Design, Testing, and Simulation of a Low-Cost, Light-Weight, Low-g,

IMU for the Navigation of an Indoor Blimp

Except where reference is made to the work of others, the work described in this thesis is my own or was done in collaboration with my advisory committee. This thesis does not include proprietary or classified information.

Abby Anderson

Certificate of Approval:

John Hung
Associate Professor
Department of Electrical and Computer Engineering

A. Scottedward Hodel, Chair
Associate Professor
Department of Electrical and Computer Engineering

David Bevly
Assistant Professor
Department of Mechanical Engineering

Stephen L. McFarland
Acting Dean
Graduate School

Design, Testing, and Simulation of a Low-Cost, Light-Weight, Low-g, IMU for the Navigation of an Indoor Blimp

Abby Anderson

A Thesis

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements for the

Degree of

Master of Science

Auburn, Alabama
May 11, 2006

Design, Testing, and Simulation of a Low-Cost, Light-Weight, Low-g,

IMU for the Navigation of an Indoor Blimp

Abby Anderson

Signature of Author

Date of Graduation

Thesis Abstract

Design, Testing, and Simulation of a Low-Cost, Light-Weight, Low-g, IMU for the Navigation of an Indoor Blimp

Abby Anderson

Master of Science, May 11, 2006
(B.E.E., Auburn University, 2003)

195 Typed Pages

Directed by A. Scottedward Hodel

In this thesis we develop an IMU for the purpose of navigating an autonomous indoor blimp. Due to the unique system properties of an indoor blimp, the developed IMU is light-weight and is capable of measuring slow rotational rates and accelerations. An emphasis is placed on maintaining low cost by using commercially available off-the-shelf components.

We present an overview of current IMU technology and evaluate that technology's suitability for use with an indoor blimp. Through this evaluation we conclude that commercially available IMUs are not viable so an IMU must be designed. We present design constraints that must be met and evaluate commercially available sensors that can meet these constraints. After selecting the most appropriate hardware, we integrate the sensors to form an IMU.

The constructed IMU is tested, modeled, and simulated. We test the IMU by applying known constant inputs and evaluating the sensors' outputs. Models of the sensors are developed from the test data. The models are then evaluated based on autocorrelation

methods. Based on experimental observations, we also develop a mathematical model of an indoor blimp in closed loop with guidance and control laws. We perform several simulations to evaluate the IMU's ability to accurately measure the blimp's states.

Acknowledgments

I would like to thank my committee members for their suggestions and contributions to this work. I would especially like to thank my advisor, Dr. A. S. Hodel, for his patience and invaluable assistance.

Thank you to Joe Haggerty, Mike Palmer, and Linda Barresi for their help in the design, building, and testing of the IMU.

Thank you to my office mates Adam Simmons and Josh Clanton for all of their help.

I would also like to thank my family, particularly my parents, Jeff and Rhonda Anderson. Without their love and support, I would have been unable to complete this.

Style manual or journal used Journal of Approximation Theory (together with the style known as "aums"). Bibliograpy follows van Leunen's *A Handbook for Scholars.*

Computer software used The document preparation package TEX (specifically LATEX) together with the departmental style-file `aums.sty`.

Table of Contents

List of Figures

Chapter 1

Introduction

In this work we present the design, construction, and testing of a light-weight, low-g IMU to be used as a navigational tool for an indoor blimp. We also present methods of integrating the IMU with a camera to improve overall system performance. By supplementing IMU measurements with measurements from a camera system, we show that state estimation errors are reduced.

In this chapter we present an introduction to the contents of this work. We begin in Section 1.1 by presenting basic concepts of inertial measurement units. In Section 1.2 we give an overview of reference frames. In Section 1.3 we present the reasons for developing an IMU specifically for an indoor blimp.

1.1 Inertial Measurement Units

An inertial measurement unit (IMU) is a device that measures a body's rotational rates and linear accelerations. Rotational rates are usually measured with gyroscopes and linear accelerations with accelerometers. An IMU generally also contains additional hardware such as a microcontroller to process the data from the gyroscopes and accelerometers.

An IMU's purpose is to provide information about a vehicle's states, usually for navigational and control purposes. States commonly of interest are position, velocity, and orientation in space. Vehicle orientation is determined by integrating the gyroscopes' measurements. Accelerometers' measurements are integrated once to obtain velocity and

integrated twice to obtain position. Information about the states is then used to navigate a vehicle to a desired position. IMUs can also be used to eliminate the need for a pilot. An autopilot can be created by combining IMU readings with control and navigation and guidance systems.

An application of IMUs in autopilots is to increase general safety. Autopilots allow unmanned vehicles to perform tasks that are hazardous to humans such as reconnaissance or surveying a chemical spill. IMUs can also be used to improve the safety of ground vehicles. Many automotive manufacturers are including gyroscopes on vehicles to detect if the vehicle is in danger of rolling so a corrective action can be taken.

1.2 Reference Frames

IMUs provide information about a vehicle's states in all of a vehicle's degrees of freedom. The number of degrees of freedom depends on how a vehicle moves through space. While vehicles such as cars are limited to having two degrees of freedom (their motion is limited to a plane or surface), other vehicles such as airplanes have six degrees of freedom (they move and rotate in 3-D space).

Generally, an IMU's accelerometers and gyroscopes are mounted in line with a body-fixed reference frame. A commonly used body-axis reference frame is shown in Figure 1.1 in which the positive x-axis passes through the front of the vehicle, the positive y-axis passes through the right side (where the right wing would be on an airplane), and the positive z-axis points downward through the bottom of the vehicle. Gyroscopes measure the rotational rates about each body frame axis while accelerometers measure the accelerations in the direction of each body frame axis. Rotation about the x-axis is

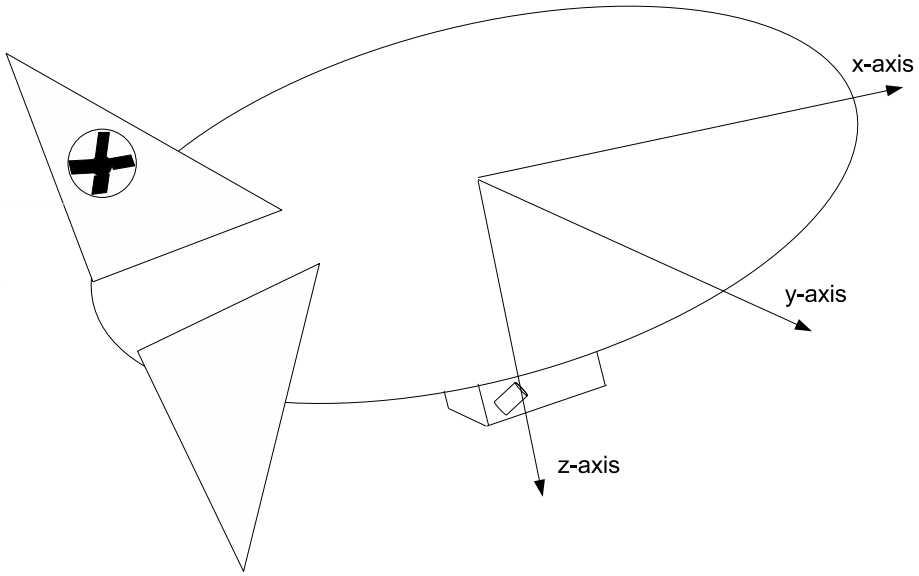


Figure 1.1: Standard Body-Fixed Axes System

roll, about the y-axis is pitch, and about the z-axis is yaw. A vehicle's roll, pitch, and yaw angles determine the vehicle's attitude in a given reference frame.

The body-fixed reference frame alone is not adequate to describe a vehicle's states in a useful manner since the reference frame moves with the vehicle. Thus, an earth-fixed reference frame, also called an inertial reference frame, is used in conjunction with the body-fixed reference frame. The inertial reference frame does not move from one instant to another. A common inertial frame is the north-east-down system in which north corresponds to the positive x-axis, east the positive y-axis, and down the positive z-axis. In order to convert between an inertial and body-fixed reference frame, a vehicle's attitude must be known. Attitude provides the angles that are between the two reference

frames, which are then used to convert a vehicle's states from the body-fixed frame to the inertial frame.

1.3 Low-g, Light-Weight IMU

IMUs are generally designed with an application in mind. Common IMU applications are for aerial vehicles that can attain relatively high velocities and rotational rates. Most of these vehicles have payloads of at least several pounds. Commonly available IMUs, therefore, are relatively heavy and are equipped with sensors designed to measure rapid accelerations and rotations. However, such IMUs are not suitable for indoor blimps, which move slowly and have small payload capacity. For such vehicles a light-weight, low-g IMU is necessary. A lack of suitable commercially available IMUs has led to the development of a light-weight, low-g IMU in the research described in this thesis.

1.3.1 Blimp Project

One motivation for building a low-g, light-weight IMU is developing an indoor autonomous blimp. The blimp available for testing in this research is fourteen feet long with an ellipsoidal mylar bag. A styrofoam gondola attaches to the underside of the bag. The gondola houses a rod connected to two fans that control the blimp's movements. A servo motor controls the fans' position. Four fins connect to the blimp's stern to provide stability. A fan is embedded in one of the fins to provide yaw control.

Because of the blimp's small size, developing autonomy presents unique challenges. The blimp's size restricts its payload capacity to approximately 4 lbs. This means any hardware added to the blimp must be light. The fans have limited power; therefore, the blimp must be flown indoors since external disturbances such as wind are too powerful

for the fans to overcome. Indoor flight excludes traditional navigational tools such as GPS. Due to the limited fan power, the blimp moves slowly compared to other aerial vehicles such as airplanes and helicopters, and so an on board IMU must be sensitive to slow motion.

In order to achieve autonomous flight, we have chosen for the blimp to be equipped with an on-board IMU. The IMU must be as light as possible to stay within the payload capacity restrictions as well as to leave room for other equipment such as cameras. In other words, the IMU's weight should not restrict the blimp's functionality. The blimp's payload capacity restrictions and low speeds require the construction of a light-weight, low-g IMU rather than the use of an off-the-shelf unit.

1.3.2 Other Applications

A light-weight, low-g IMU has applications other than the navigation of small autonomous blimps. One possible use is for a robotic arm. The attitude and position of a robotic arm is critical information to have in order for the arm to perform a desired task. Another possible application is the stabilization of helmet cameras. For instance, helmet cameras are often placed on a catcher during a baseball game to give viewers a chance to see pitching from the catcher's point of view. However, images from these cameras often give poor views of the pitch since the images move as the catcher's head moves. An IMU could correct the problem by providing information to adjust the camera to compensate for unwanted motion.

1.3.3 Why Build an IMU

Most IMUs are designed for high-speed vehicles without severe payload restrictions. With the blimp's specific restrictions, few off-the-shelf IMUs are a possibility for this project. Many IMUs weigh much more than 4lbs. With growing interest in micro Unmanned Aerial Vehicles (UAVs), light-weight IMUs are just beginning to become commercially available. However, accelerometers and gyros on commercially available IMUS are for high rates of speed and high rotational rates. The sensors are not sensitive enough to provide meaningful measurements for the low rates at which the blimp moves. Off-the-shelf IMUs also tend to be expensive.

The design of an IMU specifically for an indoor blimp has many benefits over the purchase an IMU. Careful selection of appropriate accelerometers and gyros allows payload capacity and rate restrictions to be met. Reduced power consumption over commercially available IMUs is achievable. The addition of other sensors such as range finders and altimeters to the IMU also becomes an option, which is not possible with every prefabricated IMU. Direct access to the IMU's microcontroller is another benefit because changes to the IMU's function may be necessary over time.

Chapter 2

Navigational Technology

In this chapter we present a summary of navigational technology that is specifically focused on IMUs. An emphasis is placed on the role IMUs play in the development of autonomous vehicles. In Section 2.1 we discuss technologies that are used in various IMUs. We also compare the strengths and weaknesses of these technologies as they apply to indoor blimps. We describe in Section 2.2 existing IMUs. The IMUs are broken into three categories: military units, commercial units, and units integrated with GPS. The IMUs from each category are analyzed with respect to their suitability for indoor blimps. Next, in Section 2.3 we review various technologies, including IMUs and vision, that autonomous blimp projects use for navigational purposes. Finally, in Section 2.4 we highlight the technological gaps that prevent the ideal IMU for a small blimp from being realized.

2.1 IMU Technology

In this section we provide a broad taxonomy of IMUs. Two broad categories of IMUs exist: platform IMUs and strapdown IMUs. Platform IMUs are connected to a vehicle with gimbals. The gimbals rotate in such a way that the IMU's axes are always in line with earth-fixed axes. The movement of the gimbals is used to determine the rotational rates experienced by a vehicle [22]. Strapdown IMUs are rigidly fixed to a vehicle and have no gimbals. The axes of the IMU remain in line with the body-fixed

axes of the vehicle. Rotations must be applied to the data from a strapdown IMU to get the data in an inertial reference frame rather than a body-fixed reference frame.

A platform IMU is unsuitable for an indoor blimp. Platform IMUs are heavy and require significant power relative to strapdown IMUs due to gimbals, which require precision machining. Further, a platform IMU needs motors to keep the gimbals in the proper position while the vehicle is undergoing rotations and accelerations, which compounds payload and power requirements. A strapdown IMU can weigh much less than a platform IMU because it does not need torque motors to keep it in position. While a strapdown IMU requires more computation to get the vehicle attitude data in the proper reference frame, the advance of microprocessors makes this disadvantage slight. In short, due to weight, power, and cost constraints, the IMU for the blimp must be a strapdown system.

2.1.1 Mechanical Sensors

The first sensors for IMUs were mechanical gyroscopes and accelerometers. These mechanical systems are still in wide use although they are slowly being replaced by other systems. Mechanical sensors, while capable of being highly accurate, are bulky and require more maintenance than MEMS (micro-electro-mechanical systems) sensors.

Mechanical rate gyros typically consist of a wheel rotating at a high rate on low noise ball bearings [22]. They are usually driven be an electric motor, mounted to a frame or gimbals, and often liquid filled [22]. Mechanical gyros such as the dynamically-tuned gyro are being replaced by gyros such as the ring laser gyro [7].

Mechanical accelerometers are still commonly used, especially in applications where a high degree of accuracy is required. One type of accelerometer is the pendulum accelerometer. In this kind of sensor, a proof mass is attached to a hinge, which deflects when an acceleration occurs. Another type of mechanical accelerometer is a vibrating beam accelerometer. This type of accelerometer is based on the principle that the resonant frequency of a string or bar depends on the tensile stress it is experiencing [22]. The stress is made a function of acceleration by attaching a proof mass. Acceleration is determined by measuring the frequency at which the bar or string is vibrating. The quartz vibrating beam accelerometer uses two quartz resonators to measure acceleration. Acceleration causes one resonator to undergo tension while the other undergoes compression forcing the two resonators to vibrate at different frequencies. The difference in frequencies is used to measure the acceleration [22]. The principle behind the mechanical vibrating beam accelerometer is used in MEMS technology as well.

2.1.2 Sagnac Effect Sensors

Many sensors for IMUs currently in use are based on the Sagnac effect, named after the researcher who discovered the phenomenon in 1913 [31]. Sagnac used an interferometer, which consists of a light source, a beam splitter, mirrors, and a detector. A beam of light from the source passes through the beam splitter, which causes the light beams to travel around a closed path in opposing directions. An interference pattern is detected at the detector [15]. When the path undergoes rotation, the light traveling in the direction of rotation has farther to travel to reach the detector than the light traveling against the rotation. This causes the two light beams to be out of phase at the detector causing an interference pattern. The interference is proportional to the rate of the path's rotation.

One sensor based on the Sagnac effect is the Fiber Optic Gyro (FOG), first invented in the 1960s. A fiber-optic coil, which can vary in length from meters to kilometers, acts as the closed path for the light. The light source is generally a broadband light source, and the detector is a photodetector [7]. The longer the fiber-optic coil, the more accurate the sensor. However, additional length adds weight and size. Recent advances have made the FOG a popular choice as an IMU sensor. Submarines require extremely accurate gyros and FOG technology has just become advanced enough to be used for such applications. The Navy is looking to use an interferometric fiber optic gyro (IFOG) to replace an electrostatically supported gyro (ESG), a mechanical gyro that is used for navigation on some submarines [17]. Strapdown IMUs are unable to meet the precision requirements for submarine navigation applications, and so the more expensive class of platform IMUs must be used.

Advances in FOGs are allowing them replace the ring laser gyro (RLG), another Sagnac effect sensor. Unlike FOGs, RLGs do not require a light source. Rather, the interferometer is modified to become a resonator by adding an active laser medium in the closed path [15]. Laser waves are created without a light source since the ring is designed such that the number of wavelengths in the device is an integer, and the gain of the amplifying medium is greater than the losses of the device. When the ring undergoes rotation, the two laser beams in the device become out of phase. The RLG has several advantages over other gyros: digital output, high sensitivity, insensitivity to accelerations, and a long lifetime. However, RLGs suffer from a problem called lock-in. Lock-in occurs when, at low rotational rates, the frequency of the opposing laser beams in a RLG tend to lock-in at the same frequency, preventing the sensor from measuring

rotation. Correcting lock-in adds additional weight and expense to RLGs. Honeywell produces RLGs which are several inches in diameter and weigh over a pound.

Surface acoustic wave (SAW) accelerometers are based on principles closely related to the Sagnac effect. In SAW accelerometers, acoustic waves of about 100 MHz are applied to the surface of a hinge connected to a proof mass [22]. The waves travel on either side of the hinge, which can be deflected by an acceleration. The deflection causes the waves on one side of the hinge to have a longer distance to travel than the waves on the other side of the hinge. After the waves have traveled a predetermined distance, a transducer compares the phases of the acoustic waves. The phase difference is proportional to the amount of acceleration experienced by the accelerometer.

2.1.3 MEMS

MEMS (micro-electro-mechanical systems) are devices that have static or movable components that have dimensions on the level of a micrometer [30]. Although MEMS devices were first invented in the 1960s, they did not become widely used until the 1990s. MEMS technology springboarded from IC technology. Unlike IC technology, a diversity of materials can be used as substrates for MEMS devices. MEMS devices can be grouped roughly into four categories: structures, sensors, actuators, and systems.

MEMS gyros operate on the basis of the Coriolis effect, the deflection of an object due to rotation. Many different configurations of MEMS gyros exist. For example, one MEMS gyro configuration as shown in Figure 2.1 has sensing structures connected to frames that are driven to resonance. The resonance creates the velocity necessary for a Coriolis force to exist. When the structures undergo rotation, capacitive structures

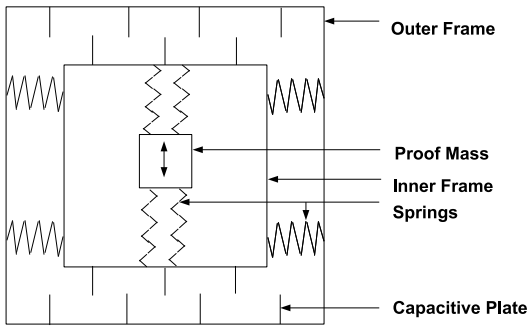


Figure 2.1: MEMS Gyro Diagram

sense deflections due to the Coriolis force and produce an electric output that represents rotational rate [5].

Several types of MEMS accelerometers exist, but two types in particular are well suited IMUs: piezoresistive and capacitive sensors. In piezoresistive accelerometers a small mass is connected to a frame. Acceleration causes the mass to move within the frame, which causes the resistance of the frame to change. The change in resistance is then used to measure the acceleration experienced by the device [24]. In capacitive accelerometers, a structure is suspended above a silicon substrate with springs. A differential capacitor that has plates connected to the structure are used to measure the structure's deflection when it experiences acceleration. The deflection causes the capacitance of the capacitor to change. The change in capacitance is proportional to the acceleration experienced by the device [4].

2.2 IMU Survey

IMUs are usually application specific. For instance, the military generally requires highly accurate IMUs that can endure extreme conditions. IMUs for recreational applications such as RC aircraft do not require the same degree of accuracy but have stricter weight restrictions. In this section we discuss various IMUs currently available and their applications. First, IMUs used by the military are discussed followed by commercial IMUs. Finally, IMUs that are specifically designed for integration with GPS are evaluated.

2.2.1 Military IMUs

A large portion of IMUs are developed for military use. Military grade IMUs are used to aid in navigation of rockets, missiles, aircraft, and other weapons. The military is showing interest in the development of smaller, lower cost IMUs. Thus, many advances in IMUs have been for tactical grade IMUs. Tactical grade IMUs can weigh several pounds since most military platforms can support the weight. In addition, tactical grade IMUs measure high rotational rates and rapid accelerations. Tactical grade IMUs also require a high level of accuracy since weapons are required to make precision strikes, and aircraft need precise navigational measurements.

Gun-Hard IMUs

Currently, the military is interested in the production of gun-hard IMUs. A gun-hard IMU is a unit that can survive the extreme accelerations (up to 20,000 g) and vibrations of a projectile launched from a gun. AlliedSignal Precision is developing a

gun-hard IMU [23]. The goal of the project is a low-cost, miniature IMU that can survive accelerations of greater than 12,000 g. To achieve their goal, AlliedSignal Precision uses a MEMS accelerometer, the Endevco 7951, which is gun-hard, can measure up to 100 g accelerations, and costs between $500 and $600. Rather than a MEMS gyro, the AlliedSignal Precision IMU uses an open loop fiber optic gyro (FOG). A FOG is chosen because it does not require any moving mechanical parts, which makes it less sensitive to vibration than other types of gyros.

BAE SYSTEMS is developing a series of gun-hard IMUs for multiple applications [34]. The SiIMU02 is the second generation of a commercially available IMU that incorporates MEMS technology. All components of the SiIMU02 use digital electronics unlike the first generation which uses analog components. The switch to digital circuitry makes the IMU less sensitive to noise associated with high vibration environments. The IMU is designed for versatility. It can be programmed as a gun-hard IMU for guided munitions, as a higher-accuracy unit for short range guided weapons and UAVs, or as a 3-axis rate pack for systems that do not require linear acceleration measurements. The SiIMU02 uses gyros that are designed by BAE SYSTEMS and off-the-shelf MEMS accelerometers. Since the IMU is designed for high velocity situations, the unit can measure roll rates of over 15,000 °/s and accelerations of up to 100 g while surviving launch accelerations many times greater. The unit is also small with a volume of less than 4 in^3 and has a mass of less than 200 g.

In addition to BAE SYSTEMS's development of a gun-hard IMU, a three-phase program has been initiated to develop a gun-hard IMU [35]. The program's ultimate goal is to create an IMU that can survive a 20,000 g gun launch. In addition the IMU has the performance requirements of a 1 °/hr gyro resolution and a 1/2 mg accelerometer

resolution. The IMU is to be developed in three phases over the course of several years with the final product having a volume of less than 2 in^3 and costing less than $1200 to manufacture. A second goal of the program is to produce an IMU that can be deeply coupled with a Selective Availability and Anti-Spoofing Module (SAASM) GPS military receiver with anti-jamming capability. The military desires the product to be small and light-weight to make the unit compatible with as many devices as possible. Price is also a major issue since most projectile's IMUs are destroyed when the projectile reaches its target.

General Purpose IMUs

The military's interest in IMUs in not limited to gun-hard units; multi-functional units are also of interest. Honeywell produces several different IMUs used by the military. The HG1700 is an IMU that is hailed as low-cost and miniature [29]. The unit, first introduced in 1994, is comprised of tactical grade ring laser gyros and linear accelerometers. Weighing less than 1.95 lb., the IMU has a volume of 25 in^3 and a price of $10,000. The unit is designed to be versatile with applications ranging from supersonic targets to test ship protection systems to high altitude UAVs to Naval missiles.

AlliedSignals has developed a MEMS IMU as an initial product in their μSCIRAS line [18]. The IMU, which uses only three silicon sensors, is intended for applications such as guided artillery shells, remotely piloted vehicles, dismounted soldier location devices, and technology insertion to decrease missile costs. The IMU weighs less than 5 oz. and is contained in a 2 in^3 package. In addition, the unit requires less than 0.8 W of power. Since one of the intended uses of the IMU is munitions, the accelerometers can

withstand inputs of up to 20,000 g. Processing is accomplished with an Intel 25 MHz 80960 microprocessor while an FPGA handles system timing and control.

