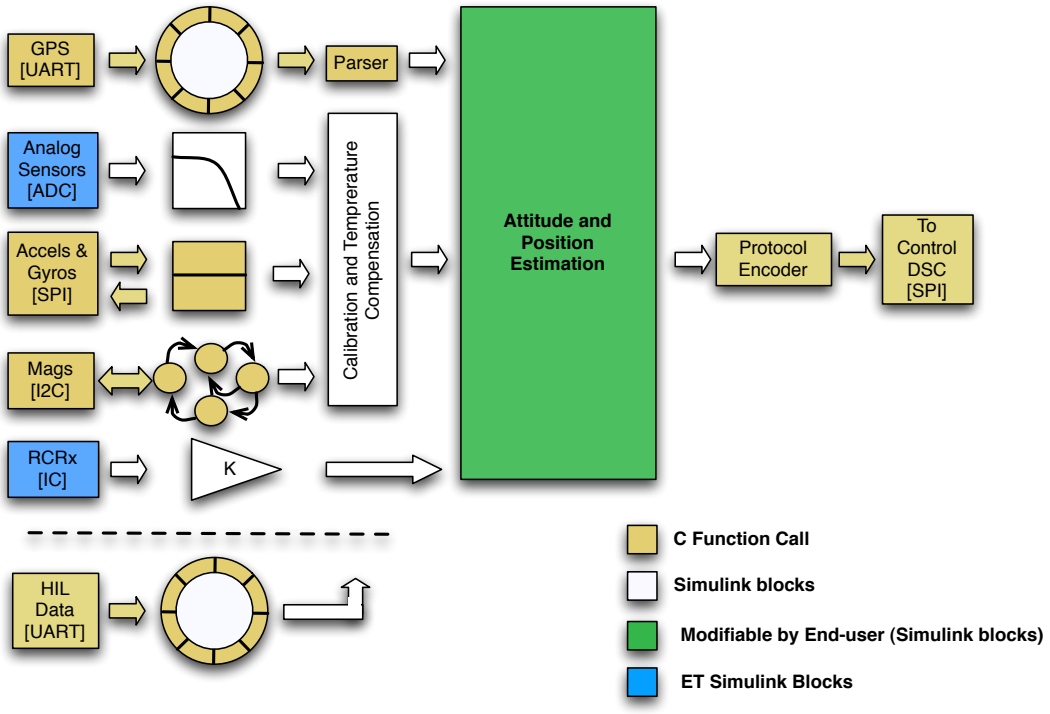


Figure 2.6: Low-level Software Architecture for the Sensor DSC

The GPS serial stream is read and placed into a circular queue[9]. An NMEA 0183 [36] protocol parser reads data from the circular queue and decodes the protocol sentences producing position, velocities and heading information.

The analog sensors are directly read from the hardware and passed through a digital low-pass filter bank to attenuate sensor noise. The accelerometers and rate gyros are queried and recovered using a protocol defined by the device's manufacturer. The magnetometers are configured and queried through an interrupt-driven state machine[10]. The analog sensors, accelerometers, rate gyros and magnetometers readings are passed through a calibration and temperature compensation block to produce temperature compensated data in meaningful units. Data being received by the RCRx is received and scaled from PWM duty cycle to angular values.

In case of having the autopilot in HIL simulation all the sensor data is replaced by externally generated data received via a serial port circular queue.

Regardless of whether in HIL simulation or real flight, the attitude and position estimation algorithm (shown in green) processes the data and presents its results

[9]Circular queues data structures were employed throughout the development of the low-level device drivers. This is a very common design pattern employed in embedded systems that enables the decoupling between two asynchronous processes. It enables a *producer - consumer* paradigm where one piece of software *produces* the data (thus fills the queue) and another *cosumes* the data (i.e. empties the queue) without one interfering the other.

[10]The interrupt-driven state machine design pattern was selected due to the way the I2C protocol works. Devices start a communication sequence by issuing an I2C *start* and after that communication is exchanged based on the state of the devices. For further understanding of the I2C Bus the reader is referred to Reference [37]

to a communications protocol encoder which separates it into sentences that are transmitted to the control DSC.

2.2.2.2 Control DSC

The control DSC's low-level software architecture (shown in figure 2.7) is less resource-intensive than that of the sensor DSC. The data sent from the sensor

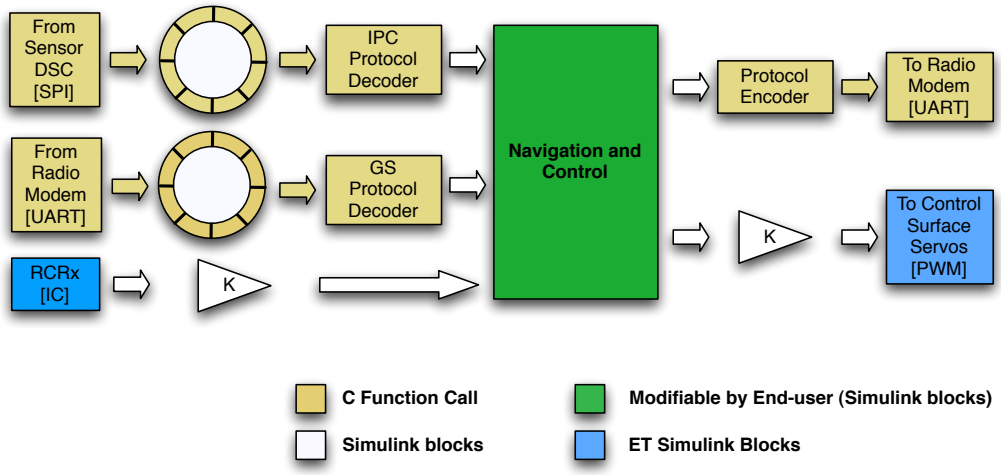


Figure 2.7: Low-level Software Architecture for the Control DSC

DSC is received and placed in a circular queue. The inter-processor communications (IPC) protocol decoder reads from the buffer and provides it to the navigation and control algorithm (shown in green). In a similar manner, data received from the radio modem (containing GCS commands) is stored in a separate circular queue. The GCS commands protocol decoder reads the buffer, interprets

those commands, and passes them on to the navigation and control algorithm. The data being received by the RCRx is received and scaled from PWM duty cycle to angular values and passed to the guidance and control algorithm to be used as trim conditions for the manual-to-autonomous transition.

The guidance and control algorithm generates the required control surface commands which are scaled from angular values to PWM duty cycle and sent to each of the control surfaces' servos. These commands are also passed on (along with the data sent from the sensor DSC) to a protocol encoder which splits it into different sentences, schedules them for transmission, and sends them to the radio modem.

2.2.3 Communications Protocol

The complete autopilot system generates 168 different meaningful variables. Attitude, position, sensor biases, temperature, absolute and differential pressure, control surface commands, controller gains, and many more are generated on every cycle. These values are not only used by the autopilot but also sent to the GCS and logged to facilitate debugging and post-flight analysis. To do so, a communications protocol assembles the data into coherent sets and calculates a checksum which is appended to the message. The checksum guarantees the validity of the sentence to the receiver. This becomes relevant when the communication is asyn-

chronous, such as in the case of the sensor DSC to control DSC inter-processor communication.

The autopilot presented here implements a very simple, yet effective binary communications protocol where each message contains a four-byte header and a three byte trailer (see Figure 2.8).

The header consists of two begin-of-message (BOM) indicators ($@); a message identifier, which specifies the type of message, and the message length.

The trailer contains two end-of-message (EOM) indicators (*@) [11] and the XOR checksum, which is computed from the message identifier to the last end-of-message byte.

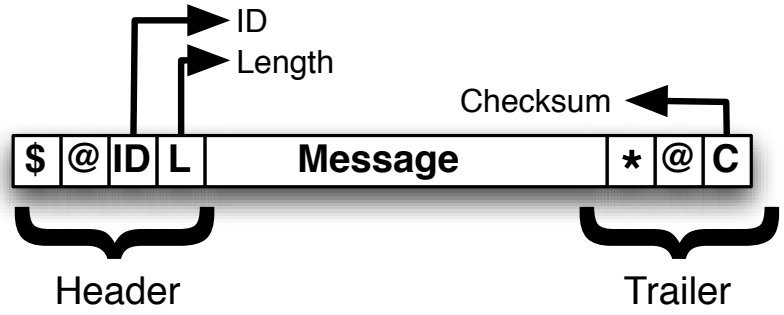


Figure 2.8: Communications Protocol Message Structure

[11] Two bytes are used for the BOM and EOM delimiters to make the procol more robust. If a single byte were to be used, the protocol parser would have no way of distinguishing if the byte is indeed a BOM indicator or just a payload byte. By using two bytes the probability of the payload actually containing two sequential bytes identical to the BOM and EOM patterns is $\frac{1}{65536}$.

There are a total of 34 different messages used to pass information between the sensor DSC, the control DSC and the GCS. The description of each message can be found in Reference [34].

2.3 Ground Control Station

As previously described, the GCS consists of pilot console, a radio modem to communicate with the autopilot and its ground station software[12]. The GCS and autopilot are both radio independent. This means that they do not depend on any specific radio modem. The autopilot is even flexible enough to take either 3.3 Volts UART signal or RS232 level for the radio modem serial stream. The only requirement is that both radios have enough bandwith to support serial communications at 115 Kbps baud rate.

For the UAV flight tests presented in this dissertation, the radio modem employed was a Digi XTend OEM 900 MHz RF Module with 1 Watt power output. Two identical modules were employed, one in the GCS and one onboard the UAV. Both were configured to transmit and receive data at a 115 Kbps baud rate.

The autopilot is also independent of which pilot console and RCRx is employed. The only requirement is that the PWM signals received from the pilot console and generated by the RCRx are within 3.0 and 5.1 volts.

[12] Although the UAV was equipped with an off-the-shelf video camera as its payload, this will not be discussed since it is not directly related to the work presented in this dissertation.

For the UAV flight tests presented in this dissertation, the pilot console was an off-the-shelf radio control for hobby RC airplanes. The model was a Spektrum DX7 2.4 GHz spread spectrum controller with support for up to 7 independent channels (see Subfigure 2.9a). It communicates directly with the Spektruum AR7000 RCRx onboard the UAV (see Subfigure 2.9b).

(a) (b)

Figure 2.9: Pilot Console and Radio Receiver. (a) Spektrum DX7 pilot console. (b) Spektrum AR7000 RCRx. From Reference [1]

The ground station software was explicitly developed for the autopilot and is described in the following subsections.

2.3.1 Ground Station Software Feature Set

The ground control station software is an application (see Figure 2.10) developed in C++ that runs on any off-the-shelf PC with Windows XP operating system. It reads the communications protocol stream coming from the autopilot over a serial port. It provides an intuitive way, via a graphical user interface, to control the aircraft when in autonomous mode. It is also used in HIL simulation mode to pass data between the autopilot and the simulation engine (see Chapter 3).

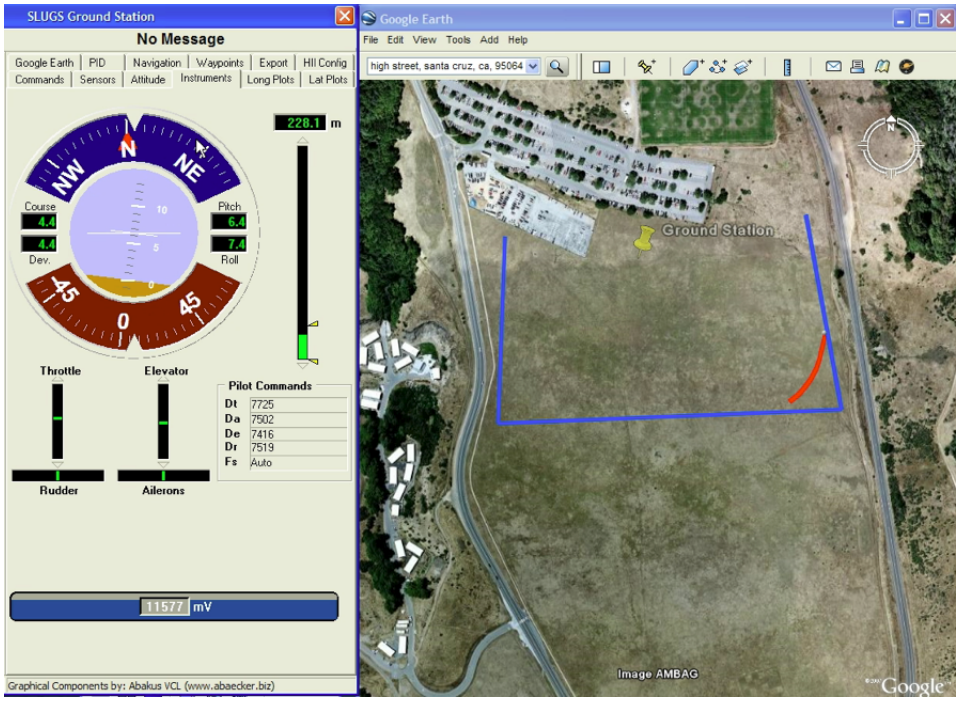


Figure 2.10: Ground station software screenshot during a flight test.

Specifically, the purpose of the ground station software is to provide the operator with an interface to:

- **Display Data.** Receive and display information from the sensors and the onboard calculations such as: attitude, position, control surface commands, pressure, etc.
- **Display the Trajectory.** Send the received position to Google Earth for display of the UAV's trajectory.
- **Command the UAV.** Send either low-level commands such as turn-rate, or high-level commands such as way-point tracking.
- **Tune the Gains.** Modify the gains for the various control loops on board.
- **Configure Waypoints.** Capture and update the waypoints directly from Google Earth and upload them to the autopilot.
- **Configure the HIL Simulator.** Configure the simulation PC's IP address and UDP ports (see Chapter 3).
- **Log Data.** Generate, as the incoming data stream is decoded, a Matlab-ready log file for post-flight analysis.
- **Generate Real Time Plots.** Present, in real-time, plots of the commanded vs. measured values used to control the UAV's attitude and position.

Figure 2.10 shows the ground station software in operation during a flight test at UCSC.

2.3.2 Software Architecture

During normal operation (as opposed to HIL simulation, which will be detailed in Chapter 3) the ground station software data flow (shown in Figure 2.11) is as follows: The GCS radio modem receives the data stream from the UAV packed in the communications protocol. This data stream is decoded into physical values and is used to **(i)** update the software's Graphical User Interface (GUI) and the data log; and **(ii)** to update a KML[13] file from which Google Earth reads and displays the UAV's trajectory.

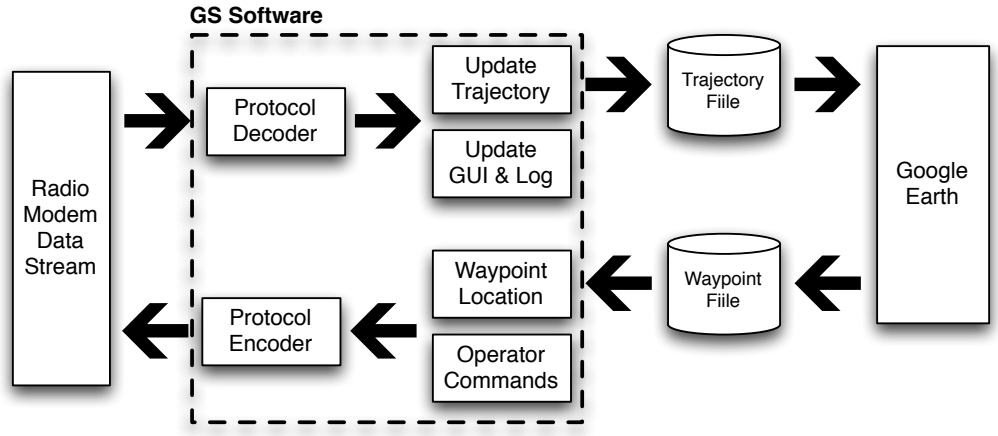


Figure 2.11: Ground Station Software Data Flow

[13]A KML file is a special type of XML file that Google Earth uses to present data in its interface.

When the GCS operator needs to change any of the UAV's waypoints, it uses Google Earth's graphical interface to layout the waypoints in a geo-referenced satellite image and then save them to a file. The ground station reads that file, encodes the new waypoints in communications protocol sentence and sends it to the UAV.

Autopilot commands and configuration values, such as control system gains, are set directly in the ground station software. These values are encoded in the appropriate communication protocol sentences and then sent to the UAV via the radio modem.

Chapter 3

Hardware in the Loop Simulator

3.1 Introduction

Hardware-in-the-loop (HIL) simulators are essential tools for flight control systems development. In essence, a HIL simulator generates synthetic data as if the onboard sensors were recording data from an actual flight. This data is sent onto the avionics which in turn, produces control commands accordingly. These control commands are relayed back to the simulator closing the control loop [38]. Specifically, UAV HIL simulators are generally comprised of: **(i)** The flight-ready avionics, **(ii)** software to simulate the aircraft dynamics, engine, weather, actuator dynamics, etc. which, in this dissertation are referred as the *plant model*; and **(iii)** the hardware and software that allows the plant model simulator to send data to the avionics and receive control commands back.

HIL simulators have long been used for rapid prototyping of flight control systems in missiles[39, 40], helicopters[41] and fixed-wing aircraft [42]. Their wide adoption is mainly due to the fact that HIL simulators allow the saving of time and money compared to real flight tests [43]. For instance, in missile development, it is estimated that the cost of a single firing test covers the cost of between 3,000 and 10,000 HIL simulated firings [39]. Another advantage of HIL simulators is that they allow preliminary runs of high-risk scenarios such as fault-tolerance [4, 43] without actually exposing the hardware to danger. This is extremely useful specially in the initial phases of the design process. One final advantage is that, provided that the software simulating the plant model is accurate, the amount of parameter tuning when transitioning from HIL to real flight implementation is greatly reduced.

There have been recent reports in the literature of several research laboratories employing HIL simulators for rapid prototyping of UAV flight control algorithms. Johnson and Schrage reported on Georgia Tech's rotary wing UAV research platform, the GTMax, which included a HIL simulator developed in-house written in C++ [44, 45]. Tisdale et al. reported on UC Berkeley's UAV platform which made use of the commercially available Piccolo autopilot[16] and its HIL simulator to develop and flight test autonomous collaborative patrolling [46]. More recently Dobrokhodov et al. reported on the Naval Postgraduate School's custom

built serial interface [31] and the Rapid Flight Test Prototyping System which made extensive use of the HIL simulator available with the Piccolo autopilot for development of vision-based target tracking, 3D path following, UAV control over the network and high-resolution imagery on the fly[5]. Nevertheless, these HIL simulators share a common limitation: a lack of a simple and intuitive way to modify the plant model model and its operating environment.

Many of the current Commercial Off-The-Shelf (COTS)[16, 17, 18] and open source[23] autopilots offer software simulators to perform HIL testing, and training. This software allows the end user to conduct overall system check, training, and, in the R&D case, algorithm verification and validation without the need to fly. The HIL simulators are often important to refining and developing algorithms before the flight hardware is ready or integrated into the airframe.

Although the HIL simulators shipped with currently available autopilots work well in general, they are quite limited once one tries to simulate complicated phenomena or use a different aircraft model. For example, simulating a component failure, a combination of failures or employing a different plant model from the one pre-programmed is difficult (and often not possible).

Due to these limitations, laboratories conducting research in control systems require a more flexible HIL simulator. Ideally the HIL simulator should be an enabling tool that can easily be modified to closely replicate the conditions under

which real flight will be conducted. It should be capable of being gradually extended from a simple linear plant model during the initial design phases, all the way to a full and detailed non-linear one once the project so requires. Another desired feature is that the end-users should be able to add plotting and logging capabilities with ease. To the author's knowledge there is currently no HIL simulator readily available that satisfies these goals for UAV development. Currently available HIL simulators offer, on the commercial side, a limited capability to change the plant model by means of only a few modifiable parameters; while the open source ones offer freedom to modify but have a steep learning curve and require significant proficiency in a high level programming language. These key factors drove the decision to develop a flexible and easily modifiable HIL simulator. This newly developed HIL simulator has the ability to simulate any dynamic model using the Simulink [47] modeling environment. Simulink-based design is used to generate synthetic sensor data for a UAV autopilot and has the capability to receive the control commands. This offers both a flexibility and ease of use not previously available.

From the previous discussion it is easy to infer that central to the HIL simulator is the mathematical description of the plant model. Undoubtedly the most crucial part of the plant model is that of the aircraft dynamics. Much has been written in the literature of how to go about obtaining an accurate plant model; from

simple approximation using a panel method [48], to a full regression model [49], or a hybrid between the two [50]. The system identification presented in this dissertation is similar to that of Reference [50] in the sense that it first employs a panel method, but it differs in the fact that the final refinement of the obtained aerodynamic coefficients is done by employing the lesser-known Parameter Space Investigation (PSI) method as described in Reference [2].

This chapter presents a new versatile modeling capability that greatly improves the probability of success in a new control system design. This new Simulink-based design allows the simulation of complex dynamic models and phenomena in one of the most versatile and convenient design environments supported by a wide variety of Control Design toolboxes. This eliminates the need for high-level language programming and coding to simulate the environment under which the control-loop design will be tested.

3.2 System Identification

As previously discussed, adequate modeling of the aircraft and the environment is key in the success of any HIL Simulator. Although very important and time consuming by itself, the system identification part of this dissertation will be presented briefly, providing only enough details to understand the process. Although the aerodynamics of the small, conventional high-wing UAV are tradi-

tional and in general not difficult to identify, there are multiple steps to obtain a reliable model. The following overall strategy of system identification (see Figure 3.1) was employed:

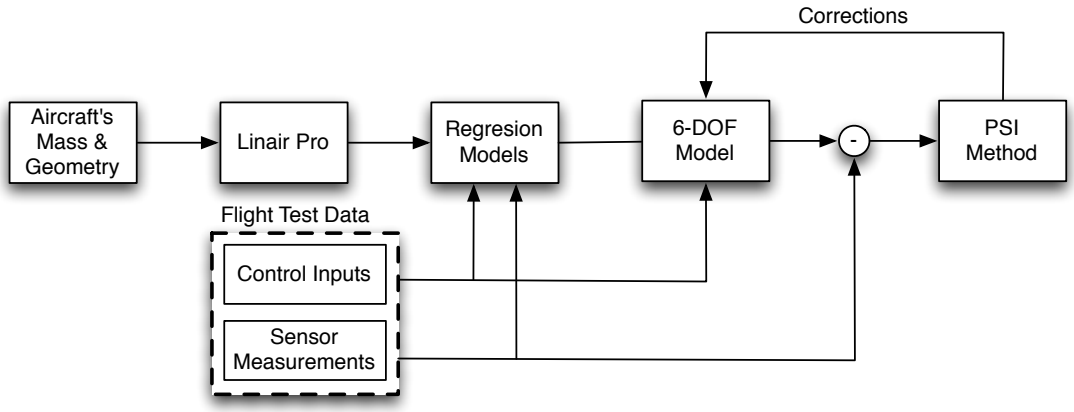


Figure 3.1: System identification architecture

Starting with initial measurements of the mass and geometry characteristics of the aircraft, a panel method software (LinAir Pro[51]), was employed to produce initial estimates of the 27 stability derivatives that define the basic aerodynamic terms. These terms were represented by the traditional regression model [49] for the aircraft plant model and are further described in Appendix A. Next a series of flight experiments were designed to measure the UAV's response to a single-input (one channel at a time) doublet command[1] in open loop. The data were acquired at a 100 Hz data sampling rate during the interval of the preprogrammed

[1]Instrumentation of the airplane to measure data adequately followed general instrumentation recommendations that can be found in Refs.[52, 49].

doublet. A series of doublets for open-loop response were performed separately in aileron, rudder and elevator channels in order to excite the UAV so that the data contained sufficiently rich information for accurate system identification. After that, the classical regression based off-line identification technique [49] was used to improve initial estimates of basic aerodynamics and control derivatives of the nominal aircraft model. The traditional regression technique was augmented with the lesser-known Parameter Space Investigation (PSI) method (as described in [2]). PSI was used to assist in identifying the structure of the regression model for the typical flight regimes. The motivation behind complementing the traditional regression methods is due to the fact that the persistency of excitation condition is very difficult to guarantee in a real flight-identification experiment[49]. By employing the multi objective *identification* in the PSI methodology with the usually information-limited flight data resulted in a high accuracy plant model.

The following subsections describe in more detail each of the steps in the process.

3.2.1 Moderate Fidelity Model

The values of the dimensionless aerodynamic coefficients in the equations of motion of the UAV (presented in Appendix A) are obtained from a Taylor series expansion about a trim flight condition. Typically these values are experimentally

obtained using wind tunnel or flight test data. Nevertheless, there exists numerous software packages that allow one to obtain reasonably good approximations of these coefficients. LinAir Pro [51] is one of such packages. It provides an intuitive way to compute the aerodynamic characteristics of nonplanar lifting surfaces. To do so, an input file is created which contains the geometry of the airplane divided in panels (thus the *panel method*) as shown in figure 3.2.

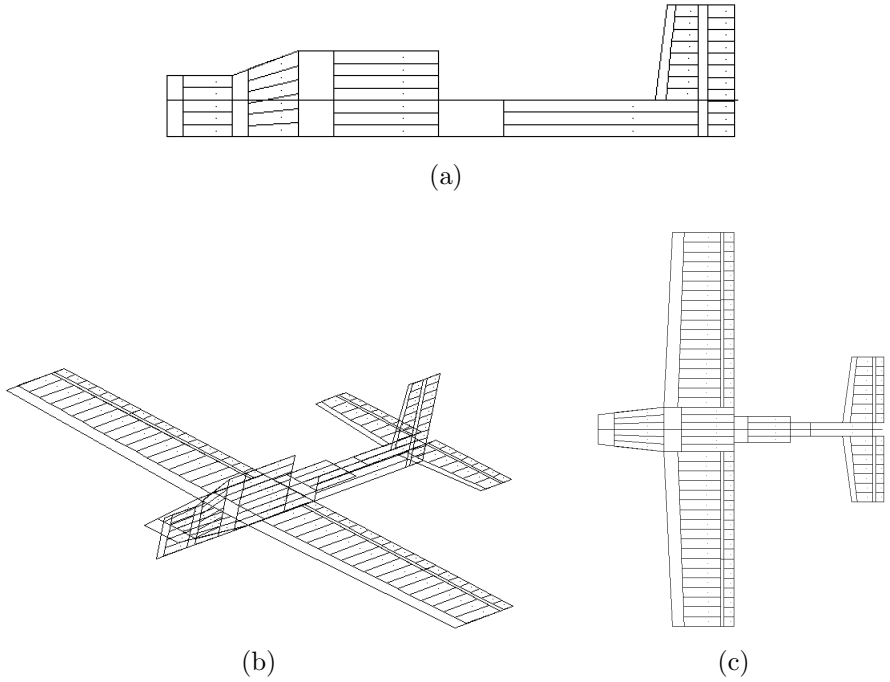


Figure 3.2: Panel Structure of the UAV's panels. (a) Side view. (b) Perspective view. (c) Top view.

LinAir Pro internally solves the Prandtl-Glauert equation [53] and computes the contributions of each panel's forces and moments as shown in Figure 3.3. Several aircraft configurations were used to compute the forces and moments.

