The Instrumentation And System Identification
For A Small Agile Helicopter

by

Alex Shterenberg

Submitted to the Department of Electrical Engineering and
Computer Science
in partial fulfillment of the requirements for the degree of
Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2000

© Alex Shterenberg, MM. All rights reserved.

The author hereby grants to MIT permission to reproduce and
distribute publicly paper and electronic copies of this thesis
document in whole or in part.

Author

Department of Electrical Engineering and Computer Science

August, 20 2000

Certified by

Eric Feron
Associate Professor
Thesis Supervisor

Accepted by

Arthur C. Smith
Chairman, Department Committee on Graduate Students
The Instrumentation And System Identification For A Small Agile Helicopter

by

Alex Shterenberg

Submitted to the Department of Electrical Engineering and Computer Science on August, 20 2000, in partial fulfillment of the requirements for the degree of Master of Engineering in Computer Science and Engineering

Abstract

Unmanned model-sized helicopters can perform aggressive maneuvers that can be very useful in challenging environments. An instrumentation package that enables a small radio controlled helicopter to perform aggressive maneuvers autonomously is described. The same package serves as the data logging platform that records the commands of a human pilot during aggressive maneuvers along with the indications of the sensors. The input-output data from several flights are analyzed and the system identification for the rotational dynamics of the helicopter based on a theoretical model is performed. The identified rotational dynamics is used in the hardware-in-the-loop simulator, which is also designed and built as a part of the project. Several aggressive maneuvers are identified and catalogued to aid the design of controllers.

Thesis Supervisor: Eric Feron
Title: Associate Professor
Acknowledgments

I would like to thank Professor Eric Feron for giving me the opportunity to work on such a rewarding, interesting, and challenging project.

The work described here is a team effort. Vlad Gavrilets played a key role in designing the vibration isolation system, the Kalman filter, the simulation, and many other components of this project. The generous help and patience of Emilio Frazzoli are much appreciated.

My special thanks to my mother, Berta Shterenberg, and my fiancee, Marina Piller, for their sincere support and understanding.
Contents

1 Introduction and Problem Statement ................................................. 11
  1.1 Motivation and Objective ...................................................... 11
  1.1.1 Unmanned Aerial Vehicles .................................................. 11
  1.1.2 LIDS Autonomous Helicopter Project ...................................... 13
  1.2 Objectives Overview ............................................................ 14
  1.2.1 Instrumentation Package Design and Implementation ..................... 14
  1.2.2 System Identification .......................................................... 15
  1.2.3 Hardware-in-the-loop Simulation ............................................ 15
  1.3 Thesis Document Road Map ..................................................... 16

2 Overview of the Autonomous Helicopter System .................................. 17
  2.1 Test Vehicle .................................................................. 17
  2.2 Mounting Avionics on the Helicopter .......................................... 18
  2.3 Vibration Isolation ............................................................... 20
  2.4 Avionics Description ............................................................. 21
    2.4.1 Computing and Telemetry ................................................... 23
    2.4.2 Sensors .................................................................. 24
    2.4.3 Actuators .................................................................. 26

3 The Micro-Controller Unit (MCU) ..................................................... 27
  3.1 Functional Description ............................................................ 27
  3.2 Hardware Description ............................................................. 30
3.2.1 Base Board ........................................ 30
3.2.2 PWM Input Filtering and Level Adjusting Hardware ... 33
3.2.3 PWM Output and Servo Power Isolation Hardware ...... 34
3.3 MCU Software ....................................... 36
3.3.1 MCU Parallel Timing .............................. 37
3.3.2 MCU Serial Timing ................................. 38
3.3.3 Interrupt Handlers ................................. 40

4 Sensors and Interfaces .................................. 42
4.1 Inertial Measurement Unit ............................ 42
4.1.1 General Characteristics ............................ 42
4.1.2 Core Sensors ..................................... 43
4.1.3 Filtering and Sampling ............................ 44
4.1.4 Serial Interface ..................................... 45
4.2 Global Positioning System Receiver .................... 46
4.2.1 General Characteristics ............................ 46
4.2.2 Serial Interface .................................. 47
4.3 Barometric Altimeter ................................. 48
4.4 Magnetic Compass .................................... 48

5 Main Computer, Telemetry, and Software .................. 51
5.1 Main Computer Hardware ............................ 51
5.2 Telemetry Hardware ................................. 52
5.3 Software ............................................. 53
5.3.1 Operating System ............................... 53
5.3.2 Peripheral Interfaces ............................ 54
5.3.3 Logflight Software .............................. 59
5.3.4 Ground Display ................................. 64
5.3.5 What Needs to Be Done For Controller Software ..... 66
6 Hardware-in-the-loop Simulation (HILSIM)  
6.1 HILSIM Overview  
6.2 HILSIM Implementation  
6.2.1 Torture Rack  
6.2.2 A/D Board  
6.2.3 Serial Interfaces  
6.2.4 Visualization  
6.3 HILSIM Software Organization  

7 Identification of the Rotational Dynamics of the Helicopter  
7.1 System Identification Method Overview  
7.2 The Experiment  
7.3 Noise Shaping  
7.3.1 Noise in the Data  
7.3.2 The Design of the Filters  
7.3.3 The Results of Noise Shaping  
7.4 The Model  
7.4.1 Flapping  
7.5 Measuring the Parameters  
7.5.1 Swing Tests  
7.6 System Identification  
7.6.1 The Effect of Flybars  
7.6.2 System Identification Approach  
7.6.3 Results of Applying the Parametric Model  
7.6.4 Data Set # 1
7.6.5 Model Validation ........................................... 96
7.6.6 Non-parametric Model .................................. 100
7.6.7 A Hypothesis About The Effect of Forward Speed .. 103

8 Aggressive Maneuvers ........................................ 104
  8.1 Overview .................................................... 104
  8.2 The Knife-Edge Maneuver ................................ 105
  8.3 The Loop Maneuver ........................................ 105
  8.4 The Aggressive Turn Maneuver ......................... 105
  8.5 The Saturation of the Yaw Gyro ........................ 107
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>Test Vehicle</td>
<td>18</td>
</tr>
<tr>
<td>2-2</td>
<td>Mounting Diagram</td>
<td>19</td>
</tr>
<tr>
<td>2-3</td>
<td>Block Diagram of Electronic Components and Interconnections</td>
<td>22</td>
</tr>
<tr>
<td>3-1</td>
<td>MCU Block Diagram</td>
<td>28</td>
</tr>
<tr>
<td>3-2</td>
<td>MCU State Diagram</td>
<td>29</td>
</tr>
<tr>
<td>3-3</td>
<td>MCU PWM Functions Diagram</td>
<td>32</td>
</tr>
<tr>
<td>3-4</td>
<td>MCU PWM Inputs Diagram</td>
<td>33</td>
</tr>
<tr>
<td>3-5</td>
<td>MCU PWM Outputs Diagram</td>
<td>35</td>
</tr>
<tr>
<td>3-6</td>
<td>MCU Timing Diagram</td>
<td>39</td>
</tr>
<tr>
<td>4-1</td>
<td>GPS Protoboard Diagram</td>
<td>47</td>
</tr>
<tr>
<td>4-2</td>
<td>Altimeter Serial Connection</td>
<td>49</td>
</tr>
<tr>
<td>5-1</td>
<td>IMU State Diagram</td>
<td>56</td>
</tr>
<tr>
<td>5-2</td>
<td>Flight Logging Software</td>
<td>60</td>
</tr>
<tr>
<td>5-3</td>
<td>Ground Station Graphical User Interface</td>
<td>61</td>
</tr>
<tr>
<td>5-4</td>
<td>Ground Display Software Diagram</td>
<td>65</td>
</tr>
<tr>
<td>5-5</td>
<td>Timing Diagram</td>
<td>68</td>
</tr>
<tr>
<td>5-6</td>
<td>Software Organization Chart</td>
<td>69</td>
</tr>
<tr>
<td>6-1</td>
<td>Interconnection Diagram of the HILSIM</td>
<td>73</td>
</tr>
<tr>
<td>6-2</td>
<td>Block Diagram of ADDA1110's A/D</td>
<td>75</td>
</tr>
</tbody>
</table>
List of Tables

7.1 Measured and Estimated Physical Parameters ........................................... 93
7.2 Fitness Criteria for the identified physical model (#1), the model
    with $S_\beta = 0$ (#2), and the 4th order non-parametric model (#3) ........ 99
Chapter 1

Introduction and Problem Statement

1.1 Motivation and Objective

1.1.1 Unmanned Aerial Vehicles

Recent advances in computers, battery technology, and sensors have stirred interest in small autonomous unmanned air vehicles (UAV's). Such UAV's have a variety of military and civilian uses. In the military, they can serve as surveillance platforms, landmine detectors, even fighting machines. Civilian applications include aerial photography, crop dusting, disaster area management, remote area exploration, rescue operations, and others.

Many of these applications require vertical take-off and landing capability, which conventional fixed wing airplanes lack. Among the aircraft capable of such operations, the helicopter stands out as the most efficient in terms of power, noise, and versatility.

Unfortunately, these advantages come with a price. In comparison to airplanes, helicopters are more complex mechanically. They are also inherently unstable, making the design of a controller more difficult. In addition, high vi-
bration levels interfere with the operation of sensors and imaging equipment.

Despite these difficulties, many commercial and academic institutions have taken on the task of designing the autonomous helicopters for a variety of applications.

The "Hummingbird" project at Stanford’s Aerospace Robotics Laboratory has been successful in producing an autonomous helicopter capable of navigation using no sensors but 4 GPS antennae and receivers.

Carnegie Melon University’s Aerial Robotics Group has achieved considerable success in autonomous flight, obstacle avoidance, and target recognition.

These and other universities, including MIT, participate in the International Aerial Robotics Competition. The objective of the competition is to perform a rescue operation in a simulated disaster environment.

Draper Laboratories, Boeing, and other aerospace companies are working on military contracts involving small autonomous helicopters. Among their disclosed achievements are slow autonomous waypoint navigation and automatic take-off and landing.

While the achievements of other academic and commercial institutions are considerable, none of them with the exception of Draper Laboratories, have tried to exploit one advantage of unmanned air vehicles – their expanded flight envelope.

Small unmanned air vehicles are capable of executing the maneuvers manned machines are not capable of. Inverted flight, high-g loops and rolls, pirouettes are some of the examples. These additional capabilities may prove invaluable in situations where agility and quickness are critical. In a battlefield environment, disaster area, or on a motion picture set, the ability to perform aggressive maneuvers autonomously can make the helicopter much more effective.
1.1.2 LIDS Autonomous Helicopter Project

MIT Laboratory for Information and Decision Systems is undertaking a project of designing a small unmanned helicopter capable of utilizing the full flight envelope available to small helicopters.

The project consists of several major tasks. At the highest level of abstraction, students are working on creating efficient algorithms for path planning and obstacle avoidance using the full library of aggressive maneuvers. Other students work on low-level control, state estimation, simulation, and system identification of the helicopter platform. At the implementation level, a physical platform is being built for current and future experiments.

The objective of the presented work is to design and implement an instrumentation package to enable a small radio controlled helicopter to perform aggressive maneuvers autonomously. The same package should serve as the data logging platform that records the commands of a human pilot during aggressive maneuvers along with the indications of the sensors. The input-output data from several flights are analyzed and the system identification for the rotational dynamics of the helicopter based on a theoretical model is performed. The identified rotational dynamics is used in the hardware-in-the-loop simulator, which is also designed and built as a part of the project. From the recordings of several aggressive flights, the state trajectories of the aggressive maneuvers are identified and catalogued to be used for designing the autonomous controller.
1.2 Objectives Overview

1.2.1 Instrumentation Package Design and Implementation

Small helicopters offer an extremely challenging environment for the development of an instrumentation package. An instrumentation package good enough for an aggressive flight is an even more challenging task. Several issues are addressed: sensor combination, actuation, computing power, operating system and software, power, vibration, mechanical design, telemetry, ground station, and safety.

Sensors are selected to provide full state information about the helicopter. Their accuracies are carefully chosen to enable the design of adequate control. Enough redundancy is implemented to avoid crashes in case of outages of less reliable sensors such as GPS.

The actuation system is designed in such a way that the delay from the computer command to the motion of the surfaces is minimal. This is done to maximize the phase margin available to the controller. At the same time, the actuation has to be independent from the computer to allow the pilot to salvage the helicopter in the case of the main computer failing.

The main computer onboard is powerful enough to be able to sample the sensors, run the Kalman filter, the controller, initiate actuation, and have CPU time left for the tasks of high-level path planning and obstacle avoidance. The computer also has enough I/O for all of the sensors and communication hardware.

Software design and implementation for the onboard computer comprise a significant portion of the instrumentation package. The operating system has to be well suited for controlling fast dynamic systems. The delays caused by context switching and interrupt latencies have to be minimal.

The project spans several engineering domains, from software to hardware to mechanical design. Mechanical design issues include dealing with vibration, and
designing and manufacturing the supporting structure for the avionics.

The requirement that the system perform aggressive maneuvers creates additional constrains on the design. The dynamics of the helicopter during aggressive flight is significantly faster. Faster dynamics requires higher bandwidth control, which in turn requires faster sensors and actuators. Higher bandwidth control requires that the vibration isolation subsystem works well at higher frequencies while the frequencies of the vibration loads stay the same. Safety requirements also increase as the helicopter performs higher energy maneuvers.