2.2.2 Commercial IMUs

IMUs have commercial as well as military applications. Commercial IMU applications include autopilots for airlines and instrumentation to measure a robot's position and attitude information [10],[25]. IMUs are also used in recreational products. RC controlled airplanes and helicopters can be equipped with autopilots that include IMUs along with various other sensors. IMUs are even being developed for use with wireless communication networks [9].

Ohio University's Avionics Engineering Center is beginning to develop autonomous vehicles. Brumby, a small airplane with a payload of 17.5 lb, has been equipped with an IMU in an effort to make the vehicle autonomous. The generous payload allows off-the-shelf IMUs to be used: the Crossbow IMU300CB and American GNC coremicro IMU [10]. The IMU300CB costs $3500 and measures rotational rates of up to 100 °/s and accelerations of up to 2 g [12]. The coremicro IMU is only available in conjunction with other American GNC products. The unit operates at 8-12 V and consumes 11 W of power [2]. Altitude is measured using a Honeywell Precision Barometer. The aircraft is also equipped with GPS, which future research will integrate with the IMU. Altitude is measured using a Honeywell Precision Barometer.

IMUs are also of interest in robotics applications. An IMU called the MotionPak is currently being developed for industrial, commercial, and aerospace applications with an emphasis on robotics [25]. The unit is powered by a 15 VDC voltage source, and its sensors are three solid-state gyros and three servo accelerometers. The gyro and

accelerometer sensitivities are determined by the application for which the MotionPak is used. The MotionPak can accommodate gyros with rates of 50, 100, 200, and 500 °/s and accelerometers which measure 1, 2, 5, and 10 g. The gyros output ±2.5 VDC, and the accelerometers output ±7.5 VDC. The MotionPak weighs only 2 lb.

Sensor networks can also benefit from IMU data. Research is underway to develop an IMU for an autonomous wireless sensor network [9]. Each node in a wireless network consists of a layered 25 mm square unit. One layer is an RF transceiver with a microcontroller while another layer is the IMU. Besides accelerometers and gyros, the IMU includes a 3-axis magnetometer to compensate for sensor bias. The IMU is mounted on a 25 mm square motherboard with four orthogonally connected daughter boards. Two 2-axis ADXL202E accelerometers by Analog Devices detect accelerations. Three single-axis Analog Devices ADXLRS150 gyros measure rotational rates. The magnetometer consists of two 2-axis HMC1052L magnetic sensors made by Honeywell.

2.2.3 GPS Integration

There is much interest in combining IMU and GPS data to get reliable solutions over long periods of time. IMUs have a fast update rate but are subject to errors from bias and drift. Since position information is obtained by integrating the IMUs readings, the errors accumulate quickly. Position estimates from IMUs are reliable only for short periods of time. On the other hand, GPS measurements are highly accurate, but GPS has a slow update rate compared to that of IMUs. In addition, GPS data can become unavailable due to jamming, satellites in poor configurations, or the signal simply being blocked by objects such as buildings or trees. Combining GPS and IMU data eliminates the problems each method experiences. The IMU is used to get position estimation in

between GPS updates. The GPS updates are then used to correct the errors in the IMU estimates. The IMU continues to give estimates in the event that GPS becomes unavailable.

IMUs are being developed that are specifically intended to integrate with GPS. Researchers are developing an attitude and heading reference system (AHRS) to aid in the development of a GPS/IMU prototype [16]. The AHRS consists of an IMU, magnetic sensor, and microcontroller. The IMU consists of two dynamically tuned-rotor gyros and three single-axis tuned-rotor accelerometers. The AHRS measures 368 x 124 x 194 mm and weighs over 13 lb. The unit is designed for high-g applications; it can measure accelerations of ± 5 g and body rates of ± 130 °/s. The GPS/IMU prototype is being developed based on the results of the AHRS.

Some manufacturers of IMUs only sell their products in conjunction with GPS units. The American GNC coremicro AHRS/INS/GPS Integration Unit contains an IMU as well as a GPS receiver [2]. The unit weighs 1 lb. with dimensions 5.65" W x 3.35" L x 1.6" H. The unit measures position with an accuracy of about 3 m and velocity with an accuracy of about 0.2 m/s. Crossbow produces a unit called μNAV navigation servo and control board that integrates an IMU and GPS [13]. Intended as an autopilot for RC aircraft, the unit includes accelerometers, gyros, magnetometers, and a GPS receiver. The unit outputs commands to servos using PPM (pulse position modulation) signals. The unit measures angular rates of ± 150 °/s and accelerations of ± 2 g. The unit weighs 33 g but costs $1,495.00.

The military is also interested in integrating IMUs with GPS. Rockwell International has developed a digital quartz IMU for tactical weapons specifically designed to work in conjunction with GPS [20]. Both the gyros and accelerometers are batch-fabricated

quartz sensors. The gyros are of quartz tuning fork type, and the accelerometers are of vibrating quartz type. The unit can measure rates of up to 1000 °/s and forces of up to 70 g. The IMU, which costs $5000, is intended for applications such as missiles, aircraft, and guided munitions. Furthermore, Northrop developed a FOG for a low-cost IMU for tactical weapons applications [31]. The IMU is integrated with a GPS receiver.

2.2.4 Future Developments

The design of an IMU on a single chip is currently being investigated [11]. Cardarelli proposes an IMU consisting of three planar gyroscopes and three planar dynamically tuned accelerometers on a chip that is 1 cm x 1 cm. The chip size includes all electronics such as phase lock loops and pick-offs to drive the sensors. Cardarelli states that all but a z-axis gyro have been developed to accomplish the design. The ultimate goal of the project is to design a tactical grade IMU with gyros capable of measuring rates of ±1000 °/s and accelerometers that can measure accelerations in the range of ±50 g.

Archangel is in the process of developing an IMU based on MEMS technology [6]. The unit, called the IM^3-1, is a military grade IMU that overcomes many of the inaccuracies inherent in solid-state sensors. The IM^3 includes processing in its 3/4 in x 3/4 in x 3/4 in package. Weighing only 10.5 g, the IMU can be powered in the range of 8-24 VDC. The unit is to give a body rate resolution of 0.0006 °/s at 200 Hz and acceleration resolutions of 50 μg also at 200 Hz. The unit is still in the prototyping stage.

2.3 Blimp Navigation

The technology to allow small blimps to become autonomous is a recent occurrence. Indoor blimps especially have small payload capacity, so light-weight instrumentation is

required. Before the advent of MEMS technology, most inertial sensors were too heavy to be used in with small blimp. Even with sensors becoming lighter, IMUs designed for ultra light-weight craft have only begun to come in demand with recent interest in micro UAVs. The micro UAVs for which IMUs have been designed are capable of flights much faster than blimps. Thus, few off-the-shelf IMUs exist which are suitable for autonomous blimps. An exception is a small off-the-shelf autopilot used by JPL in the development of a small autonomous blimp [27]. The autopilot weighs only 170 g and is equipped with GPS capabilities and ultrasound.

While JPL's blimp uses an IMU, most other indoor autonomous blimp projects do not use an IMU. Rather these projects rely on vision information to derive position and attitude [14],[36]. The Swiss Federal Institute of Technology in Lausanne (EPFL) is developing several autonomous indoor vehicles, among them a blimp [38]. The blimp, weighing only 100 g, is equipped with CMOS cameras, a controller board, and a communication board that uses Bluetooth. The controller board contains motor drivers and a PIC18F452 microcontroller. The GRASP Laboratory of the University of Pennsylvania is also developing an indoor blimp that navigates with vision [37]. Rather than an IMU, the 8 foot blimp with a 1.1 lb payload capacity is equipped with a monocular camera, control circuitry, and wireless communication equipment.

Large scale outdoor autonomous blimps are also being developed. The LAAS/CNRS is developing an autonomous blimp to be flown outdoors [21]. The blimp, an AS-500 by Airspeed Airship, is 7.8 m long with a volume of 15.0 m^3 and has a payload capacity of 11.0 lb. Since an outdoor blimp has a large payload capacity, more instrumentation is available for it than for an indoor blimp. The AS-500 is equipped with a fluxgate compass to measure roll rates, DGPS (differential GPS), and, since one of the project's

goal is terrain mapping, stereovision rather than an IMU. The fluxgate compass is the only instrument used that performs any inertial sensing.

One of the first autonomous blimp projects is the AURORA project [33]. The AURORA project is developing an outdoor AS800 blimp by Airspeed Airships for applications such as terrain mapping and surveillance. The blimp is 9 m long and has a 24 m^3 volume. The on-board processor is a PC104 card with a Pentium processor, six serial ports, a parallel port, and an Ethernet network interface. The instrumentation includes a GPS receiver, inclinometer, compass, 3-axis Crossbow CXL01M3 accelerometer, an Analog Devices ADXL05JH accelerometer, wind sensor, and a color CCD camera. The blimp does not include an IMU in the sense of a unit of accelerometers and gyros.

2.4 Technological Gaps

Most IMU applications have been for vehicles that do not have the strict payload capacity restrictions of an indoor blimp. Ground vehicles, jets, and missiles all have payload capacities much heavier than that available for this project. The general need for ultra-small, light-weight IMUs has been a recent occurrence.

The few light-weight IMUs available are inadequate for an indoor blimp. The inadequacies range from a unit's dependence on GPS to a design based on high speed applications. The existing technology that is suitable for developing a light-weight, low-g IMU suffers from accuracy problems.

2.4.1 Current Technology

Many IMUs currently being developed are designed to be integrated with GPS. However, GPS is not available in many environments: indoors, urban areas, forests, and

areas with jamming. Since the blimp for this research operates indoors, GPS is not available. Therefore, any IMU used must not be dependent on GPS. Since the advent of ultra light-weight IMUs is recent, most are designed for operation in conjunction with GPS. This is one of the facts that necessitated the development of a new IMU for an indoor blimp.

Military grade IMUs are inappropriate for an indoor blimp. The first prohibitive factor is the weight of such IMUs. While many manufacturers of tactical grade IMUs claim their units are light-weight, light-weight is a relative term. For a missile a pound may be light-weight, but for a blimp with a payload capacity of only four pounds a one-pound IMU is excessively heavy. In addition, IMUs for military applications are designed for the high-g environments fighter aircraft and projectiles experience. The IMU developed in this research is specifically for a low-g environment. The harsh environments in which tactical grade IMUs operate require special design considerations driving up cost. Like light-weight, the term low-cost is relative. While a low-cost tactical grade IMU may be $10,000, this is prohibitively expensive for an indoor blimp. The IMU developed in this research costs only a few hundred dollars rather than a few thousand dollars.

IMUs for commercial use are too application specific to be useful for an indoor blimp. In addition to being too expensive, IMUs designed for RC airplanes are calibrated for high speeds that are unreachable for a blimp. The MotionPak requires too much electrical power to be useful for a blimp. A power supply is the main source of weight for an IMU, and the greater the power required, the heavier the power supply. While the IMU developed for the sensor network described in [9] neither weighs too much nor requires too much power, it is too application specific to be used for a blimp. The IMU is specifically

designed to operate with a field programmable gate array (FPGA) and other hardware that is not necessary for a navigational IMU. However, the sensor network IMU does use gyros and accelerometers that are similar the IMU developed in this research.

The ideal IMU for an indoor blimp application is yet to be developed. Technologies other than MEMS are not suitable for an indoor blimp. Mechanical systems are immediately eliminated as a possibility due to their bulk. Other sensors such as FOGs and RLGs can be found that just fall within the blimp's payload restrictions, but these sensors are too expensive. MEMS sensors are both affordable and extremely light-weight. The best MEMS IMU for an indoor blimp would be a single chip with both accelerometers and gyros. This chip would measure in all three degrees of freedom. But this chip does not currently exist. Since MEMS technology is the only suitable choice for an indoor blimp's IMU, the IMU must consist of sensors on several chips rather than on a single chip.

2.4.2 Accuracy

All IMUs are subject to bias and drift errors. Some sensors are less prone to these errors than others. Mechanical gyros and accelerometers have been used for high precision navigation applications such as submarines, missiles, and aircraft. Since mechanical sensors are the oldest inertial sensing technology, they tend to be some of the most accurate sensors available. High precision sensors require precision manufacturing and calibration, which makes them expensive.

MEMS technology has drastically reduced the size of sensors but has yet to reach the accuracy levels of older technologies. While high precision IMUs based on mechanical technology can operate for minutes before readings become too unreliable, low-cost MEMS IMU readings become unreliable after a few seconds. However, MEMS sensors

are the only practical choice for an application with extremely small payload capacities. The errors associated with MEMS sensors are, therefore, unavoidable, and other methods must be employed to correct the errors.

Chapter 3

Instrumentation

We address in this chapter the selection of commercial off-the-shelf sensors for the design of a light-weight IMU, and we discuss their advantages and disadvantages. We also discuss the integration of selected sensors to form a low-cost, light-weight, low-g, IMU. The components of our IMU system include a microprocessor, accelerometers, and ultrasonic range finders.

The remainder of this chapter is organized as follows. In Section 3.1, we present and evaluate various microprocessors suitable for a light-weight IMU. Next, in Section 3.2 we present a similar discussion in terms of the selection of a gyroscope. We evaluate available accelerometers and discuss the accelerometer chosen for the IMU in Section 3.3. We evaluate various options for altimeters in Section 3.4. In Section 3.5 we present methods of obstacle detection focusing on ultrasonic sensors. In addition, we discuss the principles of ultrasonic sensor operation and a method of communication with ultrasonic sensors. We address power considerations for the IMU in Section 3.6. In Section 3.7 we present the physical parameters of the designed IMU. Finally, in Section 3.8 we address software design for the IMU including programming language, communications, and data acquisition and formatting.

3.1 CPU

Several features are necessary for a microcontroller intended for use in a light-weight IMU. Suitable microcontrollers must be in a light-weight package and must have low

power requirements. Multiple I/O lines and AD converters are needed for interfacing with external sensors. Serial ports are needed to relay data to other devices. Numerous microcontrollers exist, but only a few have the necessary features for a light-weight IMU.

National Semiconductor and Atmel manufacture microcontrollers that are suitable for a light-weight IMU. National Semiconductor produces the CP3UB17. This microcontroller offers several communication ports: SPI, UART, USB, and an Advanced Audio Interface. It has up to thirty-seven general purpose I/O pins and requires between 2.5 V and 3.3 V to operate. A C development environment is available for the chip. Atmel manufactures the ATmega8, an 8-bit microprocessor that operates at 16 MHz. The chip features a USART, a two-wire serial interface, and an SPI serial interface. Up to 8 AD converters and 23 I/O lines are available, and the microcontroller is programmable in C.

Rabbit Semiconductors manufactures several microprocessors including the Rabbit 2000 and the Rabbit 3000. The Rabbit 3000 Microprocessor has 56 digital I/O lines that are TTL compatible. The chip can operate at voltages ranging from 1.8 V to 3 V and has six serial ports. The Rabbit 2000 Microprocessor is an 8-bit microprocessor with 40 serial ports. It has 40 I/O lines and six timers. Both the Rabbit 2000 and 3000 have a C development environment.

Zilog manufactures the Z8F2421 and the Z8F042A microcontrollers. Both chips operate at 20 MHz, have eight AD converters, and have flash program memory. The Z8F2421 has SPI, IIC, and two UART serial ports while the Z8F042A only has a UART port. The Z8F2421 has up to 31 I/O ports in a 40- or 44-pin package. The Z8F042A has a 20- and 28- pin package.

Microchip offers a variety of microcontrollers that are suited for a low-g, light-weight IMU. In the PIC16F series, Microchip offers the PIC16F77, PIC16F737, PIC16F777,

PIC16F873, and PIC16F877A, and in the PIC18F series are the PIC18F4320, PIC-18F452, and PIC18F6520. The PIC16F series can operate at up to 20 MHz while the PIC18F series operates at up to 40 MHz. Each chip has several AD converters, I/O ports, and serial ports. Package sizes range from 28-pin to 40-pin. All PICs are low power.

We present in this section a survey of several microcontrollers that may be considered in the design of a low cost, lightweight, low-g IMU. The processors considered are an OOPIC-R (Section 3.1.1) and a PIC18F4320 (Section 3.1.2). The PIC18F4320 is found to be better suited than the OOPIC for an IMU microprocessor.

3.1.1 OOPIC

We first discuss the OOPIC-R microcontroller, an object-oriented microcontroller produced by Savage Innovations. The OOPIC-R is a unit that features a serial port, thirty-one I/O port connectors, and power ports. The OOPIC-R contains a removable Microchip EEPROM in a DIP package and is powered by a singe 9 V battery. The unit is light-weight, and the addition of a single battery can supply power for multiple external sensors. The numerous I/O ports are also ideal for interfacing sensors such as accelerometers and gyros. Several of the I/O ports feature AD converters, which are necessary for sensors with analog outputs. Connection to a serial port is simple with an adapter mounted to the OOPIC circuit board.

The OOPIC-R has several qualities that are desirable for the microprocessor in a light-weight IMU. Since the OOPIC is object-oriented, programming is an easier task than with other microcontrollers. A free integrated development environment (IDE) is available for the OOPIC. The OOPIC compiler allows software to be written in Basic,

C, or Java syntax. The software contains many predefined objects to make certain programming tasks simple. For example, an object called oSerial allows the user to interface the OOPIC, using only a few lines of code, with a PC's serial port. Predefined objects are available for common sensors such as for the Devantech SRF04 Ultrasonic Range Finder. The OOPIC software has an object for an IIC (Inter-Integrated Circuit) bus, a common bus for many sensors [32].

The OOPIC has many desirable features, but it is ultimately unsuited for use as the IMU's microcontroller. The connection of multiple external sensors to the IIC bus requires an additional voltage source. The OOPIC's packaging is bulkier than needed for a light-weight IMU. Product instability sometimes causes the OOPIC to become unresponsive, and the unit's EEPROM must be manually removed to fix the problem.

3.1.2 Microchip 18F4320

The PIC18F4320 is a low cost, low-power microcontroller manufactured by Microchip [26]. In a 40-pin DIP package, this chip features flash program memory and a clock speed of up to 40 MHz. It can operate at voltage levels ranging from 2.0 V to 5.5 V. The chip has 36 I/O lines and thirteen, 10-bit AD converters. A USART module and IIC bus perform serial communications. The PIC18F4320 is chosen for the IMU because of its low power requirements, numerous I/O lines and AD converters, and its serial communication ports.

The microcontroller's clock speed used is a critical consideration for the IMU. The PIC can function with clock speeds up to 40 MHz by supplying an external clock, but 40 MHz is determined to be too fast. At 40 MHz the PIC's internal timers cycle too quickly for sensors that operate slowly compared to the clock speed. A 20 MHz clock

speed, provided by a crystal oscillator, is chosen to balance the need for processing speed and sensor requirements.

To integrate the PIC18F4320 with other hardware, I/O port pins must be configured properly and pin functions must be verified. The first step in configuration is to choose appropriate I/O lines, which depends on whether the hardware to be interfaced has digital or analog outputs. The PIC has 13 I/O lines, labeled AN0-AN12, that are software configurable as either digital or analog input pins. Successive pins must be configured as the same type of I/O line so for the IMU lower numbered I/O lines are for analog inputs, and higher numbered I/O lines are for digital inputs. Since most pins on the microcontroller are multiplexed to have several functions, configuring one pin can often have unintended effects on another pin. Verifying that each pin operates as intended is necessary.

The PIC18F4320 features a Master Synchronous Serial Port (MSSP) module that is configurable to operate as an IIC bus. The MSSP module supports both the IIC bus's master and slave functions; however, only the master mode is required for the IMU. The SCL (serial clock) and SDA (serial data) lines of the IIC bus are multiplexed with two of the PIC's general I/O lines. With the I/O lines configured properly in software and the addition of pull-up resistors, the SDA and SCL lines become an IIC bus. The IIC bus is used to interface with the IMU's four Devantech SRF08 ultrasonic sensors.

Relaying information from the IMU to a PC or other device is accomplished through serial communications by combining a MAX232 with the PIC18F4320's USART (universal synchronous asynchronous receiver transmitter). The USART is configured to operate as an RS-232 port but has TTL output levels (voltages ranging from 0 V to 5 V). Since RS-232 levels range from -15 V to +15 V, the PIC's serial transmit and receive

lines as is are incompatible with a PC. To make a standard serial port and the PIC's USART module compatible, a MAX232 is used. The MAX232 is a 16-pin chip, requiring only four external capacitors, that converts voltages between RS-232 and TTL levels.

While chosen to be the microcontroller for IMU, the PIC18F4320 has some drawbacks. In order to program the microcontroller, the chip must be in the bulky DIP package rather than a surface mount package. The PIC18F4320 is also more difficult to program than the OOPIC. The object-oriented nature of the OOPIC makes operations such as communication through a serial port simple. Serial communications with the PIC18F4320 require considerable programming.

3.2 Gyroscopes

There are many factors for determining whether a gyro is suitable for a light-weight, low-g IMU. The gyro must be low power, weigh as little as possible, and be able to sense low rotational rates. The gyro must require as few external components as possible. Since one goal is to sense roll, pitch, and yaw, a triaxial gyro is desirable. However, due to the lack of demand for low-g IMUs, a low-rate triaxial gyro suitable for this application is not known to exist.

Analog Devices produces the ADXRS150, a low-rate (± 150 °/s), single-axis MEMS gyro. The gyro weighs less than half a gram, has a volume of less than 0.15 mm^3, and requires a 5 V source. For ease of testing, Analog Devices manufactures the ADXRS150 on an evaluation board, the ADRXS150EB [3]. The evaluation board contains all necessary external capacitors on a DIP package. The bandwidth of the evaluation board is set to 40 Hz, but it can be altered by adding an external capacitor which reduces the bandwidth and improves the signal to noise ratio. Although the evaluation board adds

additional weight to the IMU, it is chosen because of ease of implementation and low cost.

The ADXRS150 has linear output characteristics that are affected by the addition of external components. A rotational rate of -150 °/s or less produces an output of 0.25 V which increases linearly with clockwise motion until a rate of 150 °/s or greater is reached. This produces an output of about 4.75 V. While the linear relationship between input and output is unaffected by external components, the sensor's bandwidth can be altered with the addition of capacitors. The gyro's -3 dB frequency is given by

$$f_{out} = \frac{1}{2\pi R_{OUT} C_{OUT}} \tag{3.1}$$

where C_{OUT} is chosen by the designer and R_{OUT} is set by the manufacturer to be 180 kΩ. Adding external resistors affects the effective value of R_{OUT}, increases the measurement range, and adjusts the null setting.

3.3 Accelerometers

In this section, we present an overview of accelerometers considered for the design of a low-g, light-weight IMU. We discuss the ACH-04-08-05 in Section 3.3.1. In Section 3.3.2 we discuss an Analog Devices accelerometer and design choices to make the accelerometer suitable for the IMU.

3.3.1 MSI ACH-04-08-05

The ACH-04-08-05 is a triaxial capacitive accelerometer manufactured by MSI [28]. The chip weighs only 0.35 g, can operate on voltages ranging from 3-40 V, and has a

maximum power dissipation of 100 mW. Connecting external components adjusts the accelerometer's sensitivity to acceleration. The sensor also features a linear relationship between output and acceleration.

While the ACH-04-08-05 has many desirable features, it has too many drawbacks to be used in the final IMU design. Although the chip is light-weight, the additional external components needed for operation add extra weight and unnecessary complexity to the design of an IMU. Each of the chip's three outputs require an op-amp, several resistors, and capacitors. The accelerometer requires another op-amp circuit to generate a reference voltage. Even with the addition of high gain circuits on each axis of the accelerometer, the ACH-04-08-05 still lacks the sensitivity for the blimp's low-g accelerations.

3.3.2 Analog Devices ADXL213

The ADXL213 is a ±1.2 g dual axis capacitive accelerometer manufactured by Analog Devices with many features that make it our choice for a light-weight, low-g IMU [4]. Although the blimp cannot reach accelerations with a magnitude of 1.2 g, the accelerometer still provides enough sensitivity to give measurements in the desired range. The sensor requires minimal external components: three capacitors and one resistor, which adjust the sensor's sensitivity. The accelerometer has a light-weight, small package that measures 5 mm x 5 mm x 2 mm. The sensor's outputs are in pulse width modulated (PWM) form making it easy to interface with a microcontroller.