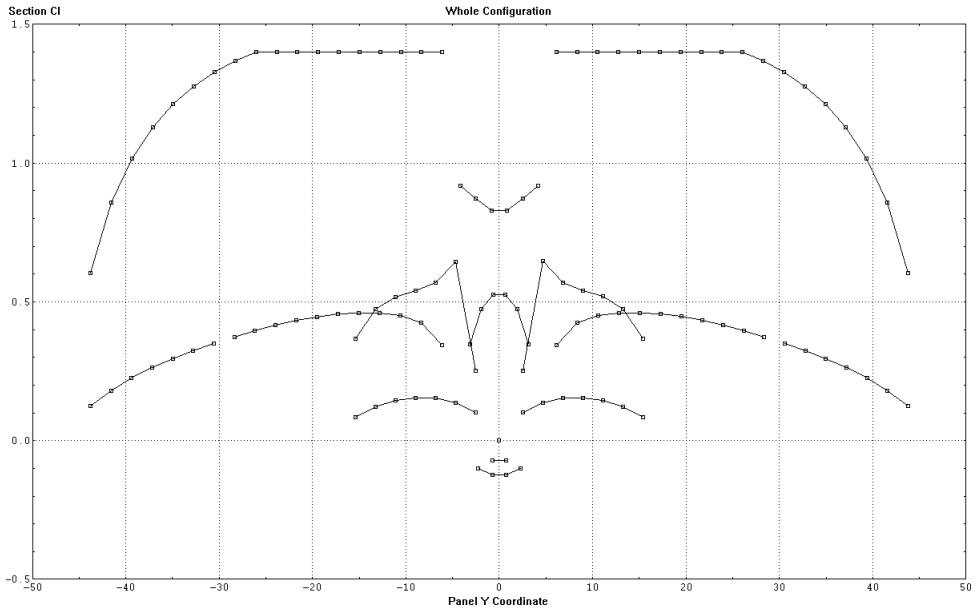


Figure 3.3: Lift contribution of each panel in LinAir Pro.

Each configuration had a control surface deflected at a certain angle. With this deflection, angle of attack (α) and sideslip angle (β) sweeps were performed and the contributions of the forces and moments were recorded. These data were used in a regression model to perform curve fitting and obtaining a first iteration of the aerodynamic coefficients. The aerodynamic coefficients assembled into the standard model forms the moderate-fidelity model of the UAV.

3.2.2 High Fidelity Model

The Parameter Space Investigation (PSI) method was first presented in the late 1970's in the Soviet Union [54] as solution to certain engineering-type multi-

criteria optimization problems[2]. The principal advantage of this method consists in the fact that, although lengthy, the formulation and solution of a problem framed in the PSI method comprises a single, iterative process[3]. The following, briefly describes the PSI Method as presented in Reference [55].

Let the aerodynamic coefficients presented in Appendix A determine a set A of *design variables*. Let this set be $A = \{\alpha_1, \ldots, \alpha_r\}$ which defines an r-dimensional design variable space, Π. By the nature of these design variables, they are constrained as

$$\alpha_j^* \leq \alpha_j \leq \alpha_j^{**}, \qquad j = 1, \ldots, r, \tag{3.1}$$

where α_j^* and α_j^{**} are the minimum and maximum allowable values respectively for design variable α_j. These bounds are chosen around the values of the aerodynamic coefficients predicted by LinAir Pro.

The design variables in A are also subject to one or multiple *functional constraints* that limit the set of feasible solutions. These functional constraints depend on one or more design variables. They define the mathematical relations which must be satisfied by the independent values of the design variables in an optimization problem. The functional constraints usually represent some essential physical principles which define the relationship among these parameters; for

[2]A rather complete presentation (in English) of the PSI method can be found in ref. [55].
[3]Iterations are typically done in a Personal Computer using a software.

example, the lift coefficient C_L (considered as a design variable) of a wing cannot be less than the ratio of weight (m) divided by the product of dynamic pressure (q) times the wing's surface area (S) – $C_L \geq m\frac{g}{qS}$.

These functional constraints can be written as:

$$C_l^* \leq f_l(A) \leq C_l^{**} \qquad l = 1, \ldots, n, \tag{3.2}$$

where C_l^* and C_l^{**} are the minimum and maximum allowable values respectively for the functional constraint $f_l(A)$.

Finally, all engineering problems have multiple (and often contradicting) performance *criteria*. If it is assumed that all the criteria have the same weight in the final decision process, then it is desired that these criteria, $\Phi_v(a)$, obtain extreme values. Furthermore, if these are formulated correctly, it can be assumed, without the loss of generality, that these are to be minimized. If the criteria are to be minimized, it is common to have a maximum threshold where these become unacceptable. Therefore the criteria can be formally stated as:

$$\Phi_v(A) \leq \Phi_v^{**}(A) \qquad v = 1, \ldots, k, \tag{3.3}$$

where $\Phi_v^{**}(A)$ is the maximum allowable performance criteria.

Figure 3.4 shows a graphical representation of the PSI method. In a general

way it can be stated that equations 3.1, 3.2 and 3.3 formulate the problem. The

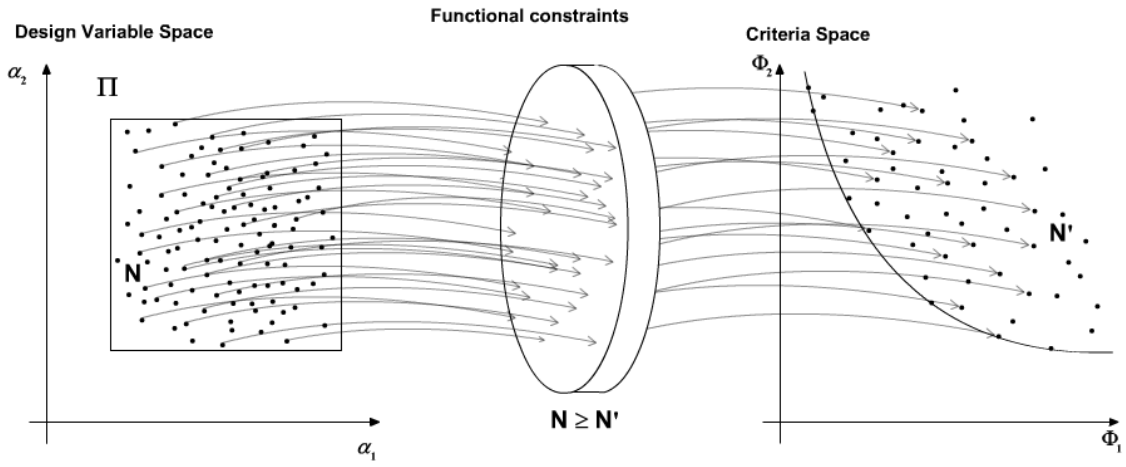


Figure 3.4: Graphical representation of the PSI method. From [2]

PSI method uses a set of computer algorithms to produce a uniformly distributed design variable LPτ space grid [56]. This variable (or parameter) space is then evaluated (or investigated) with the functional constraints which filter out the non-feasible solutions. Those feasible are then evaluated with the performance criteria and placed in a k-dimensional criteria space. All these criteria naturally form a Pareto set [57] of feasible solutions from which one of these Pareto-optimal solutions is selected.

The PSI method implementation used for this project is that of the software package MOVI 1.3 [2]. To enable MOVI to interact with the Simulink plant model a shared-memory interface (via a C++ library) was employed as described in Reference [58]. Figure 3.5 shows how the MOVI 1.3 package was used to

improve the the aerodynamic coefficients obtained with LinAir Pro. The workflow

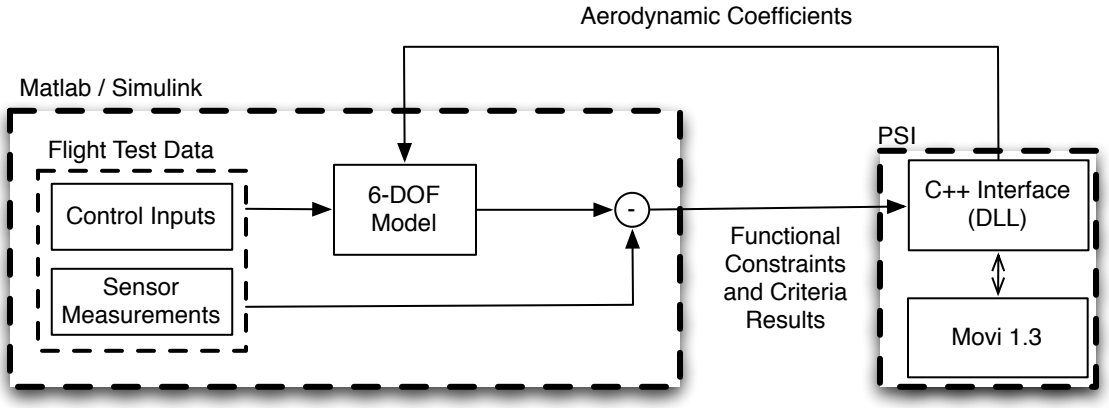


Figure 3.5: Interaction between Matlab/Simulink and the PSI implementation in MOVI 1.3 through a C++ DLL

starts with the Simulink plant model with the LinAir aerodynamic coefficients. This model's control inputs are the control surface commands from the doublet flight experiments. Its output is the plant model response based on the current set of coefficients. This output was used to generate a result for each of the functional constraints and criteria. The resulting values were sent back to MOVI for comparison purposes. MOVI in turn generated a new set of aerodynamic coefficients which were the design variables of the problem. More than 10,000 iterations of this were performed until a sufficiently large feasible set was obtained. From the feasible set a Pareto-optimal solution was automatically selected.

Figure 3.6 shows a response comparison, in the longitudinal channel of the UAV, between the flight experiments (in blue), the Simulink model response using

Linair Pro's set of coefficients (in red) and the new set of coefficients obtained with MOVI (in green).

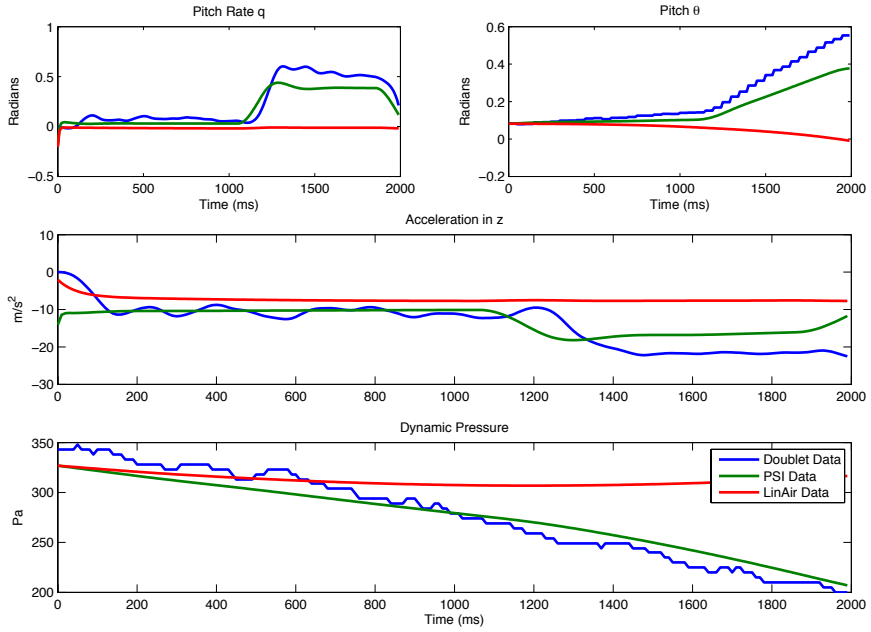


Figure 3.6: Response comparison between flight data and data obtained in HIL simulation using the aerodynamic coefficients obtained from LinAir Pro and MOVI.

It clearly shows an improvement over the initial set of coefficients. All the simulation and HIL results shown in the rest of this dissertation use these coefficients for the plant model. This new 6-DOF model based on the initial stability derivatives improved using the PSI method serves as the high-fidelity plant model of the aircraft.

3.3 HIL Architecture

The HIL simulator makes use of two different PCs (see Figure 3.7) named after their main tasks: One in charge of running the Simulink plant model **simulation**, and another to run the **ground station** software discussed in Section 2.3.1. It is assumed that both of these PCs are connected to the same network and thus UDP communications is readily available. These components and their interactions are described in the following sections.

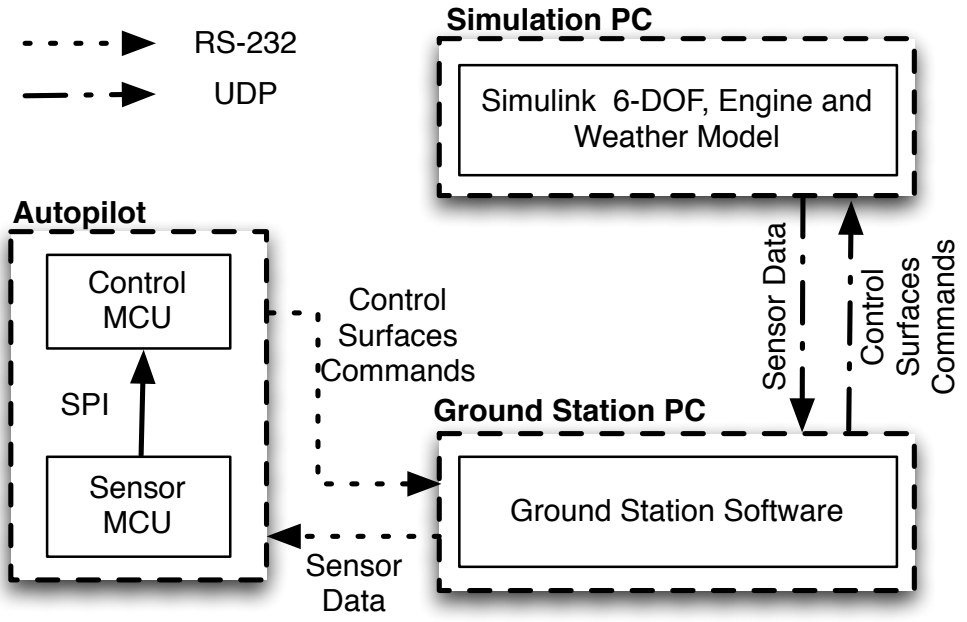


Figure 3.7: HIL setup; new Simulink-based 6-DOF model is a centerpiece of the Simulator

3.3.1 The Ground Control Station PC

The GCS PC has a Pentium 4 microprocessor with 2 GB of RAM running Windows XP operating system. It acts as a proxy between the autopilot and the Simulink plant model. The ground station software receives (via UDP) the synthetically generated sensor data from the Simulink plant model and passes it on to the autopilot (via serial). The autopilot generates control surface commands which the ground station receives. The ground station then sends these commands to the Simulink plant model closing the loop.

This computer also offers the complete ground station functionality to configure the autopilot gains, set navigation waypoints, monitor the sensor data and display the UAV in a Google Earth.

3.3.2 The Simulation PC

The simulation PC has a Pentium 4 microprocessor with 2 GB of RAM running Windows XP operating system. It runs the aircraft, engine, actuators and weather Simulink models at a constant sample period of 0.005 seconds. To facilitate the HIL simulator integration into the autopilot, a Simulink blockset was developed consisting of two components: **(i)** a source which listens on a predefined UDP port and parses messages in order to translate them into the control surface commands (up to 10 different control surfaces); and **(ii)** a sink, with inputs for the traditional

sensors (accelerometers, gyros, magnetometers, GPS, dynamic and static pressure, temperature, engine RPM). This sink parses the data and writes it back to a predefined UDP port in the correct format for the ground station software.

One of the key capabilities built into the autopilot, regarding HIL simulation, was its capability to echo back a particular timing data frame. This frame allowed the HIL simulator to send a packet to the autopilot and effectively measure the time it took to travel back. This in turn provided an effective way to monitor the delay between sending sensor data and receiving back a control command associated with that data. During multiple runs it was noticed that this delay was never higher than 0.03 seconds.

3.3.3 The Autopilot

The autopilot, besides the manual and autonomous modes described in Section 2.2, has a third mode – the *HIL mode*. The autopilot can be placed in this mode using the ground station software. Once in this mode, the autopilot ignores the data from its sensors and waits to receive sensor data via a serial port. When in HIL mode, all the computations in the autopilot are identical to those conducted in-flight, except that they are based on simulated sensor data.

3.4 HIL Simulator Validation

Validating the HIL simulator had two objectives. The first was to compare, via doublet commands, how accurate the identified model resembled that of the real aircraft. The second was to validate the overall HIL simulator architecture.

A flight experiment was set where the UAV was flown in open loop with the Piccolo Autopilot and performed doublet commands in elevator, aileron and rudder. The sensor data was collected and used to refine the 6-DOF model as described in Section 3.2. After that, the 6-DOF model was run in software simulation with the exact same doublet commands collected from the flight experiment. Finally the HIL simulator with the Piccolo Autopilot in the loop was configured and once again ran with the doublet command data as inputs to the model.

Figure 3.8 shows a comparison between the flight data, the 6-DOF software simulation and the HIL simulation response to the elevator doublet command[4]. The relevant variables for the longitudinal channel are shown: acceleration in Z-axis (Figure 3.8a) , q rotation (Figure 3.8b) rate and pitch angle θ (Figure 3.8c). It is worthy of notice how close the HIL results (shown in green) resemble those of real flight (shown in blue). Figure 3.9 shows the results for the rudder doublet. The relevant variables for the lateral channel are shown: acceleration

[4]The 6-DOF model was implemented in a Simulink model; the software and HIL simulations use exactly the same model and are expected to have very similar responses.

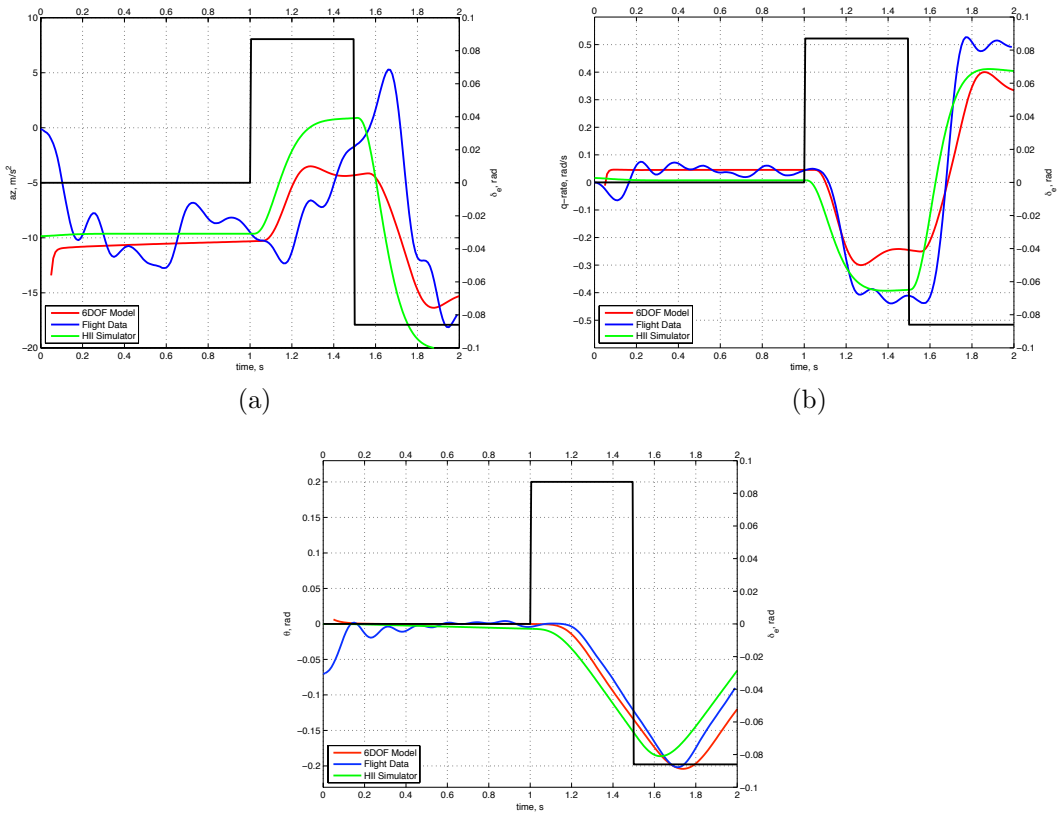


Figure 3.8: Elevator doublet response. (a) Z-Axis acceleration. (b) Rotation rate along Y-Axis. (c) Pitch.

in Y-axis (Figure 3.9a), p rotation rate (Figure 3.9b), r rotation rate (Figure 3.9c), and yaw angle ψ (Figure 3.9d). Worthy of notice is how the HIL results exhibit a *non-minimum phase* response in acceleration in Y-axis, and the p and r rotation rates, similar to that exhibited by the aircraft. For the lateral channel, system identification was significantly more complicated, mainly due to the fact that the sensors were noisier than that of the longitudinal channel. This had

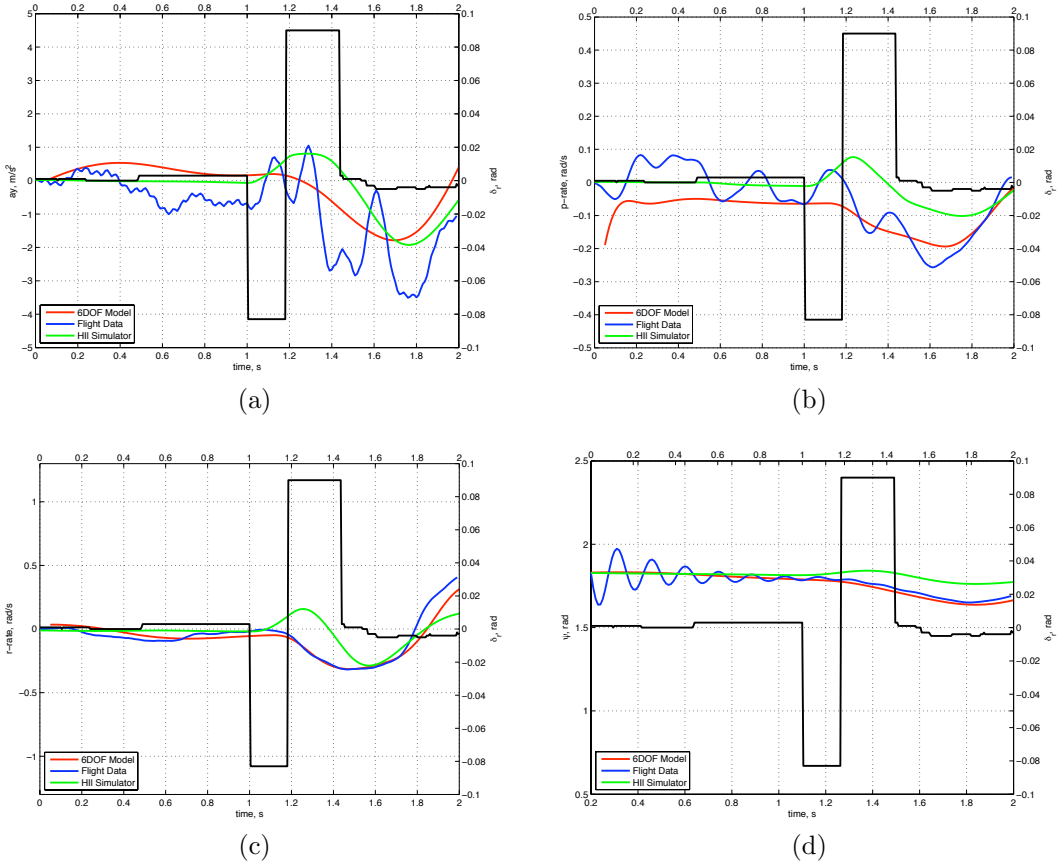


Figure 3.9: Rudder doublet response. (a) Y-Axis acceleration. (b) Rotation rate along the X-axis. (c) Rotation rate along the Z-axis. (d) Yaw.

the consequence of having the responses not follow as closely (as the longitudinal channel) the behavior of the real aircraft. Nevertheless this was considered to be sufficiently close to continue development.

Chapter 4

Navigation and Control

4.1 Introduction

Guidance, Navigation, and Control (GNC) refer to the three separate processes that are required to fly the aircraft on a desired trajectory. Guidance is the choice of desired path; within the structure of this work, it is a sequence of three dimensional waypoints which are left to the end user to specify. Navigation is the process of determining where in three dimensional space the aircraft is located, and Control is the process of using the navigation information to cause the aircraft to fly along the desired trajectory. Within the control framework, the structure is subdivided into inner loop (or airframe stabilization) and navigation (trajectory following).

The inner and navigation loops presented in this chapter were designed to showcase the complete *simulation* → *HIL simulation* → *flight test* workflow using the same Simulink model proposed in Chapter 3 and Appendix A. All the control and navigation loops presented in this chapter were implemented directly in Simulink. The Simulink models used for software simulation, HIL simulation, and testing are exactly the same as those compiled and then downloaded to the autopilot hardware.

The controller design presented in this chapter is based on the classical successive loop closure (or inner/outer loop) approach and is completely implemented in the control DSC (See Chapter 2). Figure 4.1 shows a top-level block diagram of the inner and outer/navigation loops. The inner loop is comprised of a set of Proportional, Integral, Derivative (PID) controllers separated into lateral and longitudinal channels. These are designed to stabilize the UAV and receive *mid-level* commands of turn-rate ($\dot{\psi}_c$), altitude(h_c), and airspeed (U_c). This loop generates the direct commands to the control surfaces – ailerons (δ_a), rudder (δr), elevator (δ_e), and throttle (δt). The outer loop implements a *high-level* waypoint-based navigation which in turn generates commanded turn-rate and altitude for the inner loop.

The autopilot itself has two different submodes of operation. It can be placed in either manual-*direct commands* or in autonomous-*waypoint navigation* mode.

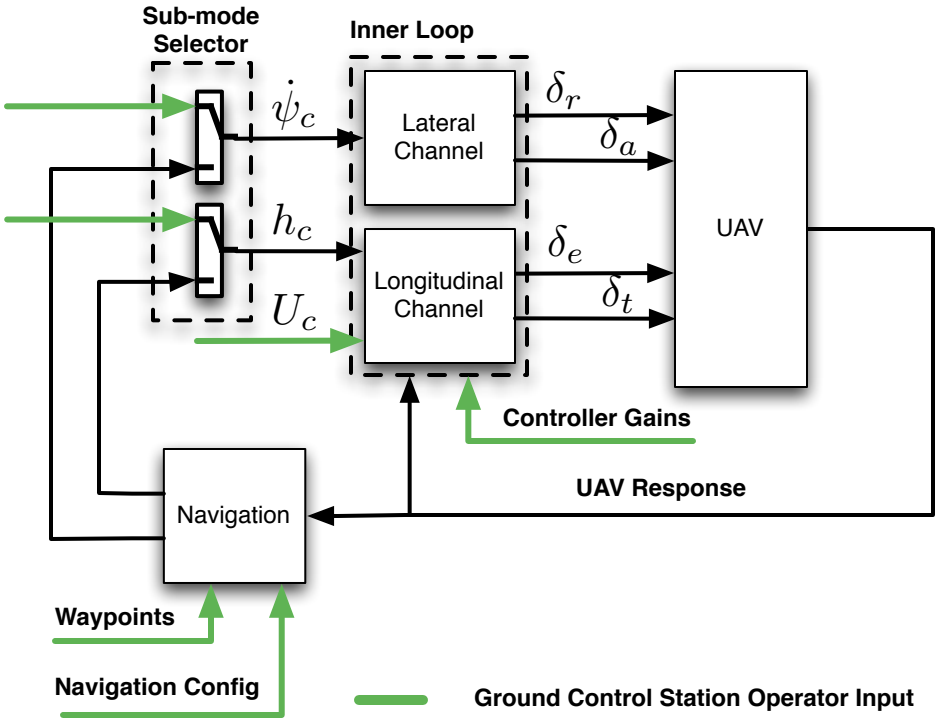


Figure 4.1: High level block diagram of the inner loop and navigation.