1.2.2 System Identification

Identifying the dynamics of a helicopter is an extremely complex task. The helicopter system is unstable and therefore requires constant corrections from the pilot. Consequently the data available for the system identification already contains feedback from the pilot.

This document describes the identification of the rotational dynamics of the helicopter. The flapping response of the main rotor to pitch and roll commands near hover is studied, and the prediction of a theoretical model is compared with the data obtained during the flights. The flights are designed to excite all of the important responses as much as possible given the safety restrictions.

An analysis of the importance of such factors as the presence of flybars is given. An attempt is made to justify and correct the discrepancies between the theoretical model and the flight data.

1.2.3 Hardware-in-the-loop Simulation

The hardware-in-the-loop system serves the purpose of verifying the operation of the avionics package by testing the package in an environment simulated by another computer in the laboratory. The hardware-in-the-loop simulator consists of:
- the hardware and software that interface with the onboard computer and acquire the outputs of the onboard computer
- a model of the helicopter dynamics and the environment surrounding it.
- the implementation of the model in software

Only the hardware and software interfaces, and a part of the helicopter model are included in the presented work and are described here. The design of the interfaces for the hardware-in-the-loop simulator is a task similar to the design of the onboard interfaces.

1.3 Thesis Document Road Map

This paper presents the description of the hardware and software design for the autonomous helicopter platform. In addition to giving a high-level description of the design choices and trade-offs, this document also goes into the details of hardware and software implementations. It should serve as a manual to those whose task will be to maintain and expand the system.

Chapter 2 gives an overview of the system with an emphasis on the justification of the design choices in view of the functional requirements.

Chapters 3 through 6 describe the hardware and software parts in greater detail. Chapter 3 focuses on the Micro-Controller Unit (MCU), which contains the actuator and the pilot command interfaces. Chapter 4 goes into the details of the sensors and their interfaces. Chapter 5 inspects the CPU and the software running on it. Chapter 6 describes the hardware and software of HILSim, the hardware-in-the-loop simulator.

Chapter 7 contains the description of the system identification methods and the results of applying them. Chapter 8 identifies and catalogues the aggressive maneuvers.
Chapter 2

Overview of the Autonomous Helicopter System

2.1 Test Vehicle

The test vehicle (see Figure 2-1) is based on an XCell-60 helicopter manufactured by "Miniature Aircraft USA." The size of the engine is 60 cm$^3$. The rotor spans 60.6 inches. The helicopter is equipped with fly-bars and a 2-blade teetering carbon-fiber rotor. The frame is made of carbon fiber. The weight of the helicopter is about 8 lb.

This R/C aircraft was chosen because of its reputation as a reliable and stable aerobatics platform with a record of winning many aerobatics contests.

Flights with dumb weight were performed to determine the maximum payload the XCell-60 can handle while performing aggressive maneuvers. The helicopter performs reasonably well carrying up to 7.5 lb [9].

The platform is equipped with an electronic governor that maintains commanded rotor speeds by adjusting throttle commands. A hobby gyro provides proportional negative feedback of the measured yaw rate to tail rotor pitch, thereby augmenting yaw rate damping.
2.2 Mounting Avionics on the Helicopter

All of the external electrical components with the exception of the GPS antenna and the compass are mounted in a single aluminum enclosure suspended below the landing gear of the helicopter. Such arrangement is dictated by the requirements of the vibration isolation subsystem. Vibrations are easier to isolate for larger masses and higher moments of inertia. If the electronics were mounted outside of the box, additional weight would have to be added to the box to maintain adequate isolation.

In order to increase the moments of inertia even more, the components are mounted along the walls of the enclosure as shown in Figure 2-2. The landing gear is extended. Specially manufactured mounts accommodate the avionics enclosure.

Figure 2-2 shows that the box is suspended on four elastic isolators that are placed in the carefully measured locations inside the outer walls of the box [1].
List of Parts:
1. Proxim Wireless LAN Adapter
2. Power Regulator
3. DSP Design TP300 PC104 CPU
4. CMC Superstar GPS Receiver
5. R/C Receiver & Battery
6. Honeywell HPB200A Altimeter
7. Inertial Sciences ISIS IMU
8. 68HC12 Processor Board
9. Main Batteries
10. Servo Battery
11. Aluminum Enclosure
12. Mounting Bracket
13. ME500-1 Neoprene Isolators

Figure 2-2: Mounting Diagram
The elastic material on one side of the isolators is attached to a pair of metal bars that are mounted on the landing gear of the helicopter. The metal bases on the other side are fastened to an aluminum bracket which attaches to the body of the enclosure.

2.3 Vibration Isolation

Such an arrangement allows for effective isolation of unwanted vibrations. The vibration environment on a small helicopter is rather complex. The main source of large-amplitude high-frequency vibration inputs is the main rotor, spinning roughly at 26-27 Hz. The harmonics are 1 per revolution, 2 per revolution (blade passage frequency near tail rotor), engine frequency (around 230 Hz), and tail rotor frequency (around 115 Hz). These inputs also have sidebands, which excite lateral and vertical first bending modes of the tailboom (20 Hz). The combined amplitude of the vibrations is on the order of 40 deg/sec and 1 g in all axes.

A passive vibration isolation system is designed for the avionics box weighing 7 lbs [2]. The resulting rotational cutoff frequencies are in the range of 7-9 Hz, the translational frequencies are around 11 Hz in horizontal plane, and 13 Hz in vertical direction. The cutoff frequencies have to be low enough to provide sufficient attenuation of the high-frequency vibration sources. At the same time the suspension system should be sufficiently stiff to sustain high-g maneuvers without bottoming out.

The low rotational cut-off frequencies are achieved by spacing the isolators closely inside the avionics box (see Figure 2-2). The isolators are located in the corners of a rectangle, the geometric center of which coincides with the center of gravity of the isolated assembly. Thus the rotational and translational modes are decoupled[4]. Neoprene isolators (ME500-1 from Barry Controls) were chosen primarily for their reliability and low cost ($10 a piece). Neoprene has a damping ratio of 0.05, which results in fast decay of the transmissibility function, and also
has 10:1 amplification factor at resonance. The suspended assembly’s resonances get excited by broadband vibration inputs. Since these resonant frequencies are significantly faster than helicopter rigid-body dynamics, digital notch filters are also used to remove them.

The vibration isolation system cuts the high-frequency vibration inputs down to negligible levels. In addition to the high frequencies, two low-frequency modes prevail. These modes are very likely associated with the flapping dynamics of the Bell-Hiller stabilizers. These frequencies for the Xcell-60 with the avionics payload are 2.7 and 3.1 Hz in pitch and roll, respectively. The slightly higher roll frequency is due to lower roll inertia. The amplitudes of the low-frequency vibrations are up to 6 deg/sec.

Digital notch filters are used to attenuate the low-frequency modes. The filters contribute 8 deg phase penalty at the design crossover frequency of 2 rad/sec in pitch and roll axis.

Lastly, the high hobby gyro gain leads to increased vibration levels. There are two main reasons for this. The airframe is flexible, which couples into gyro measurement leading to resonance. The tail rotor servo pushrod is also flexible, acting like a second order system, and decreasing phase margin in the yaw rate control loop. Stiff airframe and pushrod allow for a much higher yaw gyro gain, which leads to better handling qualities in manual flight.

2.4 Avionics Description

Safety, high bandwidth requirements for the flight control system, and adequate sensor information were the key factors in designing the avionics package. A block diagram of the electronic components and their interconnections is given in Figure 2-3.
Figure 2-3: Block Diagram of Electronic Components and Interconnections
2.4.1 Computing and Telemetry

Most of the processing is done in the Central Processing Unit (CPU). The CPU is a 266 MHz PC104 board with 32 MB of RAM and 16 MB of permanent flash RAM. The input and output channels of the CPU include 4 serial ports, 4 A/D channels, ethernet and parallel port, most of which are used by the sensors and actuators. The CPU runs the real time operating system QNX[4].

In order to facilitate helicopter recovery in case of the CPU or main battery failure, a separate microprocessor powered from a different power source handles the task of driving the servo actuators with pulsewidth modulation (PWM) waveforms. This device is labeled Micro Controller Unit (MCU) in the block diagram. During the normal automated control operation, the MCU receives serial commands from the CPU and converts them to five PWM signals. In the datalogging mode, the MCU digitizes R/C pilot's commands coming from the receiver and passes them through to the servos while simultaneously sending the commands to the CPU via the serial link. In the event of the main computer failure, the pilot has to flip a switch on the transmitter, which directs the MCU to pass the PWM signals from the receiver directly to the servos.

The MCU is implemented on a Motorola 68HC12 prototyping board. In addition to its regular components, the MCU also has RF filtering electronics for the receiver commands and optical isolation circuitry to isolate the servo bus from the rest of the electronics.

The CPU communicates with the ground station via a wireless LAN telemetry system. While wireless LAN offers the benefit of high throughput, its poor reliability and timing prompted the decision to leave the telemetry system out of the safety critical loops. If the helicopter's antenna is blocked, the device may drop packets or delay them until the helicopter comes back into the view. The speed of the transmission also drops as the distance to the helicopter increases. Wireless LAN link is used only for monitoring and data logging. Having an ethernet
link is also an extremely valuable debugging tool. Proxim RangeLAN2 ethernet adapter is chosen for the task. The bandwidth is as high as 1.6 Mbits/s for large packets. Since it connects to an existing ethernet controller, the Proxim does not require writing any QNX drivers.

2.4.2 Sensors

The Inertial Measurement Unit (IMU) provides the bulk of sensor information to the CPU. The unit contains three gyroscopes and three accelerometers. The outputs of the sensors are temperature compensated internally. The IMU is chosen for its accuracy over extended input ranges and light weight. The unit weighs 250 grams. After 10 minute preflight warm-up, the biases of the gyro drift on average by .03 degrees/sec and the biases of the accelerometers drift by 5 mg during 15 minute flight. The full scale of the gyros is chosen at +/- 300 deg/sec, which is enough for most high angular rate maneuvers around any axis. The IMU provides 100 Hz updates of the rates and specific forces through the 115.2 Kbit/sec serial port. The device is manufactured by Inertial Sciences Inc.

The Superstar GPS receiver from Canadian Marconi provides 1 Hz updates of the inertial position and velocity. Since the practice of scrambling the GPS signal for civilian users, known as Selective Availability, was discontinued on May 1, 2000, short term relative navigation with the GPS provides horizontal accuracy on the order of several meters. The duration of flight is limited to 15 minutes by the amount of fuel the helicopter carries. The main source of GPS errors in the absence of selective availability is ionospheric delay. Since the path traveled by the signal from a satellite to the helicopter does not change significantly in 15 minutes, the errors are small. Blending the GPS and IMU measurements, as well as other sensors in the Extended Kalman Filter (EKF) further improves navigation accuracy. Differential GPS is not required for a fully autonomous 15 minute flight.
A tri-axial magnetoresistive sensor HMC2003 from Honeywell is used to measure the three components of Earth magnetic field in projection to the body axis. It is mounted on the horizontal fin, far away from the engine and the avionics box. The compass requires scale factor and bias calibration to compensate for the magnetic fields induced by the helicopter-mounted components. In addition the EKF estimates of roll and pitch angles are used to resolve the magnetic field in local North-East plane. The resolution of the magnetic field measurement on a 12 bit A/D converter is 1 milliGauss, which approximately corresponds to 0.3 degrees in a 200 milliGauss Earth magnetic field. Although the EKF heading updates are made only once a second, the analog readings are sampled at 20 Hz, and passed through a first order analog filter to avoid aliasing.

A HPB200A barometric altimeter from Honeywell is used to complement the altitude information from GPS. This is an absolute pressure sensor with the range of 0-17 psi. Sensor resolution is 0.001 psi, which is roughly 2 ft. The altimeter is sampled 10 times a second. The pressure change due to the induced velocity from the rotor turns out to be small. According to the momentum theory, the dynamic pressure change due to the induced velocity is equal to \( T/4/A \), where \( T \) is the thrust, and \( A \) is the rotor disc area. For the helicopter with 18 lbs gross weight and 5 ft rotor diameter this value is 0.0016 psi, which results in roughly 3 ft altitude error. A wind of 10 knots would have the same effect. The effects of gusts and thrust variations due to collective changes are on the same order, and these higher-frequency errors are low-pass filtered by the EKF.

During take-off and landing there is high pressure region between the helicopter and the ground (ground effect) resulting in negative altitude indications from the altimeter. Thus, compensation is necessary for the automatic take-off and landing logic.
2.4.3 Actuators

High bandwidth servos, especially for the tail rotor pitch, are essential to achieve closed loop bandwidth adequate for aggressive maneuvers. Fast hobby servos were chosen for all channels except throttle, where the response is dominated by the slow time constant of the engine. JRDS8417 servos were used. Their slew rate is 600 degrees/second, which translates to roughly 7 Hz bandwidth (frequency at which 90 degrees phase lag occurs).

Servo Interface

The servos are driven by pulse width modulated waveforms (PWM) whose duty cycle range from approximately 1 ms to approximately 2 ms. One and two ms pulses correspond to angular position limits of the servos.
Chapter 3

The Micro-Controller Unit (MCU)

3.1 Functional Description

The MCU is a microprocessor board that carries out three functions:

- Convert the command that the main CPU uses to drive the servos from a serial message to 5 pulse-width-modulated (PWM) digital waveforms.