The ADXL213 does have drawbacks for the blimp application. No 3-axis, low-g accelerometer could be found that is suitable for this project. Since the ADXL213 is dual axis, two accelerometers are necessary to measure acceleration in three degrees

of freedom. This leaves one axis of one accelerometer unimplemented, which means extra weight. An ideal accelerometer for this application has a smaller measurement range. However, since low-g navigational applications are unusual, manufacturers do not produce such accelerometers.

The ADXL213 has an adjustable bandwidth, which controls the sensor's noise characteristics. The equation

$$F_{-3dB} = 5\mu F / C_{(x,y)} \tag{3.2}$$

where $C_{(x,y)}$ is the value of the capacitor at the x- and y-axes outputs, governs the adjustability of the accelerometer's -3 dB bandwidth. Reducing the bandwidth reduces the effects of noise on sensor readings. Since the blimp's acceleration cannot vary greatly over a short time, a bandwidth of 1 Hz is chosen for the IMU's accelerometers. Using Equation (3.2) a capacitor value of 5 μF gives the desired bandwidth. Since this value is not a standard capacitor value, 4.7 μF capacitors is used instead. The relationship between bandwidth and noise is

$$rmsNoise = (160\mu g/\sqrt{Hz}) \times (\sqrt{BW \times 1.6}) \tag{3.3}$$

For a bandwidth of 1 Hz, the sensor has the RMS noise value 0.2024 mg.

The accelerometer's bandwidth places restrictions on PWM frequency. Due to the effects of aliasing, the PWM frequency must be at least five times greater than the bandwidth. The period of the accelerometer's PWM output, $t2$, is given by

$$t2 = R_{SET}/125M\Omega \tag{3.4}$$

where R_{SET} is a resistor value. $t2$ must be less than 0.2 s (a PWM frequency of 5Hz); thus, R_{SET} must be at least 25 MΩ. A frequency of 5 Hz does not take advantage of the microprocessor's speed. For this design a frequency of 83.33 Hz ($R_{SET} = 1.5\ M\Omega$) is chosen because the frequency is slow enough for the microprocessor to get an accurate reading of the high time of the PWM signal but fast enough to be comparable to the speed of the other sensors on the IMU. Knowing the PWM signal's frequency allows the accelerometer's output to be interpreted by applying the formula

$$acceleration = ((t1/t2) - Zero\ g\ Bias)/Sensitivity \tag{3.5}$$

where $t1$ is the positive pulse width of the sensor output, $t2$ is the period of the sensor output, $Zero\ g\ Bias = 50\%\ nominal$, and $Sensitivity = 30\%/g\ nominal$. At 0 g the output is a PWM waveform with a 50% duty cycle.

3.4 Altimeter

In this section we present various products that can measure altitude including hand-held laser range finders, barometric pressure sensors, and ultrasonic sensors. Each of these sensors can be used as an altimeter either as is or with a few modifications. We evaluate the sensors as they pertain to a light-weight, low-g IMU.

Small laser range finders, for applications such as golf and hunting, can be converted to an altimeter. The AccuRange 4000 Sensor (a golf scope) measures distances ranging from 0-50 ft and weighs 22 oz. This model has the complexity of an RS-232 connection. A similar product is the LaserCom 200, which has a range of up to 300 m. This unit weighs 600 g, which is a considerable percentage of the blimp's payload capacity.

Interest in areas such as micro UAVs and model rocketry has spurred the development of small altimeters. MRA[1] produces small altimeters specifically for small UAVs. The Mk V is an altimeter that measures ranges from 20 cm to 100 m with an accuracy of 2 cm. While the unit measures ranges appropriate for an indoor blimp, the unit weighs 400 g and requires 12 V to operate. Many model rockets employ light-weight altimeters, but these altimeters only measure peak altitude. The blimp project requires continuous measurements.

One application of barometric pressure sensors is altimeters. Honeywell produces the AM-250 Barometric Altimeter, which has a large range and is highly accurate. This unit, however, is expensive and weighs approximately 3 lb. The unit reports altitude on a digital display making interfacing difficult. Intersema produces the MS5534 series of barometric pressure sensors. The MS5534B is a lightweight chip, requiring only 3 V and typically drawing 0.6 mA, that measures 9.15 mm x 9.15 mm. The sensor has a 0.1 mbar accuracy for a pressure reading which results in about a 1 m resolution in altitude. A three-wire interface connects the barometric pressure sensor to a microcontroller. However, the MS5534B proves to be difficult to mount to a circuit board without damaging the sensor.

The Devantech SRF08 is an ultrasonic sensor that can be used as an altimeter for low altitude flight. Because of its reliability and light weight, the SRF08 is chosen as the IMU's altimeter. A more detailed discussion of the Devantech SRF08 follows in Section 3.5.

[1] http://www.roke.co.uk/aviation/miniature_radar_altimeter.asp

3.5 Obstacle Detection

In order to navigate the blimp around obstructions, obstacles must be detected. Unless an obstacle is exceedingly sharp, an indoor blimp would suffer no damage from a collision; however, the blimp's attitude and position would change abruptly. The IMU by design is unable to process abrupt changes.

One method of obstacle detection is with laser range finders. Laser range finders measure large ranges accurately but tend to be too heavy and too expensive for a light-weight, low-g IMU. For example, a small unit produced by Laser Optronix called the Minilynx SL 210 M has a range of 50-100 m but weighs 6.6 lb.

Sonar based range finders are another method of obstacle detection. Acroname produces various small sonar units that have smaller ranges than laser range finders but lower power requirements. Two sonar range finders are the Devantech SRF04, which has a range from 3 cm to 3 m, and the SRF08, which has a range from 3 cm to 6 m. Both units weigh only 0.19 lb. Although the SRF04 is half the cost of the SRF08, the SRF08 is preferable because of its larger measuring range. The SRF08 requires 5 V, typically draws 15 mA during a ranging operation, and is packaged with dimensions 43 mm x 20 mm x 17 mm. With its light-weight, low-power characteristics, the Devantech SRF08 ultrasonic sensor complies with the IMU's design requirements.

In this section we make a detailed presentation of the Devantech SRF08's operation. We begin with an overview of the SRF08's operation followed by a description of IIC bus operation as it relates to the SRF08. We conclude the section with a discussion of how to interface multiple sonars on a single IIC bus.

3.5.1 Sonar Operation

The Devantech SRF08 Ultrasonic Range Finder manufactured by Acroname is a sonar-based sensor that measures the distance between itself and another object. The SRF08 emits a sonic pulse and measures the time between signal emission and the detection of the reflected signal to determine the distance of an object. A PIC16F872 microcontroller on the SRF08 stores ranging data, converts time to distance, and interfaces with other devices. The results of a ranging operation can be given in inches, centimeters, or microseconds by writing a specific value to one of the SRF08's internal registers. The microcontroller stores the first seventeen echo recordings in registers which can then be read via an IIC bus. Further information on the operation of the SRF08 is available in [1].

3.5.2 IIC Bus Protocal

The ultrasonic sensors connect to other devices through an Inter-Integrated Circuit (IIC or I^2C) bus. The IIC bus was originally developed by Phillips for television, but the bus has become an industry standard for communicating with devices such as EEPROM, clocks, and sensors [19]. The bus consists of two lines: a serial clock line (SCL) to synchronize the communicating devices and a serial data line (SDA) for transmitting and receiving data. One device acts as the master while the other device is the slave. In the case of the ultrasonic sensor, the SRF08 is always a slave device.

The following describes the operation of an IIC bus (detailed information is available at [32]). The master device can either perform a write to or a read of a slave device. Multiple slaves can exist on a single bus by assigning each slave a unique address. All IIC operations begin by sending a start condition, a unique pulse that identifies the beginning

of an operation and causes all slave devices on a bus to wait for the next signal. Once the start condition is sent, the master transmits the seven bit address of the slave with which it wants to communicate. Only the slave device with the transmitted address responds to further signals from the master. A R/W bit (the most significant bit of the address byte) determines whether the operation is a read or write. Whenever the master is transmitting the address of the slave, the R/W bit is always low. For the SRF08 the master must send the internal register number to which the master wishes to write followed by the data byte to be written. Multiple bytes can be written to a device with each byte sent stored in successive memory addresses. After each byte received, the slave sends an acknowledge signal to the master. Once all data has been transmitted, the master sends a stop sequence to the slave.

Performing a read operation is similar to performing a write operation on the IIC bus. The master first sends a start sequence followed by the desired slave's address with the R/W bit low. For the SRF08 the address of one of the slave's internal registers is next transmitted over the SDA line. The master next sends another start sequence followed by the address of the desired slave with the R/W bit high to signify a read operation. The slave then transmits a data byte to the master. The master sends an acknowledge signal, and the slave transmits the next data byte. When the master has read the last desired byte, the master fails to send an acknowledge signal instead sending the stop sequence.

The SCL line controls the timing between the master and slave. The SDA line transitioning from high to low while the SCL line is high generates a start condition. The SDA line transitioning from low to high while SCL line is high generates a stop condition. During the start and stop conditions is the only time when the SDA line

transitions when the SCL line is high. During data transmission the SDA line changes after the SCL line pulses so data is read while the SCL line is high. All data transmissions are 8-bits followed by an acknowledge bit. Although the master controls the SCL line, the slave can hold the line low during read operations. Called clock stretching, this action prevents the master from reading data from the SDA line before the slave is ready to transmit.

3.5.3 Changing a Sonar's Address

Each SRF08 must be assigned its own address since multiple sensors communicate over one bus. The default address of an SRF08 is E0h. A sensor can have an even address from E0h to FEh; thus, sixteen SRF08s can exist on one bus. The following sequence, which instructs the sonar that its address is about to be changed, must be written to the SRF08's command register (located at address 00h) in order to change the default address: A0h, AAh, A5h. After writing A5h, the desired address, bit-shifted right once, is written to the command register. For example, writing the number 71h to the command register changes the sonar's address to E2h. When the sonar is initially powered, an LED will blink in a specified pattern given in [1] to indicate the sonar's current address.

3.6 Power

Several options are available for supplying power to the IMU. The majority of the IMU's weight comes from the power supply so it is important to balance weight and power. Other than weight, factors considered in choosing a power supply are rechargability and cost. Rechargeable batteries tend to cost more but have the advantage of being

Type	Voltage (V)	Weight (oz.)	Capacity (mAh)	Rechargeable
Alkaline	9.0	5.0	3135	No
Lithium	9.0	4.1	3000	No
Ni-Cad	7.2	5.25	700	Yes
Ni-Cad	9.6	7.0	700	Yes
NiMH	7.2	5.6	1000	Yes
NiMH	9.6	7.4	1000	Yes
Lith-Ion	7.2	6.3	4400	Yes

Table 3.1: Battery Comparison

reusable. In Table 3.1 we summarize the properties of various batteries. Although the lithium battery weighs less than the alkaline, the alkaline battery has a larger capacity. The alkaline battery is also inexpensive. The rechargeable batteries all weigh more than the alkaline battery, are more expensive, and except for the lithium-ion battery, have much less capacity. With these considerations in mind, the alkaline battery is chosen to power the IMU. A 9 V alkaline battery is sufficient to power the IMU. It is found through experiment that a single battery can power the designed IMU, which has an impedance of 25 MΩ, for approximately three hours.

Voltage regulators are used to convert the voltage levels from the batteries to voltage levels usable by the IMU's various sensors. Alkaline batteries produce 9 V, too much for the components of the IMU rated for 5 V. The LM7805C voltage regulator, rated for up to 32 V, outputs a steady 5 V, but this regulator does not take advantage of the batteries power supply. The LP2954 regulator also outputs 5 V, but it continues to output 5 V even with a significant drop in output from the battery. For this reason, the LP2954 is used over the LM7805C.

Part	Cost
18F4320	$7.83
SRF08	$59.50
ADXRS150	$30.00 (x3)
ADXL213	$9.70 (x2)
Total	$176.73

Table 3.2: Summary of IMU Cost

3.7 IMU Board

Once the design of the IMU was complete, a board was designed on which to mount the components. A schematic of the board is shown in Figures 3.1 and 3.2. The board consists of a mother board and two daughter boards. The mother board measures 69 mm x 103 mm while the daughter boards both measure 23 mm x 39 mm. The daughter boards are mounted at ninety-degree angles to the mother board with one mounted to the short side and the other mounted to the long side. The purpose of the daughter boards is to ensure the proper alignment of the accelerometers' and gyros' axes. Without the ultrasonic sensors, the entire board including the battery weighs 131 g. The addition of four ultrasonic sensors adds 80 g to the total weight.

The IMU developed in this research costs little compared to other IMUs currently available. In Table 3.2 we summarize the cost of the IMUs main components. Components such as resistors and capacitors are not included as they add a negligible amount to the overall cost.

3.8 Software

The PIC18F4320 is programmable in two languages, C and assembly, both of which have advantages and disadvantages. C is a portable, high level language while assembly

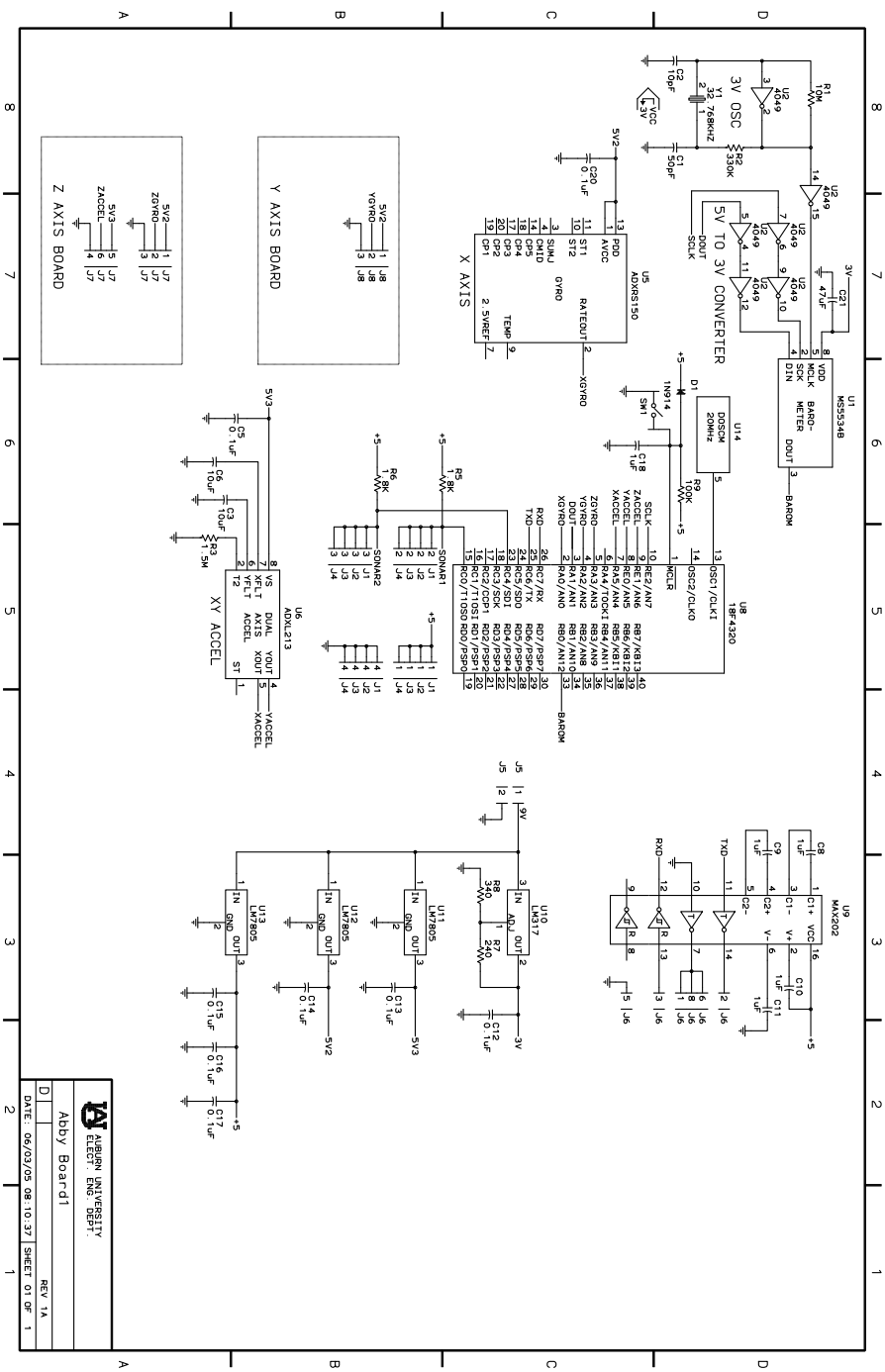


Figure 3.1: Mother Board

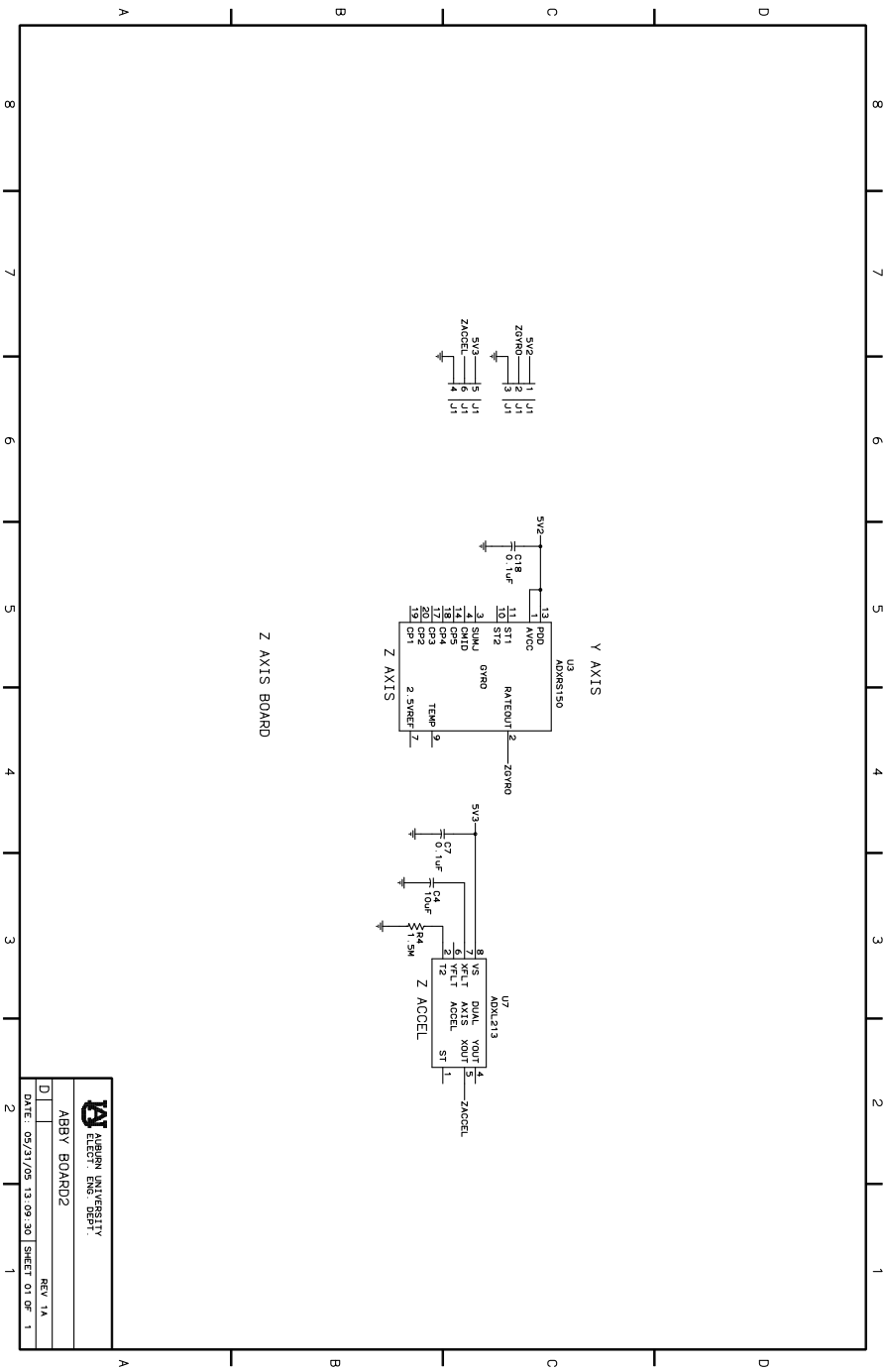


Figure 3.2: Daughter Boards

is as close as possible to machine language without actually having to write ones and zeros. Assembly commands depend specifically on the processor being programmed with instructions varying from processor to processor. Operations such as multiplication and division can be complicated to perform in assembly while in C multiplication and division can be accomplished with a single line. C's efficiency comes at the cost of knowledge of exactly how commands are executed. Free software, called MPLAB, is available for the development of PIC microcontrollers in assembly. Microchip offers the C18 software for programming the PIC18 series in C. Although it is easier to program in C than in assembly, the software for the IMU is written in assembly. Cost considerations are one factor for this choice. While C is an easier language in which to program, assembly programs have more control over the precise timing of operations. The programmer knows exactly how long it takes to execute an instruction in assembly while this knowledge is not readily available in C programs. Since timing is an important consideration for this application, assembly is a better choice.

In this section we present the design process of the IMU's software. In Section 3.8.1 we present software considerations in performing serial communications with the PIC18F4320's USART module. We next explain in Section 3.8.2 the software required to interface gyros with analog outputs to the microcontroller. In Section 3.8.3 we develop the software to interface sensors with digital outputs with the PIC18F4320. We explain the sonar interfacing software in Section 3.8.4, and in Section 3.8.5 we present the method for formatting the IMU data. Finally, in Section 3.8.6 we present the IMU program flow.

3.8.1 USART Communications

The IMU communicates with a PC through the PIC's USART (universal synchronous asynchronous receiver transmitter) module, which has two main registers. The TXSTA register is for USART transmissions and the RCSTA register is for USART receptions. While the IMU never receives data with the USART, the RCSTA register configures the necessary pins as serial port pins rather than ordinary I/O pins. The TXSTA register determines whether the USART is in synchronous or asynchronous mode as well as whether the BAUD rate, a measure of the rate of information flow, is high speed or low speed. In addition, the TXSTA register also enables and disables transmissions.

The PIC18F4320 allows the programmer to decide various properties of data transmission including BAUD rate. For the IMU, asynchronous transmission occurs with a high speed BAUD rate of 9600. A BAUD rate of 9600, a common BAUD rate, makes the IMU easier to connect to such devices as a wireless link. The BAUD rate is set by writing an appropriate value to the SPGRB register. This value x is determined by the equation

$$BAUD\ rate = F_{OSC}/(64(x+1)) \tag{3.6}$$

for low speed, where F_{OSC} is the clock frequency and

$$BAUD\ rate = F_{OSC}/(16(x+1)) \tag{3.7}$$

for high speed. Given that the oscillator frequency is 20 MHz, x for a low speed BAUD rate is 31.55. Since the SPBRG register can only take whole numbers, 32 must be used, which is a source of BAUD rate error. By replacing x with 32 in Equation (3.6), the

actual BAUD rate achieved is 9470. The error is then given by

$$\%error = \frac{|actual - desired|}{desired} \times 100 \tag{3.8}$$

Thus, for a low speed BAUD rate, the error percentage is 1.35%. For a high speed BAUD rate x is 129.2, which rounds to 129. By using Equation (3.7), the actual BAUD rate achieved is 9615, which has an error of 0.15625%. Thus, the high speed BAUD rate is chosen because it reduces errors in the BAUD rate.

The TXREG and TXSTA registers control data transmission through the USART. Setting the data transmission bit in the TXSTA register and loading TXREG with the data to be transmitted begins data transmission. When all of the data in TXREG has been transmitted, a flag bit is set that causes an interrupt if the interrupt is enabled. In the IMU program, the first byte from a data buffer is loaded into the TXREG register to begin data transmission. A routine then polls to see if the transmit is complete. Once the transmit is complete, the next data byte is loaded into TXREG, and the process is repeated until all of the data has been transmitted.