In the former the Ground Control Station (GCS) operator sends direct mid-level commands; in the latter the GCS operator configures the waypoints and the navigation "block" becomes responsible for generating the mid-level commands[1]. Both loops receive configuration settings from the GCS operator; the inner loop gets the proportional, integral and derivative gains for each controller; the outer loop receives the waypoints (in latitude, longitude and height) and additional configuration values that will be presented later in this chapter.

Note that in Figure 4.1 the inner and navigation loops receive the UAV re-

[1]Note that the commanded airspeed U_c is always set by the GCS operator regardless of the submode.

sponse directly. In reality these measurements come from the attitude and position estimation performed in the sensor DSC. This block has been omitted from the figure for the sake of clarity.

The following sections describe in greater detail the inner and navigation loops.

4.2 Inner Loop

One of the commonly used techniques that simplify analysis of the spatial motion of an aircraft is the decoupling of its motion onto longitudinal and lateral components. Although this decoupling indeed simplifies the analysis, it has the drawback that is only valid for a small set of linear flight regimes where the lateral-longitudinal cross coupling can be neglected.

4.2.1 Decoupling of Motion

A study of equations A.6, A.11, and A.19 in Appendix A, demonstrate that three nonlinear vector equations are required to describe the spatial motion of an aircraft. Nevertheless these equations can be separated into two subsets using small disturbance theory [59]. This leads to a linear model for an aircraft that has been partially disturbed from a steady-state trajectory when in trim condition. Assume that the aircraft is in steady-state, level flight, and in trim condition, and that this condition is disturbed by the deflection of the aircrafts elevator or an

increase in throttle. This change generates a pitching moment, without affecting the rolling or yawing moments. It also produces a change in the forces applied in the longitudinal (X and Z) direction in the body frame $\{B\}$, without affecting the dynamics in the lateral (Y) direction. A similar analysis can be performed on the lateral dynamics affected by the deflection of the ailerons or the rudder. Overall this lateral/longitudinal separation significantly simplifies the analysis of aircraft dynamics and control and is valid for most UAV flight regimes.

Based on this separation, the inner loop has been designed to replicate how a trained pilot flies a manned aircraft. The following subsections describe the inner loop implementation of both channels.

4.2.2 Lateral Channel

The lateral channel block diagram is shown in Figure 4.2. It controls two of the four control surfaces – the ailerons and the rudder. The rudder, although traditionally used to damp the Dutch roll mode [60], in this design is used exclusively to keep the aircraft in a coordinated turn[2] by driving the lateral acceleration in the body frame $\{B\}$ to zero.

[2]For a turn to be considered *coordinated* two conditions must be met: **(i)** the angular velocity must be constant; and **(ii)** the resultant of gravity and centripetal acceleration from the aircraft in the turn, lies in the aircraft XY plane [59].

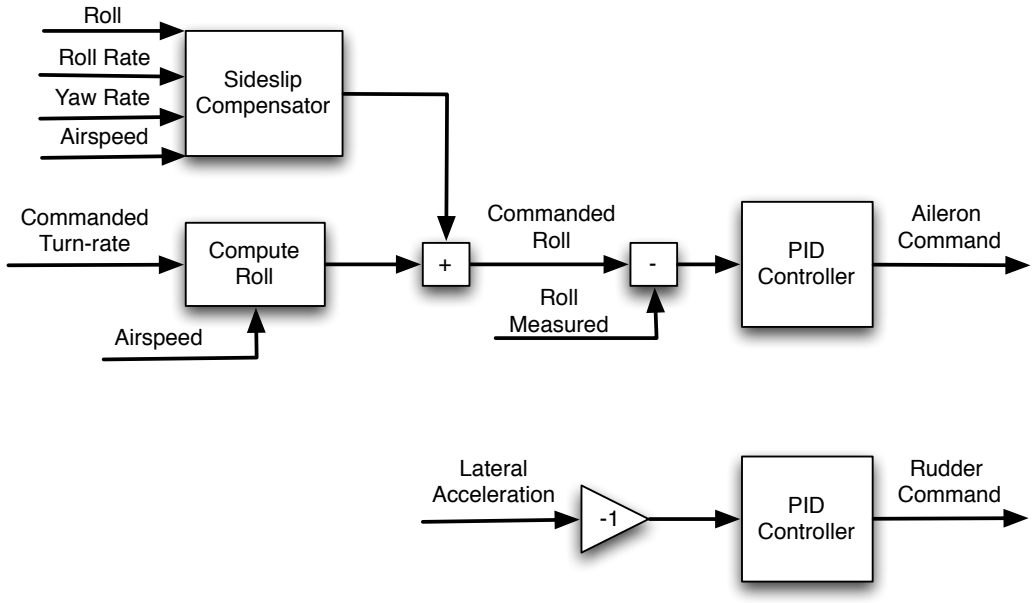


Figure 4.2: Block diagram of the inner loop's lateral channel.

The ailerons are used to control the UAV's turn-rate by controlling its roll angle. The ailerons loop receives, either from the navigation loop or the GCS direct commands, a commanded turn rate ($\dot{\psi}_c$). Using the commanded turn rate and the measured airspeed (U_m) it converts that to a commanded roll (ϕ_c):

$$\phi_c = \tan^{-1}(\frac{\dot{\psi}_c \, U_m}{g}). \tag{4.1}$$

The commanded roll is compared with the roll measured to produce the error signal that the PID controller will drive to zero by controlling the ailerons.

An additional component in Figure 4.2 is the *sideslip compensator* which is described in the following subsection.

4.2.2.1 Sideslip Compensator

Equation 4.1 assumes that the aircraft, when turning, is always in a coordinated turn and that the sideslip (β) and pitch (θ) angles are small[3]. Nevertheless, if these assumptions do not hold, either because the turn is not coordinated or because the β or θ angles have become large, an additional compensation is needed to generate the roll command that would make the UAV follow the commanded turn rate.

Equation A.19 in Appendix A ($\dot{\psi} = \frac{q\sin(\phi)+r\cos\phi}{\cos(\theta)}$) shows the computation of the turn rate $\dot{\psi}$ with none of the assumptions taken for coordinated turns. If one compares both computations of turn rate under nominal flight conditions (and during coordinated turns), the results are practically the same. On the other hand, if the assumptions taken to derive Equation 4.1 don't hold, there will be a difference between the results produced by these two equations. Let the error between both computations be defined as:

$$\dot{\psi}_{err} = \frac{g\ \tan(\phi)}{U_m} - \frac{q\sin(\phi) + r\cos\phi}{\cos(\theta)}, \tag{4.2}$$

this error, can in turn be converted to a roll (using Equation 4.1) compensation

[3]For a complete derivation of Equation 4.1 and the assumptions taken that make it valid see pages 238-242 of Reference [60].

(ϕ_{ssc}) component to be added to the roll command to account for that difference:

$$\phi_{ssc} = \tan^{-1}(\frac{\dot{\psi}_{err}\ U_m}{g}). \tag{4.3}$$

The flight test results in Chapter 6 will show how this simple, yet effective, compensator played a key role in achieving control surface failure tolerance by complementing the commanded turn rate under the presence of high sideslip angles, such as those induced by a rudder failure[4].

4.2.3 Longitudinal Channel

Traditionally, there are two modes in which pilots exert the longitudinal control of the aircraft [61]. The *climb/descend* mode in which the throttle is usually fixed (high if climbing and low if descending) and airspeed/climb rate is controlled by varying the aircraft's pitch (θ) with the elevator.

In the *level flight* mode, the control surfaces are used in the opposite way. The pilot controls airspeed by varying the aircraft's thrust (with the throttle). The altitude is maintained by adjusting the elevator to pitch the aircraft as necessary. In this mode, since airspeed is kept constant by the throttle (unless saturation occurs) even small changes in pitch are rapidly reflected in altitude and changes.

[4]Note that similar compensator structures can be used for each actuator failure, but for this work, only the rudder failure was included.

During the design phase of the autopilot it was decided that most of the flight time would be conducted at a constant altitude (following waypoints). This made the level flight mode a better candidate to implement in the longitudinal autopilot.

Figure 4.3 shows a block diagram of the inner loop's longitudinal channel. Based on the measured altitude (h_m) the air density (ρ) is computed as [62] :

$$\rho = \rho_0 \left(1 - \frac{h_m}{145,442}\right)^{4.255876}, \tag{4.4}$$

where ρ_0 is the air density at sea level in Kg/m^3.

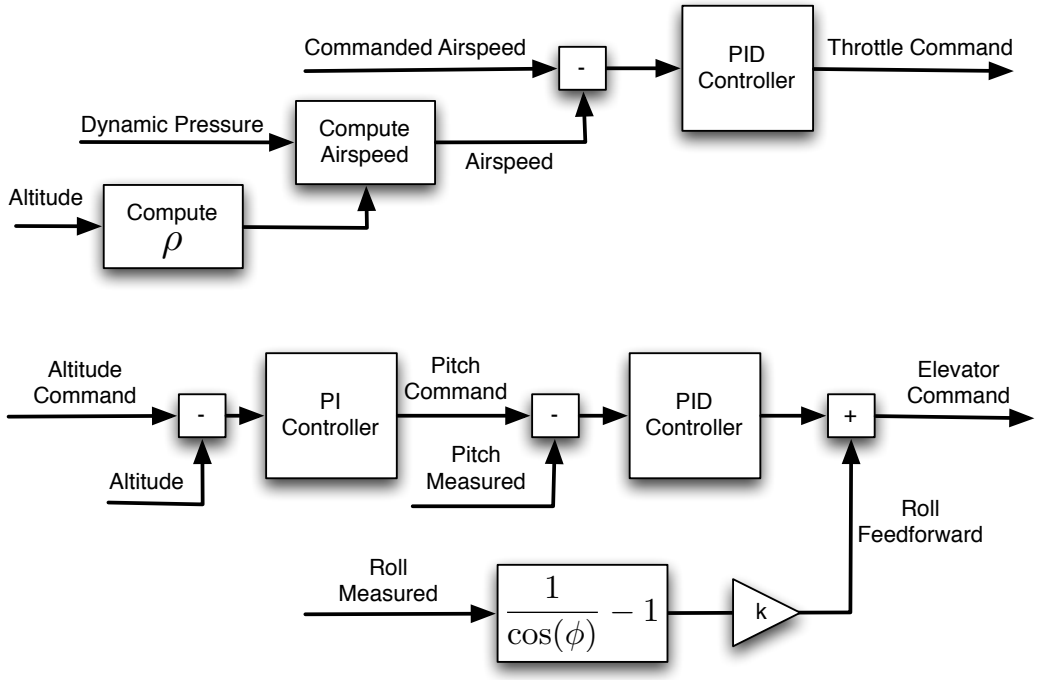


Figure 4.3: Block diagram of the inner loop's longitudinal channel.

The current air density (ρ) is used with the dynamic pressure (q) to compute the measured (indicated) airspeed (U_m) :

$$U_m = \sqrt{\frac{2q}{\rho}}. \tag{4.5}$$

The commanded airspeed (U_c) is then compared with the measured airspeed (U_m) to generate the error signal that the the PID controller will drive to zero by controlling the throttle.

The elevator control consists of two cascaded controllers. The commanded altitude (h_c) is compared to the measured altitude (h_m) to generate an error signal to a PI controller. This PI controller in turn generates a pitch command (θ_c) which is compared with the measured pitch (θ_m) to generate an error signal that the PID controller will drive to zero by controlling the elevator.

Rolling the aircraft (to place it in a turn) has the consequence of making the aircraft loose lift in the vertical direction and thus altitude. Because of this, if it is desired to maintain the altitude during the turn, trained pilots usually increase the aircraft's pitch when banked to turn. To mimic this behavior a feedforward term in the elevator control was introduced.

When in level flight, lift (L) is equal to the aircraft's weight (W). If turning, then lift, weight, and roll (ϕ) are related as $L = W/\cos(\phi)$. Therefore a change

in lift (Δ_L) can be expressed as:

$$\Delta_L = W\left(\frac{1}{\cos(\phi)} - 1\right), \tag{4.6}$$

which suggests that the feedforward term should be proportional to $\left(\frac{1}{\cos(\phi)} - 1\right)$. This has the desired effect of increasing the pitch angle proportionally to the lost lift.

4.3 Navigation Loop

The control law in the navigation loop follows directly that presented in Reference [3] with some minor modifications presented later in this chapter. The paper presents a control law to generate lateral acceleration commands to track circular paths. The reader is referred to such paper for a complete discussion of the control law including a detailed performance and Lyapunov stability analysis. The following briefly presents the control law derivation as presented in [63].

Based on the geometry shown in figure 4.4, let $V_{g_{uav}}$ be the UAV's XY velocity vector with respect to ground and C be sector of a circle of radius R. Let L_2 be a lookahead distance defining the UAV's *forward-looking position* in the desired path. Let the intersection of this distance with C define a secant that subtends from the UAV's position to the intersection with the desired path. Let L_2 be

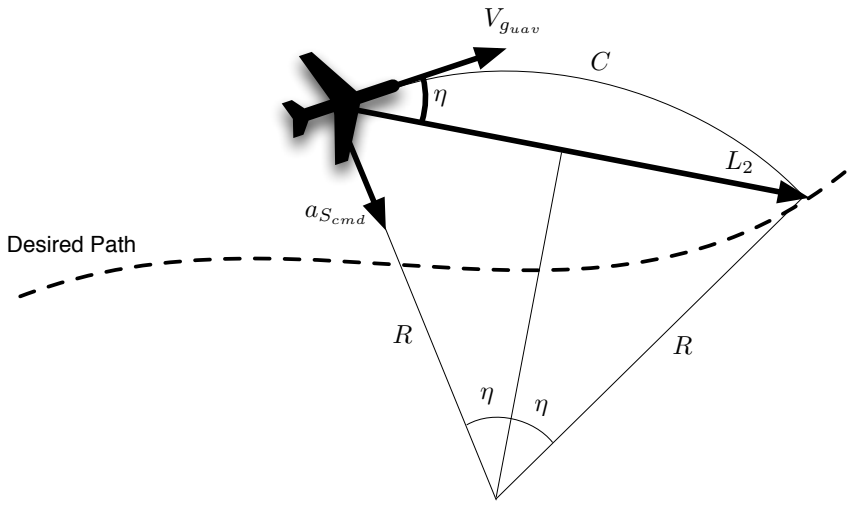


Figure 4.4: Navigation control law geometry. After Figure 1 in Reference [3]

divided into two equal segments by the line that bisects the angle tended by C. Then from elemental trigonometry:

$$\frac{L_2}{2} = R\sin(\eta). \tag{4.7}$$

From elementary kinematics it is known that centripetal acceleration (a_c) required to follow the circular sector of C is given by:

$$a_c = \frac{||V_{g_{uav}}||^2}{R}, \tag{4.8}$$

thus if the UAV is to follow C it must command a_c lateral acceleration. Solving Equation 4.7 for R and substituting it into Equation 4.8 produces the following

control law:

$$a_{S_{cmd}} = 2\frac{||V_{g_{uav}}||^2}{L_2}\sin(\eta). \tag{4.9}$$

It is clear that the only requirements for the implementation of the control law presented in Equation 4.9 is to select the lookahead distance L_2 and determine the angle η. The following subsections present the computation of η assuming L_2 has been selected. The reader is referred to Reference [3] for an analysis on how to select an appropriate value of L_2.

4.3.1 Lookahead Angle η

Let P_{e_y} be the the lateral position error of the UAV with respect to the desired path (see Figure 4.5). Let E be the closest point on the path from which the lateral

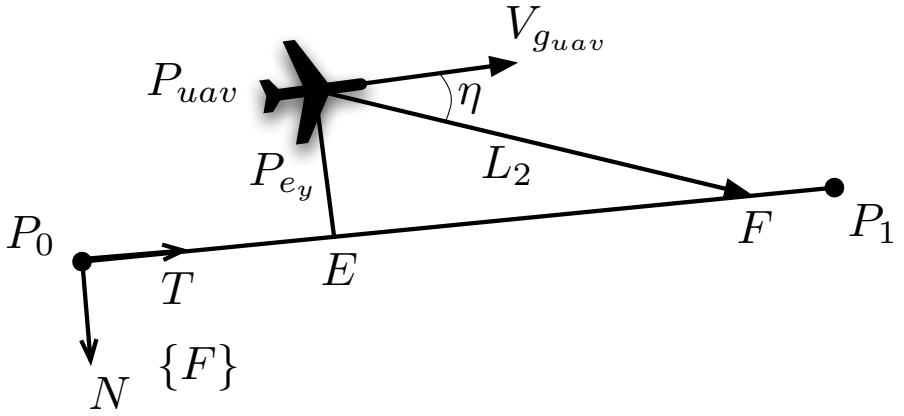


Figure 4.5: Geometry Description of the angle η computation.

error is measured (i.e. $|P_{uav}E| = P_{e_y}$), the angle η be the angle between the UAV's

ground velocity vector ($V_{g_{uav}}$) and L_2[5], and F be the location of the intersection of L_2 with the desired path.

Assume there exists a Serret-Frenet frame [64] $\{F\}$ attached to the desired path's origin waypoint (P_0). Assume the UAV is traveling from P_0 to P_1 and that the Serret-Frenet frame, attached to P_0, is defined as follows:

$$\begin{aligned} T &= \frac{1}{D}\left(P_1 - P_0\right), \\ N &= \frac{1}{|[-y_T \quad x_T \quad 0]|}[-y_T \quad x_T \quad 0]^\top, \\ B &= T \times N, \end{aligned} \tag{4.10}$$

where x_T and y_T are the x and y components of T, and D is the distance from P_0 to P_1. with the T axis pointing towards the next waypoint.

From the geometry shown in Figure 4.5, the EF separation can be computed as:

$$|EF| = \sqrt{|L_2|^2 - |P_{e_y}|^2}, \tag{4.11}$$

and the vectors E and F as:

$$E = P_{uav} - |P_{e_y}|N, \tag{4.12}$$

[5]In the following discussion, assume L_2 is a vector with the desired lookahead length in the sense presented in the previous section.

$$F = E + |EF|T. \tag{4.13}$$

With these values then L_2 can be computed as:

$$L_2 = F - P_{uav} = |EF|T - |P_{e_y}|N, \tag{4.14}$$

where

$$P_{e_y} = N^\top (P_1 - P_{uav}),$$

and the angle η can be defined as:

$$\eta = \sin^{-1}\left(\frac{|V_{g_{uav}} \times L_2|}{|V_{g_{uav}}||L_2|}\right). \tag{4.15}$$

4.3.2 Dynamic Computation of L_2

The derivation of the control law presented in Equation 4.9 assumes that the UAV is tracking circular trajectories (or circular segments). Reference [3] presents an analysis for following linear trajectories. This analysis shows that, if some small angle assumptions hold, linearization of the nonlinear guidance logic yields a PD (proportional and derivative) controller for the cross-track error. The controller's damping ratio ξ is 0.707 and the natural frequency ω_n is given by $\sqrt{2}\frac{V}{L_2}$: (see Section C in Ref. [3]).

This result although ideal for the damping ratio, shows that the poles of the

controller move as the UAV's groundspeed changes. This is undesirable because the system could become unstable, therefore, Equation 4.9 was modified to have L_2 not be a constant but rather a function of ground speed:

$$a_{S_{cmd}} = 2\frac{||V_{g_{uav}}||^2}{L_2(V_{g_{uav}})}\sin(\eta), \tag{4.16}$$

where $L_2(V_{g_{uav}}) = L_{2_{base}}||V_{g_{uav}}||$ and $L_{2_{base}}$ is a constant. Now the natural frequency of the linearized response is independent of ground speed. Equation 4.16 can now be simplified to:

$$a_{S_{cmd}} = 2\frac{||V_{g_{uav}}||}{L_{2_{base}}}\sin(\eta). \tag{4.17}$$

This is the navigation law that was implemented in the autopilot.

4.3.3 Waypoint Transition

Previous sections have shown the navigation loop as if only two waypoints existed in the track. Nevertheless, transitioning from one waypoint segment to the next has a big impact on the the overall performance of the navigation. Correct timing for the waypoint transition, significantly reduces the overshoot and at the same time brings the UAV as close as possible to the desired waypoint.

For purposes of the waypoint transition logic presented in this section it was

assumed that the UAV did not have to fly exactly over the waypoint. It was rather preferred to have the UAV follow as closely as possible the path and turn as sharp as the UAV bank angle limit allowed. To achieve this, it is necessary to *pre-turn* the UAV, i.e. start changing waypoint segments before actually reaching the end of the segment.

Based on the geometry shown in Figure 4.6 let the segment P_0P_1 be the segment which the UAV is currently traversing, the segment P_1P_2 be the next segment in the navigation track, and the vectors T_1, N_1, T_2 and N_2 be the Serret-Frenet frame vectors as previously described[6]. Let a circle of radius R_t be the smallest circle the UAV can track based on its maximum bank angle (ϕ_{max}) and commanded airspeed (U_c). Let this circle be inscribed between the two segments and thus tangent to two points, one of each segment. The circle's radius can be computed as:

$$R_t = \frac{U_c^2}{g \tan(\phi_{max})}. \tag{4.18}$$

Then the pre-turn distance P can be computed as follows: Let the angle γ between the vectors T_1 and T_2 be computed using an inner product:

$$\gamma = \cos^{-1}(T_1 \cdot T_2),$$

[6]Note that Figure 4.6 presents the frame comprised of T_2 and N_2 displaced from its location for the sake of clarity only.

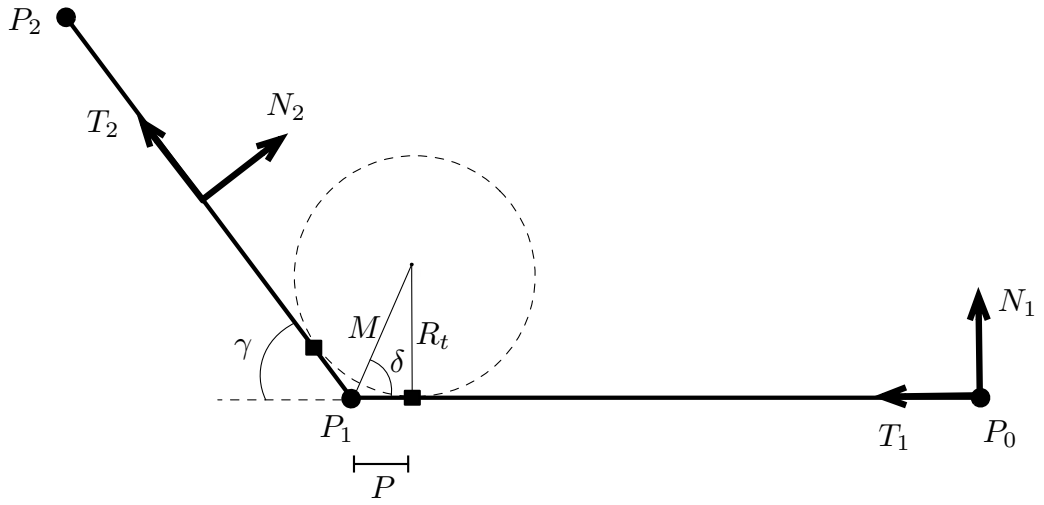


Figure 4.6: Waypoint transitioning geometry

the bisection angle δ be:

$$\delta = \frac{\pi - \gamma}{2}.$$

Finally, the pre-turn distance P is given by:

$$\begin{aligned} M &= \frac{R}{\sin(\delta)} \\ P &= M\cos(\delta) \\ \therefore \quad P &= R\frac{\cos(\delta)}{\sin(\delta)} = R\cot(\delta). \end{aligned} \tag{4.19}$$

At the implementation level, the navigation loop computes, at each waypoint transition, the pre-turn distance P and the total segment run T_r. The segment run T_r is computed by subtracting P from the waypoint separation for the corre-

sponding segment.

To ensure adequate waypoint switching regardless of the lateral error, the navigation loop computes the projection of the UAV current position onto the segment: $D_g = (P_{uav} - P_0) \cdot T_1$. To provide an additional lead time for turns, a pre-turn distance is computed based on the UAV's groundspeed as:

$$D_{pt} = |V_{g_{uav}}| P_k,$$

where P_k is a configurable value. To determine when to switch to the next segment, the obtained distance (D_g) is compared to $T_r - D_{pt}$.

This waypoint transition logic generates a progression of the lateral error P_{e_y} as shown in Figure 4.7.

At time T_0 the UAV is coming into the current segment (either just from starting or from a previous waypoint transition), initially the lateral error is large (shown in blue). As time progresses, the lateral error is reduced by the commands generated by the navigation loop (times T_1 and T_2). Waypoint segment switch occurs at time T_3 at this moment the lateral error suddenly changes from being approximately zero to the waypoint transition distance as described in this section.

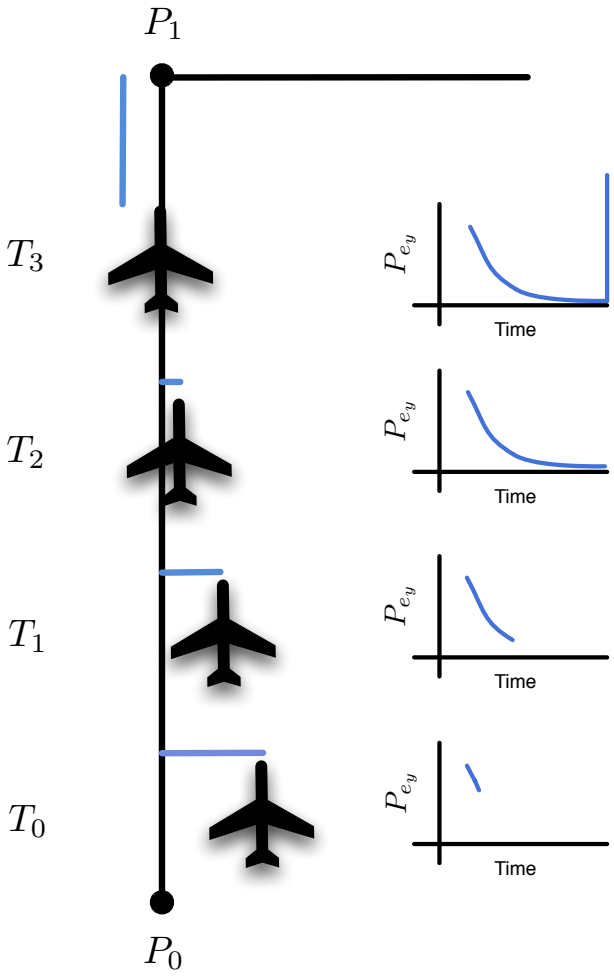


Figure 4.7: Evolution of the lateral error P_{e_y} upon waypoint switch.

4.4 Inner Loop and Navigation Simulation

In order to implement the inner and navigation loops described in this chapter a new Simulink model was created. The *the plant model* used for the simulations was the 6-DOF model described in Appendix A. The plant model's aerodynamic coefficients were obtained as described in Section 3.2. The complete Simulink

model is laid out as shown in Figure 4.1 where the UAV block has been replaced by the plant model.

The simulation scenario consisted of having the UAV fly under manual control for the first five seconds of the simulation. After that, waypoint tracking is engaged and the UAV maneuvers to capture waypoint 1 and start navigation in a six waypoints track. The following presents the implementation and simulation results of the navigation, lateral, and longitudinal blocks for such scenario.

The performance of the developed outer loop navigation was first analyzed in simulations. Figure 4.8 shows an XY plot of the simulation results of the waypoint tracking scenario. The UAV starts at position $(0, 200)$ flying due southeast. After five seconds, waypoint tracking is engaged and the aircraft maneuvers towards waypoint 1 (red trajectory). Once in the waypoint track (shown in green), when the pre-turn point is reached the UAV transitions waypoint segment (blue trajectory) and continues navigation.

Figure 4.8 also shows how by pre-turning, the algorithm avoids most of the overshoot that would otherwise be present. The system correctly tracks the straight segments, and the simple algorithm for waypoint transition presented in Subsection 4.3.3 was deemed adequate. Position errors during simulation never exceeded 15 m even during waypoint transition.