- Convert the PWM commands given by the pilot via the transmitter and the receiver to a serial message and send the message to the main computer.

- Act as an emergency backup device that allows the pilot to switch from computer control to manual control where the commands are passed directly to the servos without involving the main computer.

The MCU is equipped with the interfaces shown in Figure 3-1. The unit is connected to the main computer via a bi-directional serial link operating at 38400 Baud. On the input side, the pilot commands are converted to 7 PWM waveforms by a standard R/C receiver and fed to the MCU’s 7 digital I/O lines. At the output, the MCU uses its other 5 digital lines to drive the helicopter’s
servos with PWM commands. Two lines out of five have other standard R/C devices inserted in the path between the MCU and the servos. A R/C governor is positioned between the MCU and the throttle servo. It maintains commanded speed setting of the engine. A yaw rate gyro is inserted between the MCU and the tail rotor servo to stabilize the yaw rate.

The MCU must operate in three modes: emergency pilot control, normal automatic control, and normal pilot control with logging.

During the emergency mode, the main computer is not required for operation. Pilot’s commands are passed directly to the servos without measuring their pulse widths. Once this mode is entered, the helicopter must be landed and the MCU must be reset manually.

In both normal modes, the MCU is programmed to drive the servos only after a serial command from the computer is received. The MCU processes the command and outputs corresponding PWM waveforms to the servos. At the same time, the MCU keeps logging pilot inputs and sending them to the computer.
The normal manual control mode facilitates logging pilot commands for system identification purposes. It is also used for take-off and landing. A switch on the transmitter is used to command the transitions between the two normal modes. Special logic has to be implemented in order to avoid sudden servo position changes when the modes are switched.

Figure 3-2 contains the state diagram of the MCU modes. The MCU starts in the normal pilot control mode for take-off. In the normal logging mode, the MCU measures the widths of the pulses from the receiver and reports to the main computer via the serial link. The main computer logs the pilot commands and returns them immediately back to the MCU via the serial link. The MCU then creates PWM waveforms to drive the servos.

If the mission is to log the pilot inputs and sensor outputs to identify the system, the MCU stays in the normal pilot control mode until the end of the flight. Otherwise, if the mission is automatic control, the pilot switches the MCU into the automatic control mode after take-off by toggling the switch labeled "Gear Invert/Mix" toward him/herself. This switch corresponds to channel #5 on the receiver. The MCU is not aware of the change in modes. After the control mode is entered, the board keeps measuring pilot inputs and supplying them to the main computer. This time, however, instead of passing the pilot inputs back...
to the MCU for driving the servos, the main computer disregards all messages from the MCU. The commands created by the controller code are supplied to the MCU instead.

Another possible regime is control augmentation. In that regime, the main computer processes the commands of the pilot and controls the MCU with the results.

Therefore, only two of the three modes have to be implemented in the MCU. The main computer does the switching between the other two modes by using the pulse width on channel #5, the auto/manual switch, passed to it by the MCU.

3.2 Hardware Description

3.2.1 Base Board

The MCU is implemented on a Motorola HC12 evaluation prototyping board, manufacturer part number M68EV8912832. The board consists of a 5.15 by 3.4 inch double-sided printed circuit board. Its features include: an HC12 16-bit processor, 32 Kbytes of flash EEPROM, 1 Kbyte of single-cycle-access RAM, 8 channel timer, 16 bit pulse accumulator, up to 63 digital I/O lines, RS-232C interface with a DB9 connector, remote debugging capability, 8 MHz bus operation, prototyping area, reset button.

All of the aforementioned features are exploited in the MCU design. The timer channels in conjunction with the pulse accumulator are used to measure seven and produce five pulse width waveforms. The digital I/O lines are used to output the PWM waveforms. The serial interface is used to communicate with the main computer. The prototyping area is filled with filtering and isolation circuitry required to drive the servos and to measure pulses accurately.

The HC12 comes with a program called D-Bug12 which allows the users to download small programs to the HC12 from the host computer via a serial con-
The program limits the amount of EEPROM space available to the users. Additional hardware and software from PE-Micro, Inc. allows the users to erase D-Bug12, load it with programs up to 30 Kbytes, and use the board’s convenient remote debugging features. These tools consist of a parallel port to HC12 BDM connector cable, an assembler, a programmer, and a debugger.

**Hardware For PWM Inputs and Outputs**

The timer module of the HC12 is connected to 8 digital I/O channels and 8 16-bit registers that can be configured as either input capture or output compare. In the input capture mode, the registers can be used to latch the value of the free running counter when a transition occurs on the corresponding digital line. In the output compare mode, the HC12 can be configured to trigger an interrupt when the timer reaches the value contained in the register.

The MCU makes use of 7 inputs capture lines to measure the width of the PWM waveforms sent to it by the receiver. The remaining channel is configured as output compare and used for generating programmable timed interrupts that output measured PWM waveforms on the 5 output channels. Figure 3-3 summarizes MCU’s PWM functions.

The clock of the HC12 runs at 8 MHz. It is divided by 2 by the Prescaler/Divider. The resulting clock speed of 4 MHz can be used for timer functions. Therefore, the resolution of the MCU pulse width measuring system is 1/4 MHz = 250 ns. With its 16-bit counter, the system is capable of measuring the maximum pulse length of 250 ns * 2^16 = 16.38 ms, which is more than sufficient since the pulse width of a servo command has to be between 1 ms and 2 ms.

**Serial Function of the MCU**

The MCU uses the built in serial port of the HC12 to communicate with the main computer. The serial communication is carried out at 38400 baud with
Figure 3-3: MCU PWM Functions Diagram
8-bit words, no parity, and 1 stop bit. The hardware raises interrupts whenever a byte is received on the serial line and whenever the transmission path is open for another byte. These two interrupts are used to enable the MCU-main computer communication. More details about the communication protocol are given in the MCU software section later on in this chapter.

3.2.2 PWM Input Filtering and Level Adjusting Hardware

Figure 3-4 contains a circuit diagram of the hardware used for processing the PWM inputs. Digital inputs to the MCU have to be filtered to attenuate high frequency noise. Passive Electro-Magnetic Interference filters packaged into a D-connector are used for that purpose. The filters are Pi-type with a 3 dB cut-off frequency set at 0.8 MHz.

The receiver produces digital outputs with the high voltage level of 3 Volts and low voltage level of 0 volts, while the MCU requires +5 on its digital inputs.
In order to condition the input signals and at the same time clean up the transitions, the MCU uses comparator chips, LM339s. The LM339s require a reference voltage, Vref, which connected to all of the input pins marked with the minus sign on the diagram. Vref is obtained by dividing the +5 Volt battery supply using a voltage divider circuit. If the voltage on the input pins marked with pluses is greater than the voltage on the corresponding minus pins, the outputs of the corresponding channel is +5 V, otherwise, it is 0 V.

Batteries provide poor voltage reference. A possible improvement for the MCU would be a power regulator that regulates the output of the R/C battery to 5 Volts.

### 3.2.3 PWM Output and Servo Power Isolation Hardware

It is essential for the HC12 to be completely electrically isolated from the servo motors since the motors produce high current spikes on both their power lines and their signal lines. In the MCU, the electrical isolation is achieved by means of the optical isolator chips, 74OL6000, produced by QT Optoelectronics. A circuit diagram of the PWM output and power isolation is given in Figure 3-5.

Inside the chips, LEDs light up when the input signal level is high and the photoelements on the output side sense the transition of light. Thus the inputs and the outputs are completely electrically isolated.

The signal driving the servos has to be at least .5 Volts different from the power source. To accomplish this, a diode is inserted between the servo battery and the positive voltage source of the optoisolators. In addition, the noise is reduced by bridging power and ground on both sides of the optoisolators with .1 \( \mu F \) capacitors.
Figure 3-5: MCU PWM Outputs Diagram
3.3 MCU Software

The MCU runs a program that is written in the Assembly language on the host computer, assembled on the host computer and downloaded to the MCU through the parallel port. The program is entirely interrupt driven everywhere except for the emergency mode. In the emergency mode, all sources of interrupts are turned on and the current values of the input ports are continuously fed to the output ports. In the normal mode of operation, the main loop does not contain any operations, which are instead performed in interrupt handlers.

The MCU has to deliver computer commands to the servos as fast as possible. Since the MCU is in the control loop, any delays incurred while executing main computer commands to the servos decrease the phase margin of the feedback system and make the controller less robust and harder to design.

The biggest sources of delays are the pulses themselves. The pulses range from 1 ms to 2 ms. When the servos are driven in series, the total delay until the last servo is driven can be as large as 10 ms. At 25 Hz, the frequency of the helicopter control system, such a delay cuts out a big chunk from the time allocated to other processes in the control loop. The receiver also drives the servos serially. This is fine for a human pilot whose bandwidth is not larger than 5 Hz, but not acceptable for a controller with a cross-over frequency of 2 rad/second.

The servos can also be driven in parallel. Pulses are started on all 5 channels simultaneously and are terminated according to their lengths. It is important that the microcontroller has enough time to change the outputs and setup the output capture channel for the next channel before the time the next channel has to go off. Even for the smallest difference in pulse length, the MCU has to work fast enough to get out of the output capture interrupt service routine on time.

With 8-bit resolution of the computer commands the smallest difference between two channels is 4 μs. The MCU’s 8 MHz clock allows only 32 instructions during that time, which is not enough for a valid interrupt service routine. Tricks
have to be implemented to approximate small delays. Thus, accuracy is reduced and the servos "jerk" by 1-2 counts. In addition the complexity of the program increase dramatically since the pulse widths have to be sorted. With parallel driving, the total latency between the time the main computer issues a serial command and the time the pulse is delivered to the servos is 3.6 ms.

Another major source of delays is the serial transmission. The maximum speed of the MCU’s serial communications is 38400 Baud. At this speed, each byte takes 260 $\mu$s. In order to minimize the time of the serial transmission, the resolution of the control commands is limited to 8 bits, decreasing the length of the main computer’s commands to 8 bytes (1 header + 6 channels * 1 byte + channel mask). Therefore, the time of the transmission of a complete command is $260\mu s \times 8$ bytes = 2 ms. It is important to note that the resolution of the servos is also 8 bits. The servos measure pulses using a clock with a 4 $\mu$s period. With carefully set boundaries, it is possible to achieve no loss in accuracy. In reality, however, the boundaries for each servo are a little bit different. Since the range of the commands has to be big enough to cover all servos, some accuracy is lost. In general, feedback compensates for the lost resolution of commands.

Both serial and parallel approaches to driving the servos are implemented in the MCU. If the phase lag is large enough to make the 25 Hz controller impossible, the parallel program can be used at the expense of accuracy. Feedback will compensate for the small "jerks." Otherwise, it is better to use the serial approach because of its accuracy and simplicity.

### 3.3.1 MCU Parallel Timing

An MCU cycle starts with the arrival of the first byte of the new control command on the serial line (Figure 3-6). The MCU starts a state machine to receive and verify the control command. Each subsequent byte invokes the same interrupt and serves as an input to the state machine. After all 8 bytes have been received,
the MCU starts pulses on all channels, determines the order in which the pulses have to be turned off, and enables the output capture interrupt with the first alarm set to the smallest command. From this point on, the output capture interrupt plays the important role. When each channel’s pulse expires, the output capture interrupt goes off. The MCU loads the next pulse’s length into the output capture register and changes which channels are high and which are low. After all of the channels are output, the interrupt routine initiates the serial transfer of the measured pilot inputs to the main computer by writing the first byte. The serial handler takes care of the other bytes using the interrupt raised by the MCU when the transfer buffer is empty.

The MCU measures the commands of the pilot by using the input capture interrupts. Since the input capture registers buffer the value of the timer when each edge is detected by the MCU, these interrupts can run at a lower priority relative to the output compare interrupt. It is still important to keep their interrupt service routines short because once the routines are entered, all other interrupts are temporarily masked. By holding the processor past the time of the next output capture interrupt, they can indirectly contribute to the timing error of the control commands.

### 3.3.2 MCU Serial Timing

In the serial mode, the MCU does not sort the commands after they are received from the main computer. Instead, it starts the first pulse by writing 0x01 to the output port. It also sets the alarm to go off after the first command’s pulse is over. Then it simply shifts the value in the port to the left by one, making it 0x02. The MCU continues shifting outputs until the last channel is driven. It then sends a serial message to the computer as in the parallel case.
Pilot inputs are asynchronous with other functions

Figure 3-6: MCU Timing Diagram
3.3.3 Interrupt Handlers

Serial Interrupt, SCIVec

The serial interrupt handler consists of the receiver state machine, the sorting algorithm, and the transmitter state machine. The receiver state machine examines incoming serial bytes looking for the header byte, 0xff. When the header byte is received, the machine is re-started. The next byte is assumed to be the channel mask, and the following 6 bytes are the commands. As a result, the header byte cannot appear anywhere else in the serial message. This limits the range of the possible commands to 0-254, but makes the serial message small and the protocol easy to implement. The channel mask was originally included to permit the channels to be driven by different sources: the masked channels by the pilot, and the others by the computer. This approach was abandoned, but the channel mask remains in the protocol in case the approach is reinstated.