3.8.2 AD Converters

The PIC18F4320 features several I/O lines that are configurable as analog input pins, an essential feature for interfacing the gyros, through control registers. The main register for controller analog-to-digital (AD) conversions is the ADCON0 register. This register allows the programmer to select on which input pin to begin an AD conversion as well as allows the programmer to enable or disable AD conversions. The other register important for AD conversions in the ADCON1 register. This register allows the

programmer to configure the pins with AD converters to be either analog or digital input pins.

The gyros, the only sensors with analog outputs, connect to the pins labeled an AN0, AN2, and AN3, which are equipped with AD converters. It must be ensured in software that the capacitor used for AD conversions has sufficient time to fully charge between each conversion to guarantee accurate sensor readings. An amplifier settling time, capacitor charging time, and a temperature coefficient determine this time, called T_{ACQ}. The amplifier settling time is given as 5 μs in the microcontroller's data sheet [26]. The temperature coefficient is given by the equation

$$T_{COFF} = \frac{temp - 25}{0.05} \tag{3.9}$$

where the temperature is given in degrees C and the temperature coefficient is calculated in microseconds. For temperatures below 25°,which is assumed for the IMU's operation, the temperature coefficient is 0 μs. The capacitor charging time is 9.61 μs. Thus, the time between selecting an AD channel and the beginning of a conversion must be at least 14.61 μs. This time can be ensured by inserting "nop" commands in the code.

Using the AD converters, the code for the IMU reads the gyro data. ADCON0 is set to the channel of the first gyro. Several nops are then included to give the AD converter's capacitor time to charge. Once the capacitor is charged, setting a bit in the ADCON0 register begins a conversion. A flag bit, which is set when an AD conversion is complete, is polled. When the code determines that the flag bit is set, the flag bit is cleared, and the channel for the second gyro is selected. Data from the first gyro is read from the ADRESH and ADRESL registers containing the result of the 10-bit AD

conversions and stored in RAM. With the results of the previous conversion stored, the second AD conversion begins. When the conversion is complete, the process is repeated for the final gyro.

3.8.3 Digital Inputs

The accelerometers' outputs are digital in the form of a pulse width modulated (PWM) signal, which are read using one of the PIC18F4320's timers, TMR0. The PIC18F4320's Timer0 module is configurable as either an 8- or 16-bit timer through its control register T0CON. This register is used to enable/disable the timer and determine the prescaler value. The timer's rate is affected by the prescaler, which determines the timer's increment rate. When the timer is incrementing and overflows, a flag bit is set, which can cause an interrupt.

By design the accelerometers' outputs have a frequency of 83.33 Hz or a period of 0.012 s, which means that Timer0's resolution must allow it to capture times as slow as 0.012 s. The amount of time it takes for Timer0 to overflow can be determined with the following equation:

$$T_i = 4PS \times T_{OSC}(2^b - x) \tag{3.10}$$

where T_i is the time it takes for Timer0 to overflow, PS is the prescaler value, T_{OSC} is the period of the PIC's clock, b is the number of bits assigned to Timer0, and x is the value loaded into the Timer0 registers TMR0H and TMR0L. Registers TMR0H and TMR0L determine at what value the timer begins to increment. The higher the value in these registers, the shorter the amount of time before Timer0 overflows. The four is in the equation because Timer0 increments every four clock cycles for a prescaler value

of one. By setting the prescaler to 32, setting the timer as 8-bit, and setting x to zero, the Timer0 can count times as long as 0.016384 s. These values are chosen because they allow Timer0 enough time before overflow to capture the period of an output while cycling as quickly as possible.

Once Timer0 is properly configured, the accelerometers' output can be read. Although the duty cycle of the accelerometers' PWM output is set by an external resistor, the program still measures the period because of uncertainties in the resistor value. Measuring the period of each accelerometer is accomplished by calling a subroutine. The routine first checks the state an accelerometer's output. If the output is high, which means the PWM waveform is somewhere in the middle of its cycle, the program waits until it goes low. Once the output is determined to be low, the program waits for the output to go high. These steps are to ensure that the period is measured at the beginning of a cycle rather than at some unknown point. Once the output goes high, Timer0 is enabled and begins counting. The program waits for the output to go low and then high again. When the output goes high, Timer0 is disabled and the value in its TMR0H and TMR0L registers are stored in RAM. The TMR0H and TMR0L registers are then cleared for the next operation. Once the periods of the accelerometers' outputs are known, the output's duty cycle can be measured. The program first measures the output of the x-axis accelerometer testing if the output is high. If it is high, a routine is called which delays until the output goes low. Once the output is low, a routine is called which measures the amount of time that the output is high. This routine polls the accelerometer's output until it goes high. Once the signal is high, Timer0 begins counting. The program then polls until the signal goes low. With the signal low, Timer0 is disabled. The value in the TMR0H and TMR0L registers are then stored in RAM

until it is processed. The Timer0 counting registers are then cleared. The process is repeated for the y-axis and z-axis accelerometers.

3.8.4 MSSP Module

The PIC18F4320 includes a master synchronous serial port (MSSP) module that can be configured as either a serial peripheral interface (SPI) bus or an inter-integrated circuit (IIC) bus. For the IMU program only the IIC bus configuration is needed since the Devantech SRF08s use an IIC bus to interface with other devices. The PIC18F4320 can be configured as a master or a slave, but the sonars are always slaves so the microcontroller is always the master.

The MSSP module is configured with several registers. The SSPCON1 register configure the IIC bus's SCL and SDA pins, which are multiplexed with general I/O pins. By setting a bit, SSPCON1 configures the pins as a serial port rather than a general I/O port. The SSPCON1 register allows the user to select whether the microcontroller is a master or a slave as well as whether 7- or 10-bit addressing is used. For the ultrasonic sensors, 7-bit addressing is used. The other control register for the IIC bus is the SSPCON2 register. This register has the ability to send and receive acknowledge signals, send stop and start signals, and determine whether the microcontroller is in send or receive mode. SSPSTAT is the status register for the IIC bus. This register contains information on whether a start or stop bit has been detected, whether or not a transmit is in progress, and whether or not a receive operation is complete. The rate of information flow is controlled by the SSPADD register when the microcontroller is in master mode. The SSPADD register contains the value that controls the BAUD rate generator for the IIC serial bus. The equation to determine the clock frequency of the

IIC bus is

$$F_{cl} = F_{OSC}/(4(SSPADD + 1)) \tag{3.11}$$

where F_{cl} is the frequency of IIC bus clock, F_{OSC} is the microcontroller's clock frequency, and $SSPADD$ is the value written to the SSPADD register. The final register associated with the IIC bus is the SSPBUF register. The SSPBUF register receives data from a slave device. It also contains the data to be written to a slave during a transmit operation.

Before an IIC operation can begin, the control registers must be properly configured. SSPCON1 is set so that IIC mode is enabled and master mode is selected. The IIC BAUD rate is then set. Since 100 kHz is the frequency at which the IIC bus is designed to operate, this is the frequency chosen to determine the value to load into the SSPADD register. By using Equation (3.11) where $F_{OSC} = 20$ MHz, the value to loaded into SSPADD is 49. The register SSPBUF is also cleared to ensure that no data is loaded in the register.

In order for the sonars to begin a ranging operation, each sonar must receive the ranging command. To prepare the sonar to receive commands, the IMU code generates a start condition by setting a bit in the SSPCON2 register. A routine called IICPoll is then called to delay, by checking a flag bit, until the start condition is complete. Once the start condition has been sent, SSPBUFF is loaded with the address of the first sonar, E0h. It must be ensured that the LSB (least significant bit) is a zero to signify a write operation to the sonar. SSPBUF is then loaded with the sonar's command register, 0h. After the write operation is complete, the command 51h, which tells the sonar to format its data in centimeters, is loaded into SSPBUF and written to the sonar's address 0h. When IICPoll has determined that the write operation is complete, a stop condition is

sent to let the sonar know that the communication has ended. The process can then repeated for each remaining sonar, located at addresses E2h, E4h, and E6h, if they are connected to the IMU.

Once the command to begin ranging is written, the sonar must be given to time to complete its ranging operation. Timer1 is used to ensure that ranging operation is complete before attempting to read the sonar's data by waiting 65 ms. Timer1, a timer similar to Timer0, is set to be a 16-bit timer with a prescaler of 8 by writing to the T1CON register. Using Equation (3.10) to determine a value for x results in 24,911 or 614Fh. This value is loaded into the TMR1L and TMR1H registers. The Timer1 flag bit causes an interrupt to occur when Timer1 overflows indicating that 65ms has elapsed.

With the ranging operation complete, data can be read from the sonars. A read operation begins by sending a start condition. Once IICPoll determines that the operation is complete, the address of the first sonar to be read, E0h, is loaded into SSPBUFF. Next, the address from which the data to be read is written to the sonar. The sonar contains data at address 02h-23h, but only the data in addresses 02h-03h are read with the even address containing the high byte of data and the odd address containing the low byte of data. After the address has been sent, a restart condition is sent by setting a bit in register SSPCON2. The address E1h, the address of the sonar with the R/W bit set to let the sonar know that a read operation is about to follow, is loaded into SSPBUF. A bit is then set in the SSPCON2 register to enable IIC reception mode. The IIC flag bit is polled with the routine IICPoll to determine when the microcontroller has completed reading a data byte. The contents of SSPBUF is then saved in RAM, and an acknowledge signal is sent to tell the sonar that the PIC is ready to receive the next byte. The second byte is then received and stored. Since the second byte is the final

byte to be read from the sonar, a not acknowledge signal is sent to the sonar rather than an acknowledge signal. A stop condition is then sent to terminate communication with that sonar. The process can be repeated for each sonar.

3.8.5 Data Formatting

Once data from the sensors is obtained, it is formatted. Unformatted data is stored in a buffer in RAM. Each byte in the buffer is sent to a routine, called ASCII, that converts binary data into ASCII format. The ASCII routine converts one 8-bit number into two ASCII characters. First, the first four bits of a number are converted followed by the last four bits. Each nibble is compared to the value 0Ah to determine whether the ASCII character representation should be a letter or number. If the nibble is less than 0Ah, the routine processes the nibble as a number. If the nibble is greater than or equal to 0Ah, the routine processes the nibble as a letter. Numbers are converted to ASCII characters by adding the value 30h while letters are converted by adding 37h. When both nibbles have been converted, the new data is stored in a buffer. Every data byte undergoes the ASCII conversion. Once all data is converted to ASCII characters, it is transmitted serially with the USART to another device with each sensor reading separated by a comma. The end of a transmission is designated by a semicolon followed by a newline character.

3.8.6 Program Flow

The entire IMU program can be summarized by a flow chart as shown in Figure 3.3. The PIC18F4320 registers are initialized to the desired configurations. The initialization process includes enabling the proper interrupts, configuring the serial ports, initializing

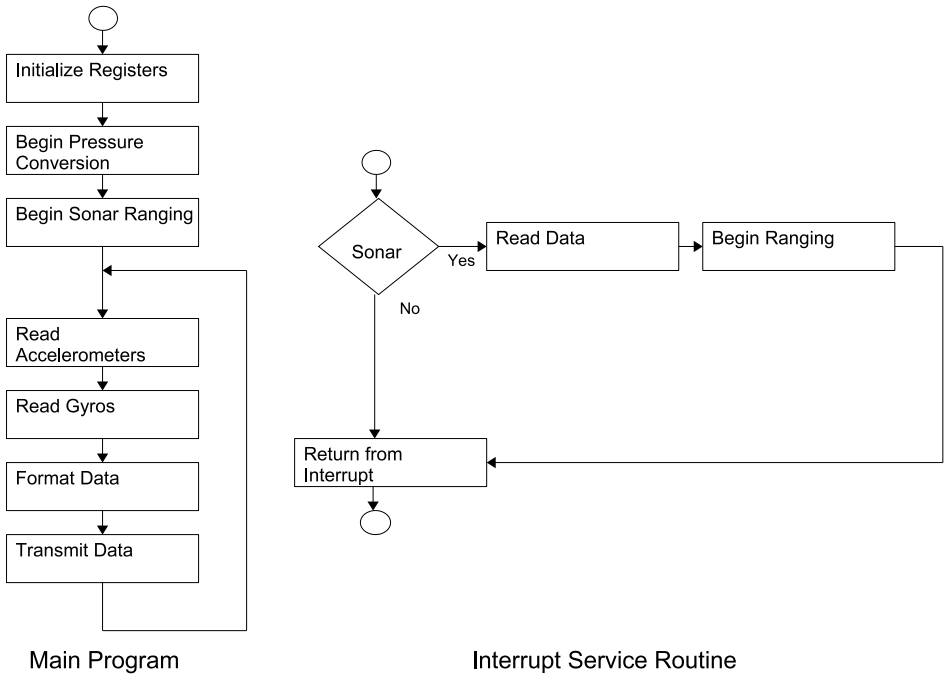


Figure 3.3: Flow Chart of IMU Program

timers, and establishing pin functions. The direction of the I/O pins must be set as well as their states set to a known value. A summary of pin functions is shown in Table 3.3. After initialization the sonar is handled with an interrupt service routine (ISR).

With initialization complete, the main part of the program commences. Readings from the accelerometers are taken with interrupts disabled. Since accurate readings of the accelerometers are dependent on Timer0, an interrupt during this section of code could result in inaccurate readings. If an interrupt occurred just before Timer0 overflows, an ISR would occur, which would add an unknown amount of time to the accelerometer reading. After all three accelerometers are read, interrupts are enabled and readings are taken from the gyros. Once reading the gyros is complete, the data gathered are converted to ASCII characters and stored in a buffer. The buffer is then serially transmitted to an outside device via the USART. During transmission interrupts are disabled so a complete data set can be sent at once. After data transmission is complete, the program returns to the section of code where accelerometer readings are taken. The process then repeats in an infinite loop.

The only deviation in the infinite loop is when an interrupt occurs when the sonars are done ranging. Inside the ISR, the source of the interrupt is checked by polling flag bits. If the Timer1 overflow flag is set, the sonars are done ranging. If this flag is not set, the interrupt cause is unknown, and the ISR is exited. When a sonar causes an interrupt, the data are read from each sonar. A new ranging operation is then begun, and the ISR is exited.

Pin	Pin Name	Function	Pin	Pin Name	Function
2	AN0	Yaw axis gyro input	9	RE1	Z-axis accel. input
4	AN2	Roll axis gyro input	18	SCL	IIC bus clock
5	AN3	Pitch axis gyro input	23	SDA	IIC bus data line
7	RA5	X-axis accel. input	25	TX	USART transmit line
8	RE0	Y-axis accel. input	26	RX	USART receive line

Table 3.3: Pin Functions

Chapter 4

Sensor Modeling and Blimp Simulation

In this chapter we present the results of sensor testing and modeling as well as a simulation of the blimp in flight. We begin in Section 4.1 by outlining several tests performed on the IMU's sensors and then construct models of the sensors based on the collected data. In Section 4.2 we develop two models of the blimp followed by guidance and control laws. We conclude the chapter with Section 4.3, which outlines the results of performing various simulations with the developed blimp and sensor models.

4.1 Sensor Data and Modeling

In this section we present information pertaining to an IMU's sensors. In Section 4.1.1 we explain methods for gathering data from sensors and validating sensor operation. We present tests performed on gyros, accelerometers, and an ultrasonic range finder. In Section 4.1.2 we develop a simple model for each of the sensors for the IMU designed in Chapter 3. We then evaluate the validity of each model based on autocorrelation methods.

4.1.1 Sensor Testing

The operation of each sensor in an IMU can be validated through various tests. Ideally, test stands and tables designed specifically for testing IMUs would be used. Test tables give precise control of spin rates and accelerations so that IMU measurements can

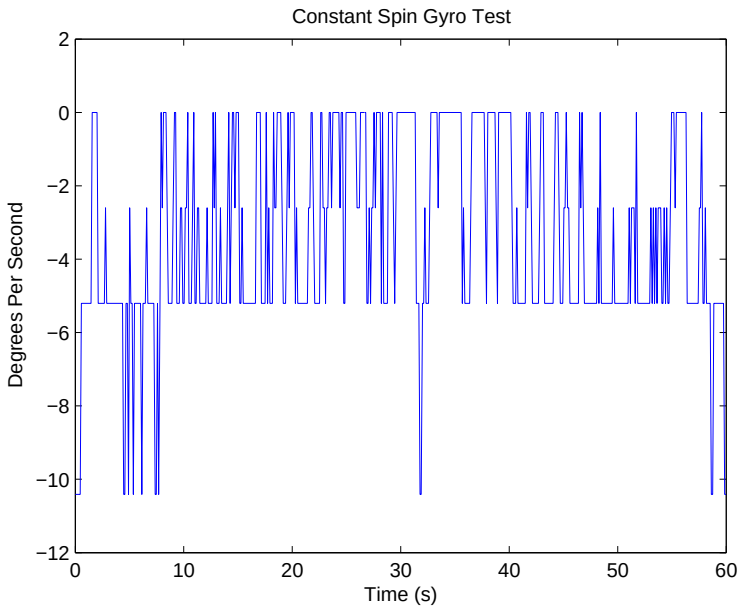


Figure 4.1: Gyro Output for a Rotational Rate of -6 $^\circ/s$

be compared to known truth data. However, such test stands and tables are unavailable for testing the IMU presented in this work.

To test the operation of the IMU's gyros, constant rates are applied as input, and the output is measured. The simplest test is to apply an input of a zero rotational rate. In Figure 4.10 we show the results of applying zero input. The gyro outputs a noisy reading of zero degrees per second as expected. Another test is to apply a known constant rotation and compare the measurements to the actual rotational rate. Placing the gyro in a chair and spinning the chair provides a rough approximation to a constant rotational rate. In Figure 4.1 we show a plot of the results of spinning the gyro in a chair at a rate of -6 $^\circ$/s. (The low rate of spin is chosen as a test rate because the blimp has low rotational rates.) The gyro has a noisy output centered at -5.2 $^\circ$/s. Some fluctuations in the gyro data are due to noise while others are due to actual fluctuations

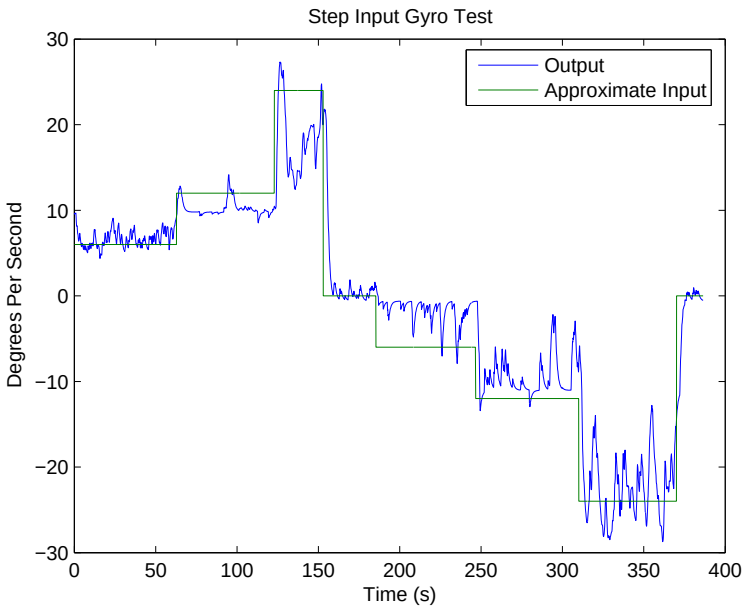


Figure 4.2: Step Inputs Applied to a Gyro

in the chair's rate of spin. The test verifies that the gyro is suitable for measuring the blimp's rotational rates. Another test performed on the gyros is applying various step inputs. The step inputs are applied by placing the IMU in a chair and spinning the chair at an approximately constant rate. The rate of the chair is set by measuring the amount of time one rotation takes. We apply rotational rates of 6 °/s, 12 °/s, 24 °/s, 0 °/s, -6 °/s, -12 °/s, and -24 °/s where a positive sign indicates clockwise rotation and a negative sign indicates counterclockwise rotation. In Figure 4.2 we show the results of this test with filtered gyro output and bias removed (gyro bias is presented in Section 4.1.2). With the green line in Figure 4.2 we approximate the actual input to the gyros (the actual input is not really constant) while the blue line is actual gyro data. The data show that the gyro is sensitive enough to detect low rotational rates and relatively small changes in rotational rates. In Figure 4.3 we plot the mean and median of the gyro

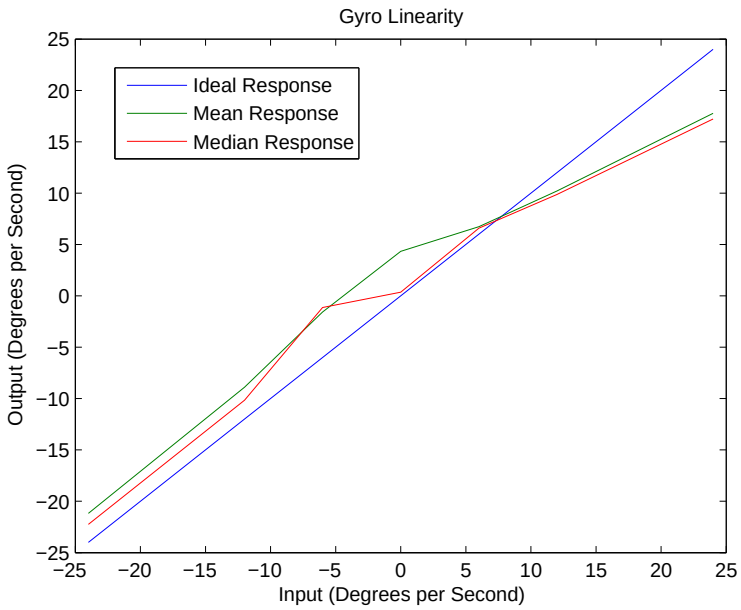


Figure 4.3: Linearity of Gyro Response

output at the various inputs to test the sensor's linearity. By comparing the mean and median values to the ideal response, it can be seen that the sensor output is not exactly linear. However, the uncertainty of the actual input rates directly affects the plot in Figure 4.3.

To get a more reliably smooth input, another experiment is designed. As shown in Figure 4.4, a square wooden platform is constructed with a string attached to each corner. The four strings are tied to a single string. The platform has a series of twenty evenly spaced holes in which steel balls can be placed to change the platform's moment of inertia and thereby the natural frequency at which the platform rotates. Once the IMU is placed at the center of the platform, the string that suspends the platform is wound, and the platform is released to spin freely. The number of revolutions before the platform changes spin direction is counted, and the time is recorded in order to compute

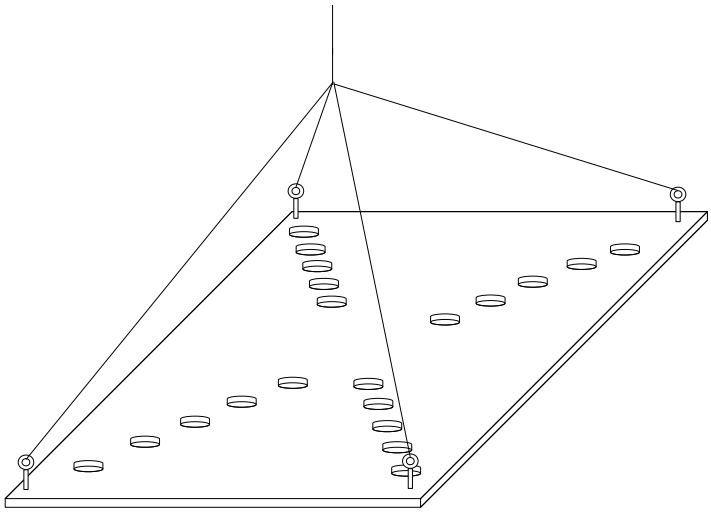


Figure 4.4: Platform Design

a mean rotational rate (the green curves in Figures 4.5-4.7). The mean rate is then compared to the IMU's measured rate (the blue curves in Figures 4.5-4.7). The results of the tests are shown in Figures 4.5-4.7.