Figure 4.9 shows the Simulink implementation of the inner loop's lateral chan-

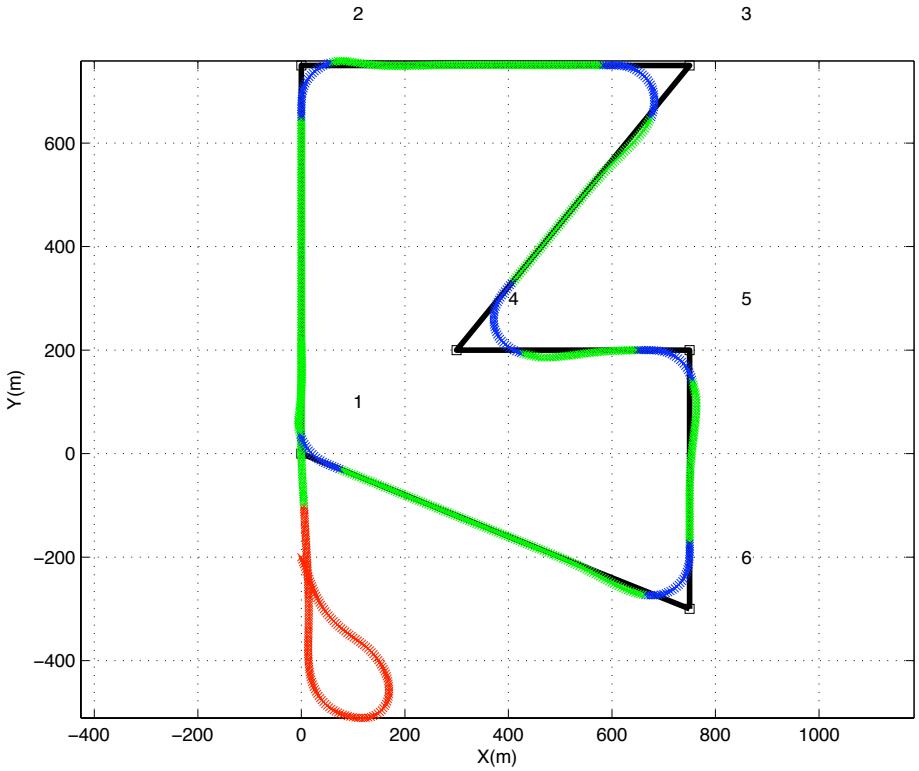


Figure 4.8: Waypoint tracking simulation results. Red indicates the UAV is in manual and waypoint capture mode. Green indicates line tracking. Blue indicates waypoint transition.

nel. By using Simulink, the actual implementation resembles a design block diagram as the one shown in Figure 4.2. It is important to note that this exact implementation was used for simulation, HIL simulation and actual flight tests. This demonstrates the ease of use and great potential offered by the SLUGS platform.

Figure 4.10 presents the inner loop's lateral channel controlled variables comparing commanded *vs.* measured values.

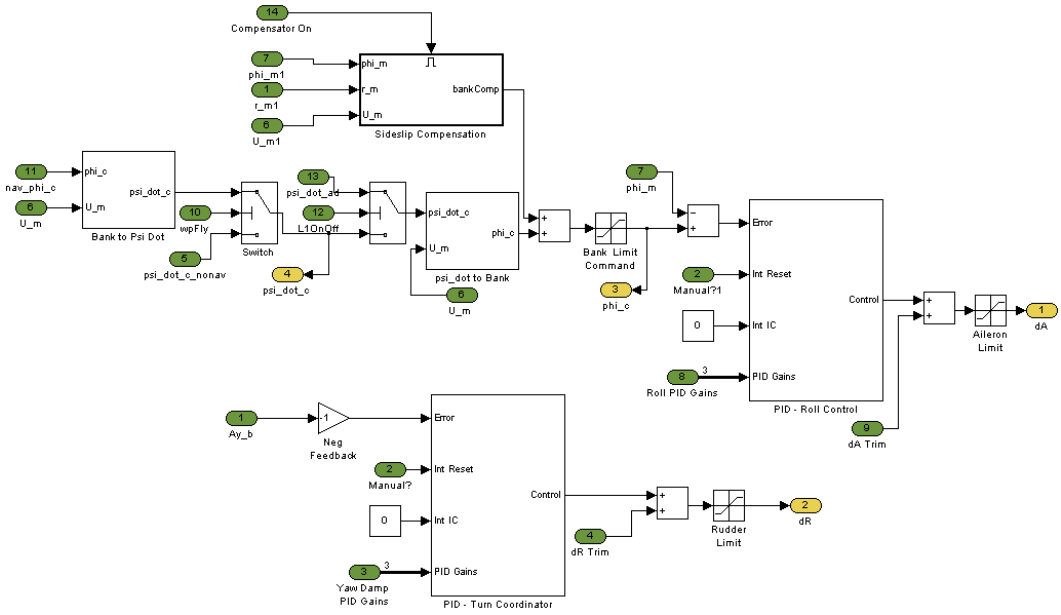


Figure 4.9: Inner loop's lateral channel implementation in Simulink.

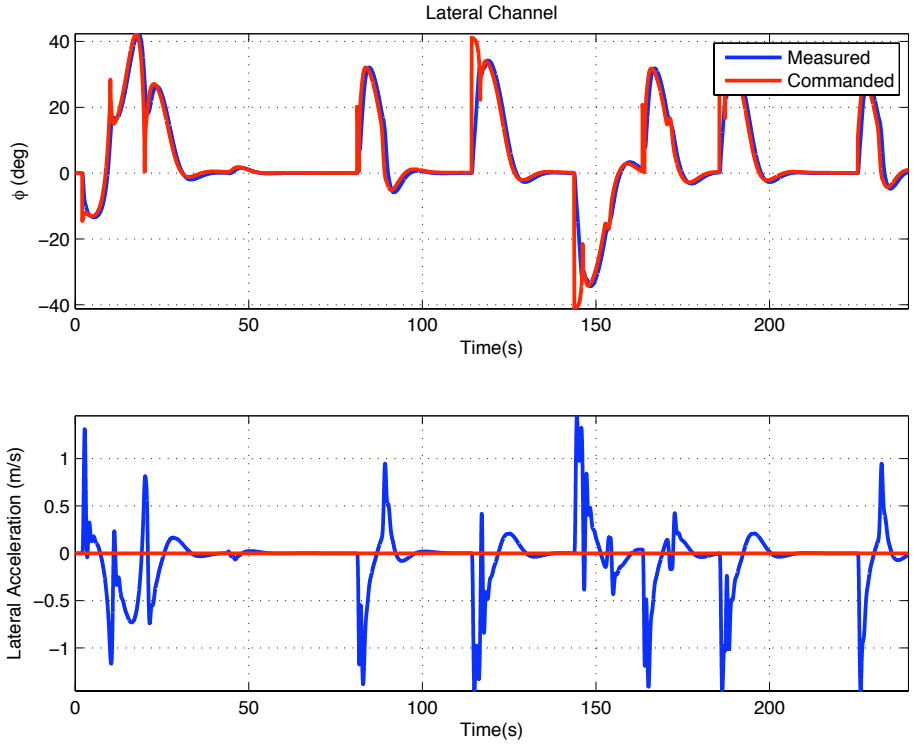


Figure 4.10: Inner loop's lateral channel controlled variables: Roll angle ϕ and lateral acceleration in body frame A_{y_b}. Red is commanded, blue is measured.

The measured roll angle (ϕ) closely follows the command. The measured roll angle differs from the commanded only on those abrupt changes such as those shown around 110 and 145 seconds of the simulation run. These variations are a direct consequence of changes in ground speed which in turn change the length of L_2 and thus the commanded turn rate.

Figure 4.10 also shows how during turns the rudder controls the lateral acceleration. The acceleration never exceeds 1.5 m/s^2.

Figure 4.11 shows the Simulink implementation of the inner loop's longitudinal channel. As for the lateral channel, the actual implementation resembles that of a design block diagram as the one shown in shown in Figure 4.3. It is worthy to emphasize that converting this to a high level programming language would have been far more challenging than replicating the block diagram in graphical environment.

Figure 4.12 presents the inner loop's longitudinal channel controlled variables comparing commanded *vs.* measured. Note how when the aircraft descends the airspeed increases. This is a direct consequence of the fact that the inner loop design uses only the *level-flight* implementation in which airspeed is controlled via throttle. So the maximum authority that the control system has to decrease the airspeed is to command the throttle to zero. To prevent an excessive increase in airspeed during descent for flight tests the negative value of pitch angle θ was

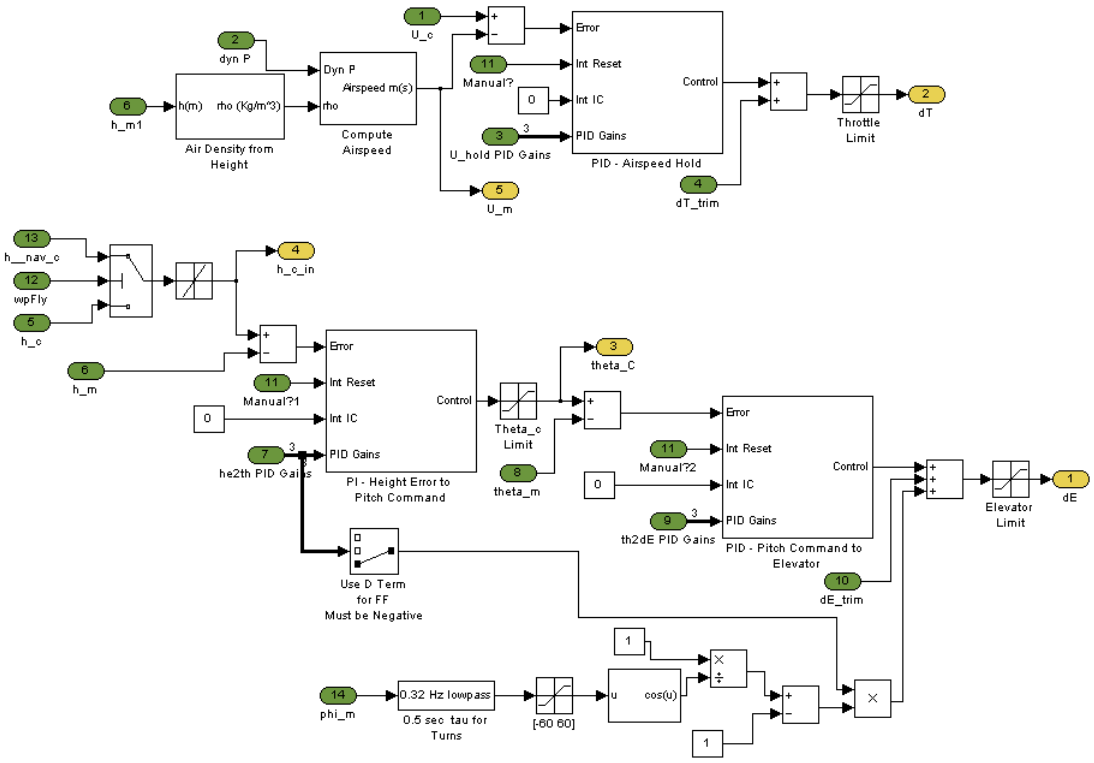


Figure 4.11: Inner loop's longitudinal implementation in Simulink.

limited to $-8°$.

Figure 4.12 also shows the pitch angle θ and the UAV's altitude. Both follow closely the commands generated by the navigation loop.

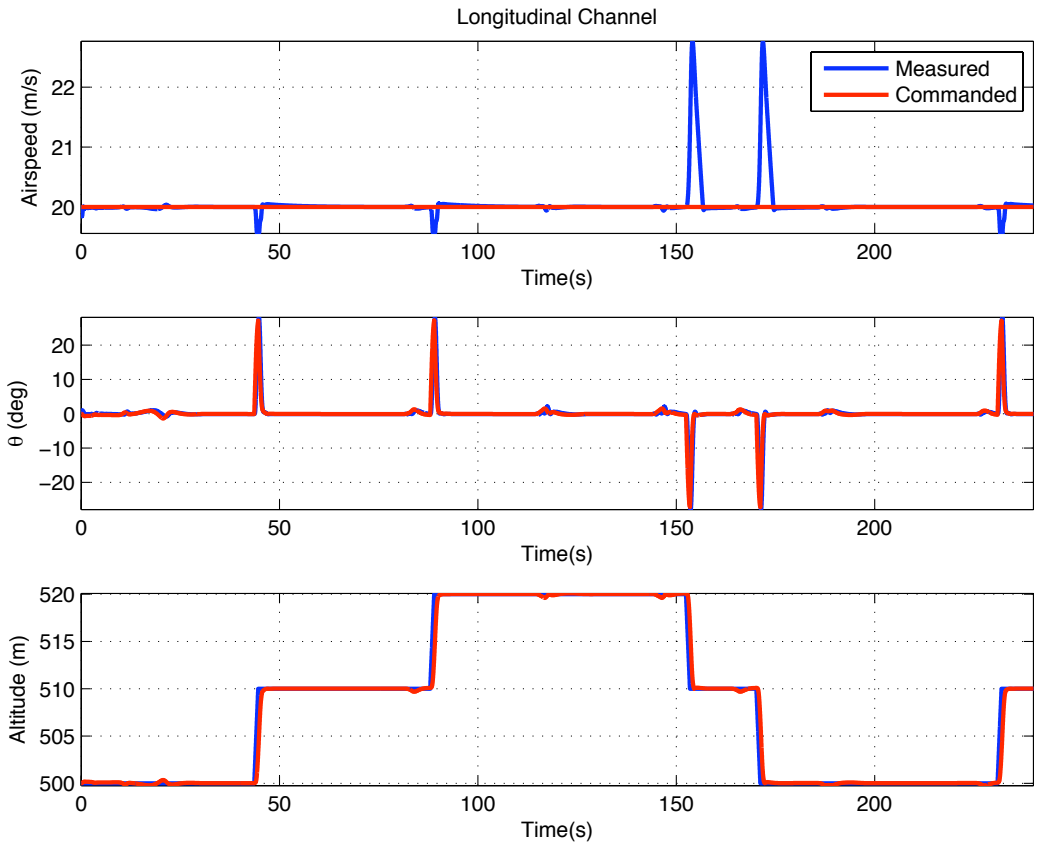


Figure 4.12: Inner loop's longitudinal channel controlled variables: Airspeed U, pitch angle θ, and altitude H. Red is commanded, blue is measured.

Chapter 5

Failure Tolerance

5.1 Introduction

In-flight failure tolerance is one of the most complicated tasks that an autopilot can perform. It is not only theoretically difficult and numerically intensive but also requires a fast response to prevent an airplane to escape its dynamic controllability envelope. The implementation of failure tolerance algorithms is often complicated and is typically preceded by simulations that show its effectiveness.

During the flight validation and verification (V&V) process of the autopilot presented herein it was decided to explore this class of control problems to showcase the effectiveness and ease of use offered by SLUGS. This chapter presents the implementation, in the SLUGS autopilot, of an adaptive control algorithm previously reported in the literature.

5.2 Background

The problem of fault tolerance in flight control systems has been addressed in literature for more than three decades. In 1978, Wensley et. al.[65] proposed the development of a fault tolerant flight control computer which achieved fault tolerance by integrating multiple redundancies into many of the aircraft systems. Error analysis, detection and reconfiguration was performed by software in a central minicomputer that implemented a voting mechanism for each of the independent flight computers. Redundancy on flight computers is a valid solution to achieve fault tolerance, and many commercial and military airplanes are equipped today with such systems. Nevertheless it is easy to argue that it is quite impractical and expensive to provide redundancy to every single vital system in the aircraft, especially in small UAVs.

Despite all the efforts to achieve some level of fault tolerance without total system redundancy, 20 years later, Patton wrote[66]: *"Research into fault-tolerant control has attracted many investigators ... Little information from these and other similar (academic) studies has ever been applied to real process plants or vehicles"* when referring to multiple studies in the area of fault-tolerant control. A year later Rago et. al. [67] showed successful simulation results for failure detection (of sensors and actuators) and fault tolerant control using an interacting multiple model Kalman filter, but did not follow up with HIL nor flight test results. Using

a similar scheme: failure detection and then fault tolerant control, Napolitano et. al. [68] approached the problem from a different perspective, and proposed using neural networks to identify the failure and reconfigure the control system accordingly. Once again only software simulation results were offered. Separately Alwi et. al [69] proposed a different scheme for fault tolerant control, where the effectiveness level of the actuators is used by the control allocation scheme to redistribute the control signals to the remaining actuators when a fault or failure occurs. This control allocation scheme showed, via simulation, that faults and even certain total actuator failures could be handled directly without reconfiguring the controller.

But (arguably) the most renowned adaptive control architecture in flight control is the Model Referenced Adaptive Control (MRAC). MRAC's modifications [70, 71, 72] have been shown to successfully recover aircraft in the presence of modeling and environmental uncertainties [73, 74].

More recently, two different UAV research groups have presented fault tolerance flight test results. The first are the autopilots made by the company Athena Control[75] [1] which have shown significant failure tolerance (see video in Reference [76]).

The second is the $\mathcal{L}_1$ output-feedback augmentation implemented using an

[1]These autopilots are based on the a patent[9] filed by David Vos,the company's founder. Recently Rockwell Collins acquired Athena Controls and all its autopilots portfolio.

off-the-shelf autopilot and a PC104 computer reported by the Naval Postgraduate School's Unmanned Systems Laboratory[77].

The $\mathcal{L}_1$ adaptive controller has the advantage, over other adaptive algorithms, that there is no failure identification, failure isolation, and controller reconfiguration process traditional in adaptive controllers [68, 67, 78]. On the contrary, this adaptive controller responds instantaneously at the appearance of failures. The $\mathcal{L}_1$ adaptive controller has numerous other advantages over conventional adaptive controllers[79] (including MRAC and its modifications): guaranteed fast adaptation, limited only by hardware; decoupling between adaptation and robustness, and; guaranteed transient performance for the system's input and output, to name a few.

There are many reports in the literature about the $\mathcal{L}_1$ controller's flight test results. These report successful flight tests [15] and significant failure tolerance to rudder and aileron failures [4, 80, 81].

All these factors made the $\mathcal{L}_1$ adaptive controller an ideal candidate to be implemented as an integral part of SLUGS. The following section presents *the implementation* of an $\mathcal{L}_1$ output feedback controller in SLUGS. It presents a brief description of the architecture and its Simulink model implementation. The reader is referred to References [77, 4, 80, 81, 79] and references therein for a complete treatment of the topic.

5.3 $\mathcal{L}_1$ Output Feedback Controller

The $\mathcal{L}_1$ output feedback architecture as presented in [77] augments the navigation loop's turn-rate command (see Figure 5.1a). To achieve the desired turn rate, the controller reads the current turn-rate and rapidly adapts to augment the inner loop's command. This implementation requires a full PC104 computer and its required power and cabling.

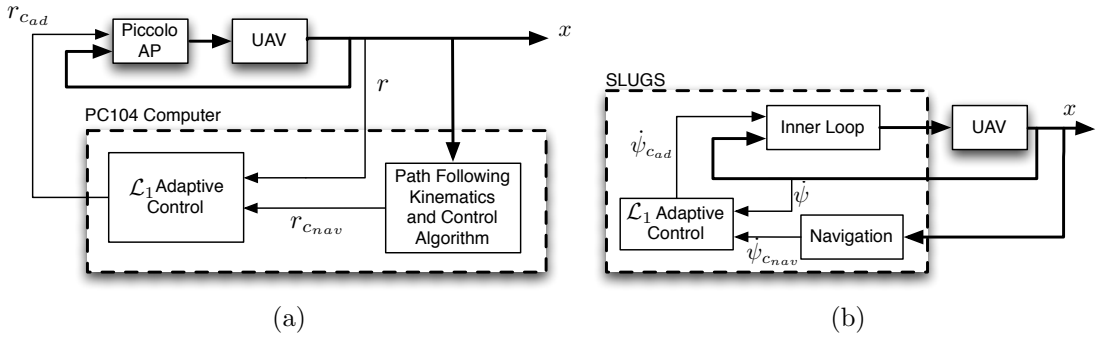


Figure 5.1: $\mathcal{L}_1$ output feedback controller augmenting the navigation command. (a) As implemented in the NPS UAV (after Figure 5 in Reference [4]). (b) SLUGS implementation.

The augmentation implemented in SLUGS, although conceptually the same, allows all the components to run *inside* the same processing unit (see Figure 5.1b). This eliminates all the time delays intrinsic to the implementation shown in Figure 5.1a. It also significantly reduces the required equipment, making it feasible for much smaller aircraft that have power and size restrictions.

Regardless of the implementation, the augmentation guarantees the tracking

of the navigation command[2] in the presence of unmodeled dynamics and bounded disturbances.

The following discussion introduces some algebra of the theory of fast and robust adaptation [85] that is necessary for the onboard implementation of the $\mathcal{L}_1$ adaptive controller.

Let $M(s)$ be the desired UAV's response model for the navigation command ($\dot{\psi}_c$), and $Q(s) = \mathscr{L}\{\dot{\psi}(t)\}$ be the Laplace transform of the UAVs response:

$$Q(s) \approx M(s)Q_c(s). \tag{5.1}$$

Consider, for this discussion, the desired model to be:

$$M(s) = \frac{m}{s+m}, \qquad m > 0.$$

Note that for the output feedback augmentation shown in Figure 5.1b the UAVs response is given by:

$$Q(s) = G_q(s)\left(Q_{ad}(s) + z_q(s)\right), \tag{5.2}$$

where $G_q(s)$ is a strictly proper transfer function and $z_q(s)$ is the Laplace transform of the bounded time-varying disturbances. Equation 5.2 can be rewritten in the

[2]As long as there is enough control authority left.

same form as Equation 5.1 as:

$$Q(s) = M(s)\left(Q_{ad}(s) + \sigma_q(s)\right), \tag{5.3}$$

where $\sigma_q(s)$ lumps all the uncertainties due to $G_q(s)$ and $z_q(s)$:

$$\sigma_q(s) = \frac{\left(G_q(s) - M(s)\right) Q_{ad}(s) + G_q(s) z_q(S)}{M(s)}.$$

The $\mathcal{L}_1$ output feedback controller aims to: **(i)** obtain an estimate of the unknown signal $\sigma_q(s)$, and; **(ii)** define a control signal that compensates for these uncertainties within the bandwidth of a low pass filter $C(s)$. This lowpass filter guarantees that the $\mathcal{L}_1$ controller stays in the low frequency range even in the presence of high adaptive gains and large reference inputs.

The $\mathcal{L}_1$ controller architecture (shown in Figure 5.2) consists of three main components: the control law, the state predictor, and the adaptive law.

The *state predictor*[3] has the same structure as that of Equation 5.3 where the unknowns have been replaced by estimates (denoted by *"hat"*):

$$\dot{\hat{\psi}}(t) = -m\hat{\psi}(t) + m\left(\dot{\psi}_{ad}(t) + \hat{\sigma}(t)\right). \tag{5.4}$$

[3]The state predictor is not to be confused with a *state observer*; the latter estimates the state for time t, while as the former estimates the state for time $t+1$.

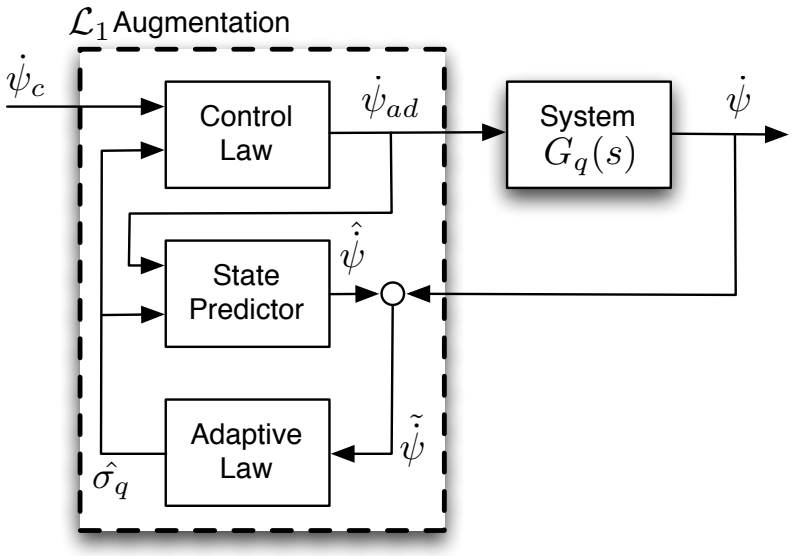


Figure 5.2: $\mathcal{L}_1$ Augmentation architecture. After Figure 4 in Reference [4]

The *adaptive law* estimates the value of the unknown signal σ_q. It is based on the projection operator [82], ensuring boundedness of the adaptive parameters by definition:

$$\dot{\hat{\sigma}}_q(t) = \Gamma_c \text{Proj}(\hat{\sigma}_q(t), -\tilde{\dot{\psi}}(t)), \quad \hat{\sigma}_q(0) = 0, \tag{5.5}$$

where $\tilde{\dot{\psi}}(t) = \hat{\dot{\psi}}(t) - \dot{\psi}(t)$ is the error between the state predictor and the output of the actual system, $\Gamma_c \in \mathbb{R}^+$ is the adaptation rate subject to a computable lower bound, and $\text{Proj}(\cdot)$ denotes the projection operator as defined in Reference [82].

The *control law* generates the augmented turn-rate command $\dot{\psi}_{c_{ad}}(t)$:

$$Q_{c_{ad}}(s) = C(s)\left(Q_c(s) - \hat{\sigma}_q(s)\right), \tag{5.6}$$

where $Q_c(s)$ is a bounded reference input signal with bounded derivative, and $C(s)$ is a strictly proper first order low-pass filer with $C(0) = 1$.

The complete $\mathcal{L}_1$ output feedback adaptive controller is given by Equations 5.4, 5.5, and 5.6 subject to the following *stability condition*: the selection of $C(s)$ and $M(s)$ needs to ensure that

$$H(s) = \frac{G_q(s)M(s)}{C(s)G_q(s) + (1 - C(s))M(s)}$$

is stable. The Simulink implementation is show in Figure 5.3

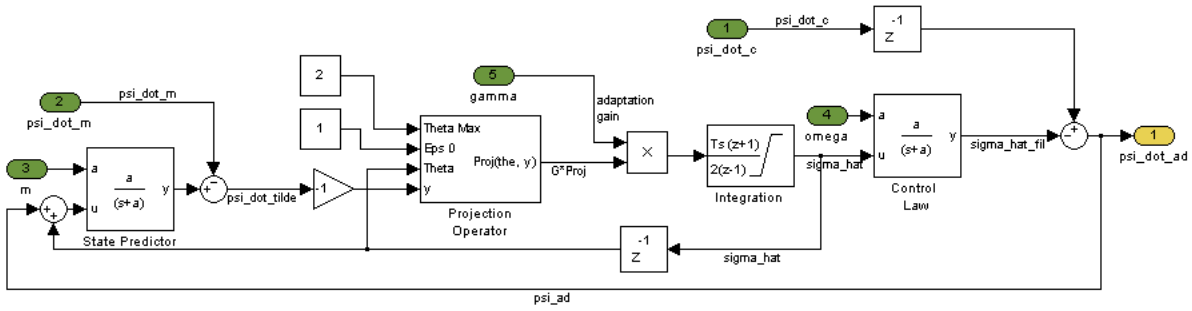


Figure 5.3: $\mathcal{L}_1$ adaptive controller Simulink implementation.

5.4 Failure Tolerance Simulation Results

The simulation scenario used to test the $\mathcal{L}_1$ controller's failure tolerance was the same as the one described in Section 4.4. The only difference was that after 30 seconds of flight a rudder failure was introduced into the 6-DOF plant model.