The sorting algorithm is a simple bubble sort. A new array is created for the sorted differences between commands. Another array of channel masks is created for the bytes that are going to be written to the ports. A one in the mask corresponds to a +5 Volt value on the corresponding channel. When two or more commands are equal, the mask is modified to turn off two or more pulses at once.

The transmitter state machine is similar to the receiver state machine. However, the output serial message protocol is different. It consists of 2 headers (both 0xff), 6 2-byte values, and a checksum making the total length of the output message 15 bytes. The resolution of the output message is increased because after the pulses, the timing is no longer critical as long as the transmission takes less than 10 ms.

Output Compare Interrupt: OC7Vec

The timer is a 16-bit counter running at 4 MHz. At this frequency, the resolution is 4000 counts per 1 ms, and the useful range of servo pulses is 4000-8000 counts.
The 8-bit serial commands have to be converted to the resolution of the timer by multiplying them by 16.

The output compare interrupt is raised when the 16-bit value in the output compare register ($9e$) is equal to the value in the timer register ($84$). The serial vector sets up the first such interrupt by writing the first serial command plus one millisecond to the output compare register. The interrupts that follow are set up in the output compare interrupt vector itself.

After the mask is written to the output port ($56$), the next interval is loaded, converted to a 16-bit value, and added to the output compare register.

In the parallel mode, the interrupt service routine can take longer than two 8-bit counts. When the routine is over, the timer could have run ahead of the output compare register impairing the next interrupt. To correct that, special measures are taken if the difference between pulses is one or two 8-bit counts. If the difference is one count, the interrupt is run again without returning to the main loop. This measure assumes that the processing time of the interrupt routine is about one count. If the difference is two counts, the processor is forced to idle in a loop for 32 clock cycles before re-running the routine. 32 clock cycles of the 8 MHz clock correspond to one 8-bit count.
Chapter 4

Sensors and Interfaces

4.1 Inertial Measurement Unit

4.1.1 General Characteristics

The Inertial Measurement Unit (IMU) used in the avionics package is manufactured by Inertial Sciences Inc.[3]. The unit packs 3 accelerometers, 3 gyroscopes, a temperature sensor, a microcontroller with A/D circuitry and a serial interface into a small, light, low power unit. The cost of the IMU ($9900) is lower than the cost of other devices of comparable performance. Its low weight (250 g), and small dimensions (3.30” x 2.5” x 1.83”), make the unit very attractive for the small autonomous helicopter application. The outputs of the sensors are temperature compensated. Even though the compensation equations are supposed to work at any temperatures, experience shows that warming the IMU up for approximately 10 minutes before the flight helps stabilize its outputs. After the warm-up period, the IMU has to be calibrated for additional 2 minutes by leaving it in place and logging the sensor indications. The IMU consumes approximately 5 Watts of power. It requires three voltage levels: +5, -5, GND. The -5 level is used mostly for generating valid RS232 waveforms. Since the IMU does not tolerate unregulated power, it is very important not to reverse the polarity of the
IMU’s power inputs, and to always feed it precisely regulated voltage levels.

4.1.2 Core Sensors

Gyroscopes

The gyroscopes form the core of the IMU. The sensors are manufactured on site at Inertial Sciences. They use a unique solid state technology developed at Sandia National Laboratories for the missile navigation application.

The gyroscopes are capable of measuring rates of up to 3000 degrees per second. The manufacturer allows the users to customize the ranges according to their applications. Initially, the rates where chosen to be +/- 1000 degrees per second on each axis, but the ranges were lowered later to +/- 300 degrees to increase accuracy.

Using a greater range of rates comes with a price. First of all, since 2 bytes are used to encode the measurements, the resolution decreases proportionally to the increase in the range. Secondly, the bias stability of the sensors decreases with extended range. This leads to higher drift rates.

The ranges were changed when the IMU was sent back to the manufacturer for other repairs (see below). Actual aggressive maneuvers proved wrong the assumptions that 300 degrees per second suffice. During the knife edge maneuver, the helicopter pitches up to 90 degrees, quickly spins around its yaw axis for 180 or 540 degrees, and then heads down. During this maneuver, the roll and pitch gyros stay within the bound, but the yaw gyro saturates as the yaw rate reaches up to 500 degrees/s. It was decided to avoid performing that maneuver until the IMU is sent to the manufacturer, where only the yaw gyro range can be increased.

The performance of the gyroscopes was verified in several experiments and in actual flights. The scale factors were determined by swinging the IMU in a pendulum setup. The Actual angular rates were calculated from the known length of the pendulum arm, from the measured deflection and the period of
oscillations. When the actual rates where compared with the rates reported by the IMU, the scale factor errors were found to be less than the inaccuracy of the test. The inaccuracy of the test was estimated as 1% of the full scale, which is equal to the scale factor nonlinearity error reported by the sensor manufacturer.

Gyro biases where measured on the ground with a stationary IMU. The bias stability was found to be .01 degrees/second in 2 minutes, which is comparable with the specifications of the manufacturer. The stationary bias errors were then verified in flight under vibrations. The biases before and after the flights do not differ by more than .01 degrees.

**Accelerometers**

Inertial Science gets their accelerometers for the IMU from a third party supplier. The range of +/- 10 g was chosen from several ranges that were available.

During the flight time, the biases drift by approximately 5 mg. The scale factors were found to drift significantly over long periods of time, but stay constant during flights. As a result, the scale factors have to be re-calibrated approximately every month.

**4.1.3 Filtering and Sampling**

The IMU uses AD7716, a data acquisition chip from Analog Devices, to filter the analog outputs of the six sensors and to sample them. The chip features 22-bit sigma-delta A/D with a programmable digital filter, and programmable output update rates.

In the original version of the IMU, the output update rate of AD7716 was set to 279 Hz, and the cut-off frequency of the digital filter to 73 Hz. The IMU further downsamples the discretized values to 100 Hz for the serial output. In effect, the overall system has an anti-aliasing filter at 73 Hz, and samples at 100 Hz, which does not prevent anti-aliasing. After the defect was discovered, the
IMU was sent back to the manufacturer, where the cut-off frequency of the filter was reduced to 36.5 Hz. The output update rate, which is related to the cut-off frequency of the filter was changed to 140 Hz.

In addition to re-programming the chip, analog passive filters with cut-off frequencies of 9 Hz were inserted on the outputs of all six sensors. These filters were designed to help the vibration isolation system in filtering the noise from the vibrations induced by the main rotor at 25-26 Hz. The filters also perform the anti-aliasing function.

The IMU’s microcontroller downsamples the 140 Hz updates from AD7716 to 100 Hz for the serial interface. This task is not performed ideally. The serial output does not occur precisely at 10 ms intervals. Oscilloscope measurements show that every several seconds, the distance between packets starts to slowly grow from 10 ms to about 10.7 ms. Immediately after that the distance slowly decreases to about 9.3 ms, and then goes back to 10 ms. The IMU outputs irregular intervals approximately 10% of the time. This does not represent a significant problem because the controller runs at 25 Hz, which is significantly slower than the IMU. With 40 ms period, +/- .7 ms timing scatter is unimportant.

### 4.1.4 Serial Interface

The IMU communicates with the main computer via a serial interface operating at 115200 Baud. Each 16 byte packet includes a header, a checksum, a counter, and 6 2-byte values.

Only 2 wires are used for the serial connection: the TX channel, and the ground. The main computer does not need to send any information to the IMU.
4.2 Global Positioning System Receiver

4.2.1 General Characteristics

The GPS provides once-per-second position and velocity updates. In GPS-INS implementations, the updates from the GPS are used to correct the integration of inertial sensor information coming from inertial sensors such as the gyroscopes and accelerometers in the IMU.

For highly dynamic applications such as the helicopter, a high number of satellite tracking channels is desirable. While most mainstream receivers carry 8 channels or fewer, a few specialized receivers carry 12. The receiver in the avionics package is a 12-channel SUPERSTAR receiver from Canadian Marconi.

The accuracy of the GPS can be significantly improved by using a complementary GPS receiver on the ground. The measurements of the ground receiver can be uploaded to the onboard receiver through a wireless link and combined with the onboard measurements to provide the capability of differential GPS. The accuracy of the SUPERSTAR in the differential mode increases to .2 meters from 30 meters. Because single GPS is adequate for the short duration of the flight, the avionics package does not use the differential GPS receiver.

Selective Availability is a measure of the U.S. government to limit access to the GPS only to authorized users. With Selective Availability on the accuracy of the GPS decreases by a factor of 3. This ceased to be an issue since the summer of 2000 when President Clinton decreed the cancellation of Selective Availability in the continental United States.

An active antenna mounted on the horizontal stabilizer is used in the helicopter. The antenna requires a supply of 5 Volts.
4.2.2 Serial Interface

The GPS receiver uses a serial interface to communicate with the main computer. The serial interface is implemented in CMOS voltage levels. This is different from the usual way the serial protocol is implemented in PCs. PCs use RS232 voltage levels which are bipolar and range from -12 to +12 Volts. Fortunately, the main computer is equipped with one TTL level serial port that is there for a touchscreen controller. The TTL serial port, COM4, is adapted for the GPS. In order to improve the reception of CMOS levels by the TTL serial chip, a series of 2 CMOS/TTL inverters was inserted into each path. A circuit diagram of the hardware in the GPS serial connection is given in Figure 4-1. The GPS interface is implemented on a small protoboard mounted on top of the receiver’s printed board. The protoboard contains a 20-pin connector, and a 74LS14 CMOS/TTL inverter chip.

The GPS receiver is highly customizable in data format, content, and interface. The following settings were chosen:
• Data Format: A CMC Binary message produced every second

• Content: Message #20 which includes latitude, longitude, altitude, velocities, solution timestamp, and confidence measures

• Interface: Serial protocol at 9600 Baud, 8 data bits, 1 stop bit, no parity

The GPS receiver does not use its receive line.

4.3 Barometric Altimeter

The GPS is an excellent source of latitude and longitude information. Unfortunately, the altitude information provided by the GPS receiver is significantly worse. To compensate for that a barometric altimeter is used. The altimeter measures ambient atmospheric pressure which can be converted to altitude using the formula: \( Z = \log\left(\frac{P_0}{P}\right) - \frac{(RT)}{g} \), where \( Z \) is the altitude, \( P \) is the measured pressure, and \( P_0 \) and \( T_0 \) are the sea level pressure and temperature respectively.

When properly calibrated, the altimeter provides excellent accuracy. Its resolution of 1 millipsi translates to roughly 2 feet of altitude.

As the GPS, the pressure sensor uses TTL serial levels. Since the main computer only has one TTL level serial port, the altimeter’s serial interface has to be converted from TTL levels to RS232 levels. This is accomplished with a MAX232 chip shown in Figure 4-2. One of the major benefits of this particular converter is that it only uses one 5 Volt power supply and requires very little power. The chip can convert up to two serial interfaces at once.

4.4 Magnetic Compass

The magnetic compass is a tri-axial magnetic field sensor implemented in a chip. The sensor provides three analog values corresponding to the three components of the magnetic field in the orientation of the helicopter.
A well calibrated compass is capable of yielding excellent attitude information. Unfortunately, after the interfaces with the compass were implemented and the sensor was tested, the compass was found to drift a lot over long periods of time and also to drift slightly during flight.

The calibration procedure involves rotating the helicopter around 2 circles inscribed into perpendicular planes: one circle in the horizontal plane, and one in the vertical. The procedure is too complex to execute with necessary precision on the field. In addition, after careful evaluation of the performance of the Kalman filter without the compass, the contribution of that sensor was found to be negligible. As a result, the sensor was dismantled, and the software pertaining to it was stripped from the rest of the software.

However, since it is still possible that the compass will find its use in the future, the hardware interface and the sensor are described briefly here.

The compass provides three analog outputs which are sampled by the A/D converter in the main computer. The compass is mounted on top of the horizontal stabilizer of the helicopter, and its cable runs to the avionics box located under the landing gear.

The analog channels of the computer are set up to sample 3 channels with a
common ground.
Chapter 5

Main Computer, Telemetry, and Software

5.1 Main Computer Hardware

The principal considerations when choosing the onboard computer were: I/O, computational power, weight, compatibility, reliability, and power consumption. It was important to select a CPU that has enough I/O for all sensors and actuators without the need to add any additional cards to it. The required interfaces (including foreseeable future expansion) consist of 4 serial ports, an ethernet card, three or more A/D channels, and a parallel port for digital I/O.

The computer used in the avionics package is the TP300 PC104 single board computer made by DSPDesign of England. The board includes all of the required I/O, is light, and powerful. It has a 266 MHz MediaGX Cyrix processor, 32 MB of RAM, 16 MB of persistent flash, an ethernet controller, 4 serial ports, 1 parallel port, and 4 A/D conversion channels. The only functional sacrifice that had to be made was the high power consumption of the computer, 10-12 Watts. A significant logistic downside is the slow response time of the manufacturer. The card has to be ordered well in advance through a U.S. distributor, Saelig Inc.
All of the necessary hardware in the computer is compatible with the QNX operating system [10].

The CPU has to be cooled off by a special small fan and a heat sink that connect directly to the computer board.

## 5.2 Telemetry Hardware

One of the design requirements was that the helicopter is capable of performing aggressive maneuvers on its own, without any operation-critical support from the ground. In view of this specification, a telemetry system with high bandwidth but low reliability was chosen. Wireless LAN not only possesses the aforementioned characteristics but also offers additional benefits.