The rotational rate of the platform can be modeled by a sinusoid that undergoes damping due to friction and air resistance

$$r(t) = Ae^{-at}\sin(\omega_N t)$$

where ω_N is the system's natural frequency, A is amplitude, and a is a damping factor. For an initial time t_0 and an ending time t_1 and ignoring the damping factor, we can say that the average rotational rate between changes of direction is

$$\begin{aligned} r_{avg} &= \frac{1}{t_1 - t_0}\int_{t_0}^{t_1} A\sin(\omega_N t)dt \\ &= \frac{-A(\cos(\omega_N t_1) - \cos(\omega_N t_0))}{\omega_N(t_1 - t_0)} \end{aligned}$$

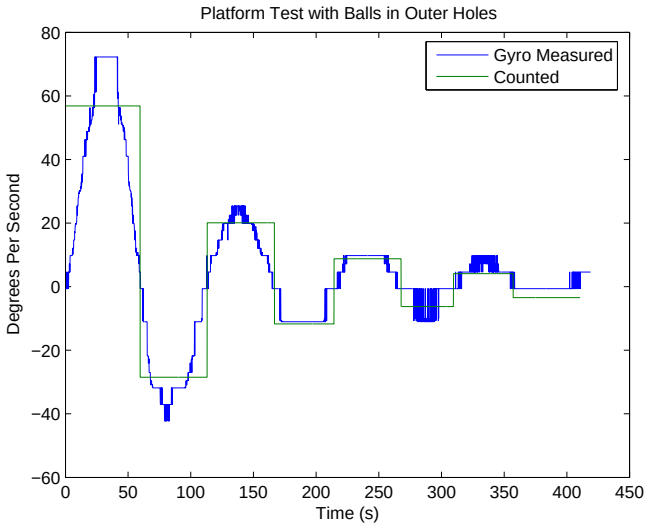


Figure 4.5: Measured Rotation Rate with Balls in Outer Holes

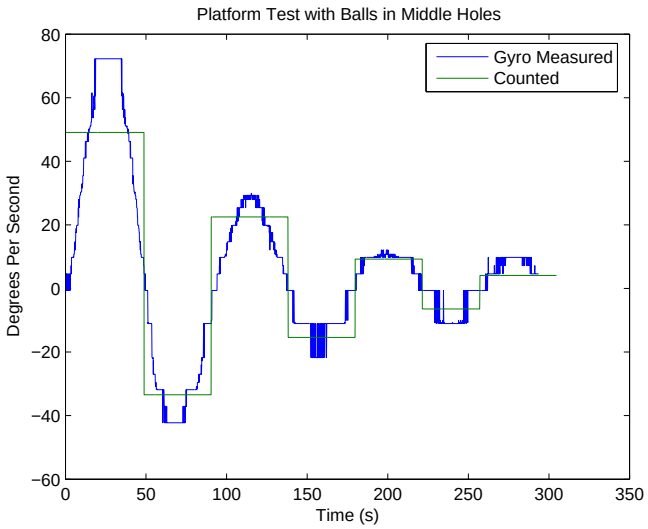


Figure 4.6: Measured Rotation Rate with Balls in Middle Holes

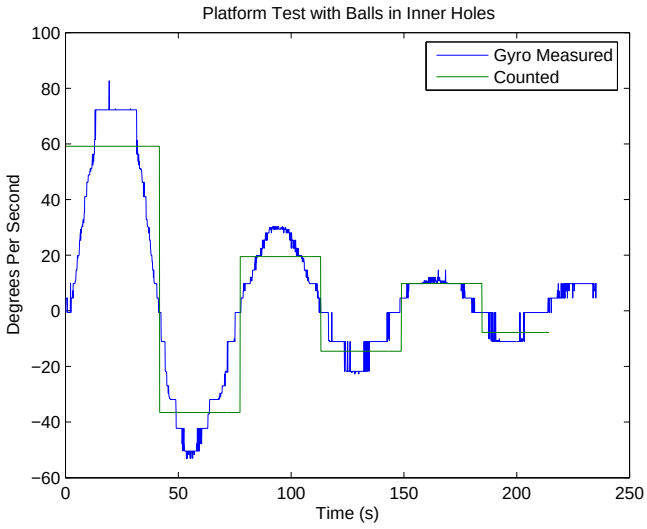


Figure 4.7: Measured Rotation Rate with Balls in Inner Holes

Without loss of generality, we let $t_0 = 0$ and $t_1 = \pi/\omega_N$ so

$$r_{avg} = \frac{2A}{\pi} \tag{4.1}$$

Thus, by comparing the average value calculated from knowledge of the amount of time and the number of revolutions between the spinning platform's changes of directions and an average value found by finding the maximum rotational rate measured by the gyro for half a period and applying Equation (4.1), we have a method to gage the gyro's accuracy.

In Tables 4.1-4.3 we summarize the results of applying the above described analysis to the data taken from the spinning platform. The greatest relative error is 40%, the smallest relative error is 0%, and the overall average relative error is 15%. As the signal to noise ratio decrease, the relative error increases; thus, the greatest errors are found at

Meas. Peak	72.2	-42.3	25.4	-11.0	9.8	-11.0	9.8
Revs	8.375	3.875	2.625	1.625	1.0	0.75	0.5
Time (s)	53.0	49.0	47.0	50.0	41.0	43.0	44.0
Gyro Mean	46.0	-26.9	16.2	-7.0	6.2	-7.0	6.2
Calc. Mean	56.9	-28.5	20.1	-11.7	8.9	-6.3	4.1

Table 4.1: Measured and Calculated Rotational Mean Value for Balls in Outer Holes

Meas. Peak	72.3	-42.3	30.0	-21.8	12.1	-11.0
Revs	6.0	3.625	2.5	1.625	1.0	5.625
Time (s)	44.0	39.0	40.0	38.0	39.0	45.0
Gyro Mean	46.0	-26.9	19.1	-13.9	7.7	-7.0
Calc. Mean	49.1	-33.5	22.5	-15.4	9.2	-6.4

Table 4.2: Measured and Calculated Rotational Mean Value for Balls in Middle Holes

the lowest rotational rates. The absolute error, however, is small at low rotational rates so the gyros are adequate for measuring a blimp's rotational rates.

Constant input tests are also used to validate the operation of the accelerometers. As in the case of the gyros, a constant input of zero is applied to each accelerometer. The results presented in Figure 4.9 for one accelerometer show that the accelerometers have an output of zero for an input of zero as desired. Since equipment to test the accelerometers is unavailable, the only other test performed is a constant input test where the input is gravity (or one g). Placing an accelerometer so that its measurement axis is aligned vertically results in the sensor experiencing an acceleration from gravity. The results of this test for one accelerometer are shown in Figure 4.8. The accelerometer has a noisy output centered around 1.025 g, an acceptably accurate measurement.

Meas. Peak	82.7	-53.0	30.3	-22.8	14.7	-11.0
Revs	5.75	3.25	2.0	1.25	0.875	0.625
Time (s)	35.0	32.0	33.0	31.0	32.0	29.0
Gyro Mean	52.6	-33.7	19.3	-14.5	9.4	-7.0
Calc. Mean	59.1	-36.6	19.5	-14.5	9.8	-7.8

Table 4.3: Measured and Calculated Rotational Mean Value for Balls in Inner Holes

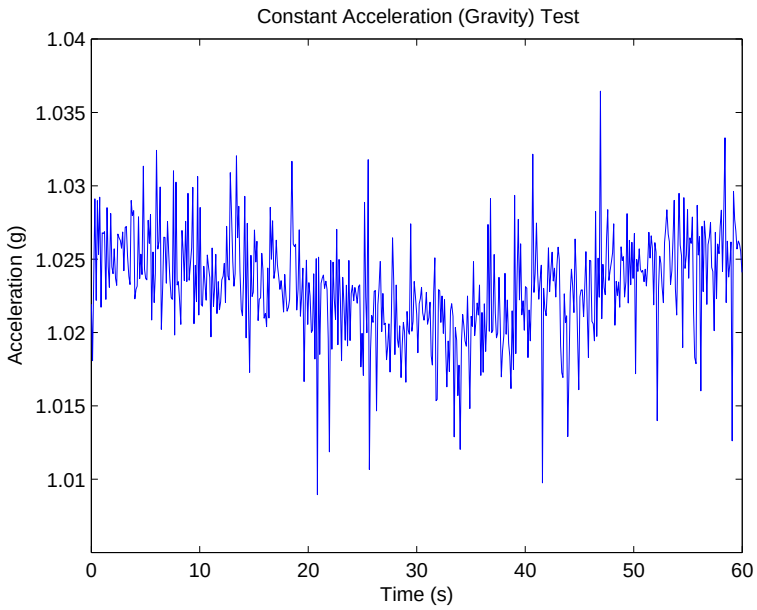


Figure 4.8: 1 g Input Applied to an Accelerometer

The ultrasonic sensor is tested by comparing its distance reading to a known actual distance. An obstacle is placed at various distances from the range finder, and the sensor's response is measured. As shown by the experimental data summarized in Table 4.4, the farther the sonar is from an obstacle, the less reliable the measurement reading is.

Actual Distance (cm)	Measured Distance (cm)	Error (cm)	Actual Distance (cm)	Measured Distance (cm)	Error (cm)
30	34	4	180	187	7
60	64	4	210	218	8
90	96	6	240	248	8
120	127	7	270	280	10
150	156	6	300	309	9

Table 4.4: Distances Measured by Ultrasonic Sensor

4.1.2 Sensor Modeling

A simple model of a sensor, applied to each of the gyroscopes, accelerometers, and ultrasonic sensors that form the IMU discussed in Chapter 3, is given by

$$m_m = m_a + b + w + \eta \tag{4.2}$$

where m_m is the measured quantity, m_a is the actual value of the measurand, b is a constant bias term, w is a walking bias term, and η is zero-mean white noise with variance σ^2. To determine values for b, w, and σ^2 that reasonably model each of the IMU's sensors, the IMU is run for several hours at rest (the IMU remains stationary while data are being gathered). Since the IMU undergoes no forces other than gravitational force, the quantity m_a is known to be zero except for the z-axis accelerometer for which $m_a = 1$ g. The constant bias b is determined by finding the average value of the collected data. The variance of the random noise affecting each sensor is found by evaluating the variance of the data. After evaluating the data, it is determined that random noise dominates the sensors' outputs so w can be neglected. A summary of model parameters for each sensor of the IMU is presented in Table 4.5. In Figure 4.9 we show a plot of actual data gathered from an accelerometer as well as a plot of the accelerometer model. In Figure 4.10 we show a plot of data from a gyro and its associated model. Data taken from the IMU's ultrasonic sensor and the corresponding model are shown in Figure 4.11.

The residuals of the actual data gathered from sensors and the sensor models are used evaluate the validity of the models. For the model to be considered sufficient, the residuals should appear to be Gaussian noise [8]. This means that the autocorrelation function of the residuals for any given model should be a Dirac delta. The autocorrelation

	Accelerometers		Gyros	
	Mean b	Std. Dev. σ	Mean b	Std. Dev. σ
x	-0.0048906	0.0044971	0.21698	0.10694
y	7.3425e-4	0.00034547	-0.0052620	0.51891
z	0.032194	0.0037683	-9.79251	0.39243
sonar	-0.1404	0.7336		

Table 4.5: Model Parameters of Sensors

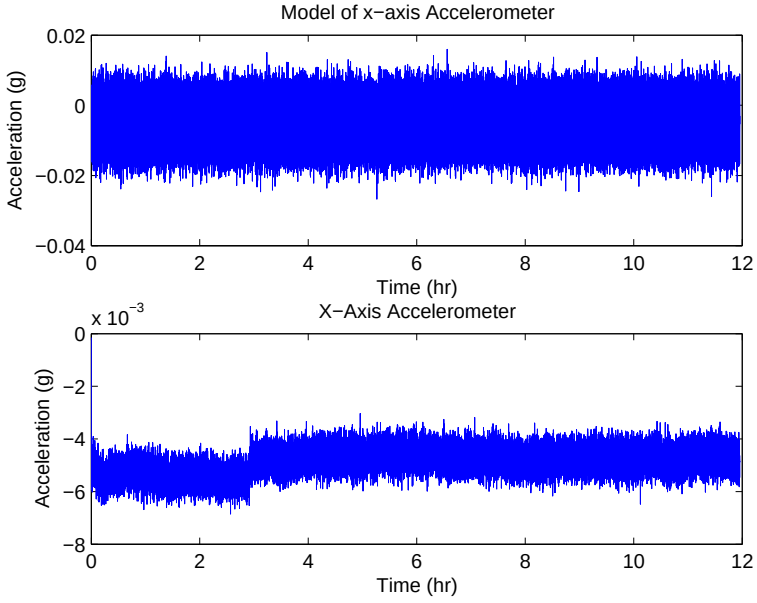


Figure 4.9: Accelerometer Data and Model

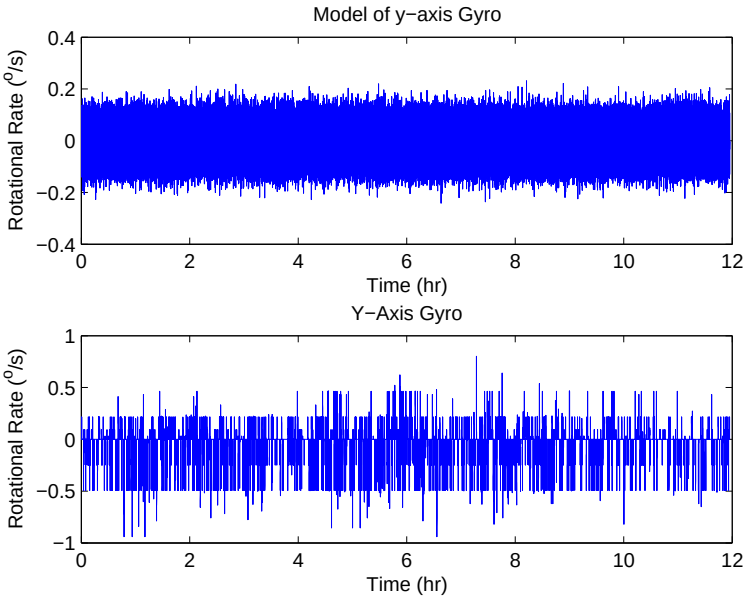


Figure 4.10: Gyro Data and Model

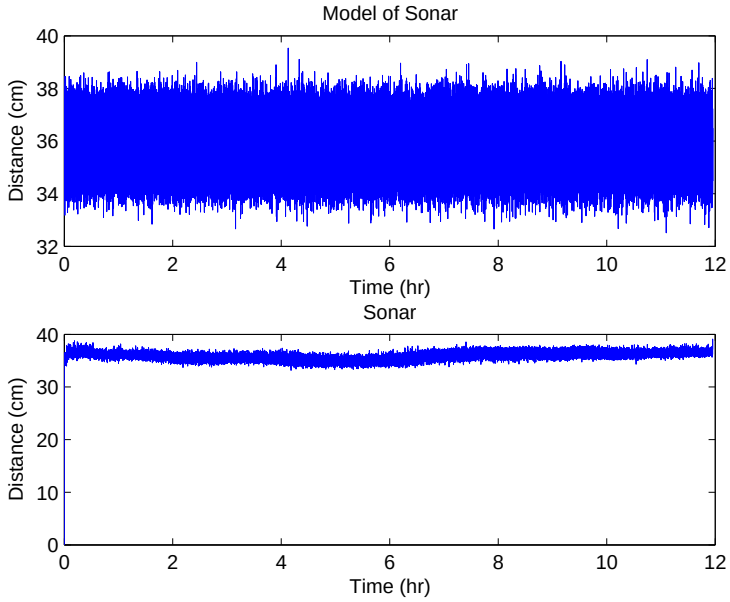


Figure 4.11: Sonar Data and Model

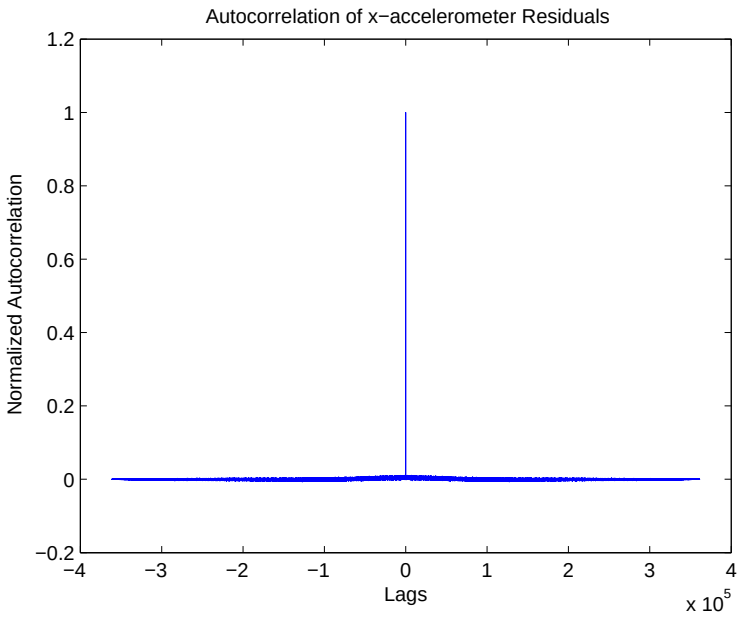


Figure 4.12: Accelerometer Residual Autocorrelation Function

function of a sequence $x[n]$ is defined as

$$R_x(\tau) = \lim_{N\to\infty} \frac{1}{N} \sum_{k=0}^{N} x[k]x[k-\tau]$$

Since $x[n]$ is a finite sequence of residuals for each sensor, the autocorrelation function is approximated as

$$R_x(\tau) = \frac{1}{N} \sum_{k=0}^{N-1} x[k]x[k-\tau] \tag{4.3}$$

The autocorrelation function of an accelerometer's residuals is shown in Figure 4.12, and the the autocorrelation function of a gyro's residuals is shown in Figure 4.13. Figures 4.12 and 4.13 are representative of all accelerometers and gyros tested. The plots show two delta functions, which validates the models for the inertial sensors. A plot of the sonar's residuals is shown in Figure 4.14. The autocorrelation function of the sonar's

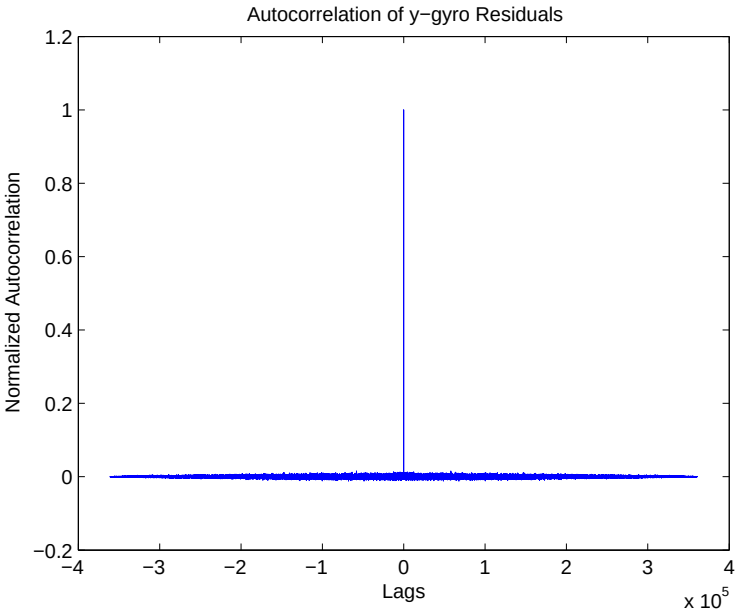


Figure 4.13: Gyro Residual Autocorrelation Function

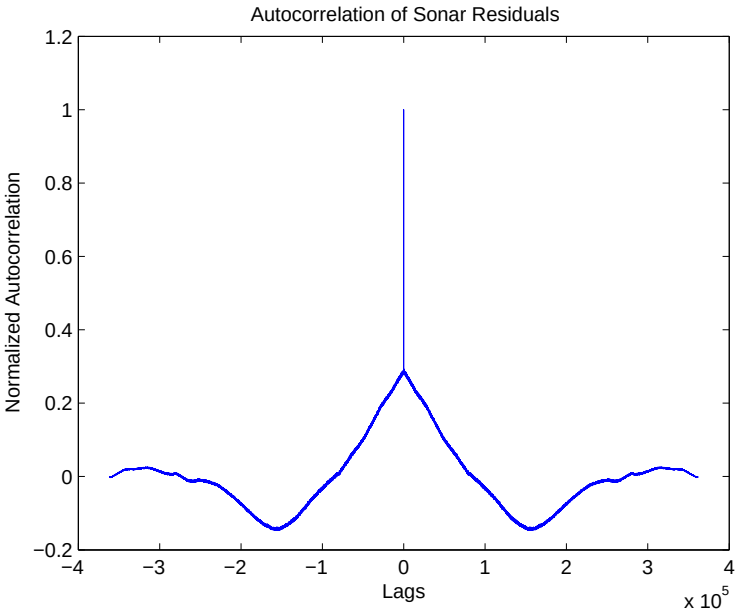


Figure 4.14: Sonar Residual Autocorrelation Function

residuals is a poor approximation of a delta function having two distinct minima rather than just a spike at the origin. This implies that the ultrasonic sensor's dynamics are driven by factors other than zero mean white noise. The model derived is inadequate to capture these dynamics. However, since the output of the sonar does not need to be integrated, ignoring the sensor's additional dynamics is not detrimental to system performance. Thus, the model is used for its simplicity.

4.2 Blimp Simulation

In this section we present simulations of the blimp. We begin by developing two models of the blimp in Section 4.2.1. One model is based on a real-valued yaw angle while the other model is based on a complex-valued yaw angle. In Section 4.2.2 we present guidance and control laws for the blimp. The control law is developed with a back-stepping approach.

4.2.1 Blimp Modeling

In developing the blimp model, we select a state vector $x \in \mathbb{R}^8$ as

$$x = \begin{bmatrix} p & v & \psi & r \end{bmatrix}^T$$

where $p(t)$ and $v(t)$ are length three vectors that respectively represent the position and velocity of the blimp in the inertial reference frame, ψ is yaw angle, and r is the body-frame rotational rate about the z axis. Translational dynamics are modeled in an inertial reference frame as shown in Figure 4.15. The rotational dynamics are defined in a body reference frame with the x-axis through the nose, the y-axis to the right, and the

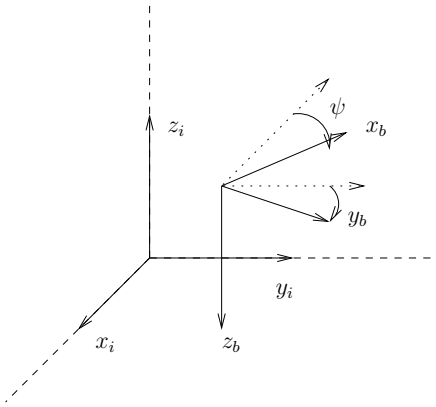


Figure 4.15: Inertial (Reference) Frame and Body Frame of Reference

z-axis "down" from the viewpoint of the blimp, so that we maintain a right-hand rule orthogonal coordinate axis set. If the origins of each frame of reference are colocated, then we transform the coordinates of a point $p_i = \begin{bmatrix} p_x & p_y & p_z \end{bmatrix}^T$ in the inertial frame to the body frame by calculating

$$p_b = \text{diag}(-1, 1, -1) \begin{bmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} p_i \triangleq \text{diag}(-1, 1, -1) G_{xy}(\psi) p_i$$

where G_{xy} is a Givens rotation in the x-y plane. For simplicity in notation, we define the diagonal matrix

$$\Xi = \text{diag}(-1, 1, -1)$$

so that we can write

$$p_b = \Xi G_{xy}(\psi) p_i$$

We derive the differential equations for this model as follows. For simplicity, we assume that the blimp mass is 1 kg and that its moment of inertia is 1 kg-m. The blimp

is assumed to be neutrally buoyant so gravity plays no role in the dynamics. The blimp has three inputs:

Thrust fan pitch angle θ_f The two main thrust fans pitch in the x-z plane and may take a commanded value of $\theta_f \in [-\pi, \pi]$ rad where $\theta_f = 0$ means that the fans drive the blimp forward in the x-direction and $\theta_f = \pi/2$ means that the fans thrust the vehicle downward (positive z-direction in the body frame). For simplicity, we assume that the thrust force is applied at the blimp center of gravity; this is not correct but is adequate for our study.