All the simulations were run for 240 seconds. During the failure the rudder was fixed at $\pm$ 9°. Four different cases were run for these failures: **(i)** the autopilot with no failure tolerance enabled; **(ii)** the autopilot with the sideslip compensator (SSC) enabled; **(iii)** the autopilot with the $\mathcal{L}_1$ adaptive controller enabled, and; **(iv)** the autopilot with the $\mathcal{L}_1$ adaptive controller and the SSC enabled.

To compare the results, two performance measures (PM) were analyzed. The first (PM_1) is the integral of the turn-rate error,

$$PM_1 = \int_0^T \dot{\psi}_e(t)dt = \int_0^T \left(\dot{\psi}_c(t) - \dot{\psi}_m(t)\right) dt \qquad 0 \leq T \leq 240.$$

This exhibits two phases of the test; on the failure onset PM_1 changes rapidly. As the failure tolerance algorithm adapts to the rudder failure, the error is drastically reduced and thus the slope of PM_1 changes. Therefore PM_1 allows one to compare convergence times.

The second PM (PM_2) is the integral of the absolute value of the turn-rate error,

$$PM_2 = \int_0^T \left|\dot{\psi}_e(t)\right| dt \qquad 0 \leq T \leq 240.$$

This PM, although ever increasing, it gives a sense of an *"accumulated error"* and provides a metric upon which to compare equally all the results.

Figure 5.4 presents XY plots for each of the simulation scenarios. Subfigure

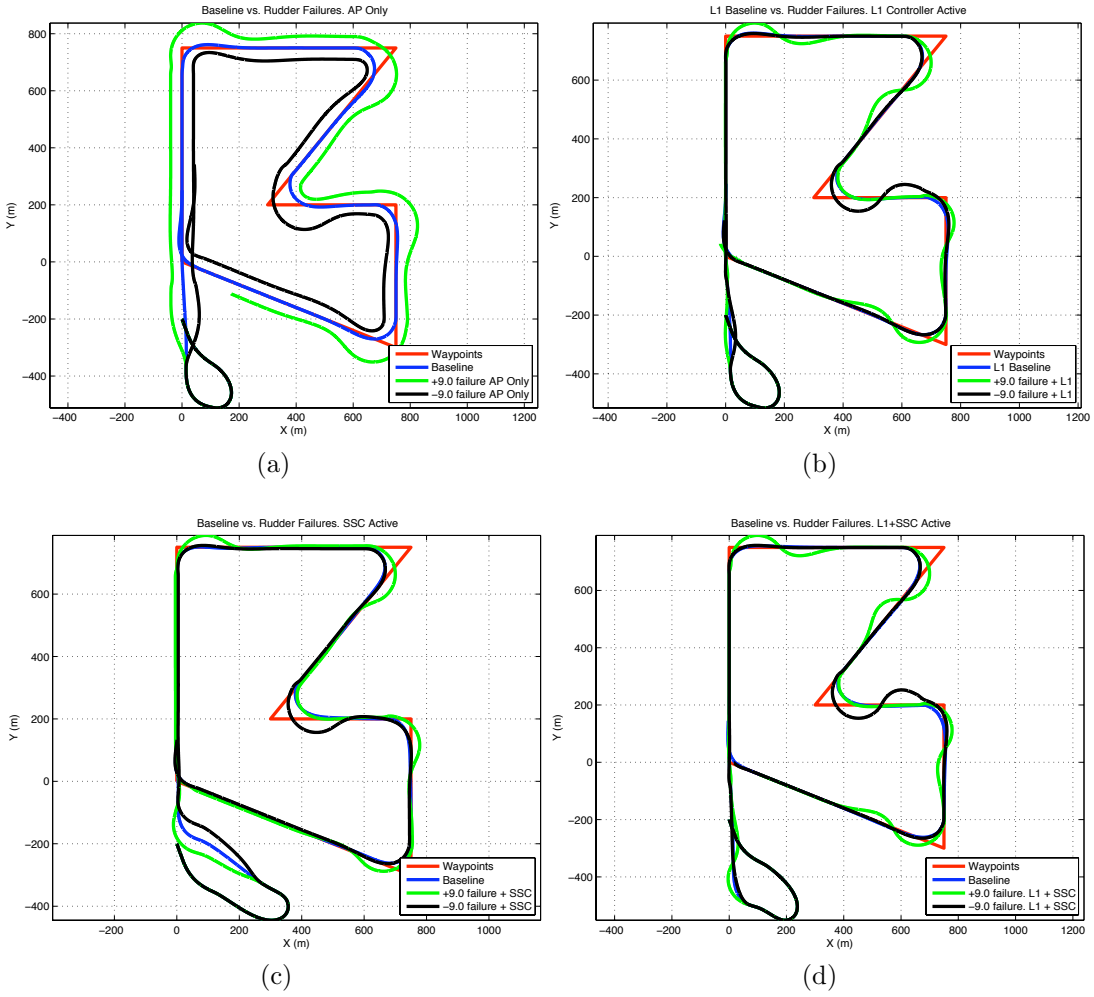


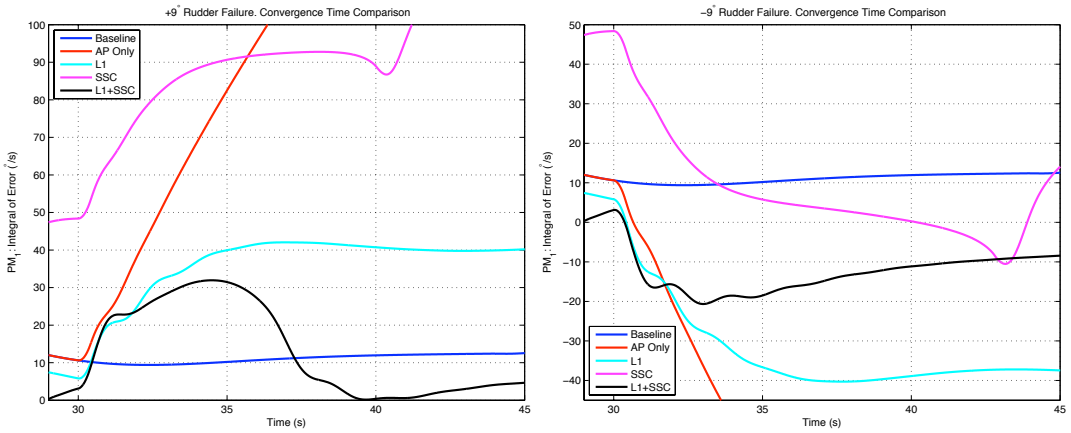
Figure 5.4: XY Plots of the different failure tolerance simulation scenarios. (a) Autopilot Only; no failure tolerance. (b) $\mathcal{L}_1$ adaptive controller.(c) Sideslip Compensator (SSC). (d) $\mathcal{L}_1$ adaptive controller plus sideslip compensator. Colors represent: (red) Waypoint path; (blue) corresponding scenario with no failure; (green) $+9^\circ$ rudder failure; (black) -9° rudder failure.

5.4a presents the UAV trajectory with no failure tolerance enabled. It is quite clear that the UAV is unable to follow the commanded track. Subfigure 5.4b shows a dramatic improvement compared to the no failure tolerance scenario. For

most of the the flight simulation, the UAV stays in the waypoint track. Subfigure 5.4c presents the autopilot with no failure tolerance enabled but using the sideslip compensator (SSC) presented in section 4.2.2.1. This simple, yet effective, addition to the inner loop enables the autopilot itself to cope with rudder failures, without any failure tolerance algorithm. Finally Subfigure 5.4d presents the UAV trajectory with both, the SSC *and* the $\mathcal{L}_1$ controller enabled with a clear improvement over the results shown in Subfigure 5.4a.

Figure 5.5 shows the progression of PM_1 for 16 seconds; one second before and 15 seconds after the failure onset. Note how the SSC by itself (shown in magenta) is noticeably faster in response, but does not converge within the time frame shown on the plot. On the contrary, the $\mathcal{L}_1$ adaptive controller (shown in cyan) and the $\mathcal{L}_1$ + SSC enabled (shown in black), converge about the same time (approximatley 12 seconds after failure onset). Although convergence time is about the same, the magnitude of the $\mathcal{L}_1$ + SSC run is significantly smaller, implying better performance.

The XY plots in Figure 5.4 present an overview of the UAV's performance for the different failure scenarios. Figure 5.6 presents the values of the other performance measure (PM_2) for such evaluation. Interestingly these seem to present different conclusions based on the sign of the rudder failure. Subfigure 5.6a shows a marginally better performance by the $\mathcal{L}_1$ adaptive controller; on the



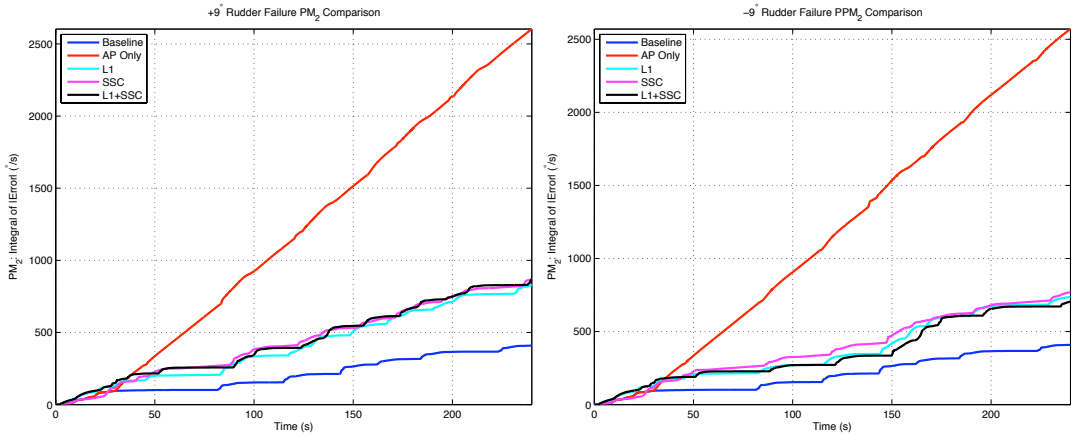
(a) (b)

Figure 5.5: Performance measure PM_1 comparison. (a) $+9°$ rudder failure. (b) $-9°$ rudder failure.

contrary, Subfigure 5.6b clearly shows the case of $\mathcal{L}_1$ + SSC performing better. This is a direct result of the fact that the waypoint track contains four right-turns and only one left-turn. This implies that if the failure *"helps"* the UAV turn right (i.e. the negative failure), then the aircraft will not achieve the commanded turn-rate only once throughout the simulation run. This can be seen on figure 5.4 where all the black plots[4] only diverge from the waypoint track on the waypoint 3 transition. Therefore this suggests that overall the $\mathcal{L}_1$ adaptive controller by itself, on the long run, outperforms the case where the SSC is enabled ($\mathcal{L}_1$ + SSC).

Table 5.1 presents the average of the last twenty computed values for PM_2. The values shown in bold present the best performance. This also shows the fact that the values of PM_2 are noticeably smaller for the negative failures than those

[4]Except in Subfigure 5.4a which has no adaptive algorithm nor compensation.



(a) (b)

Figure 5.6: Performance measure PM_2 comparison. Smaller is better. (a) $+9°$ rudder failure. (b) $-9°$ rudder failure.

obtained for the positive ones.

	No Failure	$+9°$	$-9°$
Autopilot Only	**408.84**	2602.22	2568.97
$\mathcal{L}_1$	410.34	**827.23**	753.93
SSC	413.21	866.41	768.62
$\mathcal{L}_1$ + SSC	412.22	867.32	**704.40**

Table 5.1: Average of the last 20 computed values for PM_2. Smaller is better. Quantities shown in boldface represent the best performance. Quantities represent accumulated error (in $°/s$).

Based on the simulation results presented in this section, the following can be concluded: **(i)** Enabling the SSC with the $\mathcal{L}_1$ adaptive controller ($\mathcal{L}_1$ + SSC) has a smaller initial turn-rate error, as shown in figure 5.5, and; **(ii)** In the long run the $\mathcal{L}_1$ adaptive controller is the best performing of the four scenarios simulated.

With these successful results, the UCSC's UAV was configured to flight test the rudder failures under these same scenarios. The next chapter presents the results of such flight tests.

Chapter 6

Flight Tests Results and Analysis

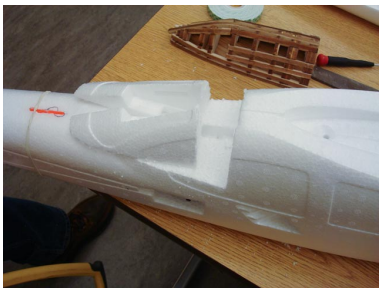
6.1 Introduction

This Chapter presents the results of extensive flight tests conducted during the months of July and August 2009 at UC Santa Cruz.

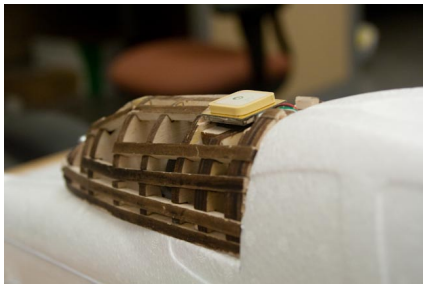
The UAV employed was a modified version of the off-the-shelf Multiplex Mentor Radio Control (RC) aircraft. The modification to the aircraft's fuselage consisted on shaving out a significant section of the "cockpit" (see Figure 6.1a) to make way for a wireframe pod[1]. This pod, made out of balsa wood, securely housed the autopilot and the radio modem (See figure 6.1b). The purpose of this pod was twofold: on one side it aimed at protecting the main avionics in case of a

[1]This pod was designed and built by Hyuk Choong Kang, another graduate student at UCSC's Autonomous Systems Lab (ASL).

crash; on the other hand, it provided a great vibration-isolation environment on which to mount the autopilot. As a side result, this also made the core avionics trivial to mount and unmount from the UAV.



(a)



(b)

(c)

Figure 6.1: External modifications to the Multiplex Mentor RC aircraft. (a) Cutting of the "cockpit" to mount the autopilot pod. (b) Autopilot pod mounted.(c) Full view of the aircraft fuselage with the wings and pod mounted (green arrow indicates the pod).

The ground station software was configured and installed on a Pentium-class

laptop computer. This laptop, along with the electronics required to power the GCS radio modem were installed inside a briefcase[2] (See Figure 6.2) to make the complete system portable and easily deployable.

Figure 6.2: Portable ground station. The ground station briefcase contained the the laptop running the ground station software and a shade screen to facilitate its operation in direct sunlight. The briefcase also housed the radio modem, battery, its antenna, and power circuitry. The red streamer is for use as a wind sock to the safety pilot.

During the first three weeks of flight tests, the UAV was flown under pilot

[2]The briefcase housing and mounting was designed and developed by Max Dunne and Craig Moriwaki, both undergraduate students at UCSC's ASL.

control. The collected flight data were analyzed offline to verify and validate the inner workings of the autopilot. This initial test period was also used to tune position and attitude estimation filters as reported in Reference [83].

Once the position and attitude estimation filters were proven to work, one week of flight tests (consisting of more than nine flights) was devoted to tune the PID gains for the inner loop. The tuning process consisted in adjusting the gain of each controller one-at-a-time during flight. The process started with the longitudinal channel and then move on to the lateral channel. For this, a SLUGS autopilot *selective passthrough* (SPT) mode was employed. The SPT mode allowed to configure, from the ground station software, which control surfaces remain under pilot's control and which are under the autopilot's inner loop control. In this way, the longitudinal channel (elevator and throttle) was first tuned while the pilot retained full control of the lateral channel (rudder and ailerons). Finally the pilot was left with only the rudder to keep the UAV within the allocated fly-zone. In order to tune the turn-coordinator (rudder) loop, aileron control was returned to the pilot in order to keep the aircraft within the fly-zone while the gains were adjusted to give reasonable performance. With that the full inner loop was tested under different conditions to verify the correct tracking of commands – altitude, airspeed, and turn-rate. This process was greatly aided by the ability to plot commanded and actual variables in real time on the GCS software.

6.2 Experimental Setup

The primary objective of experimental testing was to validate the results obtained from computer and HIL simulations. The secondary objective was to validate the complete *simulation* → *HIL simulation* → *flight test* workflow in SLUGS.

The waypoint track set for the flight test experiments consisted of four waypoints defining two short and two long legs (roughly defining a rectangle). For purposes of all the XY plots presented in this section, the ground station was physically located at the origin of the plots.

Using this waypoint track, four flight test sets were selected. Each with several different variants. These tests aimed at validating an aspect of the system and at the same time keep the aircraft safe. The flights were always performed within visual range of the safety pilot, thus the short separation between waypoints. Although great care was taken in performing each test and its variants under similar conditions, there are environment factors that are impossible to control, especially wind gusts. All these are undoubtedly reflected in the results. Care was taken not to fly if the steady wind exceeded five knots (i.e. one third of the UAV nominal airspeed).

The data presented here are as it were directly received from the autopilot through the ground station. No further processing (i.e. filtering, smoothing, etc.)

has been performed on it[3].

Each experiment and its results are presented in the following subsections. Each one clearly states the experiment's objectives, its variants, a brief description of the experiment execution, the results obtained and the analysis of such results.

6.2.1 Flight Test 1: Nominal Control

In order to validate the nominal performance of the SLUGS autopilot, a series of test were undertaken to validate the system. Both the inner and outer loop performance need to be evaluated in a flight test environment. To this end, the following objectives are posed:

Objectives:

1. To validate the autopilot's capability to follow the desired waypoint track.

2. To asses the performance of the inner loop to track the inner loop's controlled variables.

3. To asses the impact of the $\mathcal{L}_1$ adaptive controller and the SSC on the overall operation with no control surface failure using the turn rate performance measures defined in Section 5.4.

[3]The data sets used to generate the plots and tables in this chapter are available at the main SLUGS website.

4. To define new PMs that relate to the UAV's position errors and compare each variant using these PMs.

5. To verify the navigation loop L_2 and η computations.

6. To asses the DSC processing power required for each variant.

The autopilot variants tested are with the $\mathcal{L}_1$ adaptive control on and off, and with the autopilot's SSC on and off, yielding four test variants[4]:

Variants:

1. The autopilot by itself (AP).

2. The autopilot with the $\mathcal{L}_1$ adaptive controller enabled (L1AC).

3. The autopilot's SSC enabled (APSSC).

4. The $\mathcal{L}_1$ adaptive controller and the autopilot's SSC enabled (L1AC+APSSC).

Execution: The UAV was commanded to conduct two complete cycles around the waypoint track[5]. The variants were conducted back to back to attempt having similar environment conditions for all the variants.

Results: Figure 6.3 presents the XY plots for each of the four variants in this experiment.

[4]In further discussions the variants will be addressed by their acronyms listed in parenthesis at the end of the variant descriptions

[5]Two cycles around the waypoint track were selected to make sure that in the presence of overshoot or divergence, if any, it was due to wind gusts rather than performance of the variant being evaluated.

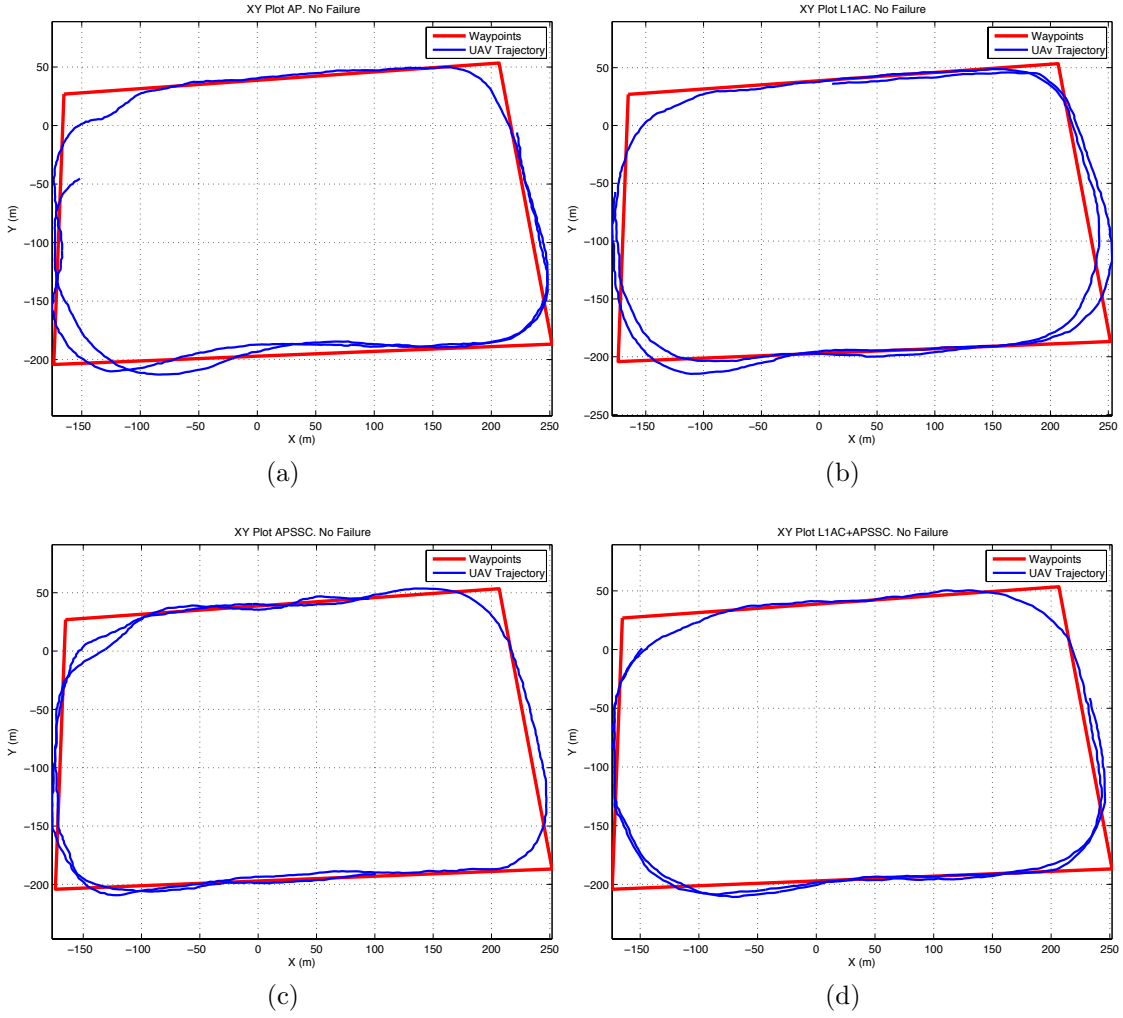


Figure 6.3: XY plots for each of flight test 1 variants. (a) Autopilot only (AP). (b) $\mathcal{L}_1$ adaptive controller enabled (L1AC). (c) Inner loop's SSC enabled (APSSC). (d) $\mathcal{L}_1$ adaptive controller and SSC enabled (L1AC+APSSC).

Figure 6.4 shows the four inner loop controlled variables – roll angle ϕ, pitch angle θ, lateral acceleration in body frame A_y, and airspeed U

Figure 6.5 presents the comparisons of PM_1 and PM_2.

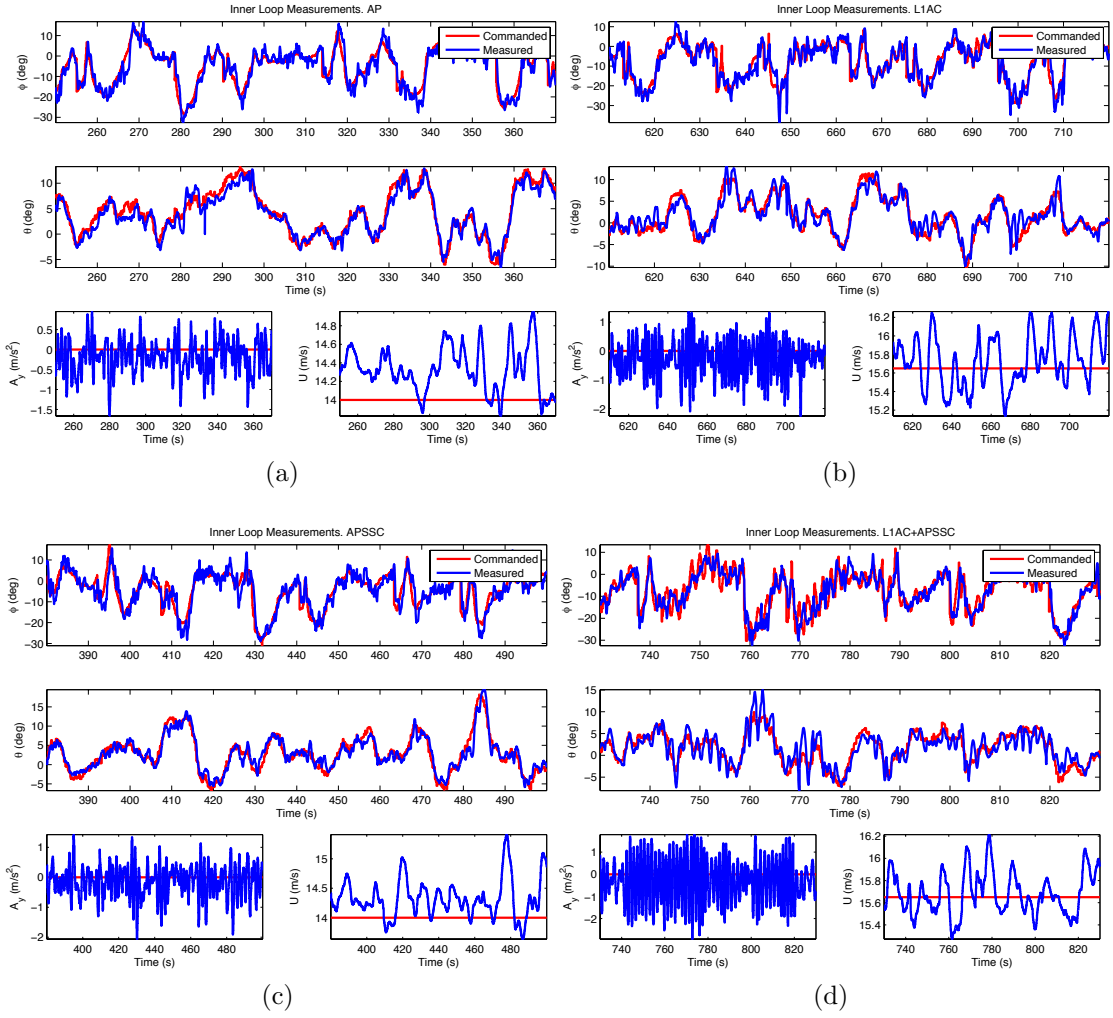
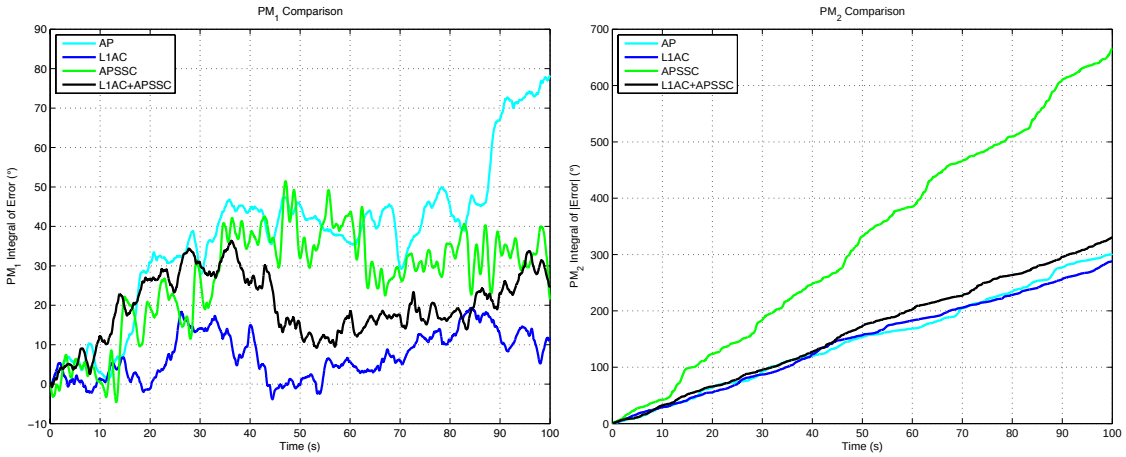


Figure 6.4: Flight tests results for inner loop stabilization. (a) Inner loop autopilot only (AP). (b) $\mathcal{L}_1$ adaptive controller enabled (L1AC).(c) Inner loop's SSC enabled (APSSC). (b) $\mathcal{L}_1$ adaptive controller and the SSC enabled (L1AC+APSSC).