First of all, wireless LAN adapters can be smoothly integrated into the internal networking of QNX. The helicopter and the ground station can be on the same relatively high speed ethernet network, which greatly simplifies the development and flight operations.

Secondly, a wireless LAN adapter does not require writing any tedious serial or network drivers since it simply plugs into the existing ethernet plug of the main computer. The adapter effectively replaces the ethernet wire running from the ground station to the helicopter.

RangeLAN2 adapters from Proxim are used on both ends of the network. The adapter’s specification is 1.6 MBits/sec which drops to 800 KBits/sec when the reception is poor. This was found not to be true. The actual performance depends on the size of the packets and ranges from 400 KBits/sec for large packets to 2000 Bits/sec for tiny packets. The reason for such a discrepancy is that it takes a long time to initiate sending a packet, but once contact is initiated, the transmission is extremely fast. For that reason multiple packets are assembled on the helicopter before being sent to the ground station.
5.3 Software

5.3.1 Operating System

Real Time

Both the helicopter and the ground station run QNX operating system. QNX is a hard-real time operating system. Its context switching time and its interrupt latencies are on the order of microseconds.

Messaging

QNX is also a messaging operating system. Messaging primitives are embedded into the kernel of QNX. Three functions, Receive(), Send(), and Reply() are provided for basic communications between processes. The sending process initiates communications by sending a message to the other process which should be blocked executing the Receive() statement. Upon the reception of the message, the second process becomes unblocked while the sending process blocks waiting for the reply. The second process has to use the Reply() function to complete the communication. Reply() can contain useful data.

This setup is used almost exclusively in the helicopter software for inter-process communications. It has many advantages over standard ways of implementing inter-process communications such as shared memory, semaphores, and signals. The messaging is conceptually simpler and easier to implement than the alternatives. In addition, the two processes remain completely separated from each other. The memory used by one process cannot be corrupted by another process. Since the data are copied into the message body of each message, transition state memory simply does not exist.

Sometimes it is important to send a message to a process without blocking to receive the reply. In such cases, QNX utilities called proxies can be used. Proxies are set up and triggered by the sending process using the commands:
qnx_proxy_attach(), and Trigger(). The sending process resumes its operation immediately after triggering a proxy. The receiving process catches the proxies using the regular Receive() command, but does not need to reply to them. Interrupt service routines and timers also trigger proxies.

Both proxies and regular messages are assigned a priority level which is used by the operating system to decide which message or proxy to deliver first.

Networking

QNX is a networking operating system. Each computer is assigned a node number. It is possible to send messages to processes running on other nodes, to start and stop programs on other nodes, and even use a console physically located on one node to operate another node. During flight operations, the helicopter is assigned node #1, and the ground station is given node #2. The ground monitoring programs use message sending capabilities to start and stop programs on the helicopter and to request data updates.

Networking also makes helicopter computer maintenance and program development much easier. In the lab, the helicopter computer is on the QNX network, which allows other computers to download software to it, start and stop programs.

5.3.2 Peripheral Interfaces

Overview

All of the sensors in the package, the MCU, and the telemetry system provide information to the program running on the main computer. Since most of these peripherals work asynchronously from the main computer, they have to be given the ability to alert processes running on the main computer that new information is available.

For the serial peripherals, this notification is done using a proxy issued by custom interrupt service routines, or, in the case of the barometric altimeter, by
the standard QNX serial driver.

**IMU**

The IMU is the fastest and the most important peripheral in the system. It is very important that the software handling it is lean and efficient.

As described in the previous chapter, the IMU generates 16 byte messages at 100 Hz. In addition to the data, these messages contain a header byte, a counter, and a checksum.

The IMU software receives the messages from the serial port, verifies the checksum, and triggers a proxy to alert other processes that a complete IMU message has just been received.

This procedure is implemented as an interrupt service routine attached to interrupt #4 that is physically connected to COM1, the serial port of the IMU.

The state machine drawn in Figure 5-1 handles the reception of IMU messages. The machine starts out in the SYNCHRONIZING state and stays there until the header of the next message arrives. The rest of the message is then inserted into the result buffer, IMUmessage, byte by byte. After each byte is received, the checksum is updated. When the checksum byte arrives in the very end of the IMU message, it is compared with the checksum computed by the interrupt service routine. If the two match, the proxy is triggered.

Since the IMU is the fastest peripheral that must accessed on every iteration, it makes sense to time other processes by the IMU messages’ arrival. The proxies generated by the IMU interrupt service routine start all other activity in the main loop. As a result, a separate process is not needed for sampling the IMU as it is needed for other sensors. The interrupt service routine reports directly to the main loop’s Receive() statement. It also follows that IMU software has to have the highest priority in the system and the sensor has to be very reliable.
Figure 5-1: IMU State Diagram
MCU

The MCU is the second highest priority peripheral in the system. It acts as a sensor when it provides pilot commands via a serial link, and as an actuator when it processes serial commands from the computer. The actuation portion of the MCU is in the control loop, so it has to be precisely timed by the system.

A low-level serial driver was written for the MCU as well. The driver consists of a write function, and a read function. Writing to the serial port is implemented as a user function that simply writes a byte to the port, waits until the UART indicates that the byte has been transmitted and move on to the next byte. Reading is implemented in a fashion similar to the IMU’s. A state machine that synchronizes to the headers and verifies the checksum is written for the MCU.

Since driving the MCU has to be precisely timed, it is put into the main loop. The MCU has to be driven every 20 ms, which is exactly the period of two IMU messages. The MCU is driven by the main loop every second IMU message.

Reading the values produced by the MCU is not a part of the critical loop. As a result, a special lower priority process is assigned the task of catching the proxies generated by the MCU interrupt service routine. This process does not block the main loop.

GPS

The driver for the GPS also includes an interrupts service routine. The routine is slightly different from the routines handling the IMU and the MCU. The GPS message contains a 4 byte header that contains no useful information, and a 2 byte checksum. The GPS service routine strips the message of the header and the checksum, reducing its size from 77 bytes to 71 bytes. The stripped version of the message is made available to a special process that only handles GPS. The GPS process is notified about a new message by a proxy. The interrupt service routine is attached to COM4, the only serial port with TTL levels.
Barometric Altimeter

In contrast with other serial devices, the altimeter does not have its own custom interrupt handler. Because the updates are relatively slow and the data is in the ASCII format, standard QNX serial device driver is used.

The QNX device driver, Dev.ser, is started at start-up by the start-up configuration script of the helicopter, /etc/config/sysinit.1. The altimeter is connected to the serial port COM2, located at the address 0x2F8, and served by the interrupt level 3.

The altimeter interface is different from that of other serial sensors in that at startup, the sensor has to be configured for the required format and for continuous operation. When the process responsible for sampling the values of the barometric altimeter is started, it writes a series of commands to the sensor configuring it. The commands are:

1. *00IN=RESET %reset the altimeter
2. *001=R5 %set the update rate to 5 Hz
3. *00P2 %start continuous updates

Magnetic Compass

Magnetic compass is no longer a part of the software package because of its limited usefulness and complex calibration procedure. When the sensor was still a part of the software package, it worked as a polled device.

A special process was assigned the task of handing the compass. The compass process did not have an internal periodic timer like the altimeter or GPS. Instead, it was receiving periodic proxies from the main loop. After receiving the proxies, the compass process initiated conversion on the A/D chip by writing words to the port of the A/D converter. The compass process was also capable of receiving messages from other processes asking for the most recent date from the compass.
When such messages are received, the compass process fills the reply with the requested data.

5.3.3 Logflight Software

All of the peripheral software described earlier was combined in a system, whose task was to log sensor outputs and pilot information for system identification and modeling purposes.

The requirements of the 'logflight' software are:

- Log outputs from all sensors in the system tagging them with timestamps
- Log pilot commands on all channels tagging them with timestamps
- Send the logged data to the ground station reliably
- Make sure that the ground station collects enough information from the sensors during flight to recover the cause of any accidents should they occur

Based on the capabilities of the ground link, it was decided to log all of the required information in the memory of the helicopter computer, and send all of the data down to the ground station in the end of each flight. At the same time, in order to fulfill the requirement that the ground station has 'blackbox' information at all times, a separate link to the ground station is used to gather data at the frequency specified by the ground station. It was experimentally established that the optimum frequency of ground updates is about 3-7 Hz.

Figure 5-2 contains a diagram of all of the processes involved in 'logflight' and their interactions.

When the helicopter boots up, it starts a program called "heli_parent." This program simply halts while listening to any messages coming from the ground station. When "heli_parent" receives a message, it spawns the logflight code as a separate process. When "heli_parent" receives the next message, it kills logflight.
Figure 5-2: Flight Logging Software
As a result, logflight software is loaded from the flash disk of the helicopter computer. Also, multiple logging sessions are possible without having to restart the helicopter computer or to re-start a program remotely.

The next step in the logging process happens at the ground station. The ground station software is started by executing the "ground.display" command. The command draws the display on the screen and stops waiting for user input. When the "Connect" button shown in Figure 5-3 is pushed, the ground station locates the "heli.parent" process running on the helicopter and sends it a message prompting it to launch logflight processes.

'Logflight"s main process performs its initialization routines and pauses waiting for the ground station to command it to start logging data.

It is up to the user to specify the suffix of the log files and the maximum duration of the logging session and to initiate the logging procedure by sending these parameters to the helicopter. The user should modify the "Duration" and
"File Suffix" fields in the graphical user interface and press the "Start Logging" button (Figure 5-3). After this, the logging software runs on its own.

'Logflight' stores the outputs of the peripherals in large data buffers allocated in advance. Since the main process carries the responsibility of writing all these data to the ground station in the end of the run, it has to have access to the data obtained in other processes. In order to accomplish that, shared memory is used between the main process, GPS process, MCU process, BAR process, and Compass process. The buffer used to store IMU data does not need to be shared since the IMU interrupts service routine reports directly to the main loop. Shared memory buffers are shown in Figure 5-2 as stacked cylinders drawn with punctured lines and connected to both processes where the memory is accessible. The internal memory buffers use the same symbol drawn with solid lines and placed inside the owner process.

After the starting parameters are received by the main process, it initializes the shared memory regions, creates file names based on the suffix passed to it, forks all other processes shown in the helicopter section of Figure 5-2, creates a proxy and attaches that proxy to the IMU interrupt service routine, and enters the main loop. The interrupt service routine receives and parses the IMU messages, checks their checksums and alerts the main loop to the new piece of data by triggering the proxies. With each IMU packet, a new iteration of the main loop begins.

Inside of the main loop, the new IMU packet is timetagged, and stored in the memory buffer. Every second IMU message, or at 50 Hz, a word is written to the MCU. The MCU, which is timed by the main computer, responds with the newest pilot command data after approximately 10 ms. The serial message with pilot commands goes directly to the MCU process. Every fifth IMU message, a proxy is triggered to start the analog compass values conversion process. As with the MCU, after a delay, the compass reports to the compass process and not back to the main loop.
When the logflight code is modified to include the controller and the Kalman filter, the main loop will also contain those two procedures.

The main loop continues its operation until a KILL signal raised by the "heli-parent" process is caught by the main process. When that happens, the main process opens a file on the ground station for each peripheral and sends all of the memory buffers to the ground station by writing to the opened files. After the data are transferred, it closes all of the open files, kills all of the running logflight processes except for "heli.parent," and quits.

Telemetry Operation

The most significant requirement of the telemetry software is to establish the regular updates of the sensor data to the ground station without influencing the operation of the logging software. The telemetry is implemented as a low-priority process that acts as follows. In the beginning of each loop, the process waits for a Send() from the ground station to tell it to start gathering a packet. After the packet is collected, it is sent back to the ground station by the Reply() command.

The telemetry then polls the peripheral processes for newest data by issuing them a Send() message of a relatively low priority. This message is served by the peripheral processes after all other higher priority activity such as responding to hardware interrupts, and storing messages in the logs. When the peripheral processes finally get to the requests of the telemetry process, they simply fill a message with the latest complete data packet and send it back via the Reply() command. The telemetry process is blocked waiting for their reply all that time.

The messages that are sent back to the ground station can have different lengths and can contain packets in any order and from any sensor. This is done in order to minimize the transmission time. For example, the GPS messages are very large, 77 bytes, and happen every second. There is no need to send these data to the ground station at 5 Hz.
The first byte of a message contains the length of the whole message in bytes. The second byte contains the type of the next sensor packet. It is followed by the packet itself, the length of which is known to both the helicopter software and the ground station software. Such arrangement makes it easy for the telemetry process to encode flexible messages and for the ground station to decode them.

The telemetry process keeps track of when to send down which sensor packets. For example, it could send the GPS packet down every five requests from the ground station, and the IMU packet every time. This has to be roughly synchronized with the frequency at which the ground station sends its requests.

5.3.4 Ground Display

The ground display is written using the Photon Application Developer, the graphical user interface editor for the QNX’s Photon windowing system.

In Photon, the objects on the screen are called 'widgets.' In order to implement ground display functionality, the properties of the widgets have to be modified by the user code to display text, change colors of indicators, and react to the buttons. A snapshot of the user interface is given in Figure 5-3. Please refer to the Application Builder documentation for information on how to build a graphical user interface in Photon.