Blimp thrust level u_f The thrust level is given in percent thrust (from 0 to 100).

Yaw torque u_τ The yaw torque is limited to $|u_\tau| \leq 0.5$ N-m with a lever arm of $r_\tau = 3$ m to the blimp's center of gravity. u_τ is assumed to be in the body frame x-y plane.

The actual applied force by the thrust fans and the yaw torque fan in the body frame is thus the vector

$$f_t = \begin{bmatrix} \cos(\theta_f) u_f / 600 \\ u_\tau / r_\tau \\ \sin(\theta_f) u_f / 600 \end{bmatrix}$$

Model with Real-Valued ψ

The translational dynamics of the blimp are modeled by

$$\begin{aligned} \dot{p} &= v \\ \dot{v} &= (f_f + f_t + f_d)/m \end{aligned}$$

where p is a position vector, v is a velocity vector, f_f is frictional force, f_t is force from the blimp's thruster fans, f_d is disturbance force, and m is the blimp's mass. The disturbance force is assumed to be negligible so the model (in the body frame) is

$$\dot{p} = v \tag{4.4a}$$

$$\dot{v} = (f_f + f_t)/m \tag{4.4b}$$

Modeling frictional forces as linear based on experimental results allows the friction term to be written as

$$f_v = -\text{diag}(a_1, a_2, a_3)v \stackrel{\Delta}{=} -\Delta v$$

where $a_1 = 4/30$, $a_2 = 4/5$, and $a_3 = 3/10$. Writing the translation equations of motion (in the inertial frame) results in

$$\begin{aligned} \dot{p} &= v \\ \dot{v} &= \frac{1}{m} G_{xy}^T \Xi \left(-\Delta v + \begin{bmatrix} \cos(\theta_f) u_f/600 \\ u_\tau / r_\tau \\ \sin(\theta_f) u_f/600 \end{bmatrix} \right) \end{aligned}$$

The rotational equations of motion are developed as a sum of damping (frictional) forces and forces (torques) applied in the yaw axis. Frictional forces come from y-axis (sideways) translational velocity, which induces a yaw moment, and by damping from the yaw-axis rotation rate r. In the translational induced yaw torque term the coefficients l_1 (moment arm length) and δ_2 (a damping term) appear. Similar terms l_2 and δ_2 appear in the damping from the yaw rate r (from observed behavior it is assumed that $l_1\delta_1 = l_2\delta_2$),

which gives

$$\dot{r} = \frac{1}{J}(l_1\delta_1 v_y - l_2\delta_2 r + u_\tau)$$

Then the complete set of equations that describe the blimp are given by

$$\begin{aligned}
\dot{p} &= v && (4.5a)\\
\dot{v} &= \frac{1}{m}G_{xy}^T \Xi\left(-\Delta v + \begin{bmatrix} \cos(\theta_f)u_f/600 \\ u_\tau/r_\tau \\ \sin(\theta_f)u_f/600 \end{bmatrix}\right) && (4.5b)\\
\dot{\psi} &= r && (4.5c)\\
\dot{r} &= \frac{1}{J}(l_1\delta_1 v_y - l_2\delta_2 r + u_\tau) && (4.5d)
\end{aligned}$$

Model with Complex-Valued $\bar{\psi}$

Another method of writing the blimp model is by representing rotations as unit complex numbers. For example, the yaw angle ψ is represented as

$$\bar{\psi} = e^{j\psi} = \cos(\psi) + j\sin(\psi) \stackrel{\Delta}{=} \psi_x + \psi_y \qquad (4.6)$$

We shall use the above notation throughout this document: a bar indicates a vector (unit complex number) representation of an angle and the subscripts x and y refer to the cosine and sine of the corresponding angle, respectively.

Let $\bar{v}_i = v_x + jv_y$ be an arbitrary vector in the inertial frame x-y plane (not necessarily a unit vector). Then the body frame representation $\bar{v}_b$ of $\bar{v}_i$ is

$$\bar{v}_b = e^{-j\psi}\bar{v}_i$$

This is verified by examining the unit vectors x_b and x_i in the body and inertial frame, respectively.

The differential equation relating the complex yaw angle $\bar{\psi}$ to the body-frame yaw rate is

$$\begin{aligned}\dot{\bar{\psi}} &= \frac{d}{dt}(\cos(\psi) + j\sin(\psi)) \\ &= r(-\psi_y + j\cos(\psi)) \\ &= j\bar{\psi}r\end{aligned}$$

where the last line follows from Equation (4.6).

For ease in the use of the complex yaw angle $\bar{\psi}$, we examine the description of the position and velocity vectors as

$$p_i = [\ p_x + jp_y \quad p_z\ ]^T \stackrel{\Delta}{=} [\ \bar{p}_i \quad p_z\ ]^T$$

and

$$v_i = [\ v_x + jv_y \quad v_z\ ]^T \stackrel{\Delta}{=} [\ \bar{v}_i \quad v_z\ ]^T$$

That is, we represent the three dimensional vectors as a two-vector whose first element is a complex number determined from the x-y components in the reference frame. Then

the body-frame x-y coordinates are related to the inertial frame coordinate by

$$\begin{aligned}
\bar{p}_b &= -(\psi_x p_x + \psi_y p_y) + j(-\psi_y p_x + \psi_x p_y) \\
&= -((\psi_x p_x + \psi_y p_y) + j(\psi_y p_x - \psi_x p_y)) \\
&= -(\psi_x + j\psi_y)(p_x - jp_y) \\
&= \bar{\psi}(-\bar{p}_i^*)
\end{aligned}$$

where $-p_i^*$ represents a rotation about the y-axis that reverses the sign of the x-component. Similarly

$$\begin{aligned}
\bar{p}_b &= -\bar{\psi}\bar{p}_i^* \\
-\bar{\psi}^{-1}\bar{p}_b &= \bar{p}_i^* \\
-\bar{\psi}^*\bar{p}_b &= \bar{p}_i^* \\
-\bar{\psi}\bar{p}_b^* &= \bar{p}_i
\end{aligned}$$

That is, due to the change in sign caused by rotating the body frame axis to be z-down, the conversion processes from complex inertial to and from complex body representation are identical.

The differential equations of motion for the blimp with complex-valued $\bar{\psi}$ are

$$\dot{p}_i = v_i \tag{4.7a}$$

$$\dot{v}_i = \frac{1}{m} G_{xy}^T \Xi \left(-\Delta v + \begin{bmatrix} \cos(\theta_f) u_f / 600 \\ u_\tau / r_\tau \\ \sin(\theta_f) u_f / 600 \end{bmatrix} \right) \tag{4.7b}$$

$$\dot{\bar{\psi}} = j \bar{\psi} r \tag{4.7c}$$

$$\dot{r} = \frac{1}{J} (l_1 \delta_1 v_y - l_2 \delta_2 r + u_\tau) \tag{4.7d}$$

Note that $G_{xy}(\psi)$ is nonlinear in ψ but is linear in the components of $\bar{\psi}$.

4.2.2 Guidance and Control Law Design

The derivation of the guidance and control laws is based on the equations of motion (4.5) with real valued ψ. The control law is developed by a back-stepping approach.

Guidance design develops a desired force to be applied to the blimp in order to reach the next waypoint. Examine Equations (4.5a) and (4.5b), and treat each of them as independent first order differential equations in inertial x, y, and z axes:

$$\dot{p} = v$$

$$\dot{v} = av + b_t f_t$$

Position Control Loop

It is desired to make x approach a constant waypoint x_f. Then the error between the desired position and actual position is defined as

$$x_e = x_f - x \tag{4.8}$$

The error dynamics are then

$$\begin{aligned} \dot{x}_e &= \dot{x}_f - \dot{x} \\ &= -v \end{aligned} \tag{4.9}$$

For the blimp to have a one minute settling time, define $c_1 = -4/60$ where

$$\dot{x}_e = -c_1 x_e$$

which means that a commanded velocity v_{cmd} must be

$$v_{cmd} = -c_1 x_e \tag{4.10}$$

Velocity Control Loop

Define the velocity error v_e as

$$v_e = v_{cmd} - v \tag{4.11}$$

which has dynamics

$$\begin{aligned} \dot{v}_e &= \dot{v}_{cmd} - \dot{v} \\ &= -c_1 \dot{x}_e - av - b_t f_t \\ &= -(a + c_1)v - b_t f_t \end{aligned} \tag{4.12}$$

For v_e to have a twelve second settling time

$$\dot{v}_e = c_2 v_e \tag{4.13}$$

where $c_2 = -4/12$. Substituting Equation (4.12) for $\dot{v}_e$ results in

$$c_2 v_e = -(a + c_1)v - b_t f_{t,cmd}$$

This means that the fan thruster force command should be

$$f_{t,cmd} = (-(a + c_1)v - c_2 v_e)/b_t \tag{4.14}$$

Attitude/Fan Command Control Loop

Rotational dynamics are restricted to the yaw axis since the blimp has fins which eliminate roll and pitching is negligible. The rotational dynamics are then described by

$$\dot{\psi} = r \tag{4.15a}$$

$$\dot{r} = a_2 r + \tau/J \tag{4.15b}$$

where τ is an input torque from the blimp's yaw fan and J is moment of inertia. Define the yaw error ψ_e as the difference between a commanded yaw angle ψ_{cmd} and the actual yaw angle:

$$\psi_e = \psi_{cmd} - \psi \tag{4.16}$$

Then the error dynamics are described by

$$\begin{aligned} \dot{\psi}_e &= \dot{\psi}_{cmd} - r \\ &= -r \end{aligned} \tag{4.17}$$

where it is assumed that ψ_{cmd} varies slowly so the derivative is approximately zero. To have the yaw angle settle in ten seconds make

$$\dot{\psi}_e = c_3 \psi_e \tag{4.18a}$$

$$= -r_{cmd} \tag{4.18b}$$

where $c_3 = -4/10$. This implies that

$$r_{cmd} = -c_3 \psi_e \tag{4.19}$$

The angular rate r must then be controlled. Define angular rate error r_e as

$$r_e = r_{cmd} - r \tag{4.20}$$

For the angular rate to converge within one second, select $c_4 = 4$ such that

$$\begin{aligned} \dot{r}_e &= c_4 r_e \\ &= \dot{r}_{cmd} - \dot{r} \\ &= c_3 \dot{\psi}_e - (\tau/J - a_2 r) \\ &= -c_3 r - \tau/J - a_2 r \\ &= -(a_2 + c_3) r - \tau/J \end{aligned} \tag{4.21}$$

Thus, the commanded torque τ_{cmd} should be

$$\tau_{cmd} = J((-a_2 + c_3) r - c_4 r_e) \tag{4.22}$$

The fan angle and thruster fan power percentage are the final quantities to be developed. The fan angle θ_f is determined by the z-axis force command and x-axis force command:

$$\theta_f = \tan^{-1}\left(\frac{-f_{tz}}{f_{tx}}\right) \tag{4.23}$$

The thrust fan power percentage is chosen to minimize the error in applied force or

$$\overset{min}{l} \; ||f_\theta l - f_c||$$

Coordinate			Time (s)	Coordinate			Time (s)
x	y	z		x	y	z	
0	0	0	0	0	0	50	460
0	0	50	10	0	0	0	610
75	0	50	160	0	0	0	1000
75	75	50	310				

Table 4.6: Destination Waypoints and Times

where l is thrust fan power level, f_θ is a unit vector in the direction of the thrust fan angle, and f_c is the force command. Solving the minimization results in

$$\begin{aligned} 0 &= \frac{d}{dt}\frac{1}{2}(f_\theta l - f_c)^T(f_\theta l - f_c) \\ &= \frac{d}{dt}(\frac{1}{2}l^T f_\theta^T f_\theta l - l f_\theta^T f_c + \frac{1}{2}f_c^T f_c) \\ &= l - f_\theta^T f_c \\ l &= f_\theta^T f_c \end{aligned} \tag{4.24}$$

4.3 Simulation Results

In this section we describe the results of simulating the blimp using various combinations of sensor systems. In each simulation it is desired for the blimp to reach a series of waypoints in a specific amount of time as shown in Table 4.6. The waypoint coordinates are points in a three-dimensional space defined by a right-handed orthogonal axis set with the z-axis pointing "up." All simulations are performed with Matlab and Simulink software packages.

The remainder of the section is organized as follows. In Section 4.3.1, we evaluate the performance of the control law developed in previous sections. In Section 4.3.2 we evaluate the ability of the IMU to accurately estimate system states. In Section 4.3.3

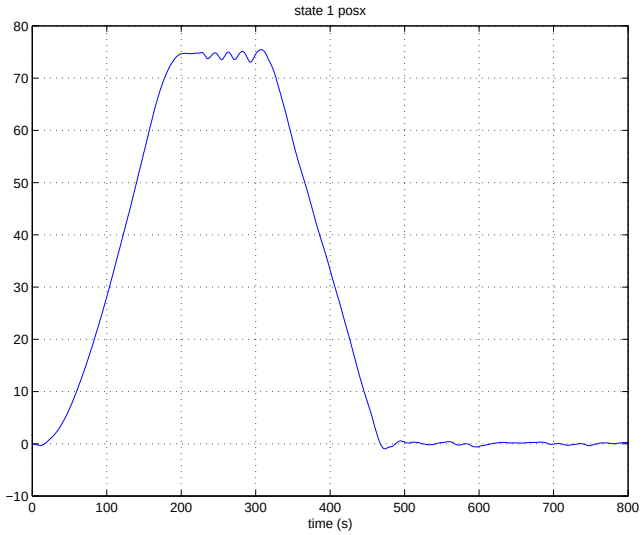


Figure 4.16: X Position with Perfectly Known States

we evaluate the effect of adding ground based camera system measurements to IMU measurements. It is assumed that the camera gives x, y, and z coordinate information.

4.3.1 Perfectly Known States

To evaluate the control law performance, the system is simulated as if the states (position, velocity, yaw, and yaw rate) are perfectly known. In Figures 4.16-4.18 we show the blimp's position through 800 s of simulation, in Figures 4.19-4.21 we show the blimp's velocity, and in Figure 4.22 we show the blimps yaw angle and yaw rate. The inputs to the blimp are shown in Figures 4.23-4.25.

As can be seen in Figures 4.16-4.18, the blimp converges to the commanded positions reasonably well, most notably in the x and y positions. The z position does not reach the position of 50 until almost 300 s later than desired. This is because given the blimp's response capabilities going from a position of 0 to 50 in 10 s is an unrealistic expectation. After 10 s, the blimp is commanded to reach a new x coordinate. Since

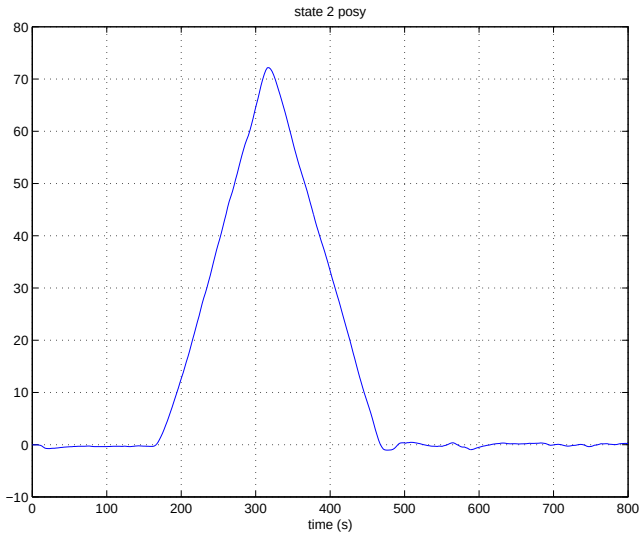


Figure 4.17: Y Position with Perfectly Known States

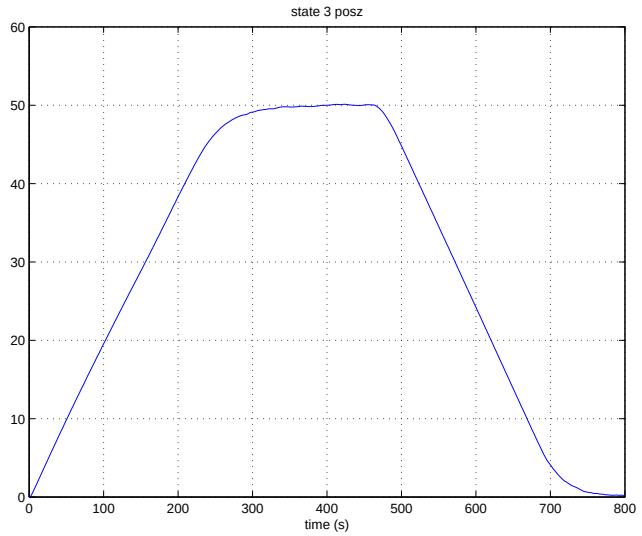


Figure 4.18: Z Position with Perfectly Known States

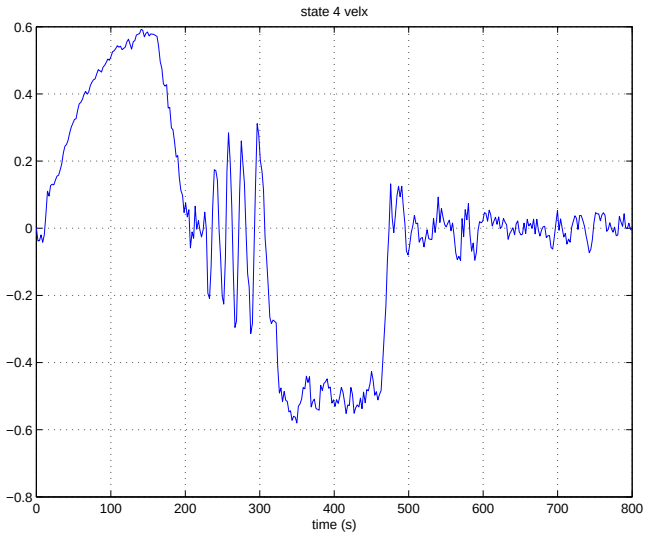


Figure 4.19: X Velocity with Perfectly Known States

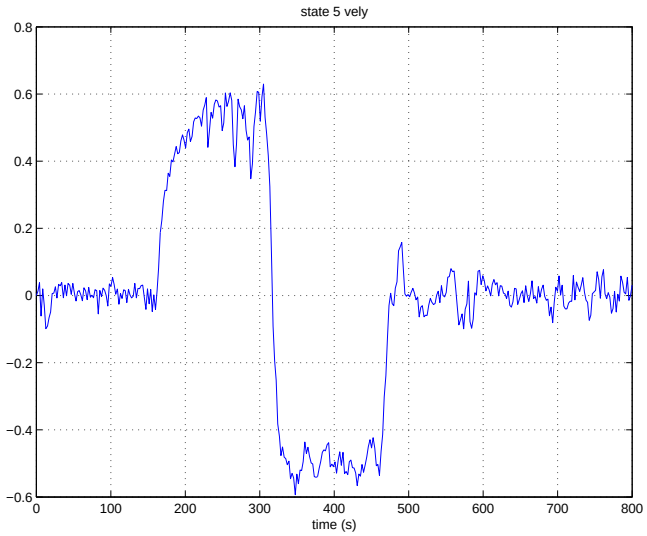


Figure 4.20: Y Velocity with Perfectly Known States

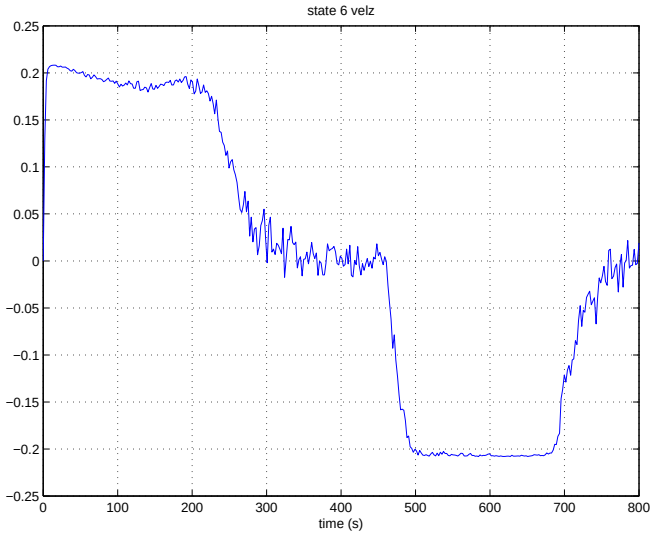
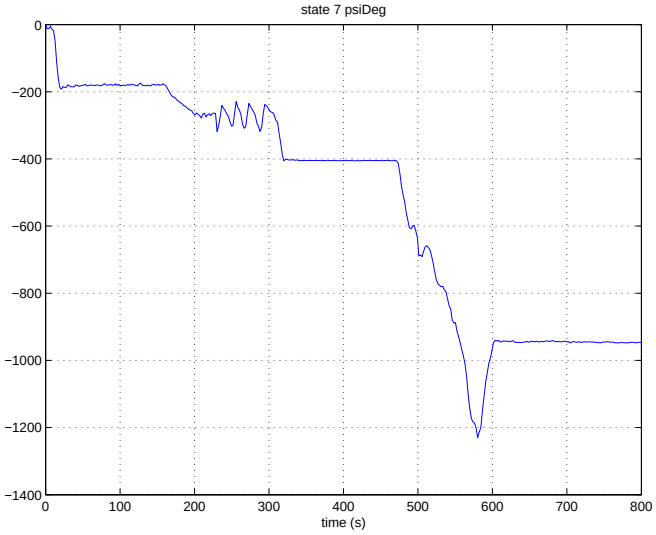


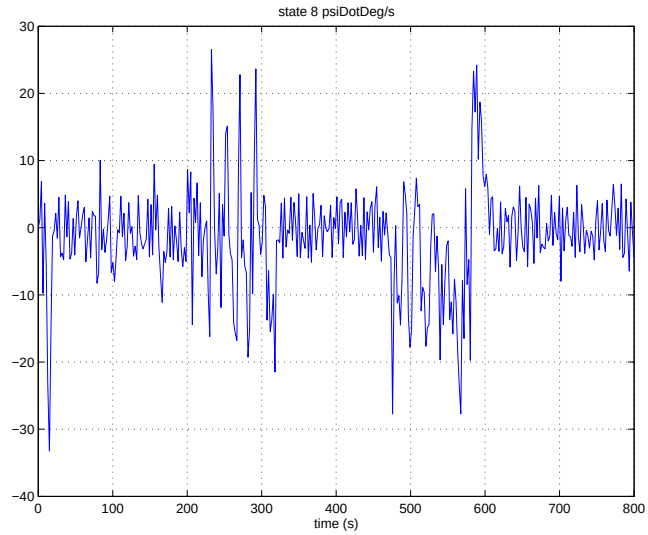
Figure 4.21: Z Velocity with Perfectly Known States

one input controls both x- and z-axis positioning, reaching the desired z coordinate is sacrificed to reach the x coordinate. The smooth positioning characteristics the blimp exhibits are due to aggressive regulation of velocity as shown in Figures 4.19-4.21. The blimp cannot converge to commanded positions without the regulation of yaw angle. In Figure 4.22a we show that the yaw angle converges to desired headings smoothly while the yaw rate, which plays a significant role in determining yaw angle, is much more aggressively controlled.

The regulation of system inputs determines the overall system performance. In Figure 4.24 we show that the thruster fan percentage is at 100 percent for almost the entire simulation. This means that converging to the waypoints in the specified amount of time is at the edge of the blimp's capabilities, but the blimp's translational motion is smooth. Yaw torque commands are applied much more aggressively than thruster fan power levels. The aggressive torque commands result in the blimp "wiggling" about its yaw axis rather than having a smooth yawing motion.