For flight tests two new PMs were developed that reflected the UAVs position errors[6]. The first, PM_3 is the integral of the absolute position error with respect

[6] PM_1 and PM_2 reflect turn rate errors. Those variants that do not include an $\mathcal{L}_1$ adaptive controller the error is with respect to the commanded turn rate $\dot{\psi}_c$. For those that do include the $\mathcal{L}_1$ adaptive controller the error is computed with respect to the reference model $M(s)$ as described in Section 5.3



(a) (b)

Figure 6.5: Flight test 1 turn rate performance measures comparison. (a) PM_1. (b) PM_2.

to an ideal reference path shown in Figure 6.6. For each point in the data set, the shortest distance to the path is computed and then integrated over time:

$$P_e = \int_0^T |P_{uav} - P_{rt_{closest}}| dt \qquad 0 \leq T \leq F_t, \tag{6.1}$$

where F_t is the flight time of for a given flight test variant.

The second PM, PM_4 is the cumulative distribution function [84] of the absolute position errors. This performance measure computes the probability (reflected in the y-axis in the plots) of a position error (in meters) to exceed a given value (reflected in the x-axis in a plot).

Figure 6.7 presents the comparison of PM_3 and PM_4.

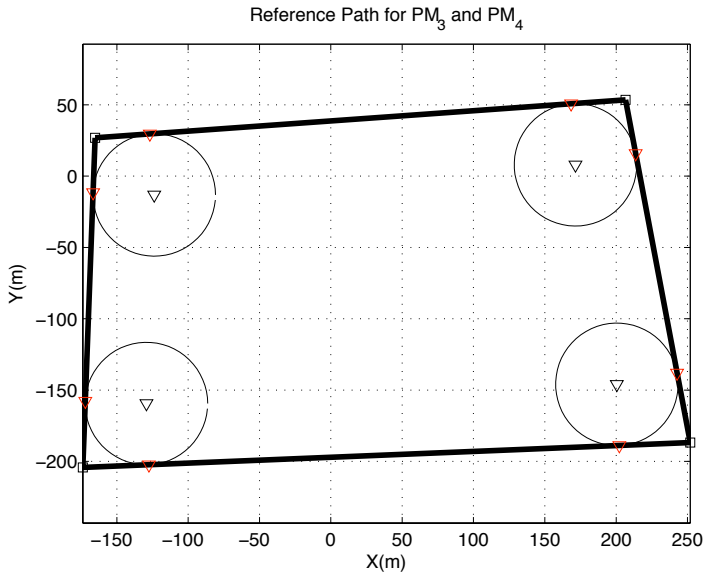
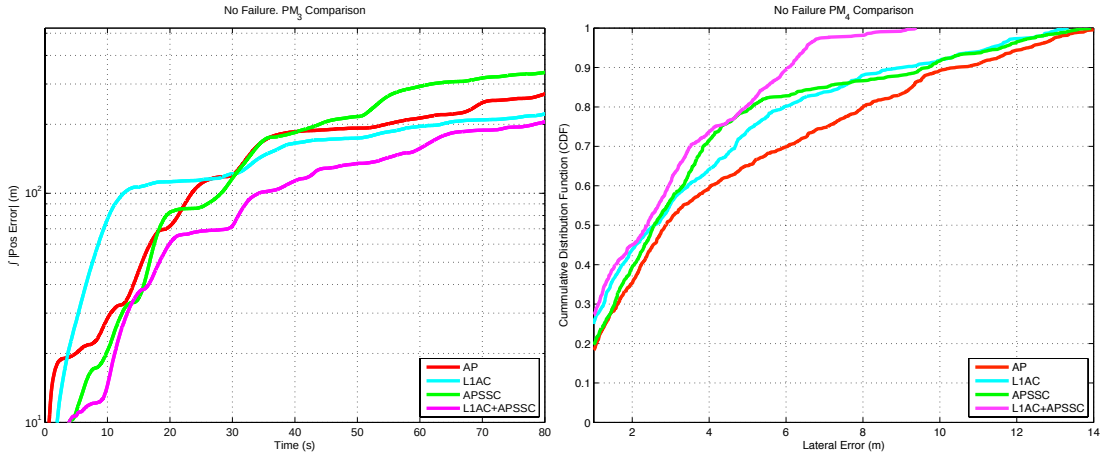


Figure 6.6: Reference trajectory used to compute the position errors for PM_3 and PM_4. The red triangles indicate the transition point as described in Section 4.3.3



(a) (b)

Figure 6.7: Flight test 1 position performance measure comparisons. (a) PM_3. (b) PM_4.

Figure 6.8 presents a sequence of eight frames showing the UAV's position (in

blue) and the location of the L_2 vector during the execution of variant AP.

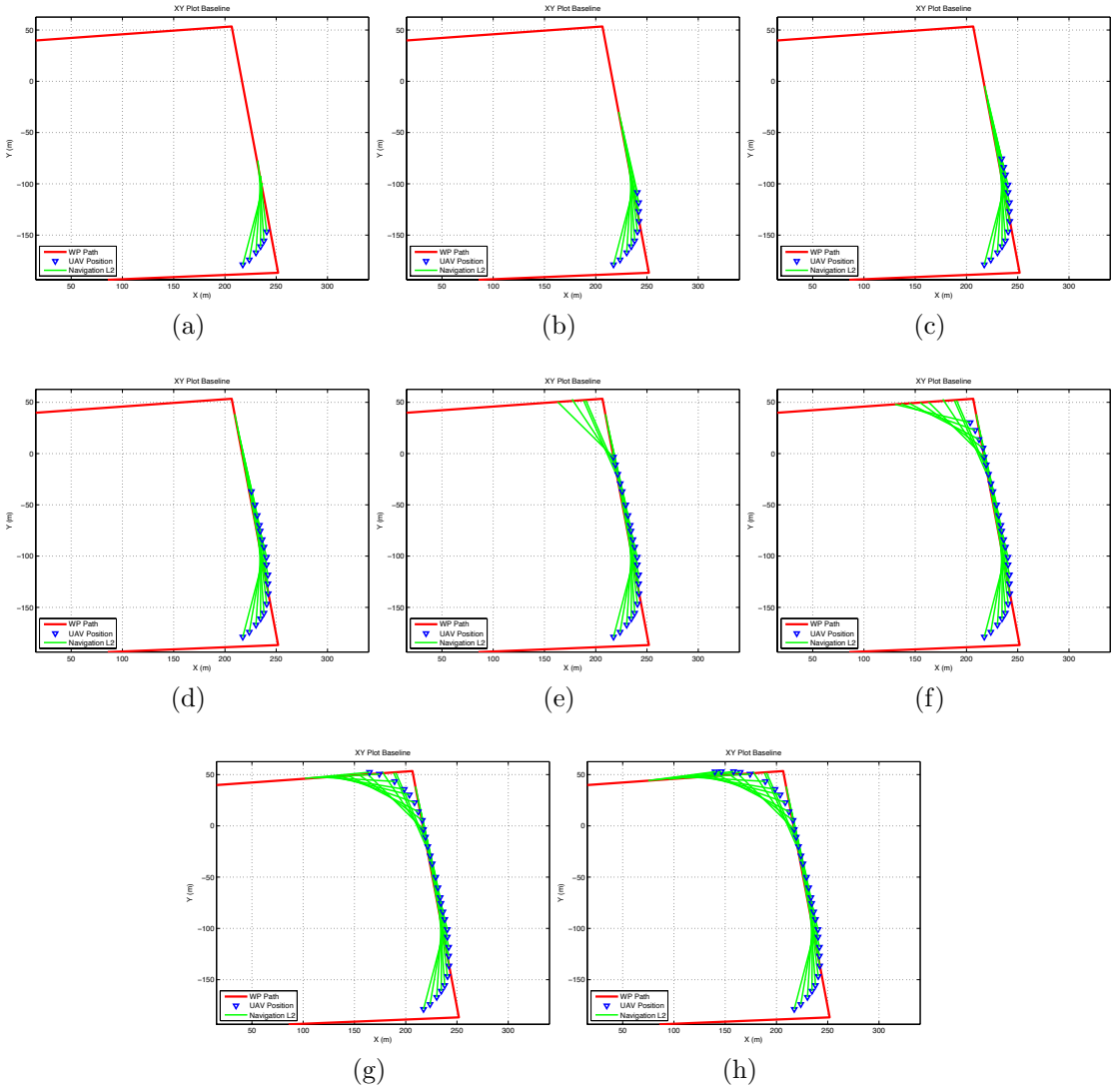


Figure 6.8: Time progression of the computation of the L_2 lookahead vector. UAV positions (in blue) are shown for every other 50 known positions. Time progression goes from (a) to (h).

Figure 6.9 compares the control DSC's computational load (in percentage) for

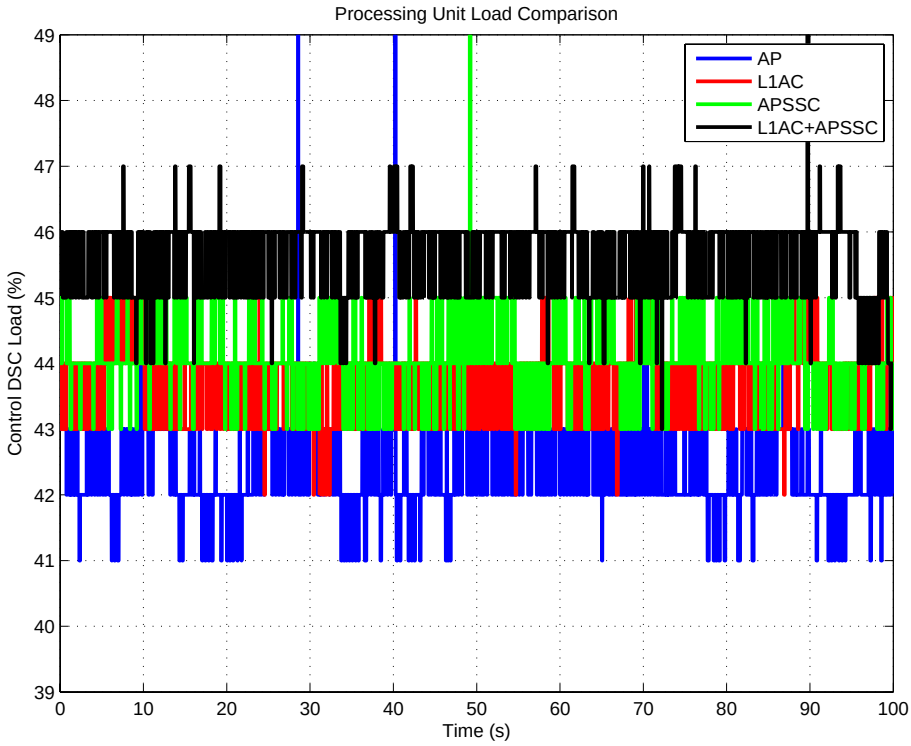


Figure 6.9: Computational load comparison for each variant in flight experiment 1.

each of the variants.

Analysis: Although Figure 6.3 shows some minor differences among the different variants, they all show that indeed the autopilot is capable of maintaining the UAV in the desired waypoint track. Analyzing the plots shown in Figure 6.3, variants L1AC and L1AC+APSSC clearly outperform the others in the straight segments. During turns, variant APSSC exhibits no overshoot during the transitions but very oscillatory performance throughout the flight for the particular choice of navigation parameters used in these experiments.

Figure 6.4 shows that the augmentations of the autopilot either by using the SSC, or the $\mathcal{L}_1$ output feedback, or both, do not hamper the inner loop's capability to track the commanded values. Worthy of noting is that some variants, specially L1AC and L1AC+APSSC do result in a more active control system with some minor oscillatory pattern. These, as supported by subfigures 6.3b and 6.3d, do not have any impact in the overall performance of the navigation. On the contrary these seem to improve the UAV's waypoint tracking performance by reducing overshoot and improve line tracking when compared to variant AP.

Recall that PM_1 was selected to show the convergence times for the $\mathcal{L}_1$ adaptive controller and the SSC. In this case since there is no failure involved, PM_1 can be though of as an "accumulator" of the error where a negative error "cancels" a positive error[7]. So With that in mind, PM_1 shown in subfigure 6.5a supports what has been previously discussed: *on average* the $\mathcal{L}_1$ and the SSC do indeed make the autopilot perform slightly better.

On the contrary, PM_2 "accumulates" the absolute value of the error, and therefore positive and negative errors do not "cancel" each other. This PM shows how the inclusion of the $\mathcal{L}_1$ adaptive controller in the autopilot has a negligible negative impact on the performance. As expected from analyzing Subfigure 6.3c and comparing it with the other variants, variant APSSC is the worst performing of the four variants.

[7]A positive or negative slope in the plot indicates a constant positive or negative error.

Regarding the position errors, both PM_3 and PM_4 confirm what is noticeable in the XY plots shown in Figure 6.3. The L1AC+APSSC variant performs much better with regards to cross-track errors. Particularly, PM_4 shows that the position errors in the L1AC+APSSC variant never exceed 9 m while all others have instances were 14 m errors occur (both likely at waypoint switch).

Regarding the computation of the L_2 lookahead vector, the time progression in Figure 6.8 shows how the vector is correctly positioned in the waypoint track. It also demonstrates the waypoint segment switching logic working successfully.

Finally, the computational load comparison (Figure 6.9) of each of the variants show how enabling either the $\mathcal{L}_1$ adaptive controller, the SSC, or both, has a negligible computational impact. On average the $\mathcal{L}_1$ adaptive controller and the SSC increase the computational load by 1% and 2% respectively. As expected enabling both on average increases the load by 3%. This increase in computational load, for all the flight tests performed, is considered to be negligible.

6.2.2 Flight Test 2: $+4.5^\circ$ Rudder Failure

The second flight test was to examine the performance of the autopilot in the presence of a moderate actuator failure. This case was with a fixed rudder failure of 4.5°.

Objectives:

1. To validate the autopilot's capability to recover from a mild rudder failure.

2. To asses the impact of the $\mathcal{L}_1$ adaptive controller and the SSC on the presence of rudder failures using the performance measures defined in Section 5.4.

3. To asses the impact of the $\mathcal{L}_1$ adaptive controller and the SSC on the presence of rudder failures using the position error performance measures defined for flight test 1.

The same four variants of the control software were used as were for Flight Test 1.

Variants:

1. The autopilot by itself (AP).

2. The autopilot with the $\mathcal{L}_1$ adaptive controller enabled (L1AC).

3. The autopilot's SSC enabled (APSSC).

4. The $\mathcal{L}_1$ adaptive controller and the autopilot's SSC enabled (L1AC+APSSC).

Execution: The UAV was commanded to conduct a single complete cycle around the waypoint track to establish the performance baseline. After that, the variants were conducted back to back, interleaving a non-failure cycle around the

waypoint track between them. The failure onset and end was controlled from the ground station software. For all variants, the failure was started after coming out the turn in the top right waypoint shown in the XY plots.

Results: Figure 6.10 presents the XY plots for the baseline as well as each of the four variants in this experiment.

The performance measures plots are shown in Figure 6.11. Subfigure 6.11a shows the values for PM_1 during the first 40 seconds of each flight. Note that failure onset time for the different variants has been adjusted to make it coincide in the plot for PM_1.

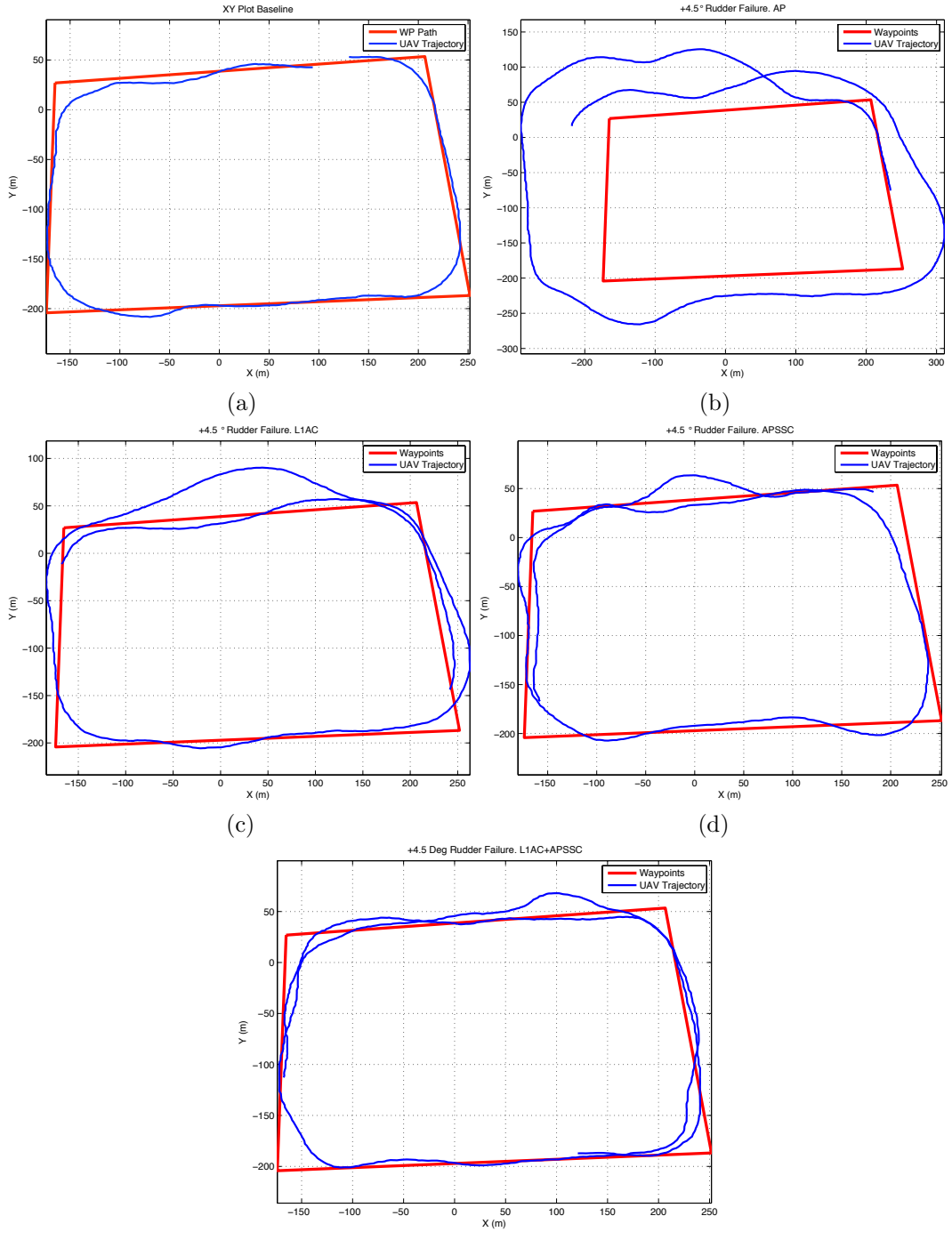


Figure 6.10: XY plots for Flight Test 2: +4.5° rudder failure. (a) Baseline run, autopilot only no failure. (b) AP. (c) L1AC. (d) APSSC. (e) L1AC+APSSC.

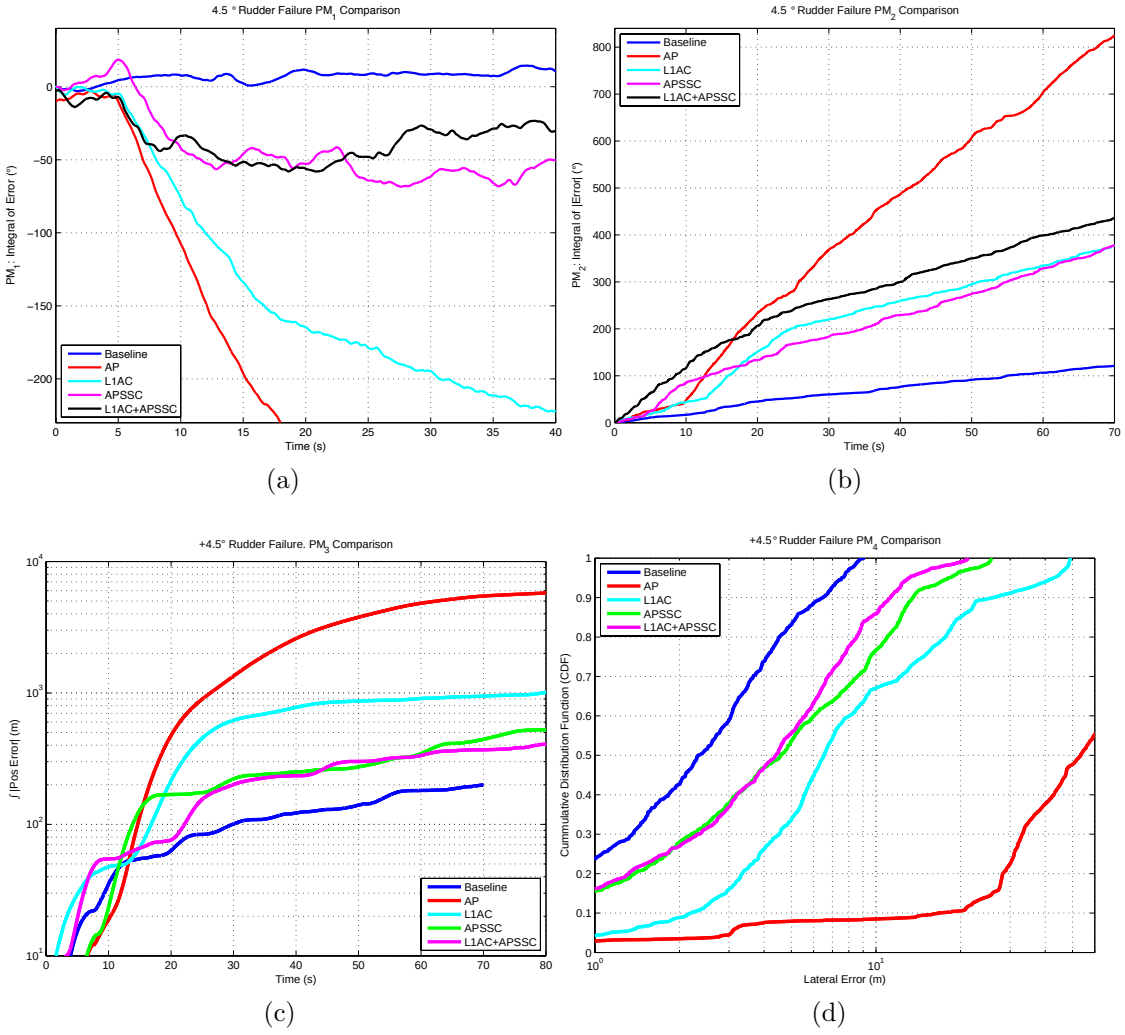


Figure 6.11: Performance measures comparison for the variants of Flight Test 2: +4.5° rudder failure. (a) PM_1. (b) PM_2. (c) PM_3. (d) PM_4

Analysis: The results shown in Subfigure 6.10b show that even after a mild rudder failure the autopilot by itself is incapable of following the desired waypoint track. This is consistent with the results obtained during simulation (see Subfigure 5.4a).

Enabling the $\mathcal{L}_1$ adaptive controller produces a dramatic improvement (Subfigure 6.10c) where now the UAV is able to follow the waypoint track with very small overshoot during waypoint segment transitions. Once again this is consistent with the results obtained during simulation. The only drawback is that the overshoot at the failure onset is significant (almost 50 m). This could be a problem if, for instance, the UAV was restricted to a tight *no-fly zone*.

The use of the SSC reduces the overshoot (see Subfigure 6.10d) but once again the performance along the waypoint track is noticeably worse than that of the $\mathcal{L}_1$ adaptive controller. Nevertheless, it is still superior in every way to that of the autopilot by itself.

Finally, using the $\mathcal{L}_1$ adaptive controller with the SSC enabled reduces the overshoot even more(see Subfigure 6.10e) and although the tracking along the lines exhibits oscillations, there is practically no overshoot during the turns.

The evolution of PM_1 shown in Subfigure 6.11a clearly shows the convergence behavior of each of the variants. The $\mathcal{L}_1$ adaptive controller (shown in cyan) although very smooth, it takes much longer than the SSC (magenta) and the $\mathcal{L}_1$ + SSC (black) variants. These two (the SSC and $\mathcal{L}_1$ +SSC) although faster, clearly exhibit a more oscillatory behavior.

Overall, variants L1AC, APSSC, and L1AC+APSSC do perform worse than the established baseline (see Subfigure 6.11b). But, considering the rudder fail-

ure, they are all significantly better than the autopilot by itself (variant AP). In the evolution of PM_2, variant L1AC (shown in cyan) gets heavily penalized for the large initial transient. On the other hand, variants SSC (magenta) and L1AC+APSSC (black) although not having a significant transient, their oscillatory behavior accumulates in the PM. Based on PM_2, despite the significant transient, the $\mathcal{L}_1$ adaptive controller outperforms all other variants. But this is not true for the position PMs; these reflect and severely penalize variant L1AC for the 50 m overshoot on failure onset. PM_3 and PM_4 support what is noticeable in XY plots shown in figure 6.10: The inclusion of the SSC reduces the overshoot on failure onset and overall offers better position tracking. Note also how in PM_4 there is only an 11 m difference between the worst case error for the baseline and that of the L1AP+APSSC variant which includes a rudder failure.

6.2.3 Flight Test 3: -8° Rudder Failure

The third flight test was to examine the performance of the autopilot in the presence of a severe actuator failure. This case was with a fixed rudder failure of -8° . The objectives were:

Objectives:

1. To validate the autopilot's capability to recover from a severe rudder failure.

2. To asses the impact of the $\mathcal{L}_1$ adaptive controller and the SSC on the pres-

ence of rudder failures using the turn rate performance measures defined in Section 5.4 and the position error ones defined for flight test 1.

3. Asses and compare the contributions of the SSC and the $\mathcal{L}_1$ adaptive controller to the turn-rate command.

For this flight test only three variants were used. For safety reasons the autopilot by itself (i.e. with no $\mathcal{L}_1$ adaptive control nor SSC enabled) was not used. The variants are:

Variants:

1. The autopilot with the $\mathcal{L}_1$ adaptive controller enabled (L1AC).

2. The autopilot's SSC enabled (APSSC).

3. The $\mathcal{L}_1$ adaptive controller and the autopilot's SSC enabled (L1AC+APSSC).

Execution: The UAV was commanded to conduct a single complete cycle around the waypoint track with the failure and controllers configured according to the variant. The variants were conducted back to back, interleaving a non-failure cycle around the waypoint track between them. The failure onset and end was controlled from the ground station software. For all variants, the failure was started after coming out the turn in the top right waypoint shown in the XY plots.