The software organization of the ground display is shown in Figure 5-4. When the program is started, a function named display_init() is called. This function attaches a routine that gets called whenever the display receives an external message. This routine is message_handler(). The handler parses the messages and shows them on the graphical display.

The next step for the user is to spawn the logflight program on the helicopter by pressing the "CONNECT" button.

The user can then start the logging process on the helicopter by sending it a message with the logging parameters in it. This can be done by pushing the
Initialize Display and attach message_handler to all messages received by the display

**start_logflight.c**
- Send() a message to heli_parent prompting it to spawn logflight
- Change the color of the indicator from red to green
- Enable the "START LOGGING" button and disable the "CONNECT" button

**start_logging.c**
- spawn a subserver to Send() telemetry requests to the helicopter @ 3 Hz
- Enable the "STOP LOGGING" button and disable the "START LOGGING" button

**stop_logflight.c**
- Kill the subserver
- Send a message to heli_parent prompting it to kill logflight
- Enable the "CONNECT" button and disable the "STOP LOGGING" button
- Change the color of the indicator to red

**Subserver**
- Send() requests to logflight @ 3Hz, receive the reply and redirect it to the ground_display where it is parsed by the message handler

Figure 5-4: Ground Display Software Diagram
"START LOGGING" button. When this button is pushed, the ground station spawns a subserver process that is responsible for periodically requesting information from the helicopter process and sending it to the ground display for the message handler.

When the user wishes to stop logging, he or she pushes the "STOP LOGGING" button, which kills the logflight process on the helicopter and initiates file transfers.

5.3.5 What Needs to Be Done For Controller Software

Logflight software was designed for the easy migration of the code into control code. The peripheral devices were kept modular, the main process is very similar to the main process in the controller code.

There are two additional components that need to be present when the code is expanded: Kalman filter state estimation, and the controller.

The Kalman filter is run as a background process. It does most of its activity when the new GPS and barometric altimeter messages arrive, which does not happen very frequently. The Kalman filter can be thought of as a source of low-frequency updates.

The integration of the high frequency, or 100 Hz, IMU data has to happen in the main loop. The main loop has to integrate the rates and accelerations since the last Kalman filter update, which could be up to one second back in time.

The control logic has to be included in the main loop as well. The calculations are performed at 25 Hz, which corresponds to every fourth IMU message. The MCU has to be driven at 50 Hz. Therefore the MCU command is repeated every second time the MCU is driven.
5.3.6 Timing

Figure 5-5 contains the timing diagram of a 40 ms period in the main loop of the program. Within the 40 ms, the state equation has to be propagated, the MCU has to be driven twice, and the control logic has to be executed once.

Every 10 ms, a new IMU message arrives on the serial port. It has to be parsed and assembled by the interrupt service routine.

When the first IMU message arrives, the rates and the accelerations are used to propagate the state equations using the fourth order Runge-Kutta algorithm with the $\delta t$ of 40 ms. The state equations have to be propagated for the period since the last valid update of the Kalman filter, which could be up to one second behind in time. The delay is due to the fact that the GPS receiver takes one second to come up with a solution for the position and velocity. With $\delta t = 40$ ms, the state propagation routine has to be run at most 25 times. The routine performs approximately 900 multiplications of doubles, which translates into approximately 3 ms for 25 runs. This estimate is based on the speed of the onboard computer, which was measured to be $2.78 \cdot 10^{-8}$ seconds per multiplication.

After the state is propagated, the control logic is executed. The control logic is a very inexpensive process. To accommodate for future expansion, one millisecond is allocated to it.

The control command is written to the MCU approximately 4 ms after the IMU message arrives. When the servos are driven in series, the maximum delay from the time the message is written to the time the last servo is driven is 10 ms. The main computer can use that period for other pending processes.

When the next IMU message arrives, the main process does not have to do anything besides assembling and storing the message.

The MCU has to be driven after the 20 ms IMU message. Since the controller is run at 25 Hz, the new control word is not computed at this time. However, in order to drive the MCU at regular intervals, a delay equal to the time required
by the combination of the state propagation routine and control logic has to be inserted. Since the time required for these two activities is not precisely known, the same delay has to be inserted after the first IMU message arrives.

5.3.7 Code Organization and Version Control

The software is kept in a version control repository under the name 'chopper.' All of the modules are stored in their own directories under 'chopper.' Figure 5-6 is a chart of software organization.

The software for GPS, MCU, altimeter, and compass, is contained in separate directories. For each of them, a standalone version of the driver also exists. When the module is compiled with the compiler directive -DSTANDALONE, or with the command make standalone, the standalone version is compiled. The standalone version is a program designed for testing the driver separately from the rest of the program. Please refer to the module help for additional information.
5.4 Unsolved Software Problems

'Logflight' software has proven to be operational and efficient in gathering data for the system identification purposes. There are, however, several issues that have to be solved before the code can be used for the controller implementation.

Skipped IMU Packets

The most important of these problems is the problem of the skipped packets in the IMU. Up to one packet per second can be lost in some logging sessions. The losses manifest themselves through the skipped counts in the IMU packet's counter field. The IMU packets contain a counter that is incremented for each new IMU packet until it reaches 99 where it rolls over to 0. In addition, the time tag attached to the packets by the logflight software reflects the skipped messages.

The amount of skipped packets seems to be constant once a logging session is started, but varies between logging sessions. The best sessions exhibit no skipped
packets, and the worst show approximately one packet per second.

The problem appears to be that of the interrupt contentions. The long interrupt service routines from other hardware devices could occupy the processor time for more than the inter-byte time in the IMU messages. As a result, one or more bytes in the IMU packet could be skipped, and the IMU interrupt service routine would not recognize a valid packet since the checksum would not match.

In order to mitigate the problem, the IMU proxy is given the highest priority when it is delivered to the main loop, which has a high priority as well. However, the problem still persists.
Chapter 6

Hardware-in-the-loop Simulation (HILSIM)

6.1 HILSIM Overview

It is hard to imagine a system where simple failures carry more serious consequences than the fly-by-wire control system. A bad solder link, or a memory leak on an aircraft flight system can put humans in harm’s way and jeopardize expensive equipment.

Even in small autonomous systems such as the helicopter described here, crashes can be dangerous to people, and extremely bad to the project.

For this reason, it was decided to build a hardware-in-the-loop simulator for the helicopter. In the simulator, the actual hardware and software of the helicopter are used and therefore carefully tested.

The hardware-in-the-loop simulator (HILSIM) replaces the environment for the flight control system. All of the sensors are disconnected from the computer and replaced by signals generated by the external computer. The commands of the flight system is measured and fed to the external computer.

In addition to the benefit of checking out all of the hardware and software
running on the helicopter, the HILSIM is used to do what a regular simulator does: help design the controller and evaluate its performance.

A visualization system allows the pilot or the flight control system to fly the helicopter using the screen of a computer on the network.

6.2 HILSIM Implementation

The HILSIM has to simulate three serial sensors, the IMU, the GPS, and the barometric altimeter. The external computer has to be able to capture and digitize the commands of the flight system: five servo angles.

The simulation is run on a desktop computer with a Pentium III 733 MHz processor and 256 RAM. At the time of the HILSIM design, the computer represented a high-end personal computer. The simulation computer has to have enough power to run a high-fidelity simulation in real time, producing inertial updates at 100 Hz.

The standard desktop system does not have enough I/O to communicate with the onboard computer. An expansion serial card along with an A/D conversion board had to be purchased.

The system is also equipped with an ethernet card that allows it to feed the position and attitude updates to the visualization computer. The visualization computer is a networked SGI Octane running an OpenGL server.

The simulation computer runs QNX because it has to be able to generate accurately timed serial messages to simulate the sensors. In addition, the simulation computer has to measure and accurately timestamp the motion of the servo motors.

The angles of the servo models are converted to analog voltage levels by potentiometers in a device called "Torture Rack."

A diagram of the HILSIM setup is given in Figure 6-1.
Figure 6-1: Interconnection Diagram of the HILSIM
6.2.1 Torture Rack

Torture rack is a device that accepts pulse width modulated servo commands from a computer and converts them to analog voltage levels proportional to the angles that the servos were commanded.

The torture rack consists of five regular servo motors physically connected to five potentiometers by metal pushrods. The pushrods are connected in such a way that the gain between the motion of the servo arm and the motion of the potentiometer arm is one.

This type of connection is particularly easy because the potentiometers have the same shape as the servo motors. They are simply the servos whose motors and control circuits are removed leaving only the feedback providing potentiometer inside.

The torture rack has to be driven by the MCU on the servo side, and has to be given a reference voltage for the potentiometers on the other side. When fed 9 Volts from a dedicated power supply, the potentiometers produce voltage levels acceptable by the A/D card of the computer without the need for amplification or attenuation. The analog voltages range from 3 Volts to 8 Volts fitting within the 0-10 Volt range of the A/D board.

The major advantage of the torture rack is that it puts the servo lag in the loop, albeit not including the effects of loading. Experience shows that the servos that are used in the helicopter do not exhibit significant deviations in their behavior when driven under load less than half of their declared maximum torque. At the same time, all servos have non-negligible delays in their transfer functions. A simple program, servo_tester was written to analyze the behavior of the servos using a frequency sweep.

Another significant advantage to using the torture rack is that the actual flight hardware, i.e. servos and their interfaces are used. The more physical and electrical connections are checked out on the ground in the HILSIM, the better.
6.2.2 A/D Board

A low-cost, 16 channel A/D board was purchased for the purpose of digitizing the angles of the servos. The board is manufactured by Real-Time Devices, USA. Its part number is ADA1110.

The board is used to sample 5 analog levels at the frequency of 100 Hz. The interface of the board is very simple, but there are several caveats and tricks that are used to make it work.

First of all, the input conditioning filter should not be used. As shown in Figure 6-2, the board includes an operational amplifier for low pass filtering and amplifying the inputs by soldering two resistors and one capacitor into specially designated placeholders. Unfortunately, the same active filter is used on all channels. This creates problems when switching channels.

For example, the original design included a filter with 16 Hz cut-off frequency. Since five channels have to be sampled almost concurrently, right after the first channel is sampled, the second one is selected after several $\mu$seconds. Other channels follow. The filter attenuates the high frequency switching and distorts the output.

The second problem of the board is its sample-and-hold chip, which is marked in Figure 6-2 as "S/H". The chip samples the analog voltage level on its inputs and holds it for several $\mu$seconds for the A/D converter. The problem is that the duration of hold depends on the output impedance of the analog circuit on the
input of the A/D board. With low impedance circuits the delay is small. With the potentiometers whose resistances are on the order of 5 KΩ, the delay is so big that when the next channel is loaded into the sample-and-hold chip, the previous value is still there. As a result, the A/D converter sees a combination of the two channels and that is what it converts.

This problem is solved by switching to channel #15, which is connected to the ground, right after each of the potentiometer channels. The output impedance of that channel is zero, resulting in the fast clearing of the sample-and-hold chip. A small delay still has to be inserted after the channels are switched, however. The delay is so small, that it is implemented by an idle counter, for(i = 0; i < COUNTMAX; i + +).

6.2.3 Serial Interfaces

A standard PC has only two serial ports. An expansion card with two more serial ports was purchased. The BlueHeat/PCI2 by ConnectTech, was selected because of its proven QNX compatibility.

The card works seamlessly. The two extra serial ports use a QNX serial driver provided by the manufacturer. The serial devices are named /dev/bh1 and /dev/bh2.

6.2.4 Visualization

Visualization is achieved by animating a 3D OpenGL model of a helicopter and the surrounding environment using the position and attitude information provided by the simulation computer.

The communication between the computers uses standard TCP/IP sockets. The simulation computer handles the timing, sending TCP/IP packets every 100 ms. As soon as a new packet is received, the helicopter position and orientation on the screen are adjusted.
The packets are ASCII strings that contain the timetag, the three positions, and the Euler angles of orientation. The first byte of each packet is a binary byte with the value of 1. When the simulation computer wants to terminate the connection, it simply replaces the first byte with the value of 255.

A snapshot of the visualization screen is given in Figure 6-3.

6.3 HILSIM Software Organization

HILSIM is organized in a very similar fashion to the logflight code described earlier. It consists of the main process that is responsible initializing the de-initializing all other processes. The main loop handles the most time critical tasks such as sampling the servo angles, performing the simulation, creating IMU messages and other sensor models. Figure 6-4 contains the software diagram of
the HILSIM.

The main loop is periodically awaken by a 100 Hz software timer. Upon receiving the timer proxy, the main process reads the servo angles and updates the simulation using the new positions of the actuators. The state of the simulation is contained in a data structure type state_t, which contains position, velocity, rates, orientation and other information. The information contained in the state along with the state of the sensor model are used to create a noise-polluted IMU message. The I/O function OutputIMUData() is then used to send the IMU message to the onboard computer.

The main loop can also receive a message from other processes requesting that other sensors are simulated or that the state information is passed to the visualization computer. In that case, the main loop uses the appropriate sensor model function to create a sensor message, and then sends it back to the calling process. In the case of the visualization routine, the visualization string is created and sent back in the reply.

The GPS and BAR processes have their own software timers working independently and asynchronously from the main loop. When the timer kicks, these processes send a message to the main loop requesting a fresh sensor message, which they then output to the onboard computer using the routines from the I/O module, hilsimio.c.