(a) Yaw Angle



(b) Yaw Rate

Figure 4.22: Yaw Angle and Rate with Perfectly Known States

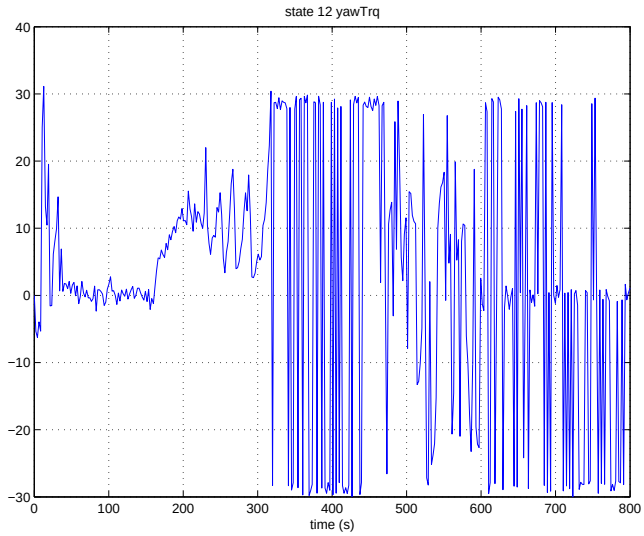


Figure 4.23: Yaw Torque Input with Perfectly Known States

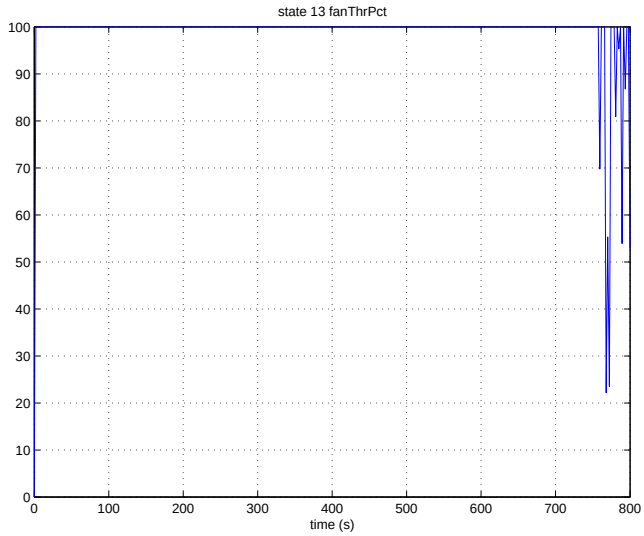


Figure 4.24: Thruster Fan Percentage Input with Perfectly Known States

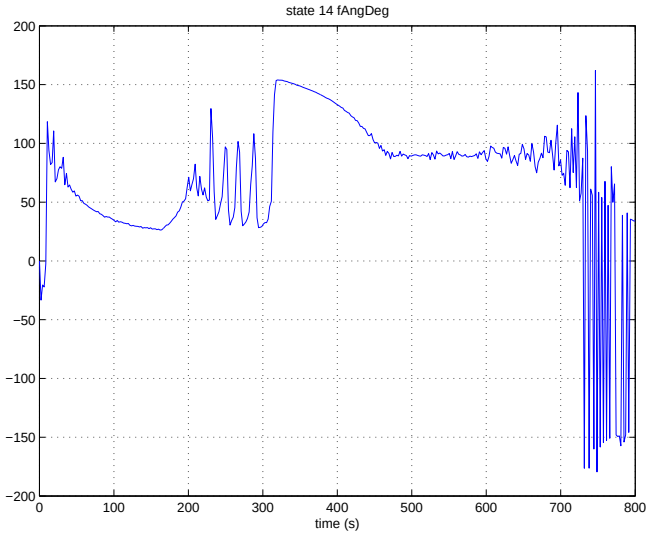


Figure 4.25: Thruster Fan Angle (Degrees) Input with Perfectly Known States

4.3.2 IMU Measured States

A simulation is performed to test the reliability of the IMU's predictions of state variables. The blimp is simulated using only models (developed in Section 4.1.2) of the sensors available on the IMU: three accelerometers, one yaw-rate gyro, and one ultrasonic range finder for altitude measurements. In the simulation all sensor biases are assumed to remain constant and are known. Thus, the biases are subtracted from the simulated IMU outputs before integration of the sensor readings. A moving average filter is also used to reduce the noise on the sensor readings. In Figures 4.26-4.28 we show the results of the simulation for the blimp's actual and measured position, in Figures 4.29-4.30 we show the actual and measured velocity, and in Figure 4.32 we show the actual and measured yaw angle and yaw rate.

As shown in Figures 4.29-4.31 and Figures 4.26-4.28, the IMU measurements give poor velocity and position estimates (the actual states are blue while the measured states

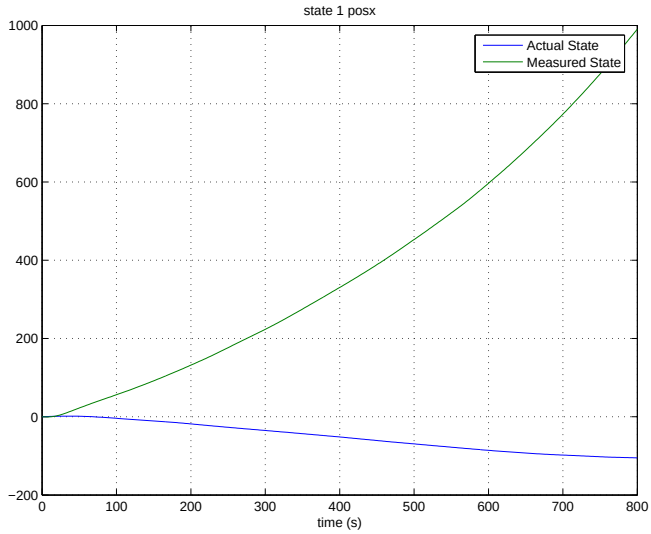


Figure 4.26: Actual and Measured X Position

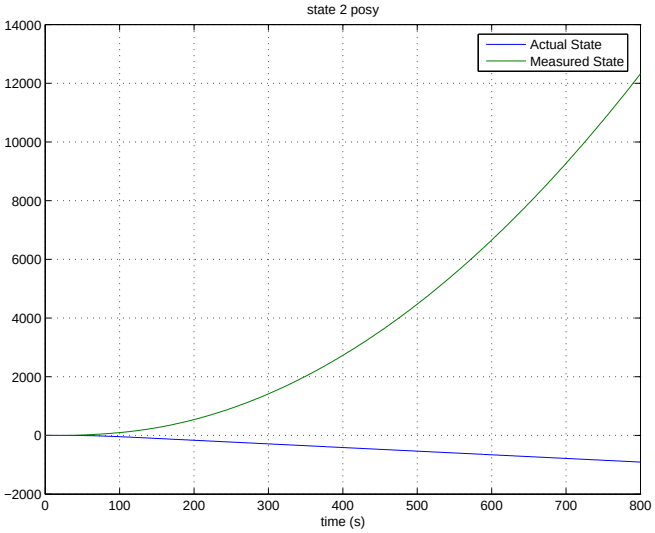


Figure 4.27: Actual and Measured Y Position

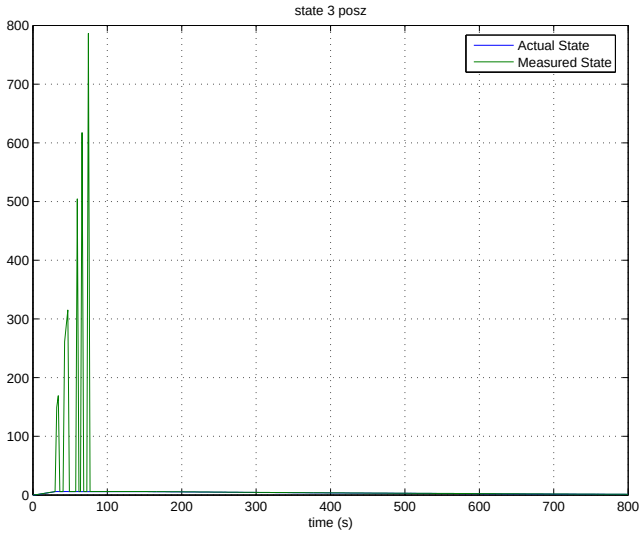


Figure 4.28: Actual and Measured Z Position

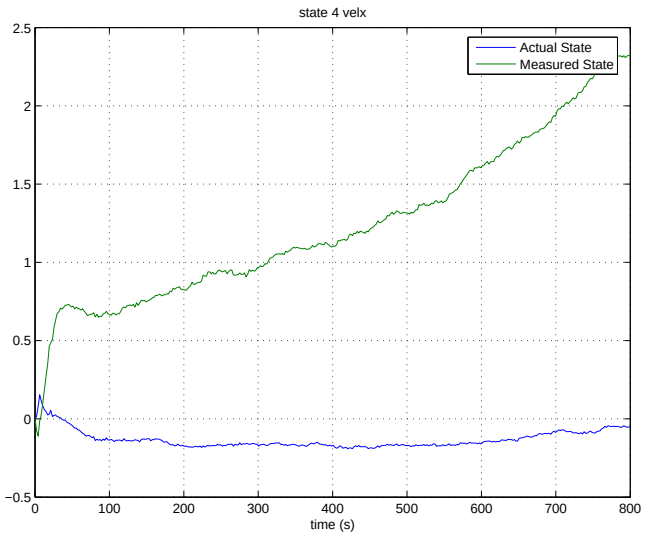


Figure 4.29: Actual and Measured X Velocity

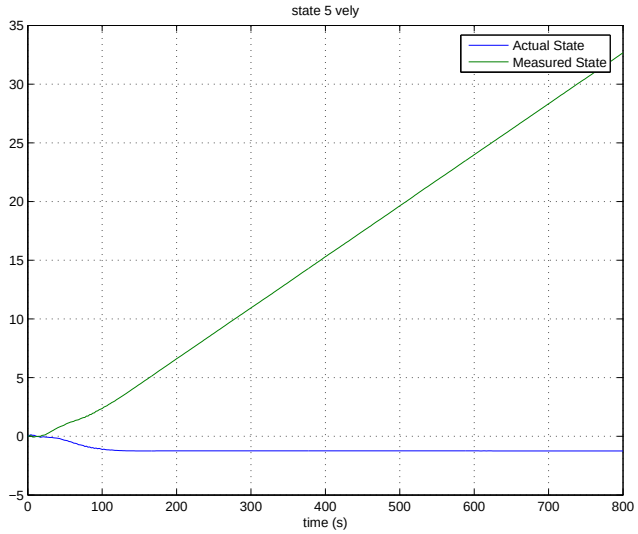


Figure 4.30: Actual and Measured Y Velocity

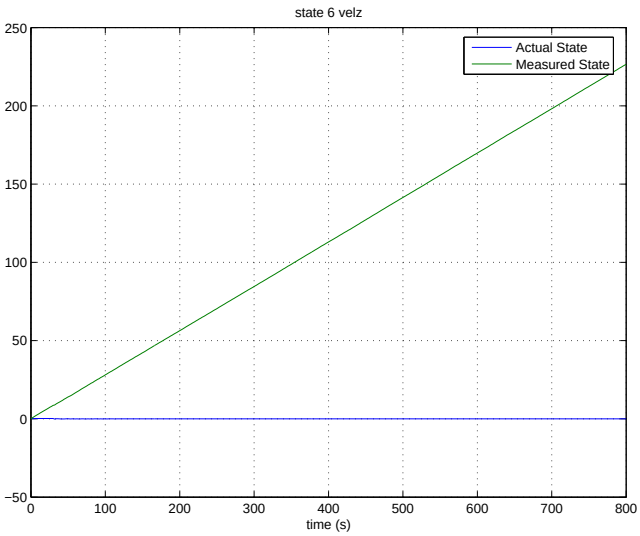
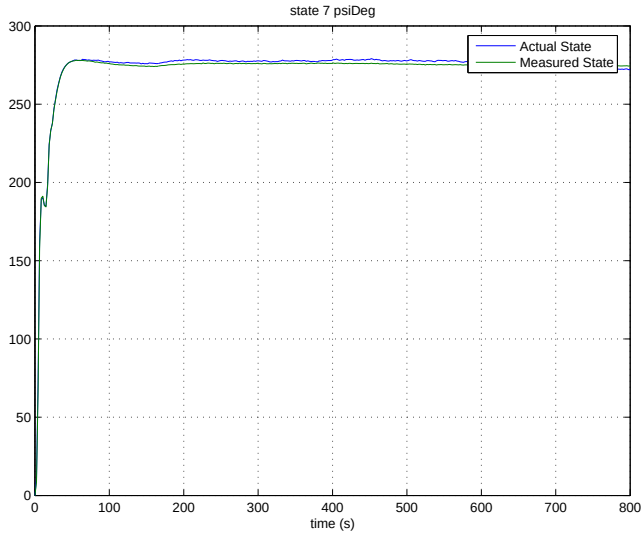
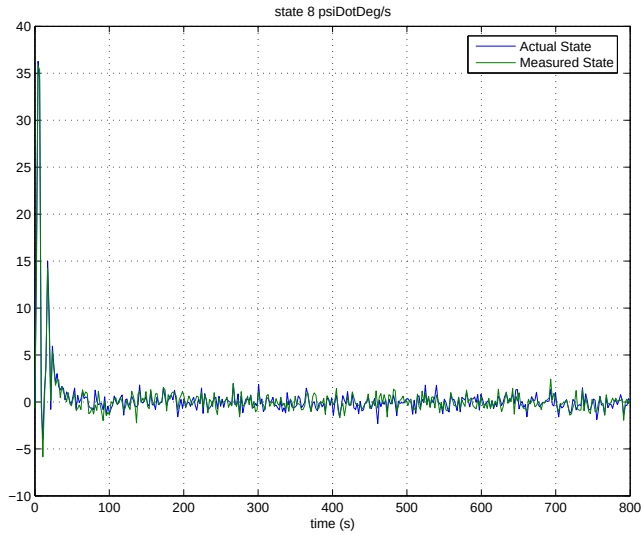


Figure 4.31: Actual and Measured Z Velocity



(a) Yaw Angle



(b) Yaw Rate

Figure 4.32: Actual and Measured Yaw Rate and Angle

are green). Integrating the IMU's acceleration measurements to get velocity results in the integration of noise as well as the actual acceleration value. This results in an error that accumulates as time progresses making the velocity estimates derived from the IMU unreliable after a short amount of time. The x-axis velocity becomes a poor estimate after about 1 s while the $y-$axis velocity is adequate until about 16 s where the error becomes over 100%. The z-axis velocity estimate is reliable only for a very short time. Poor velocity estimates result in poor position estimates since position is found by integrating velocity. The noise of the acceleration measurements is thus being integrated twice resulting in the position estimate's unbounded divergence from the actual state.

The disjunct z position estimate shown in Figure 4.28 is due to the presence of the ultrasonic range finder. Since the ultrasonic sensor data does not need to be integrated, the sonar's data is used rather than the z-axis accelerometer data when the blimp is less than 6 m from the ground. The blimp's actual z position is just less than 6 m, which is at the edge of the sonar's saturation level. The presence of noise on the sonar sometimes makes the sonar appear to at saturation level while at other times just below saturation. The points where the z-position estimate closely matches the actual position is where noise has not driven the sonar to saturation so the sonar estimate is used. Likewise, the points where the z-estimate largely diverges from the actual position is where noise has made the sonar appear to be saturated so the z-axis accelerometer data is used to determine position. This is why even though the z-axis velocity diverges quickly, the z-position estimate stays close to the actual position for periods of time.

Yaw angle and rate estimates are close to the actual yaw angle and rate as shown in Figure 4.32. Yaw rate is a state that is directly measurable by the yaw-axis gyro so there is no integration to cause error accumulation. Filtering the noisy gyro output results in

an estimate that closely matches the actual yaw rate. Integrating yaw rate produces yaw angle. Since the estimated yaw rate so closely matches the actual yaw rate, integration produces a small error; therefore, the estimated yaw angle closely matches the actual yaw angle. The small offset present eventually will accumulate over time and cause large yaw angle estimate errors.

The noisy IMU sensor readings cause unreliable state estimates after a very short time, which makes an IMU alone unsuitable for state estimation. Even small errors in measured accelerations result in large errors in velocity and even larger errors in position. Sudden changes in states would improve the IMU's state estimates, but the blimp's slow dynamics prevent this. Another sensor is needed to combine with the IMU to improve state estimates.

4.3.3 Combined Camera and IMU Estimated States

The system is simulated where it is assumed that a fixed-position camera system is tracking the blimp. The camera system gives a noisy measurement of the blimp's x, y, and z coordinates in the inertial frame. The noise is zero mean white noise with $\sigma^2 = 3$. The camera is assumed to operate at the same rate as the IMU. The results of the simulation are shown in Figures 4.33-4.39. In Figures 4.33-4.35 we show plots of the blimp's actual and measured position; in Figures 4.36-4.38 we show plots of the blimp's actual and measured position; and in Figure 4.39 we show plots of the blimp's actual and measured yaw angle and yaw rate.

The addition of the camera improves the blimp's performance over using just the IMU, but the results are still far from the results in the case where all of the states are perfectly known. As expected, the camera's position measurement is significantly

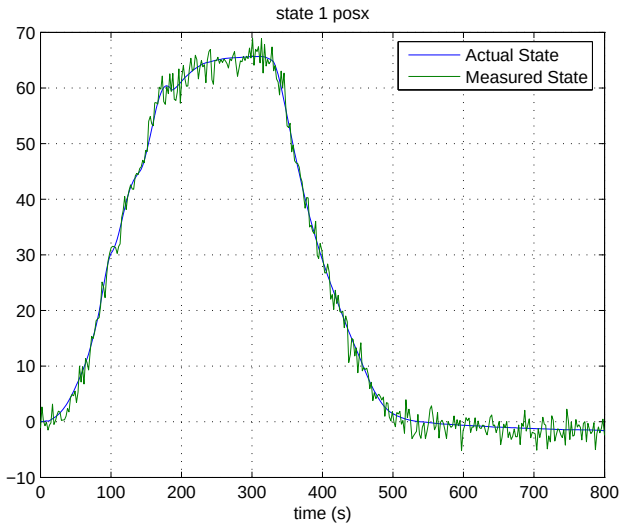


Figure 4.33: Camera Measured X Position

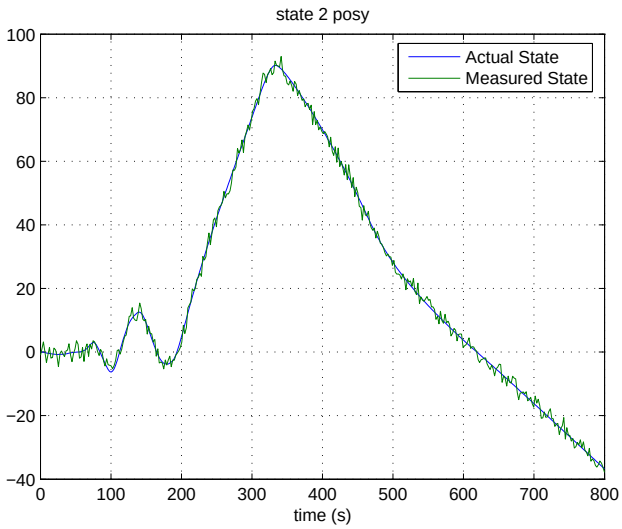


Figure 4.34: Camera Measured Y Position

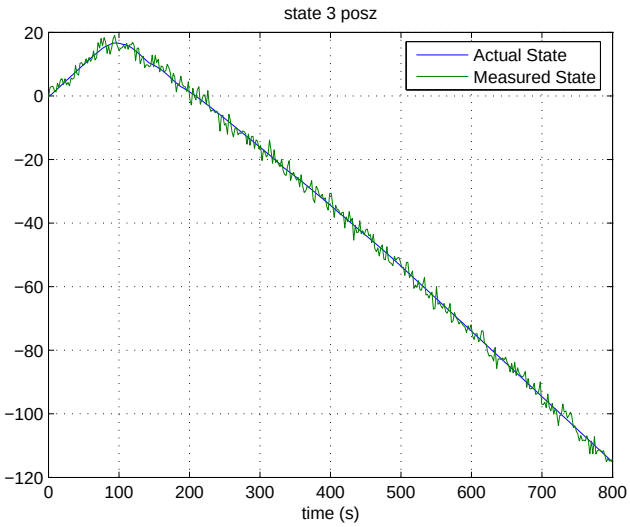


Figure 4.35: Camera Measured Z Position

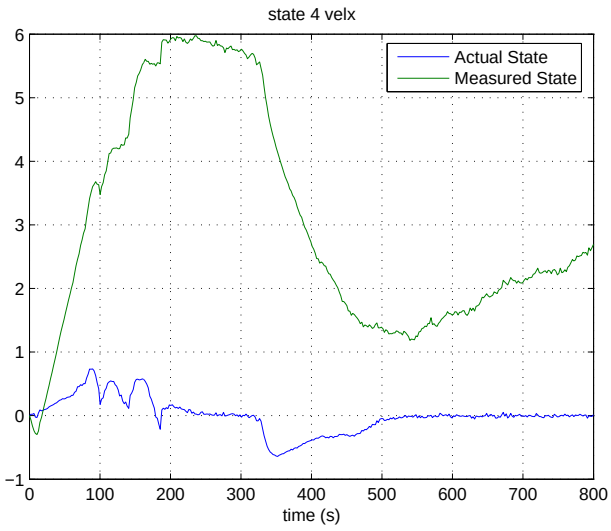


Figure 4.36: IMU Measured X Velocity

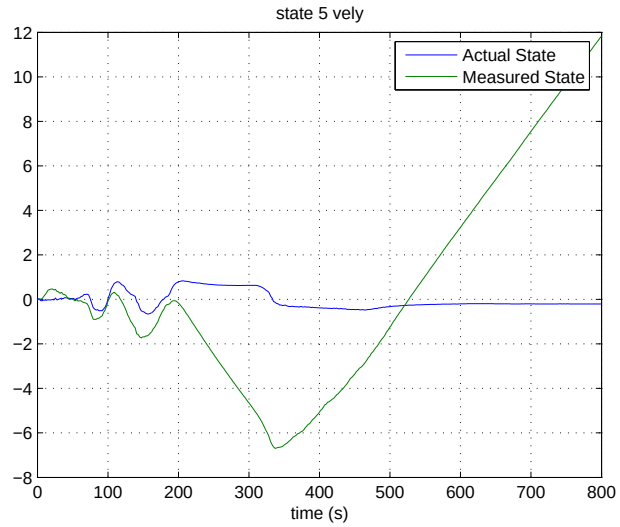


Figure 4.37: IMU Measured Y Velocity

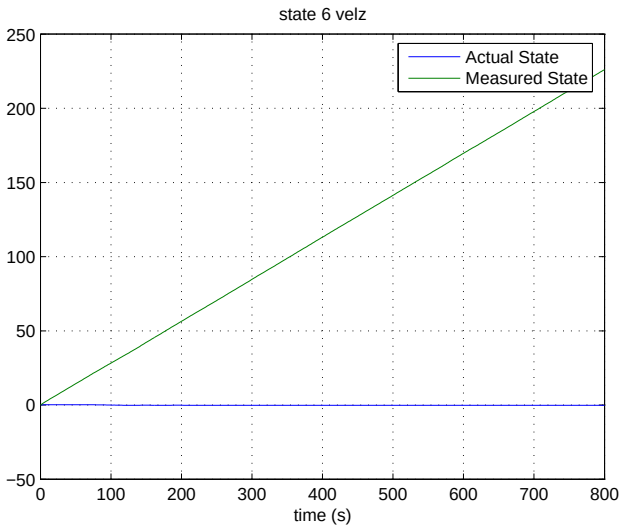
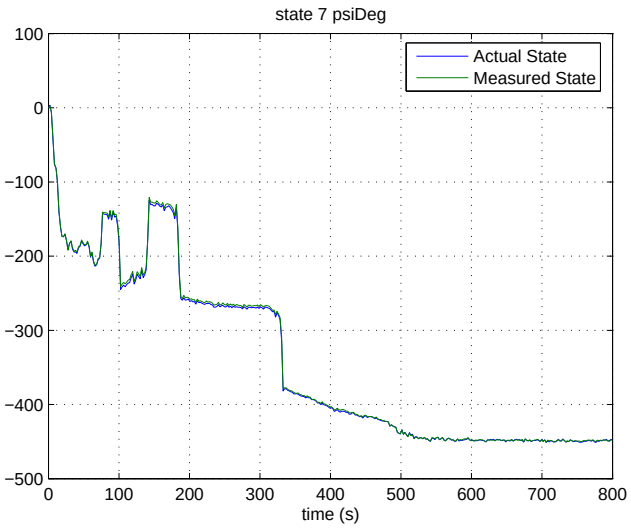
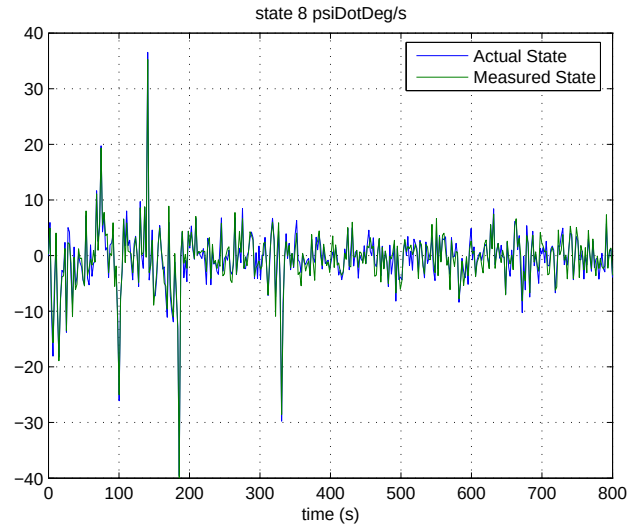


Figure 4.38: IMU Measured Z Velocity



(a) Yaw Angle



(b) Yaw Rate

Figure 4.39: IMU Measured Yaw Angle and Yaw Rate

more accurate than the IMU's position measurement. However, the blimp still fails to follow the desired path. This is because the control law depends on an accurate velocity estimate, which as seen in Figures 4.36-4.38 is poor. The camera is not used to get an estimate of velocity so the IMU estimate, which as discussed in Section 4.3.2 is unreliable after a short time, is the only available velocity information.