Results: Figure 6.12 presents the XY plots for each of the three variants in this experiment.

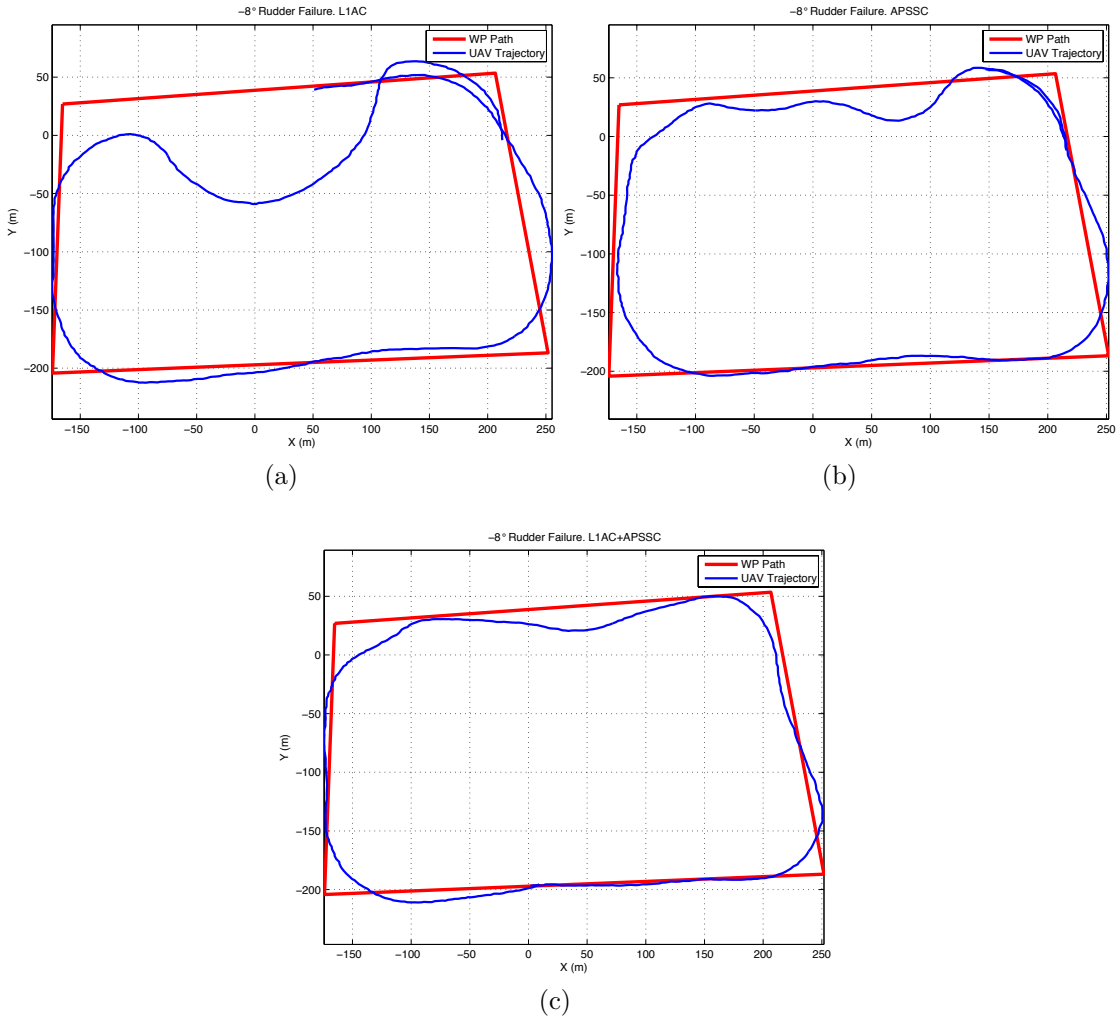


Figure 6.12: XY plots for Flight Test 3: $-8.0°$ rudder failure. (a) L1AC. (b) APSSC. (c) L1AC+APSSC.

Figure 6.13 shows the performance measures comparison[8]. Note that failure

[8]The baseline PM included in this plot is that of the result from flight test 2. Flight test 3 was conducted in the same day immediately after flight test 2, so the same baseline was used.

onset time for the different variants has been adjusted to make it coincide in the plot for PM_1.

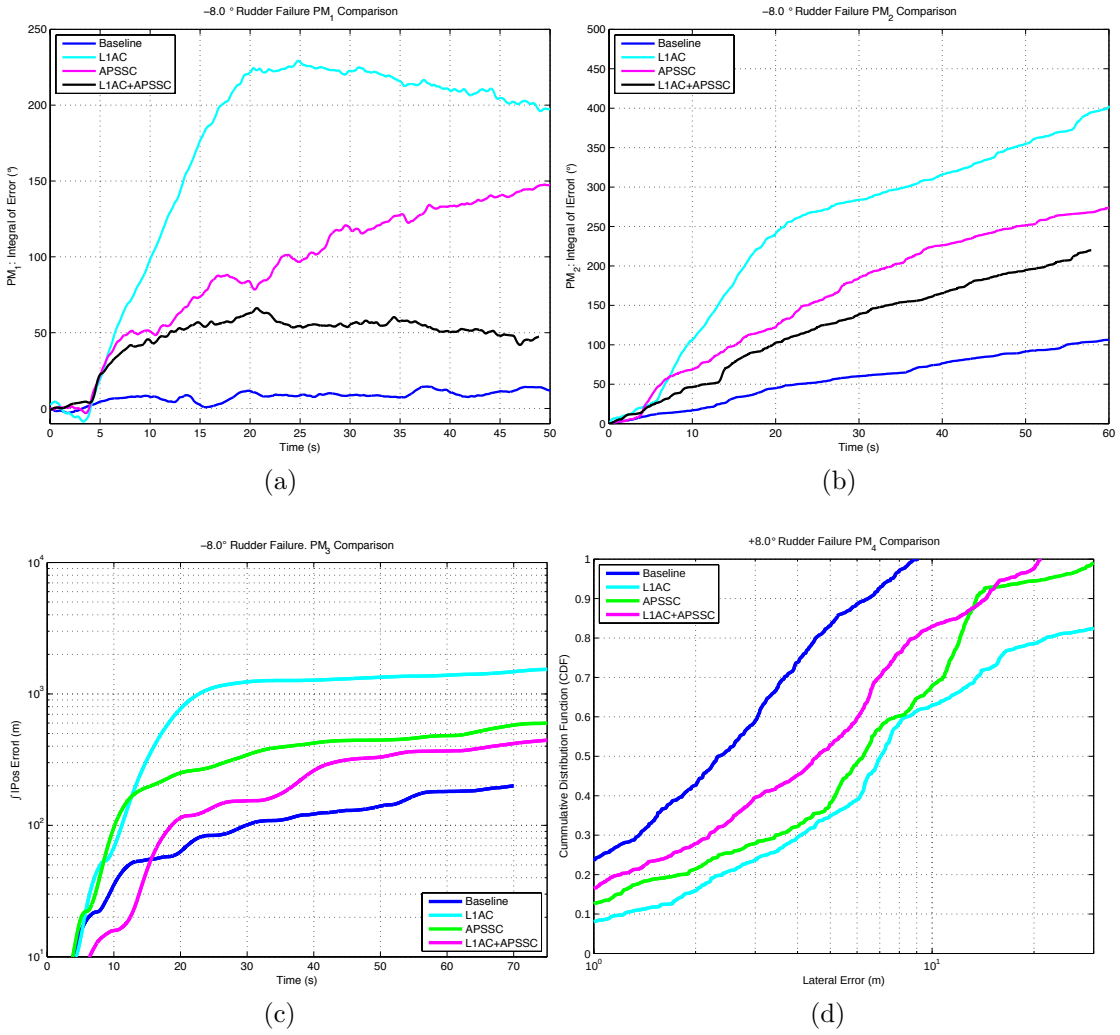


Figure 6.13: Performance measures comparison for the four variants of Flight Test 3: $-8°$ rudder failure. (a) PM_1.(b) PM_2.(c) PM_3.(d) PM_4.

Figure 6.14 presents a comparison between the $\mathcal{L}_1$ adaptive controller and the SSC contribution. The plot compares $\hat{\sigma}(t)$ to $\dot{\psi}_{ssc}(t)$. The $\mathcal{L}_1$ controller

contribution, $\hat{\sigma}(t)$, is defined in Equation 5.6. The SSC contributions, $\dot{\psi}_{ssc}(t)$, is derived from Equation 4.2.

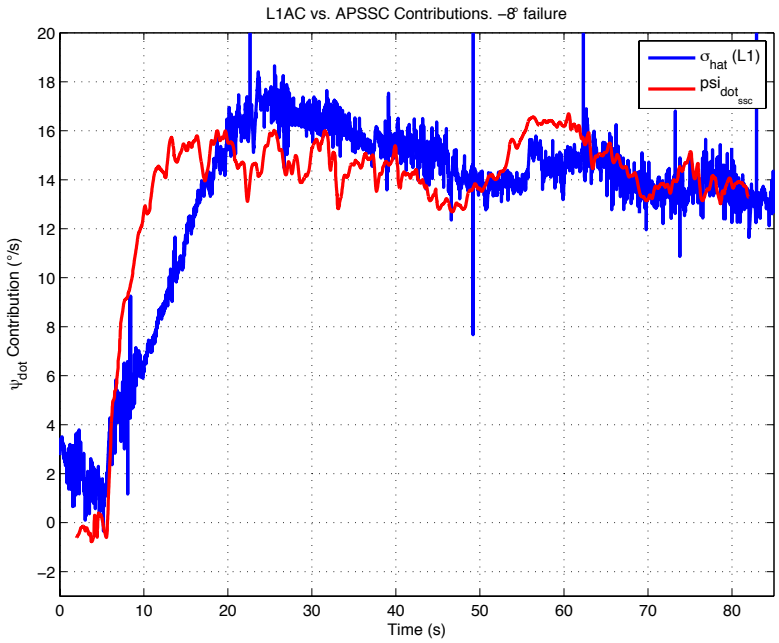


Figure 6.14: Turn-rate contribution comparison between L1AC and APSSC variants.

Analysis: With the presence of a more severe failure than in the previous experiments, the advantages and disadvantages of each variant becomes more evident. Subfigure 6.12a shows how the $\mathcal{L}_1$ adaptive controller has a very significant overshoot upon failure onset. But, once converged, the $\mathcal{L}_1$ adaptive controller effectively maintains the UAV in the desired waypoint track. Enabling the SSC (Subfigure 6.12b) noticeably reduces the overshoot present with the $\mathcal{L}_1$ adaptive controller. Nevertheless, there are some oscillations present UAV recovers from the failure onset.

By enabling both, the $\mathcal{L}_1$ adaptive controller and the SSC (Subfigure 6.12c) the significant overshoot present in variant L1AC and the initial oscillatory behavior of variant APSSC are no longer noticeable. This variant clearly outperforms the other two.

The PM_1 plot, on Subfigure 6.13a, clearly shows how, variants L1AC and L1AC+APSSC rapidly converge (within 20 seconds) thus showing a horizontal progression. On the contrary, variant APSSC seems to be ever growing as a result of a constant positive turn rate-error. Worthy of noting is that, even though the $\mathcal{L}_1$ adaptive controller converges after a relatively short transient (about 18 seconds), both PM_1 and PM_2 (shown in Subfigure 6.13b) heavily penalize its significant transient. Thus, due to its large transient, the $\mathcal{L}_1$ adaptive controller results in the worst performing of the variants. Supporting what was observed in the XY plots of Figure 6.12, the PMs show that enabling the SSC with the $\mathcal{L}_1$ adaptive controller offers a significant improvement in transient and steady state.

All four PMs coincide that variant L1AC+APSSC is the best performing. Worthy of noting is the fact that despite the difference in magnitude of the failure between this and the previous experiment, PM_4 for variant L1AC+APSSC is not significantly different suggesting an excellent performance under a wide variety of rudder failures and insensitivity to the sign of the rudder failure.

Figure 6.14 shows the computed value of $\hat{\sigma}(t)$ for the $\mathcal{L}_1$ adaptive controller case, and the $\dot{\psi}_{ssc}(t)$ for the SSC. These contributions were computed during the execution of variants L1AC and APSSC therefore they belong to separate executions. Its time progression has been matched to show the evolution of each. Note how the $\mathcal{L}_1$ adaptive controller with no *a priori* knowledge of the failure, matches quite close the computation made by the SSC which has been designed exactly to compensate for this. This plot also shows that although both converge to approximately the same magnitude, the SSC has a shorter convergence time, which in turn, results in smaller overshoot upon failure onset. Three approaches could be taken to solve this issue in the $\mathcal{L}_1$ adaptive controller implementation. These will be discussed in Section 7.3 in the final chapter of this dissertation.

6.2.4 Flight Test 4: -9.5° Rudder Failure

Objective: To analyze the effect of the SSC when used in conjunction with the $\mathcal{L}_1$ adaptive controller in reducing the overshoot on a severe rudder failure onset.

Variants: Two variants were used:

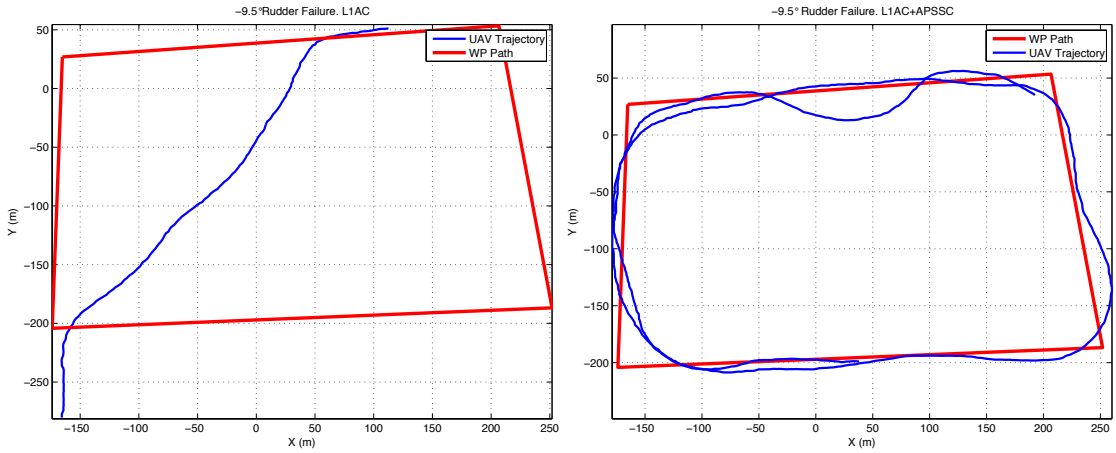
1. The autopilot with the $\mathcal{L}_1$ adaptive controller enabled (L1AC).

2. The $\mathcal{L}_1$ adaptive controller and the autopilot's SSC enabled (L1AC+APSSC).

Execution: The variants were conducted back to back, interleaving a non-

failure cycle around the waypoint track between them. The failure onset and end was controlled from the ground station software. For both variants, the failure was started after coming out the turn in the top right waypoint shown in the XY plots.

Results: Figure 6.15 presents the XY plots for both variants of this experiment.



(a) (b)

Figure 6.15: XY plots for Flight Test 4: $-9.5°$ rudder failure. (a) L1AC. (b) L1AC+APSSC.

Analysis: The XY plot of variant L1AC (shown in Subfigure 6.15a) shows how the UAV after the failure onset, exhibits an overshoot that takes it to the other side of the waypoint track. At this point the safety pilot takes control of the UAV for safety reasons. On the contrary, by enabling the SSC in conjunction with the $\mathcal{L}_1$ adaptive controller, the overshoot although noticeable, allows the UAV to continue following the waypoint track (Subfigure 6.15b).

Chapter 7

Conclusions and Future Work

7.1 Final Remarks

This dissertation has presented the complete design, development, and flight testing of a novel UAV GNC research platform: SLUGS. The autopilot and its related software are unique in the sense that they provide a complete (hardware and software) platform for UAV research and development without requiring in-depth knowledge of the low-level hardware or software. By using Simulink as the primary development platform, UAV researchers can easily transition between software simulation, HIL simulation, and flight testing without the need to *re-code* the algorithms in a traditional programming language. The HIL simulator offers full freedom to modify the simulation plant model and environment directly in Simulink, yielding unprecedented flexibility and versatility.

The UAV GNC research platform presented here was flight tested under nominal flying conditions with very successful results. To showcase the system's advantages of flexibility, versatility, and ease of re-programmability, an $\mathcal{L}_1$ adaptive controller, previously reported in the literature [77, 4, 80, 81, 79], was implemented and verified to work in simulation, HIL simulation, and flight tests. The results obtained for rudder failure tolerance are conclusive in showing in-flight failure resilience. Overall, flight tests explicitly demonstrate SLUGS as a viable and tested platform for UAV research and development.

7.2 Conclusions

In summary, this dissertation has presented:

1. The design, development, and flight test of a low-power, low-cost, lightweight UAV autopilot that is easily reprogrammable directly from Simulink. By developing the hardware and the low-level peripheral drivers, the complete hardware layer has been abstracted to a set of Simulink sources and sinks. Users of SLUGS as a development platform need only focus on the development of the GNC algorithms in Simulink. The same Simulink model used for software simulation can be compiled and downloaded directly to the autopilot for both HIL simulation and flight tests.

2. The design of a novel, two-processor hardware architecture for a UAV autopilot that decouples the tasks of position and attitude estimation from those related to control of the UAV. This allows researchers to focus on either task without the risk of disrupting or damaging the other. The use of two processing units effectively doubles the processing power with minimal impact on power consumption and overall board size. The complexity of synchronizing the two independent processing units is abstracted by the use of a simple yet highly reliable inter-processor communications protocol.

3. The design and implementation of a commands multiplexor (fail-safe) that although laid out on the same Printed Circuit Board (PCB) than the autopilot, functions and draws power completely independent of the autopilot. This allows the safety pilot to retake control of the UAV at any time, regardless of the autopilot or telemetry status.

4. The design and development of a new architecture for HIL simulator that is easily modifiable using Simulink as the main simulation engine. This HIL simulator is able to simulate anything from simple, linear models to highly detailed non-linear models of the UAV's equations of motion, its engine, actuators, the environment, and disturbances. All this can be done using

Simulink and the same ground station software normally employed during flights.

5. A two-step system identification architecture which couples the traditional regression models with the Parameter Space Investigation (PSI). This resulted in a UAV plant model that closely matched the real UAV flight dynamics. This in turn greatly improved the reliability of the results of both software and HIL simulations.

6. The design, development, and feature set of the SLUGS ground station software. This software allows one to configure the controller gains and navigation waypoints in-flight. Once the UAV is placed in autonomous mode, the ground station software can be used to send *mid-level* commands or switch the UAV to waypoint navigation mode. The ground station software produces real-time plots of the relevant variables to facilitate in-flight PID controller gain tuning. It displays and logs all the data produced by the autopilot in an easy to use graphical user interface. It utilizes Google Earth to configure the navigation waypoints and plot the UAV's trajectory in *soft* real-time.

7. The design and development of a sideslip compensator (SSC) as an integral part of the autopilot's inner loop. This has been specifically designed to reduce the adverse effect of a rudder failure in the UAV's capability to track a commanded turn-rate. This SSC proved to be a key component of the autopilot when testing rudder failures, specially when used in conjunction with an $\mathcal{L}_1$ adaptive controller, demonstrating improved transient performance to large scale rudder failures.

8. The implementation and flight tests of an $\mathcal{L}_1$ adaptive controller as an integral part of the autopilot. Simulation and flight test results demonstrate a significant resilience to rudder failures, as well as showing a very high degree of corroboration between the HIL simulation and flight test results. The implementation of the $\mathcal{L}_1$ adaptive controller as an integral part of the autopilot as well as the higher rate state updates contributed to show a much higher rudder failure tolerance than previously reported (see Reference [4]).

The complete SLUGS system, including the hardware design and all the software, has been made available under the MIT open source license to encourage its adoption and improvement by future researchers.

7.3 Future Work

SLUGS has shown successful results in software simulations, HIL simulations, and flight tests, but there is room for improvement in several of its components:

1. Although great effort was taken in tuning the $\mathcal{L}_1$ adaptive controller it still showed a significant overshoot upon rudder failure onset. The newly developed theory of fast and robust adaptation [85] shows that an increase in sampling rate directly translates into a performance improvement [86] . This has some implementation limitations that would need to be addressed. A simplistic approach would be to make the $\mathcal{L}_1$ adaptive controller Simulink implementation a multi-rate model. The $\mathcal{L}_1$ block would execute at a faster rate and the rest of the model would remain at 100 Hz. This of course would mean that the $\mathcal{L}_1$ adaptive controller would still get state updates at 100 Hz, but would perform its internal adaptation and integration and a much higher rate. A simpler approach would be to increase the sampling frequency of the complete system until the DSCs are running at full capacity. This approach would probably only allow an increase in the sampling frequency to approximately 200 Hz. A more bold approach would be to migrate the current algorithms to a fixed-point solution where only integer operations are performed. This would offer the highest gain in sampling frequency but it would require significant changes in the current implementation. Help-

fully, MATLAB and Simulink offer fixed point tools to aid in this process, but it remains a task for future work.

2. The current navigation loop implementation only supports straight-segment navigation in the definition of the path between the waypoints. More complex tasks, such as collision avoidance, target tracking and formation flight for instance, require a more flexible navigation capable of tracking curvilinear paths. The inclusion of a more versatile path-generation, and an extension of the current path-following algorithm would make the autopilot usable in a wider range of possible experiments. Path parametrization (such as via polynomials or Bezier curves) would require large changes to the navigation blocks, but again, SLUGS offers a very robust and simple way to test and simulate these changes before embarking on flight tests.

3. Although the system was designed specifically for UAVs, most of the current architecture could be easily translated into other types of autonomous platforms. Efforts are currently underway at UCSC's ASL to migrate the complete SLUGS platform to an autonomous marine surface vehicle as well as an off-road autonomous ground vehicle. Additional sensor integration (such as for a high-quality Fiber Optic Gyro) will be required to achieve the desired performance.

4. Many UAVs use a traditional ethernet connections to communicate with the UAV payload. A daughterboard that connects the autopilot to an ethernet network would allow the autopilot to interact directly with the payload. Currently many sentences of the communications protocol are updated at only 10 Hz due to bandwidth limitations intrinsic to serial radio communications. Including 802.11 wireless communications would allow for a much higher bandwidth communication channel between the ground station and the UAV at short ranges, and a daughter board that would enable this ubiquitous protocol would be extremely useful.

5. Although the project has been made publicly available with a very generous open source license, there remains an unfortunate lack of detailed documentation. A set of starter documents in the open source project would help to reduce the learning curve to those adopting SLUGS for research.

6. Lastly, while the current version of SLUGS is stable and operational, both the processor and sensor technology are continuously evolving; increasing performance while decreasing in price. Certainly some of the design tradeoffs will need to be revisited as the technological limitations are removed, and small integration issues can always be improved. The SLUGS design is not set in stone, nor should its evolution halt at the current state. Rather, as a research platform, it demonstrates the capability with the current state

of the art, and provides a baseline for continual incremental improvements as time, budget, and opportunity allow.

Appendix A

Equations of Motion and UAV Plant Model

The derivation of the equations of motion for a 6-DOF aircraft model to be used as a plant in the control system design was done in two steps. The first step was formulating the equations of motion for the aircraft, treated as a rigid body. The second step was the computation of aerodynamic, gravitational and propulsive forces that act upon the aircrafts body. The aerodynamic and propulsive forces are specific to the aircraft to be modeled and depend directly on the airfoil, aircraft geometry, mass dispersion, and engine characteristics.

A.1 Equations of Motion

In the following subsections, the development of the equations of motion for the aircrafts linear and angular dynamics as well as the attitude equations are presented

A.1.1 Linear Dynamics Equation

The equations for linear motion are governed by Newtons second law, which states that the net force applied to the center of mass of a body is equal to the product of the mass times the acceleration. However, aircraft velocities, accelerations, linear forces and attitude angles are usually measured with respect to the aircrafts body axis coordinate system since the sensors are strapped down to the aircrafts body (fuselage).

Because of this, a new reference frame is required, a frame that is attached to the aircrafts body center of gravity, hence called the Body frame $\{B\}$. This frame has its X axis defined as coming out of the nose of the airplane; its Y axis as pointing to the right wing; and the Z axis as orthogonal to these two, thus pointing straight down the fuselage.

Let a local coordinate frame $\{L\}$ be a right handed system with an arbitrary chosen origin[1], its X axis pointing north, Y axis pointing east and the Z axis

[1]Typically the ground control station location.

pointing down towards the center of the Earth. All vectors discussed in this Appendix are assumed to be resolved in the $\{L\}$ frame unless noted by capital letter superscript preceding the vector.

It is known from elementary kinematics that velocity is the time rate of change of the position:

$$V = \dot{P}. \tag{A.1}$$

This equation can be expressed in $\{L\}$, simply by premultiplying it by a rotation matrix in order to resolve the body fixed $\{B\}$ velocities into the $\{L\}$ frame:

$$\dot{P} = {}^{B}_{L}R\,{}^{B}V, \tag{A.2}$$

where the rotation matrix ${}^{B}_{L}R$, rotates a vector from $\{L\}$ to $\{B\}$, and is derived in Ref. [53].

The total derivative of vector ${}^{B}V$, which is rotating with angular velocity expressed in the body frame, ${}^{B}\omega$ with respect to $\{L\}$, is given by Coriolis' theorem [60]:

$$\frac{d}{dt_L}\,{}^{B}V = {}^{B}\dot{V} + {}^{B}\omega \times {}^{B}V, \tag{A.3}$$

where $\frac{d}{dt_L}$ denotes the total derivative in $\{L\}$. Therefore, the derivative of vector ${}^{B}V$ with respect to $\{L\}$ has two components; the total derivative in its frame and a component that is not zero when rotating (i.e.: linear acceleration and centripetal

acceleration).

Hence, the UAV's rigid body linear acceleration is given by:

$$\frac{d}{dt_L}\ {}^{B}V_{uav} = \ {}^{B}\dot{V}_{uav} + \ {}^{B}\omega_{uav}\ \times\ {}^{B}V_{uav}. \tag{A.4}$$

Therefore, applying Newton's second law, the forces exerted upon the UAV's center of gravity can be written as:

$${}^{B}F_{uav} = m\frac{d}{dt_L}\ {}^{B}\dot{V}_{uav}, \tag{A.5}$$

which can be rewritten as:

$${}^{B}F_{uav} = m({}^{B}\dot{V}_{uav} + \ {}^{B}\omega\ \times\ {}^{B}V_{uav}). \tag{A.6}$$

A.1.2 Angular Dynamics Equation

The angular equation of motion is derived using Eulers law for conservation of angular momentum in $\{L\}$. Let M_{uav} denote the moment imparted to the UAV's rigid body and $\frac{d}{dt}I_{uav}$ denote the rate of change of angular momentum. Therefore:

$$M_{uav} = \frac{d}{dt}I_{uav}. \tag{A.7}$$

Using Corilis' theorem once more, equation A.7 can be rewritten as:

$$^{B}M_{uav} = {}^{B}\dot{I}_{uav} + {}^{B}\omega_{uav} \times {}^{B}I_{uav}, \tag{A.8}$$

where the angular momentum can be rewritten in terms of the moments and cross products of inertia as:

$$^{B}I_{uav} = \begin{bmatrix} J_{XX} & -J_{XY} & -J_{XZ} \\ -J_{XY} & J_{YY} & -J_{YZ} \\ -J_{XZ} & -J_{YZ} & J_{ZZ} \end{bmatrix} {}^{B}\omega_{uav} = J^{B}\omega_{uav}. \tag{A.9}$$

However, since the UAV is symmetric with respect to its XZ plane, all the YZ and XY cross products of inertia are zero, simplifying the inertia matrix J to:

$$J = \begin{bmatrix} J_{XX} & 0 & -J_{XZ} \\ 0 & J_{YY} & 0 \\ -J_{XZ} & 0 & J_{ZZ} \end{bmatrix}. \tag{A.10}$$

So equation A.8 can be rewritten as:

$$^{B}M = J^{B}\dot{\omega} +^{B} \omega \times (J^{B}\omega), \tag{A.11}$$

where M is the angular momentum and J is the vehicle's inertia matrix.