The dynamics module is by far the most complicated part of the HILSIM. It was written by other students and is described in other documents.
Figure 6-4: HILSIM Software Diagram
Chapter 7

Identification of the Rotational Dynamics of the Helicopter

7.1 System Identification Method Overview

In this paper the rotational dynamics of the helicopter is defined as the response of the helicopter to the pitch cyclic and roll cyclic commands of the pilot. Based on the published models of helicopter dynamics[8, 12], the effects of pitch cyclic and roll cyclic commands are assumed to be isolated from the dynamics of the helicopter other than that of the main rotor and the flybars. Therefore, identifying the rotational dynamics involves determining the function of the form:

\[
\begin{align*}
\dot{x} &= \mathcal{F}(x, u) \\
y &= \mathcal{G}(x, u) \\
y &= [q\ p]^T \\
u &= [\delta_\epsilon\ \delta_a]^T
\end{align*}
\]

where \( p \) is the roll rate, \( q \) is the pitch rate, \( \delta_\epsilon \) is the pitch cyclic or the 'elevator' command, \( \delta_a \) is the roll cyclic command ('aileron'), and \( x \) contains the relevant
state of the helicopter, the main rotor and flybars.

In the vicinity of hover, the model can be linearized around the trim conditions, transforming the full non-linear equation 7.1 into a state-space model with two inputs and two outputs:

\[
\dot{x} = Ax + B \begin{bmatrix} \delta_e \\ \delta_a \end{bmatrix}, \quad (7.5)
\]

\[
\begin{bmatrix} q \\ p \end{bmatrix} = Cx + D \begin{bmatrix} \delta_e \\ \delta_a \end{bmatrix}, \quad (7.6)
\]

where \(x\) is the state of the state-space system. The number of elements in \(x\) has to be determined based on the theoretical model.

The linear model is generally valid near hover, which is the case when the forward speed of the helicopter does not exceed 5% of the rotor tip speed. For the helicopter described here, the forward velocity limit of hover is 5 m/s.

A linear model of the rotational dynamics of the helicopter near hover is determined based on a physical description of the behavior of the helicopter. The physical parameters that are not available for direct measurements are identified. These parameters can be used for designing a non-linear model of the helicopter that is valid in all flight conditions. The linear model itself can be used for tuning the non-linear model.

The dynamics of small model helicopters is quite different from the well-studied dynamics of most large helicopters. The main differences are the higher stiffness (faster response to control inputs) of the rotor compared to the usual teetering rotors, the smaller disk loading of the smaller helicopters, and the effect of the Bell-Hiller stabilizer, or the flybars[12, p. 2]

The system identification procedure described here involves the following steps:

1. Design the experiment by finding the set of maneuvers that excite all of the
rotational modes of the system. The inputs must not compromise safety during the test flight.

2. Execute the experiment and collect input-output data.

3. Examine the data, remove the trends, apply noise-shaping filters.

4. Select and define a model structure based on the theory of helicopter rotational dynamics within which a model is to be found.

5. From the theoretical model structure, determine which parameters can be measured directly and which must be identified.

6. Compute the best model for the model structure according to the input-output data and a given criterion to fit.

7. Examine the properties of the model and evaluate whether the theoretical model has to be expanded.

8. Validate the model on a different set of data.

7.2 The Experiment

The helicopter system cannot be subjected to pure pitch and roll cyclic commands because the pilot has to continuously apply stabilizing updates.

To the extent that it was possible, the cyclic commands were isolated during the data collection. The pilot was instructed to stabilize the helicopter and apply a step input in pitch cyclic. After the command, the pilot had to let the helicopter react to the command while keeping the stabilizing corrections to the minimum. The same procedure was repeated for the roll cyclic command. Additional maneuvers included coordinated turns and hover.

During the flight, the commands of the pilot were recorded using the MCU (Chapter 3). The resulting pitch and roll rates were recorded using 'logflight'
software (Chapter 5) and the IMU. A segment of the recorded data is given in Figure 7-1. The segment features a minute of flight where several coordinated turns were performed.

### 7.3 Noise Shaping

The useful data are concentrated in the frequency range from 0.1 Hz to 1 Hz. It is important to get rid of high frequency colored noise because the system identification algorithm will introduce biases into the model parameters [6].

The noise filters have to be applied to both the inputs and the outputs. When this is done, they have the effect of frequency weighting [6].
7.3.1 Noise in the Data

The roll and pitch rate data contain two main sources of noise: the resonance frequencies of the vibration isolation system, and approximately 3 Hz noise possibly coming from flybar flapping (see Section 2.3). The noise peaks for the pitch rate are clearly seen in Figure 7-2. Their frequencies are 2.7 Hz and 7.1 Hz.

The 26 Hz noise from the main rotor is filtered out by the vibration isolation system.
7.3.2 The Design of the Filters

Both peaks are attenuated by notch filters whose Bode plots are shown in Figure 7-4 and Figure 7-3.

Notch filters introduce a phase lag that reduces the phase margin of the system. At the crossover frequency of 2 rad/sec, the lag is 8 degrees. Despite the reduced phase margin, an adequate controller can be designed.

7.3.3 The Results of Noise Shaping

As the result of the filter, the noise is greatly reduced as shown in Figure 7-5.
Roll Rate Notch Filter

![Roll Rate Notch Filter Bode Plot](image)

Figure 7-4: Roll Rate Notch Filter Bode Plot
A Segment of Pitch Rate Data

Figure 7-5: A Segment of Pitch Rate Data Before and After Filtering
7.4 The Model

7.4.1 Flapping

The model presented here is taken from [8]. Several signs are changed to adapt the model from counterclockwise rotors to clockwise rotors.

The rotational dynamics of the helicopter is based on the motion of the main rotor blades called flapping. When the blades flap, the disk defined by the blade tips tilts. The tip path disk is chosen as the reference over the plane defined by the blades themselves because under aerodynamic loads the blades cone and get out of a plane (Figure 7-6). The coning of the blades is a trim effect, which does not have to be included in the linearized model. Flapping, however, is very important to the dynamics of the helicopter since it rotates the thrust vector thereby creating a moment on the helicopter.

The flapping angle at any location along the radial of the disk can be approximated as: \( \beta = \beta_0 + \beta_{1c}\cos(\psi) + \beta_{1s}\sin(\psi) \), where \( \psi \) is the horizontal angle of the leading blade with the zero position defined in the direction of the tail rotor.

The behavior of the flapping angle can be described as a second order differential equation of motion: \( I_{\beta}\ddot{\beta} = \sum \text{Moments acting on the blades} \) The following moments are modeled:
• The effect of the stiffness of the connection between the blades and the shaft is modeled by a coil spring of stiffness $K_\beta$.

• The gyroscopic accelerations of the blades rotating out of their spin plane by the roll and pitch of the helicopter.

• The moments generated by the lift produced by the blade are modeled using the blade element theory.

The resulting equation is:

$$\beta'' + \frac{\gamma}{8} \beta' + \lambda_\beta^2 \beta = \frac{2}{\Omega} (-p \cos(\psi) - q \sin(\psi)) + \frac{\gamma}{8} (\theta - \frac{4\lambda_i}{3} + \frac{-p}{\Omega} \sin(\psi) + \frac{q}{\Omega} \cos(\psi)),$$

(7.7)

where 'prime' is the derivative with respect to $\Omega t$, the angular rate of the blades multiplied by the time. The parameter $\gamma$ is the Lock number defined in terms of the radius of the blade, $R$, lift curve slope $a_0$, blade span, $c$, and air density, $\rho$, and inertia of the blades, $I_\beta$ as $\gamma = \frac{\rho c a_0 R^4}{I_\beta}$. The Lock number is an important dimensionless parameter that measures the ratio of the aerodynamic forces to the inertia forces acting on a rotorblade. $\lambda_\beta$ is the flapping frequency ratio defined as $\lambda_\beta^2 = 1 + \frac{K_\beta}{I_\beta \Omega}$, and $\theta$ is the commanded pitch of the blade.

The pilot commands are transferred to the blades using a swash plate mechanism [8] and flybars. The effect of the flybars in hover can be modeled as a transfer function between the commands of the pilot:

• $\delta_e$, the 'elevator' command

• $\delta_a$, the 'aileron' command

and the respective pitch angles of the main blades:

• $\theta_{1c}$, the lateral pitch cyclic

• $\theta_{1s}$, the longitudinal pitch cyclic
Because of the cyclic effect of the swash plate, the commands of the pilots can be written as:

\[ \theta = \theta_0 + \theta_{1c} \cos(\psi) + \theta_{1s} \sin(\psi) \]

Since the time constants involved in the flapping motion of the blade are much smaller than the time constants of helicopter motion, equation (7.7) can be solved for the steady state case response. The terms not involving the sines or cosines can be ignored since they represent the trim condition. The solution is:

\[ \beta_{1c} = \frac{1}{1 + S_\beta^2} \left( S_\beta \theta_{1c} - \theta_{1s} + (-S_\beta \frac{16}{\gamma} + 1) \frac{p}{\Omega} + (S_\beta + \frac{16}{\gamma}) \frac{q}{\Omega} \right) \] (7.8)

\[ \beta_{1s} = \frac{1}{1 + S_\beta^2} \left( S_\beta \theta_{1s} + \theta_{1c} + (-S_\beta - \frac{16}{\gamma}) \frac{p}{\Omega} + (1 - S_\beta \frac{16}{\gamma}) \frac{q}{\Omega} \right) \] (7.9)

where \( S_\beta \) is the stiffness number defined as

\[ S_\beta = \frac{8 (\lambda_\beta^2 - 1)}{\gamma} \] (7.10)

All of the parameters used in (7.8, 7.9) except for \( S_\beta \) can be measured directly. \( S_\beta \) has to be determined by matching the flight data to the theoretical model.

The flapping blades create a moment on the helicopter body in two ways: by the reaction torque of the shaft to the flapping of the blades, and by the rotation of the thrust vector which creates a moment around the center of gravity of the helicopter (see Figure 7-7).

The equations of motion for the blades can be written as:

\[ I_{yy} \dot{q} = (-K_\beta - T_{mr} \ z_{mr}) \beta_{1c} \] (7.11)

\[ I_{xx} \dot{p} = (K_\beta + T_{mr} \ z_{mr}) \beta_{1s}, \] (7.12)
where \( I_{yy} \) and \( I_{xx} \) are pitch and roll moments of inertia of the helicopter.

Substituting (7.8) and (7.9) into (7.11) and (7.12) yields the following model:

\[
\begin{align*}
\dot{q} &= \begin{bmatrix} \frac{1}{1+\beta^2} \cdot -K_\beta - T z_{mr} \cdot \frac{S_\beta}{I_{yy}} + \frac{16}{\gamma} \Omega \cdot \left( \frac{1}{1+\beta^2} \right) \cdot -K_\beta - T z_{mr} \cdot \frac{S_\beta}{I_{yy}} - \frac{16}{\gamma} \Omega \end{bmatrix} \begin{bmatrix} q \\ p \end{bmatrix} \\
\dot{p} &= \begin{bmatrix} \frac{1}{1+\beta^2} \cdot -K_\beta + T z_{mr} \cdot \frac{S_\beta}{I_{yy}} - \frac{16}{\gamma} \Omega \cdot \left( \frac{1}{1+\beta^2} \right) \cdot -K_\beta + T z_{mr} \cdot \frac{S_\beta}{I_{yy}} - \frac{16}{\gamma} \Omega \end{bmatrix} \begin{bmatrix} q \\ p \end{bmatrix}
\end{align*}
\]

(7.13) represents the state space model of the of (7.5). The dimension of the state, \( x \), is two, and the output is the same as the state, i.e. \( C \) is identity. Input does not influence the output directly, so \( D \) is zero.

Finding the elements of \( A, B, C, \) and \( D \) matrices constitutes the system identification step. All of the parameters in (7.13), but \( S_\beta \) can be measured or estimated with sufficient accuracy. Therefore, identifying \( S_\beta \) is the main task of the system identification step. Note that the inputs in this state-space model are
not the pilot inputs, but the pitch angles of the main rotor blades.

7.5 Measuring the Parameters

Table 7.1 contains the physical parameters measured on the helicopter and gives a brief description of the methods of identification. For the more elaborate experiments, a separate description is given.

7.5.1 Swing Tests

The moments of inertia of the helicopter and of the main rotor blades were estimated using an experimental setup referred to as 'swing test' in this paper\cite{2}. The part is suspended from a point located along the axis of interest and swung like a pendulum (see Figure 7-8). The resulting motion is that of a physical pendulum whose frequency is described by the equation:

\[ \omega = \sqrt{\frac{mgat^2}{I}} \]

where \( I \) is the inertia of the part, \( T \) is the period of oscillations, \( a \) is the distance to the center of mass, and \( m \) is the mass of the part. The above equation yields:

\[ I = \frac{mgaT^2}{4\pi^2} \]

The period of oscillation was measured using a stop watch and averaged over 20 complete periods. The resulting estimates for the moments of inertia are accurate to within 5%. 