A.1.3 Attitude Equation

A vector resolved in $\{L\}$ can be expressed in $\{B\}$ simply by premultiplying it by the correct rotation matrix.

$$^{B}V = {}^{B}_{L}RV, \tag{A.12}$$

If, instead of using Coriolis law to obtain the total derivative of the previous equation, the rules of calculus are used, then the vector can be expressed as:

$$\frac{d}{dt_L}(^{B}V) = {}^{B}_{L}R\,^{B}\dot{V}, \tag{A.13}$$

but, if the total derivative is taken in B, then the product rule for differentiation must be applied to obtain:

$$\frac{d}{dt_B}(^{B}V) = {}^{B}_{L}\dot{R}\,^{B}V \; + \; {}^{B}_{L}R\,^{B}\dot{V}, \tag{A.14}$$

which, if it is solved for ${}^{B}_{L}R\,^{B}\dot{V}$ and substituted in Equation A.13, gives:

$$\frac{d}{dt_B}(^{B}V) = {}^{B}\dot{V} - {}^{B}_{L}\dot{R}\,^{B}V. \tag{A.15}$$

Comparing the above equation with Equation A.3 suggests that a relation between the rate of change of the rotation matrix ${}^{B}_{L}R$ and the angular velocity vector ${}^{B}\omega$ exists. To find this relationship, let the vector cross product be expressed as a product of a matrix and a vector as:

$$ {}^{B}\omega \times {}^{B}V = \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix} {}^{B}V = {}^{B}\Omega^{B}V, \tag{A.16} $$

where ${}^{B}\omega = [p\ q\ r]^{\top}$.

By substituting Equation A.12 into Equation A.16 and comparing the second terms of equations A.3 and A.15, the following equality can be established:

$$ -{}^{B}\,\Omega^{B}_{L}RV = {}^{B}_{L}\dot{R}V, \tag{A.17} $$

and therefore:

$$ {}^{B}_{L}\dot{R} = -{}^{B}\Omega^{B}_{L}R \tag{A.18} $$

The above equation is known as the *strapdown* equation. Equating elements $(1,2)$, $(1,3)$, and $(2,3)$ of the above matrix equation (where (i,j) represents the element in the ith row and the jth column) gives a set of three equations that allows one to compute the rate of change of the Euler angles as follows:

$$\begin{aligned}
\dot{\phi} &= p + (q\sin(\phi) + r\cos(\phi))\tan(\theta) \\
\dot{\theta} &= q\cos(\phi) - r\sin(\phi) \\
\dot{\psi} &= \frac{q\sin(\phi) + r\cos\phi}{\cos(\theta)}
\end{aligned} \tag{A.19}$$

In summary, the equations to be implemented in the rigid body 6-DOF model of the UAV are given in Equations A.6, A.11, and A.19.

A.2 Forces and Moments on the Aircraft

The forces and the moments exerted on the aircraft are due to the aerodynamic, propulsion, and gravitational effects on the body. They can be expressed as:

$$\begin{aligned}
{}^{B}F &= {}^{B}F_{aero} + {}^{B}F_{prop} + {}^{B}F_{grav} \\
{}^{B}M &= {}^{B}M_{aero} + {}^{B}M_{prop}.
\end{aligned} \tag{A.20}$$

A.2.1 Aerodynamic Forces and Moments

The aerodynamic forces and moments that act upon the aircraft are produced by the relative motion of the aircraft with respect to the air mass (the airspeed). Mathematically, the aerodynamic forces and moments terms are determined by

using a first-order Taylor series expansion around the aircraft trimmed operating point. Each term in the series is a partial derivative of the forces and moments with respect to the aerodynamic variables. These aerodynamic variables are given by the deflection of the aileron, the rudder and the elevator.

Let the angle of attack α be the angle between the projection of the airspeed vector to the XZ body plane and the X body axis. Also let the sideslip angle β, be the angle between the projection of the airspeed vector to the XY body plane and the X body axis. Since the forces and moments depend on the airspeed, a new coordinate frame must be introduced. In this new coordinate frame, denoted as $\{W\}$, the X axis is aligned with the winds velocity vector (W). The rotation matrix ${}^{B}_{W}R$, which rotates a vector from $\{W\}$ to $\{B\}$, is given in [53], and thus equation (A.20) is rewritten as:

$$\begin{aligned} {}^{B}F &= {}^{B}_{W}R^{W}F_{aero} + {}^{B}F_{prop} + {}^{B}F_{grav} \\ {}^{B}M &= {}^{B}_{W}R^{W}M_{aero} + {}^{B}M_{prop}. \end{aligned} \tag{A.21}$$

For simplicity, the aerodynamic forces and moments acting on the aircraft are defined in terms of dimensionless aerodynamic coefficients that, as expected, depend on the control surfaces deflection, the aerodynamic angles α and β, and the angular rates (p, q, r). The forces and moments expressed in terms of the dimensionless aerodynamic coefficients are derived in [53] and shown in equation

(A.22) for easy reference.

$$\begin{aligned} {}^{W}F &= qS[C_D \;\; C_Y \;\; C_L]^T \\ {}^{W}M &= qS[C_l b \;\; C_m c \;\; C_n b]^T, \end{aligned} \tag{A.22}$$

where q is the dynamic pressure, S is the wing reference area, c is the wing chord, b is the wingspan, and of each of the coefficients is given by:

$$\begin{aligned} C_D &= C_{D0} + A_1 C_L + A_{polar} C_L^2 \\ C_Y &= C_{Y\beta}\beta + C_{Y\delta_r}\delta_r \\ C_L &= C_{L0} + C_{L\alpha}\alpha + C_{L\dot{\alpha}}\dot{\alpha} + C_{L\delta_e}\delta_e + \frac{C_{Lq} q c}{||{}^{B}V_{uav}||}, \end{aligned} \tag{A.23}$$

$$\begin{aligned} C_l &= C_{l\beta}\beta + C_{l\delta_a}\delta_a + C_{l\delta_r}\delta_r + \frac{b}{2||{}^{B}V_{uav}||}(C_{lp}p + C_{lr}r) \\ C_m &= C_{m0} + C_{m\alpha}\alpha + C_{m\delta_e}\delta_e \\ C_n &= C_{n\beta}\beta + C_{n\delta_a}\delta_a + C_{n\delta r}\delta_r + \frac{b}{2||{}^{B}V_{uav}||}(C_{np}p + C_{nr}r), \end{aligned} \tag{A.24}$$

where δ_e, δ_a, δ_r represent the deflection of elevator, ailerons and rudder respectively. p, q, r are the angular rates, and α and β are the angle of attack and sideslip angle respectively.

The numerical values for each of the dimensionless aerodynamic coefficients

required by equation(A.22) were derived as described in Section 3.2.

A.2.2 Gravitational and Propulsive Forces and Moments

Gravitational forces act on the rigid body but generate no moments since the forces are assumed to act at the center of gravity. The gravitational forces are resolved in the local reference frame $\{L\}$ and are given by:

$$^{B}F_{grav} =^{B}_{L} R[0 \quad 0 \quad mg]^{T}, \tag{A.25}$$

where m is the aircraft's mass and g is the gravitational acceleration ($9.81 m/s^2$).

The propulsive forces and moments are exerted in $\{B\}$ and can be stated as:

$$\begin{aligned} ^{B}F_{prop} &= [P_x \quad P_y \quad P_z]^{T} \\ ^{B}M_{prop} &= [P_l \quad P_m \quad P_n]^{T}, \end{aligned} \tag{A.26}$$

where each of the scalars P_i represent forces or moments due to the aircrafts thrust. Since for the aircraft being modeled the engine thrust axis coincides with the X axis of $\{B\}$, then the thrust projections P_y and P_z are assumed to be zero.

Assembling these together yields a full 6-DOF model that is dependent on the 23 aerodynamic coefficients shown in Equations A.23 and A.24. The complete 6-DOF is implemented using Simulink and pictured below in Figure A.1. Note that

wind gust model (from Simulink's Aerospace Blockset) acts merely to alter the aircraft velocity and thus the forces and moments acting on the aircraft through the conventional model. This model is quite general, and can be expanded to accommodate vastly different configurations and/or flight conditions.

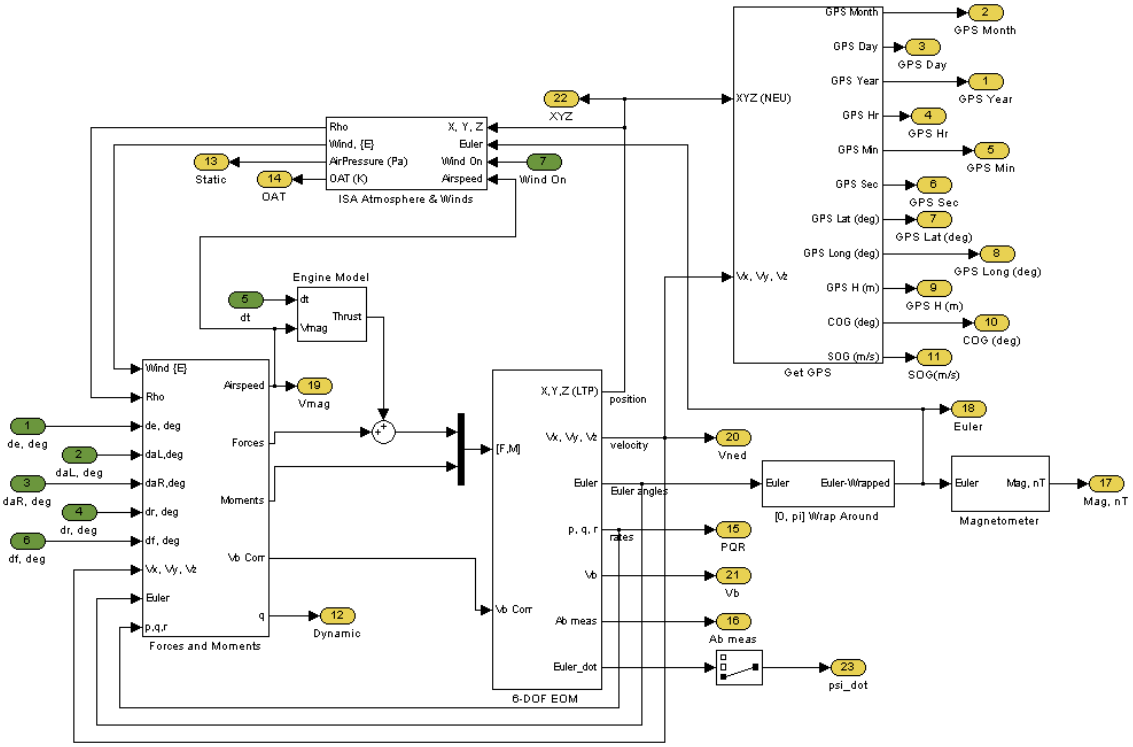


Figure A.1: Full 6-DOF model implementation in Simulink.

Bibliography

[1] Spektrum RC, "DX7 7 Channel Air with AR7000 4-DS821 MD2 servos." http://www.spektrumrc.com/Products/Default.aspx?ProdID=SPM2710, 2009.

[2] R. Statnikov, *Multicriteria Design: Optimization and Identification*. Kluwer Academic Publishers, 1999.

[3] S. Park, J. Deyst, and J. How, "A new nonlinear guidance logic for trajectory tracking," *AIAA Guidance, Navigation and Control Conference and Exhibit*, Jan 2004.

[4] V. Dobrokhodov, I. Kitsios, I. Kaminer, K. Jones, E. Xargay, N. Hovakimyan, C. Cao, M. Lizarraga, and I. Gregory, "Flight validation of metrics driven l1 adaptive control," in *AIAA Guidance, Navigation and Control Conference and Exhibit*, 2008.

[5] V. Dobrokhodov, O. Yakimenko, K. Jones, I. Kaminer, E. Bourakov, I. Kitsios, and M. I. Lizarraga, "New generation of rapid flight test prototyping system for small unmanned air vehicles," *AIAA Modeling and Simulation Technologies Conference Proceedings*, 2007.

[6] "US military's UAV missions increasing," *Armed Forces International*, January 2008.

[7] United Kingdom's Ministry of Defense, "Operations in iraq: Lessons for the future," tech. rep., Ministry of Defense, 2003.

[8] K. Nonami, "Prospect and recent research & development for civil use of autonomous unmanned aircraft as UAV and MAV," vol. 1 of *Journal of System Design and Dynamics*, 2007.

[9] D. Vos, "Fault tolerant automatic control system utilizing analytic redundancy," 2000. US Patent 6,085,127.

[10] B. Bethke, M. Valenti, and J. How, "Cooperative Vision Based Estimation and Tracking Using Multiple UAVs," *Lecture Notes in Control and Information Sciences*, vol. 369, p. 179, 2007.

[11] U. Zengin and A. Dogan, "Real-Time Target Tracking for Autonomous UAVs in Adversarial Environments: A Gradient Search Algorithm," *IEEE Transactions On Robotics*, vol. 23, no. 2, p. 294, 2007.

[12] D. Nelson, D. Barber, T. McLain, and R. Beard, "Vector Field Path Following for Miniature Air Vehicles," *IEEE Transactions on Robotics*, vol. 23, no. 3, p. 519, 2007.

[13] D. Jung, J. Ratti, and P. Tsiotras, "Real-time Implementation and Validation of a New Hierarchical Path Planning Scheme of UAVs via Hardware-in-the-Loop Simulation," *Journal of Intelligent and Robotic Systems*, vol. 54, no. 1-3, pp. 163–181, 2009.

[14] I. Kaminer, O. Yakimenko, A. Pascoal, and R. Ghabcheloo, "Path generation, path following and coordinated control for time critical missions of multiple UAVs," *American Control Conference*, pp. 4906 – 4913, June 2006.

[15] I. Kaminer, O. Yakimenko, V. Dobrokhodov, A. Pascoal, N. Hovakimyan, C. Cao, A. Young, and V. Patel, "Coordinated path following for time-critical missions of multiple UAVs via $\mathcal{L}_1$ adaptive output feedback controllers," *AIAA Guidance, Navigation and Control Conference and Exhibit*, August 2007.

[16] B. Vaglienti, R. Hoag, and M. Niculescu, *Piccolo System User's Guide*. Cloud Cap Technology, Hood River Oregon, 2005.

[17] MicroPilot, "Micropilot: World leader in miniature UAV autopilots." http://www.micropilot.com, 2009.

[18] Procerus Technologies, "Kestrel autopilot." http://www.procerusuav.com/productsKestrelAutopilot.php, 2009.

[19] J. Jang and C. Tomlin, "Autopilot design for the stanford dragonfly UAV: validation through hardware-in-the-loop simulation," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Jan 2001.

[20] J. Jang and C. Tomlin, "Design and implementation of a low cost, hierarchical and modular avionics architecture for the dragonfly UAV," in *AIAA Guidance, Navigation and Control Conference and Exhibit*, 2002.

[21] D. Jung, E. Levy, D. Zhou, R. Fink, and J. Moshe, "Design and development of a low-cost test-bed for undergraduate education in UAVs," *44th IEEE Conference on Decision and Control*, Jan 2005.

[22] R. Klenke, "Development of a Novel, Two-Processor Architecture for a Small UAV Autopilot System." Research Agreement No. W911NF-05-1-0324 Final Report, 2006.

[23] P. Brisset, A. Drouin, M. Gorraz, P. Huard, and J. Tyler, "The Paparazzi Solution," *2nd US-European Competition and Workshop on Micro Air Vehicles*, November 2006.

[24] J. Jang and D. Liccardo, "Automation of small uavs using a low cost mems sensor and embedded computing ...," *2006 IEEE/AIAA 25th Digital Avionics System Conference*, Jan 2006.

[25] H. Chao, Y. Cao, and Y. Chen, "Autopilots for Small Fixed-Wing Unmanned Air Vehicles: A Survey," in *International Conference on Mechatronics and Automation.*, pp. 3144–3149, Jan 2007.

[26] I. Kaminer, A. Pascoal, E. Xargay, C. Cao, N. Hovakimyan, and V. Dobrokhodov, "3D Path Following for Small UAVs using Commercial Autopilots augmented by $\mathcal{L}_1$ Adaptive Control." Submitted to *Journal of Guidance, Control and Dynamics*, 2008.

[27] A. Jensen, D. Morgan, Y. Chen, S. Clemens, and T. Hardy, "Using multiple open-source low-cost unmanned aerial vehicles (UAV) for 3d photogrammetry and distributed wind measurement," in *ASME/IEEE International Conference on Mechatronic and Embedded Systems and Applications*, September 2009.

[28] The Mathworks, Inc., "Land Rover Vehicles Achieve EPA Certification with MathWorks Tools for Embedded Code Generation and add2 Target Hardware." User Story, 2008.

[29] The Mathworks, Inc., "Daimlerchrysler designs cruise controller for mercedes-benz trucks using mathworks tools." User Story, 2004.

[30] M. Schwarz, H. Sheng, A. Sheleh, and J. Boercsoek, "Matlab® / simulink® generated source code for safety related systems," *Computer Systems and Applications, 2008. AICCSA 2008. IEEE/ACS International Conference on*, pp. 1058–1063, 31 2008-April 4 2008.

[31] V. Dobrokhodov and M. Lizarraga, "Developing Serial Communication Interfaces for Rapid Prototyping of Navigation and Control Tasks," *AIAA*, vol. 6099, p. 2005, 2005.

[32] The Mathworks, Inc., "NASA's X-43A Scramjet Achieves Record-Breaking Mach 10 Speed Using MathWorks Tools for Model-Based Design." User Story, 2005.

[33] Microchip Technologies, Chandler, AZ, *dsPIC33F Family Reference Manual*, 2008.

[34] A. S. L. Universty of California Santa Cruz, "SLUGS: Santa Cruz Low Cost UAV GNC System." http://slugsuav.soe.ucsc.edu, 2009.

[35] L. Kerhuel, "Matlab-simulink device driver blockset for pic/dspic microcontrollers."

[36] R. Langley, "NMEA 0183: A GPS receiver interface standard," *GPS world*, vol. 6, no. 7, pp. 54–57, 1995.

[37] D. Paret and C. Fenger, *The I2C bus: from theory to practice*. Wiley, 1997.

[38] M. Gomez, "Hardware-in-the-loop simulation," *Embedded Systems Programming*, vol. 14, December 2001.

[39] N. Kheir and W. Holmes, "On validating simulation models of missile systems," *Simulation*, vol. 30, pp. 117–128, Jan 1978.

[40] M. Sisle and E. Mccarthy, "Hardware-in-the-loop simulation for an active missile," *Simulation*, vol. 39, p. 159, Nov 1982.

[41] S. Werner, A. Buchwieser, and E. D. Dickmanns, "Real-time simulation of visual machine perception for helicopter flight assistance," vol. 2463, pp. 93–101, SPIE, 1995.

[42] K. Norlin, "Flight simulation software at nasa dryden flight research center," *National Aeronautics and Space Administration*, Jan 1995.

[43] D. Maclay, "Simulation gets into the loop," *IEE Review*, vol. 43, pp. 109–112, May 1997.

[44] E. N. Johnson and D. P. Schrage, "The Georgia Tech Unmanned Aerial Research Vehicle: GTMax," *Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit*, August 2003.

[45] E. N. Johnson, D. P. Schrage, J. V. R. Prasad, and G. J. Vachtsevanos, "UAV Flight Test Programs at Georgia Tech," *Proceedings of the AIAA Unmanned Unlimited Technical Conference, Workshop and Exhibit*, September 2004.

[46] J. Tisdale, A. Ryan, M. Zennaro, X. Xiao, and D. Caveney, "The software architecture of the Berkeley UAV platform," *Proceedings of the Conference on Control Applications*, Jan 2006.

[47] The Mathworks, Inc., *Simulink User's Guide*. Natick, MA, 2008.

[48] M. I. Lizarraga, "Autonomous landing system for a UAV," Master's thesis, Naval Postgraduate School, Monterey, CA, USA., March 2004.

[49] V. Klein and E. Morelli, *Aircraft System identification: Theory and Practise*. 2006.

[50] D. Jung and P. Tsiotras, "Modelling and hardware-in-the-loop simulation for a small unmanned aerial vehicle," *AIAA Infotech at Aerospace*, 2007.

[51] Desktop Aeronautics, P.O. Box 20384, Stanford CA 94309, *LinAir User's Guide*, 2005.

[52] E. Morelli and V. Klein, "Application of System Identification to Aircraft at NASA Langley Research Center," *Journal of Aircraft*, vol. 42, no. 1, pp. 12–25, 2005.

[53] B. L. Stevens and F. L. Lewis, *Aircraft Control and Simulation*. 2nd ed., 1992.

[54] R. Statnikov, "Solution of multicriteria machine design problems on the basis of the parameter space investigation," *Multicriteria Decision-Making Problems (in Russian)*, pp. 148–155, 1978.

[55] R. Statnikov and J. Matusov, "Use of p_τ-nets for the approximation of the edgeworth-pareto set in multicriteria ...," Jan 1996.

[56] I. M. Sobol and R. B. Statnikov, "Choosing optimal parameters in problems with many criteria," *Moscow: Science*, 1981.

[57] B. Toth and V. Kreinovich, "Verified methods for computing pareto sets: General algorithmic analysis," *Int. J. Appl. Math. Comput. Sci*, vol. 19, no. 3, pp. 369–380, 2009.

[58] V. Dobrokhodov and R. Statnikov, "Multi-Criteria Identification of a Controllable Descending System," *Computational Intelligence in Multicriteria Decision Making, IEEE Symposium on*, pp. 212–219, 2007.

[59] B. Etkin, *Dynamics of Atmoshperic Flight*. Wiley, unabridged republication. dover 2005. ed., 1972.

[60] B. Etkin and L. R. Reid, *Dynamics of flight*. John Wiley Sons, 1995.

[61] I. Gleim and G. Gleim, *Learn to fly: become a pilot*. Gleim, 2005.

[62] G. Gyatt, *Atmosculator 2.0: Atmosphere Value Calculator*, 2009.

[63] S. Park, J. Deyst, and J. P. How, "Performance and Lyapunov stability of a nonlinear path-following guidance method," *Journal of Guidance, Control and Dynamics*, vol. 30, no. 6, p. 1718, 2007.

[64] J. Stewart, *Calculus: Early Transcendentals*. Brooks/Cole, fourth edition ed., 2001.

[65] J. Wensley, L. Lamport, J. Goldberg, M. Green, K. Levitt, P. Melliar-Smith, R. Shostak, and C. Weinstock, "Sift: Design and analysis of a fault-tolerant computer for aircraft control," *Proceedings of the IEEE*, vol. 66, pp. 1240–1255, Oct. 1978.

[66] R. Patton, "Fault-tolerant control systems: The 1997 situation," *IFAC Symposium on Fault Detection Supervision and Safety for Technical Processes*, vol. 3, pp. 1033–1054, 1997.

[67] C. Rago, R. Prasanth, R. Mehra, and R. Fortenbaugh, "Failure detection and identification and fault tolerant control using the imm-kf with applications to the eagle-eye UAV," *Decision and Control, 1998. Proceedings of the 37th IEEE Conference on*, vol. 4, pp. 4208–4213 vol.4, Dec 1998.

[68] M. Napolitano, Y. An, and B. Seanor, "A fault tolerant flight control system for sensor and actuator failures using neural networks," *Aircraft Design*, vol. 3, no. 2, pp. 103–128, 2000.

[69] H. Alwi and C. Edwards, "Fault tolerant control using sliding modes with on-line control allocation," *Automatica*, no. 44, pp. 1859–1866, 2008.

[70] P. Ioannou and P. Kokotovic, "Robust redesign of adaptive control," *IEEE Transactions on Automatic Control*, vol. 29, no. 3, pp. 202–211, 1984.

[71] K. Narendra and A. Annaswamy, "A new adaptive law for robust adaptation without persistent excitation," *IEEE Transactions on Automatic Control*, vol. 32, no. 2, pp. 134–145, 1987.

[72] S. Naik, P. Kumar, and B. Ydstie, "Robust continuous time adaptive control by parameter projection," in *Decision and Control, 1991., Proceedings of the 30th IEEE Conference on*, pp. 742–747, 1991.

[73] K. Wise, E. Lavretsky, J. Zimmerman, J. Francis-Jr, D. Dixon, and B. Whitehead, "Adaptive flight control of a sensor guided munition," in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2005.

[74] K. Wise, E. Lavretsky, and N. Hovakimyan, "Adaptive control of flight: theory, applications, and open problems," in *American Control Conference*, June 2006.

[75] Rockwell Collins. Athena Family of Autopilots. http://www.athenati.com/.

[76] Rockwell Collins. Athena Autopilots, "Athena's damage tolerance." http://www.youtube.com/watch?v=dGiPNV1TR5kfeature=related.

[77] C. Cao, N. Hovakimyan, I. Kaminer, V. Patel, and V. Dobrokhodov, "Stabilization of cascaded systems via $\mathcal{L}_1$ adaptive controller with application to a UAV path following problem and flight test results," in *IEEE American Control Conference*, pp. 1787–1792, July 2007.

[78] Y. Zhang and J. Jiang, "Bibliographical review on reconfigurable fault-tolerant control systems," *Annual Reviews in Control*, Jan 2008.

[79] E. Xargay, N. Hovakimyan, and C. Cao, "Benchmark problems of adaptive control revisited by $\mathcal{L}_1$ adaptive control," *17th Mediterranean Conference on Control & Automation*, Jun 2009.

[80] V. Dobrokhodov, I. Kitsios, I. Kaminer, K. Jones, E. Xargay, N. Hovakimyan, C. Cao, M. I. Lizarraga, and I. Gregory, "Preliminary results of development, system integration and flight validation of a metrics driven $\mathcal{L}_1$ adaptive control," in *AIAA Infotech@Aerospace conference*, 2009.

[81] I. Kitsios, V. Dobrokhodov, I. Kaminer, K. Jones, E. Xargay, N. Hovakimyan, C. Cao, M. I. Lizarraga, I. Gregory, N. Nguyen, and K. Krishnakumar, "Experimental validation of metrics driven $\mathcal{L}_1$ adaptive control in the presence of general unmodeled dynamics," in *AIAA Guidance, Navigation and Control Conference and Exhibit*, 2009.

[82] J. B. Pomet and L. Praly, "Adaptive Nonlinear Regulation: Estimation from the Lyapunov Equation," vol. 37(6), pp. 729–740, June 1992.

[83] R. Curry, M. I. Lizarraga, and G. H. Elkaim, "The design of rapidly reconfigurable filters for attitude and position determination," *Submitted to the AIAA Infotech at Aerospace 2010 conference*, 2010.

[84] C. Therrien and M. Tummala, *Probability for electrical and computer engineers*. CRC, 2004.

[85] N. Hovakimyan, "The theory of fast and robust adaptation," November 2007.

[86] E. Xargay, N. Hovakimyan, and C. Cao, "Piecewise constant adaptive laws for $\mathcal{L}_1$ adaptive controller in the presence of uncertain nonlinear cross-coupling," *Submitted to the 2010 American Control Conference*, 2010.

[87] C. Coopmans, "Aggienav: A small, well integrated navigation sensor system for small unmanned aerial vehicles," in *ASME/IEEE International Conference on Mechatronic and Embedded Systems and Applications*, September 2009.