92
Table 7.1: Measured and Estimated Physical Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Value</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mass</td>
<td>m</td>
<td>6.8038</td>
<td>kg</td>
<td>weighed</td>
</tr>
<tr>
<td>Main rotor thrust</td>
<td>$T_{mr}$</td>
<td>66.7458</td>
<td>N</td>
<td>Thrust is assumed to be constant and equal to weight, $m \cdot g$</td>
</tr>
<tr>
<td>Pitch moment of inertia</td>
<td>$I_{yy}$</td>
<td>0.3856</td>
<td>kg m^2</td>
<td>Determined using 'swing test' (see section 7.5.1)</td>
</tr>
<tr>
<td>Roll moment of inertia</td>
<td>$I_{xx}$</td>
<td>0.1735</td>
<td>kg m^2</td>
<td>Determined using 'swing test'</td>
</tr>
<tr>
<td>Main rotor radius</td>
<td>$R_{mr}$</td>
<td>0.75</td>
<td>m</td>
<td>Measured</td>
</tr>
<tr>
<td>Main rotor blade inertia</td>
<td>$I_{bmr}$</td>
<td>0.0356</td>
<td>kg m^2</td>
<td>Estimated using: $1/3 \cdot 0.19 R_{mr}^2$ and confirmed by a 'swing test'</td>
</tr>
<tr>
<td>Main rotor chord</td>
<td>$C_{mr}$</td>
<td>0.06</td>
<td>m</td>
<td>Measured</td>
</tr>
<tr>
<td>Main rotor lift coefficient</td>
<td>$C_{L_{a mr}}$</td>
<td>6.0</td>
<td>-</td>
<td>Estimated from airfoil theory</td>
</tr>
<tr>
<td>Main rotor angular rate</td>
<td>$\Omega$</td>
<td>172.7876</td>
<td>rad/sec</td>
<td>Measured using hobby tachometer and assumed to be constant</td>
</tr>
<tr>
<td>Main rotor Lock number</td>
<td>$\gamma$</td>
<td>3.9168</td>
<td>-</td>
<td>$\gamma = \rho \cdot C_{L_{a mr}} \cdot C_{mr} R_{mr}^4 / I_{bmr}$</td>
</tr>
<tr>
<td>Distance from c.g. to main rotor</td>
<td>$z_{mr}$</td>
<td>0.285</td>
<td>m</td>
<td>Measured</td>
</tr>
</tbody>
</table>
7.6 System Identification

7.6.1 The Effect of Flybars

Flybars are small paddles that use aerodynamic forces to make the transfer of pitch and roll cyclic commands from the servos to the main rotor easier for the servos. Therefore, their effect on the dynamics of the rotor in hover is articulated mostly in the conversion of the command of the pilot, $\delta_e$ and $\delta_a$, into the pitch of the blades of the main rotor, $\theta_{1e}$ and $\theta_{1a}$. To model this effect accurately, the dynamics of the flybars has to be identified and the states of the flybars have to be included in the states of the rotational dynamics system. Without a good model, the effect of the flybars can only be approximated. They introduce uncertainty to the effects of pilot commands on the reaction of the helicopter. Since the flybars mostly influence the transition between the pilot cyclic commands and the pitch of the main rotor blades, the uncertainty is concentrated in the B-matrix of the state-space equation (7.13). As a simplifying approximation, the elements of the B-matrix are modeled as unknown parameters. This corresponds to modeling the flybars with arbitrary gains in the B-matrix.
7.6.2 System Identification Approach

Matlab’s System Identification Toolbox[5] is used. The input and output data sets are first de-trended to remove the effects of the trim conditions. The model of (7.13) is encoded into a user-defined parametric model with five independent parameters: $S_\beta$, $B_{11}$, $B_{12}$, $B_{21}$, and $B_{22}$.

Matlab’s version of the Prediction Error Model approach [6, p. 199] is used in the identification step. The initial guess of the unknown parameters is defined using the theoretical model with $S_\beta$ set to zero. The standard deviation of the errors in the measurement of the pitch and roll rates by the IMU is estimated to be .01 degrees/second.

7.6.3 Results of Applying the Parametric Model

The system identification procedure is applied to two sets of input-output data. The first data set contains a system identification flight with isolated pitch and roll cyclic commands. The second set contains the recordings of fairly aggressive maneuvers including fast flying and turns.

7.6.4 Data Set # 1

The system identification routine produced the following results:

\[
S_\beta = 0.0598 \quad (7.14)
\]
\[
B = \begin{bmatrix}
62.0307 & 19.8696 \\
-50.9272 & 167.2335
\end{bmatrix} \quad (7.15)
\]

Substituting $S_\beta$ into the formula for the A-matrix of the state equation yeilds:

\[
A = \begin{bmatrix}
-3.1101 & -0.567 \\
1.2601 & -6.912
\end{bmatrix} \quad (7.16)
\]
Figure 7-9: Measured and Simulated Outputs For Data Set #1, I

Figure 7-9 contains a plot of the results of applying the measured inputs to the physical model plotted with the measured outputs. The results of applying the model in which $S_{\beta}$ set to zero are also shown for reference. The segment of the flight shown in the plot contains a turn maneuver. Another flight segment is displayed in Figure 7-10. The executed there contains more roll cyclic commands.

### 7.6.5 Model Validation

The model obtained by the system identification procedure is validated using the following tests:
Figure 7-10: Measured and Simulated Outputs For Data Set #1, II
• The model is applied to a different set of input-output data and the results are compared using normalized mean square error.

• A state space model not based on physical parameters is obtained using the system identification toolbox. The performance of the non-parametric model is compared to that of the physical model.

Data Set #2

The maneuvers in the second data set were not specifically geared toward system identification. During the flight, the helicopter went through a series of high-speed coordinated turns. Despite the fact that the presence of significant forward speeds during the flight worsens the performance of the model (see Section 7.6.7), usable results were obtained. Figure 7-11 contains a segment of the outputs simulated by the model plotted with the real outputs, and the outputs obtained by the initial guess model, in which $S_\beta = 0$.

The match in Figure 7-11 is slightly worse than the one in Figure 7-9. However, the model with $S_\beta = 0$ is off by a factor of three of the fitness criterion. The disparity between the two models re-affirms the importance of non-zero stiffness between the shaft of the helicopter and the blades. The fitness criteria for the two models applied to the two data sets are given in Table 7.2. The fitness criterion is defined as the normalized mean square error,

$$ f = \sqrt{\frac{1}{N} \sum_{n=1}^{N} (y(n) - \hat{y}(n))^2}, $$

where $N$ is the number of samples, and $y(n)$ is the actual output and $\hat{y}(n)$ is the simulated output. The fitness values for the parametric model do not change significantly between the two data sets.
Figure 7-11: Measured and Simulated Outputs for Data Set #2

Table 7.2: Fitness Criteria for the identified physical model (#1), the model with $S_\beta = 0$ (#2), and the 4th order non-parametric model (#3)

<table>
<thead>
<tr>
<th>Model</th>
<th>Pitch Fitness</th>
<th>Roll Fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Set #1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#1</td>
<td>0.0948</td>
<td>0.0656</td>
</tr>
<tr>
<td>#2</td>
<td>0.2201</td>
<td>0.1893</td>
</tr>
<tr>
<td>#3</td>
<td>0.0811</td>
<td>0.0705</td>
</tr>
<tr>
<td>Data Set #2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#1</td>
<td>0.0926</td>
<td>0.0615</td>
</tr>
<tr>
<td>#2</td>
<td>0.2507</td>
<td>0.1415</td>
</tr>
<tr>
<td>#3</td>
<td>0.1214</td>
<td>0.0784</td>
</tr>
</tbody>
</table>
7.6.6 Non-parametric Model

A non-parametric model is constructed using the N4SID algorithm [7]. The N4SID method is especially well suited for multivariable systems, such as the helicopter rotational dynamics system. It is based on the subspace projection approach [6, p. 290-2, p. 340-51].

N4SID can be used to determine the effective order of the system. The algorithm is run for state-space models of several orders. For each order, the model singular values are computed[6]. A large drop between the singular value of order $n$ and the one of order $n + 1$ indicates that the system can be well modeled by a state-space model of order $n$. Figure 7-12 shows a plot of model singular values vs. the order. It can be seen that the drop between order four and order five is nearly an order of magnitude. The two additional states appear to come from the dynamics of the flybars that were only roughly approximated by the second order parametric model. The importance of the un-modeled states and the validity of the approximations can be tested by comparing the parametric and the non-parametric models.

The physical model's performance as measured by the fitness criterion is very close to that of the fourth order model. The results are summarized in Table
Figure 7-13: Measured and Simulated Outputs for the Physical and 4th Order Non-parametric Models, Data Set #1

7.2. Plots of the simulated outputs of the two models and the real output of the system are given in Figure 7-13 and Figure 7-14.

On data set #1, the fitness of the fourth order model is only marginally better than the fitness of the physical model. On data set #2, the fitness of the fourth order model is very close and only slightly worse to that of the second order physical model. This appears to further confirm the validity of the physical model.
Figure 7-14: Measured and Simulated Outputs for the Physical and 4th Order Non-parametric Models, Data Set #2
7.6.7 A Hypothesis About The Effect of Forward Speed

The poor pitch results obtained on data set #2 by the model without the stiffness, and the slightly worsened performance of the fourth order model on the same data set might be caused by the behavior of the flybars. The second data set contains the recording of fast flying and aggressive turns.

When the pilot issues a pitch cyclic command, it is first applied to the flybars. After the 90 degree delay due to the gyroscopic effect, the flybars react to the cyclic command by flapping and changing the pitch of the main rotor blades. Since the left flybar sees more air flow in forward flight than the right flybar, the flapping angle of the flybars is skewed by an amount dependent on the forward speed. As a result, the overall effect of the pitch command might vary.

The short radius of flybars could lead to the stall of the retracting flybar in fast forward flight. Consequently, the flybar would flap disproportionately.

In addition, the flapping of the flybars might occur faster and with more authority in fast forward flight. As a result, applying the model derived for hover to the data obtained during fast flight might show a mismatch in the pitch response. In particular, the simulated response appears sluggish when compared to the actual response. This is clearly seen in the case of the model with $S_{\beta} = 0$ in Figure 7-11. One reason why the physical model performs better could be that the effect of the flybars is taken into account by freeing up the B-matrix. The B-matrix is fixed in the model with $S_{\beta} = 0$. 
Chapter 8

Aggressive Maneuvers

8.1 Overview

Eric Feron and Mike Piedmonte present a human-centered approach to the design of the controller that performs aggressive maneuvers autonomously [9]. The approach is based on learning the maneuvers directly from human pilots.

Based on the data obtained in a helicopter simulation, Feron and Piedmonte concluded that the maneuvers can be differentiated by type, sequencing of steps, and duration and that the observed pilot input consists of piecewise constant functions.

Another important conclusion was that the pilot control input transitions are likely to be triggered by specific vehicle state transitions, or might be encoded as such.

The data obtained during the aggressive maneuvers performed on the real helicopter seem to support those conclusions. The attitude histories were derived with an extended Kalman filter designed by Vlad Gavrillets [1]. The following is a catalogue of several maneuvers with an attempt to identify the sequences of pilot commands and the state information that serve as the trigger for them.
8.2 The Knife-Edge Maneuver

Figure 8-1 contains the recordings of the attitude expressed in Euler angles, the downward body axis acceleration, and the pilot commands for two knife-edge maneuvers. During the knife-edge maneuver, the pilot pitches up the helicopter, quickly spins it around the yaw axis, and returns to the horizontal position.

The maneuver can be expressed in the form of a state-transition diagram of Figure 8-2.

8.3 The Loop Maneuver

Two loop maneuvers are shown in Figure 8-3 and Figure 8-4. The loop maneuver is a complete loop around the y-axis of the helicopter.

The loop in Figure 8-4 starts at approximately 4 seconds. Immediately preceding the loop maneuver, the helicopter is flying forward while banking. An aileron correction puts it back in trim by 4 seconds.

Both loops start with a ramp up input on the elevator and a ramp down on the collective. When the upside-down orientation is reached, both elevator and collective are ramped in the reverse direction until the helicopter comes out of the loop (See Figure 8-5).

In the case of the loop, Euler angles represent the orientation of the helicopter inadequately. The Euler angle representation breaks down when the pitch approaches 90 degrees as is the case for the loop maneuver. The changes in the bank angle from 5 to 7 seconds can also be attributed to the Euler angles.

8.4 The Aggressive Turn Maneuver

Two aggressive turn maneuvers are given in Figure 8-6 and Figure 8-7. The aggressive turns are fast coordinated turns that achieve the acceleration of about
Figure 8-1: Knife Edge Maneuvers
two $g$ and the banking angle of 60 degrees.

### 8.5 The Saturation of the Yaw Gyro

As explained in Chapter 2, during the recording of the aggressive maneuvers, the yaw rate gyro was saturated because the yaw rate exceeded 300 degrees per second. However, it was still possible to reconstruct most of the activity beyond 300 degrees per second. The IMU does not have a hard saturation limit. Rather, for 60 more degrees per second, the sensor provides rates that are 'rolled over' to the other side of the range. After unwrapping the rate, the activity beyond 360 degrees per second was found to be no more than 100 ms in duration. A polynomial fit was made to replace the lost data.

The Kalman filter [1] was still able to converge and provide adequate attitude information.
Figure 8-3: Loop Maneuver #1
Figure 8-4: Loop Maneuver #2
Figure 8-5: Loop Maneuver States
Figure 8-6: Turn Maneuver #1
Figure 8-7: Turn Maneuver #2
Bibliography