Chapter 5

Conclusion

In this chapter we present our concluding remarks. We begin by summarizing the content of each chapter in Section 5.1. In Section 5.2 we propose directions for future work.

5.1 Summary

In Chapter 2 we present a literature review of navigational technology. We begin by discussing the two basic kinds of IMUs, platform and strapdown, and showing that a strapdown IMU must be used when a vehicle such as an indoor blimp has an ultra-light payload capacity. Several kinds of mechanical gyros and accelerometers are next presented. We then explain the operation of sensors that are based on the Sagnac effect. Next, we present various MEMS sensors and explain their principles of operation. After completing our discussion of individual sensors, we survey various off-the-shelf IMUs and their suitability for use with an indoor blimp. We begin our survey with an overview of IMUs designed for military applications. We evaluate gun-hard IMUs based on weight, sensitivity, and cost and then evaluate general purpose military IMUs. This is followed by a presentation of IMUs designed for commercial use for applications such as robotics and aviation. Next, we discuss the increasingly common practice of integrating GPS readings with IMU readings to improve system performance. We conclude our overview of IMUs with a presentation of future technological developments. The chapter continues by presenting various technology used specifically for the navigation of blimps. We note

that many blimps use on-board vision rather than IMUs. We conclude the chapter by detailing several currently existing technological gaps which prevent the ideal IMU for an indoor blimp from being developed.

We present the design and construction of a low-cost, light-weight, low-g, IMU in Chapter 3. We begin the chapter by evaluating commercial off-the-shelf products that can be used to construct an IMU. We present several microprocessors that can possibly serve as an IMU's CPU. After comparing features, we chose a Microchip PIC18F4320 for the project. We then evaluate various gyroscopes based on their sensitivity to slow rotational rates. Next, accelerometers manufactured by MSI and Analog Devices are compared based on weight and power consumption. After we compare accelerometers, we evaluate various products that can be used as altimeters, ultimately choosing a Devantech SRF08 range finger. We then discuss in detail the SRF08's operation including its use of an I^2C bus for communication. Finally, we evaluate various power supplies, concluding that an alkaline battery is most suitable. With all hardware selected, we present design considerations of the IMU's circuit boards. In the final sections of Chapter 3, we present the software design for the IMU. We detail how data are gathered and processed from each sensor as well as how the data are transmitted to other devices.

In Chapter 4 we present the results of testing the constructed IMU and the results of simulating the blimp system. We open the chapter with a presentation of various tests performed on each sensor of the IMU. We test the gyroscopes and accelerometers by applying constant inputs and evaluating the outputs. The ultrasonic range finder is tested by recording sensor output when an obstacle is placed at various distances from the sensor. From the tests we conclude that the sensors are suitable to use on an indoor blimp. We then use the data from the tests to develop models for each sensor. Next, we

evaluate the validity of each model based on autocorrelation methods. We then develop a mathematical model of the blimp based on observational data. First, we present a model that uses a real-valued yaw angle. We additionally present a model based on a complex-valued yaw angle. The development of a guidance and control law are next presented. We then present simulation results. We first test the control law by assuming that all states are perfectly known. We then simulate the system using the modeled output of the designed IMU and then using an IMU in conjunction with a camera.

5.2 Future Directions

In this section we present work yet to be done. We begin by considering software and simulation research directions. We conclude with physical implementation possibilities.

An observer that combines IMU and camera measurements needs to be designed and implemented. An observer, such as an extended Kalman filter, could be used to get velocity estimates from the camera's position information. Another application of the observer could be to estimate sensor biases.

Simulating the system with various camera configurations has not been done. The system could be simulated with the camera having a different update rate than the IMU. Another possibility is to simulate a camera that gives only azimuth and elevation angles rather than actual position coordinates, i.e. assume a single camera rather than stereovision. A single camera would mean that new information about the blimp's position arises only when the blimp is not moving along the camera's line of sight.

The entire system has yet to be physically implemented. While the IMU has been built and tested, actual autonomous flight has never been attempted.

Bibliography

[1] Acroname Inc. *The Devantech SRF08 Ultrasonic Range Finder*, October 2002.

[2] AGNC. *AGNC coremicro AHRS/INS/GPS Specification*, 2003. June 28, 2005 http://www.americangnc.com/products/AHRS_INS_GPS_spec.pdf.

[3] Analog Devices. $\pm 150^{\circ}/s$ *Single Chip Rate Gyro Evaluation Board ADXRS150EB*, 2003. data sheet.

[4] Analog Devices. *Low Cost* $\pm 1.2g$ *Dual Axis Accelerometer ADXL213*, 2004.

[5] Analog Devices. $\pm 150^{\circ}/s$ *Single Chip Yaw Rate Gyro with Signal Conditioning ADXRS150*, 2004. data sheet.

[6] Archangel Systems, Inc. *IM*^{3}*-1 Preliminary Specifications*, b edition, 2004. June 28, 2005 http://www.archangel.com/Preliminary

[7] Neil Barbour and George Schmidt. Inertial sensor technology trends. *IEEE Sensors Journal*, 1(4):332–339, December 2001.

[8] Billur Barshan and Hugh F. Durrant-Whyte. Inertial navigation systems for mobile robots. *IEEE Transactions on Robotics and Automation*, 11(3):328–342, June 1995.

[9] J. Barton, A. Lynch, S. Bellis, B. O'Flynn, K. Mahmood, K. Delaney, and S. C. O'Mathuna. An inertial measurement unit (imu) for an autononous wireless sensor network. In 6^{th} *Electronics Packaging Technology Conference*, pages 586–589. EPTC, Dec. 8-10 2004.

[10] J. L. Campbell and J. T. Kresge. Brumby uav flight dynamics-instrumentation and fligth test results. In 22^{nd} *Digital Avionics Systems Conference*, volume 2, pages 8.A.2–10, Oct. 12-16 2003.

[11] D. Cardarelli. An integrated mems intertial measurement unit. In *Position Location and Navigation Symposium*, pages 314–319. IEEE, April 15-18 2002.

[12] Crossbow. *IMU300 DataSheet*, b edition.

[13] CrossBow. μ*NAV Datasheet*, a edition.

[14] Filipe Manuel da Silva Metelo and Luís Ricardo Garcia Campos. Vision based control of an autonomous blimp (Videoblimp). Final year thesis, Sep 2003.

[15] M. Faucheux, D. Fayoux, and J. J. Roland. The ring laser gyro. *Journal of Optics (Paris)*, 19(3):101–115, 1988.

[16] Xiufeng He, Yongqi Chen, and Jianye Liu. Development of a low-cost integrated gps/imu system. *Aerospace and Electronic Systems Magazine*, 3(12):7–10, December 1998.

[17] D. W. Heckman and M. Baratela. Interferometric fiber optic gyro technology. *Aerospace and Electronic Systems Magazine*, 15(12):23–28, December 2000.

[18] R. Hulsing. Mems intertial rate and acceleration sensor. *Aerospace and Electronic Systems Magazine*, 13(11):17–23, Nov 1998.

[19] David Kalinksky and Roee Kalinksy. Introduction to i2c. *Embedded Systems Programming*, 14(8), August 2001.

[20] D. A. Karnick. Low cost inertial measuring unit. In *Position Location and Navigation Symposium*, pages 422–425. IEEE, March 23-27 1992.

[21] Simon Lacroix, Il-Kyun Jung, Philippe Soueres, Emmanuel Hygounenc, and Jean-Paul Berry. The autonomous blimp project of laas/cnrs current status and research challenges. Technical report, LAAS/CNRS, 7, av. du Colonel Roche, F-Toulouse Cedex 4, France.

[22] Anthony Lawrence. *Modern Intertial Technology: Navigation, Guidance, and Control*. Springer-Verlag, 1993.

[23] Pei-Hwa Lo, D. Siebert, and H. T. Califano. Low cost fiber optic rate sensor inertial measurement unit. In *Position Location and Navigation Symposium*, pages 256–263. IEEE, April 20-23 1998.

[24] Jorge Lobo and Jorge Dias. Integration of inertial information with vision towards robot autonomy. *Proceedings of the 1997 IEEE International Symposium on Industrial Electronics*, pages 825–830, July 1997.

[25] Asad M. Madni, Deepak Bapna, Paul Levin, and Eric Krotkov. Solid-state six degree of freedon, motion sensor for field robotic applications. In *Proceedings of the 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1389–1398. IEEE/RSJ, October 1998.

[26] Microchip Technology Inc. *PIC18F2220/2320/4220/4320 Data Sheet*, 2003. data sheet.

[27] Miniature blimps for surveillance and collection of samples. *NASA Tech Briefs*, page 50, Jan 2004.

[28] MSI. *Accelerometer ACH-04-08-05*, August 1998. data sheet.

[29] D. J. Murphy. Characteristics of a small low cost inertial measurement unit. In *Proceedings of the 1998 Workshop on Autonomous Underwater Vehicles*, pages 75–87, August 20-21 1998.

[30] David J. Nagel and Mona E. Zaghloul. Mems: Micro technology, mega impact. *Circuits and Devices*, pages 14–25, March 2001.

[31] M. S. Perlmutter. A tactical fiber optic gyro with all-digital signal processing. In *Position Location and Navigation Symposium*, pages 170–175. IEEE, April 11-15 1994.

[32] Philips Semiconductor. *The I2C bus and how to use it (including specifications)*, April 1995.

[33] J. G. Ramos, E. C. de Paiva, J. R. Azinheira, S. S. Bueno, S. M. Maeta, L. G. B. Mirisola, M. Bergman, and B. G. Faria. Autonomous flight experiment with a robotic unmanned ariship. In *IEEE Conference on Robotics and Automation*, volume 4, pages 4152–4157. IEEE, 2001.

[34] I. W. Scaysbrook, S. J. Cooper, and E. T. Whitley. A miniature, gun-hard mems imu for guided projectiles, rockets, and missiles. In *Position Location and Navigation Symposium*, pages 26–34, April 26-29 2004.

[35] A. Warnasch and A. Killen. Low cost, high g, micro electro-mechanical systems (mems), inertial measurements unit (imu) program. In *Position Location and Navigation Symposium*, pages 229–305. IEEE, April 15-18 2002.

[36] Jason Welsby and Chris Melhuish. Autonomous minimalist following in three dimensions: A study with small-scale dirigibles. Technical report, Intelligent Autonomous Systems Engineering Laboratory, University of the West of England, Coldharbour Lane, Bristol, BS16 1QY, Mar 2001.

[37] Hong Zhang and James P. Ostrowski. Visual servoing with dynamics: Control of an unmanned blimp. In *IEEE Conference on Robotics and Automation*, May 1999.

[38] Jean-Christophe Zuffrey, Antoine Beyeler, and Dario Floreano. Vision-based navigation from wheels to wings. Technical report, Swiss Federal Institute of Technology in Lausanne (EPFL), CH-1015 Lausanne, Swizterland.

Appendices

Appendix A

PIC18F4320 Microcontroller Code

In this appendix we present the assembly code for the IMU's microcontroller. This code interfaces the IMU's three accelerometers, three gyros, and one ultrasonic range finder with a Microchip PIC18F4320. The program also outputs the IMU's data serially in ASCII format.

```
;*****************************************************************
; Author: Abby Anderson
; Company: Auburn University
; Last Revision: July 25, 2005
;
; *****************************************************************
;  This code interfaces 3 ADXRS150EB gyros manufactured
; by Analog Devices, two ADXL213 accelerometers manufactured
; by Analog Devices, and 1 Devantech SRF08 ultrasonic
; sensor with a PIC18F4320 manufactured by Microchip.
; These components are integrated to form a low-g IMU.
; The sonar is connected to the PIC through an IIC bus. The
; gyros are connected to ADCs. The accelerometers are digital
; inputs.
;*****************************************************************

LIST P=18F4320 ;directive to define processor
#include <P18F4320.INC>;processor specific variable
;definitions
```

```
errorlevel -207 ;suppresses label warning

;************************ RAM Variables *****************
FLAG equ 0x00 ;flag register
TEMP equ 0x01 ;register for ASCII routine
SWAP equ 0x02 ;register for ASCII routine
HI equ 0x03 ;result from ASCII routine
LO equ 0x04 ;result from ASCII routine

STATUS_TEMP equ 0x05
WREG_TEMP equ 0x06
BSR_TEMP equ 0x07

Buffer equ 0x10 ;start of unformatted data
;buffer
FormBuff equ 0x30 ;start of formatted data buffer
;*********************************************************
;*********************************************************
; This begins the RESET vector.
;*********************************************************
ORG 0x0000

goto Initialize ;go to register set up
```

```
;*********************************************************
; This begins the interrupt vector.
; Priority interrupt is off.
;*********************************************************
ORG 0x0008

ISR
banksel STATUS_TEMP
movff STATUS, STATUS_TEMP ;save STATUS register
movff WREG, WREG_TEMP ;save working register
movff BSR, BSR_TEMP ;save BSR register

banksel PIR1
btfsc PIR1, TMR1IF ;did Timer1 overflow?
goto SonarRead ;yes, read sonar data
goto Restore ;interrupt cause unknown

;*******Here a read operation from the sonar begins********
SonarRead
banksel T1CON
bcf T1CON, TMR1ON ;disables Timer1

banksel TMR1L
movlw 0x4F ;low byte of 65ms timer
```

```
movwf TMR1L ;for sonar

banksel TMR1H
movlw 0x61 ;high byte of 65ms timer
movwf TMR1H ;for sonar

banksel PIR1
bcf PIR1, TMR1IF ;clears timer1 interrupt flag

banksel SSPCON2
bsf SSPCON2, SEN  ;generates a start condition

call IICPoll ;delays until startup complete

banksel SSPBUF
movlw 0xE0 ;write to device at address 0xE0
movwf SSPBUF ;loads buffer with address

call  IICPoll ;delays until write is complete

banksel SSPBUF
movlw 0x02 ;address to be read
movwf SSPBUF
```

```
call  IICPoll ;delays until write is complete

banksel SSPCON2
bsf SSPCON2, RSEN  ;generates a restart condition

call IICPoll ;delays until startup complete

banksel SSPBUF
movlw 0xE1 ;read from sonar at address 0xE0
movwf SSPBUF ;loads buffer with address

call  IICPoll ;delays until write is complete

banksel SSPCON2
bsf SSPCON2, RCEN ;enables IIC reception mode

call  IICPoll ;delays until read is complete

banksel SSPBUF
movff SSPBUF, 0x20 ;puts contents in RAM

banksel SSPCON2
bcf SSPCON2, ACKDT ;clears acknowledge data bit
bsf SSPCON2, ACKEN ;sends an acknowledge signal
```

```
call  IICPoll ;delays until write is complete

banksel SSPCON2
bsf SSPCON2, RCEN ;enables IIC reception mode

call  IICPoll ;delays until read is complete

banksel SSPBUF
movff SSPBUF, 0x21 ;puts contents in RAM

banksel SSPCON2
bsf SSPCON2, ACKDT ;not acknowledge data bit
bsf SSPCON2, ACKEN ;sends a not acknowledge signal

call  IICPoll ;delays until write is complete

banksel SSPCON2
bsf SSPCON2, PEN  ;begins the stop condition

call IICPoll ;checks if stop condition
;transmitted

;******************** Sonar Section ****************************
```

```
Sonars
banksel SSPCON2
bsf SSPCON2, SEN ;generates an IIC start condition

call IICPoll ;delays until startup complete

banksel SSPBUF
movlw 0xE0 ;write to address 0xE0
movwf SSPBUF ;loads buffer with address

call IICPoll ;delays until write is complete

banksel SSPBUF
movlw 0x00 ;loads the buffer with address 0
movwf SSPBUF

call IICPoll ;delays until write is complete

movlw 0x51 ;command to get range in cm
movwf SSPBUF

call IICPoll ;delays until write is complete

banksel SSPCON2
```

```
bsf SSPCON2, PEN  ;begins the stop condition

call IICPoll ;checks if stop condition transmitted

banksel T1CON
bsf T1CON, TMR1ON ;starts Timer1 counting
;gives sonars the needed 65ms
Restore
banksel STATUS_TEMP
movff BSR_TEMP, BSR ;restore BSR register
movff WREG_TEMP, WREG ;restore working register
movff STATUS_TEMP, STATUS ;restore STATUS register

retfie

;*************************************************
; Code begins here.
;*************************************************

Initialize
banksel INTCON
movlw b'01000000';global interrupts disables
movwf INTCON ;external interrupt 0 enabled
```

```
banksel RCON
bcf RCON, IPEN ;disables interrupt priority

banksel INTCON3
clrf INTCON3 ;disables unnessary interrupts

banksel PIR1
clrf PIR1 ;clears peripheral interrupt
;flag register
banksel PIE1
bsf PIE1, TMR1IE ;enables Timer1 interrupt

banksel RCSTA
movlw b'10000000';configures pins for serial
movwf RCSTA ;port transmit

banksel TXSTA
movlw b'00000100';asynchronous, high Baud rate,
movwf TXSTA ;8-bit transmission (disabled)

banksel SPBRG
movlw d'129';gives Baud rate of 9600
movwf SPBRG
```

```
banksel SSPCON1
movlw b'00111000';master mode with serial port
movwf SSPCON1 ;enabled for IIC bus

banksel SSPADD
movlw 0x31 ;sets IIC Baud rate at 100kHz
movwf SSPADD

banksel SSPSTAT
movlw b'10000000';slew rate control disabled
movwf SSPSTAT

banksel SSPCON2
clrf SSPCON2 ;clears IIC state flags

banksel SSPBUF
clrf SSPBUF ;clear IIC data transmission register

banksel T1CON
movlw b'00110000';sets prescaler to 8
movwf T1CON

banksel TMR1L
movlw 0x4F ;low byte of 65ms timer
```

```
movwf TMR1L ;for sonar

banksel TMR1H
movlw 0x61 ;high byte of 65ms timer
movwf TMR1H ;for sonar

banksel TMR0H
clrf TMR0H

banksel TMR0L
clrf TMR0L

banksel T0CON
movlw b'10000111'
movwf T0CON

call TMR0Poll

banksel T0CON
bsf T0CON, TMR0ON

call TMR0Poll

banksel PORTA
```

```
clrf PORTA ;clears Port A data latches

banksel PORTB
clrf PORTB ;clears Port B data latches

banksel PORTC
clrf PORTC ;clears Port C data latches

banksel PORTE
clrf PORTE ;clears Port E data latches

banksel TRISA
movlw b'00101101';makes RA5 input for X-axis
movwf TRISA ;accel., RA2-RA3 input for
;gyros 1 and 2, RA0 an input
;for gyro 3
banksel TRISC
movlw b'10011001';sets port C serial comm direction
movwf TRISC

banksel TRISE
movlw b'00000011';makes RE2 an output for SCLK,
movwf TRISE ;RE0-RE1 inputs for y- and z-
;axis accelerometers
```

```
banksel ADCON1
movlw b'00001011';makes RA3, RA2, and RA0 analog
movwf ADCON1 ;inputs; rest inputs are digital

banksel ADCON2 ;left justified ADC, conversion
clrf ADCON2 ;rate is half of fosc

banksel FLAG
clrf FLAG ;clears the FLAG register

banksel INTCON
bsf INTCON, GIE ;enables global interrupts

;******************** Sonar Section ****************
; This section begins the first sonar write. After
; this the interrupt initializes all sonar actions.
;***************************************************
banksel SSPCON2
bsf SSPCON2, SEN ;generates an IIC start condition

call IICPoll ;delays until startup complete

banksel SSPBUF
movlw 0xE0 ;write to address 0xE0
```

```
movwf SSPBUF ;loads buffer with address

call IICPoll ;delays until write is complete

banksel SSPBUF
movlw 0x00 ;loads the buffer with address 0
movwf SSPBUF

call IICPoll ;delays until write is complete

movlw 0x51 ;command to get range in cm
movwf SSPBUF

call IICPoll ;delays until write is complete

banksel SSPCON2
bsf SSPCON2, PEN  ;begins the stop condition

call IICPoll ;checks if stop condition transmitted

banksel T1CON
bsf T1CON, TMR1ON ;starts Timer1 counting
;gives sonars the needed 65ms
```

```
Main
banksel FSR0
lfsr FSR0, 0x10 ;address of unformatted data buffer

banksel FSR1
lfsr FSR1, 0x30 ;address of formatted data buffer

;************* Accelerometer Section ***************
Accelerometer
banksel INTCON
bcf INTCON, GIE ;disables interrupts

banksel T0CON
movlw b'00010000';makes 16-bit timer with prescaler
movwf T0CON ;of 32

banksel TMR0H
movlw 0x00
movwf TMR0H ;clears Timer 0 high register

banksel TMR0L
clrf TMR0L ;clears Timer 0 low register

call Period1 ;gets the period of the PWM cycle
```

```
call Period2


banksel PORTA
btfsc PORTA, RA5 ;tests state of x-axis accelerometer
call HighDutyX ;calls routine to wait for low
;duty cycle
call LowDutyX ;calls routine to begin data
;acquisition
banksel PORTE
btfsc PORTE, RE0 ;tests state of y-axis accelerometer
call HighDutyY ;calls routine to wait for low duty
;cycle
call LowDutyY ;calls routine to begin data
;acquisition
banksel PORTE
btfsc PORTE, RE1 ;tests state of z-axis accelerometer
call HighDutyZ ;calls routine to wait for low duty
;cycle
call LowDutyZ


;*********** Gyro Section ************************
Gyro
banksel ADCON0
movlw b'00001001';selects Channel 2 AD converter
```

```
movwf ADCON0

nop ;gives time for capacitor
nop ;to charger
nop
nop

bsf ADCON0, GO ;begins AD conversion

call ADPoll ;waits for conversion to end

banksel ADCON0
movlw b'00001101';selects Channel 3 conversion
movwf ADCON0

banksel ADRESH
movff ADRESH, POSTINC0 ;puts high byte in buffer
banksel ADRESL
movff ADRESL, POSTINC0 ;puts low byte in buffer

bsf ADCON0, GO ;begins gyro 2 AD conversion

call ADPoll ;waits for conversion to end
```

```
banksel ADCON0
movlw b'00000001';selects channel 0 AD conversion
movwf ADCON0

banksel ADRESH
movff ADRESH, POSTINC0 ;puts high byte of gyro 2 in buffer
banksel ADRESL
movff ADRESL, POSTINC0 ;puts low byte of gyro 2 in buffer

bsf ADCON0, GO ;begins gyro 3 AD conversion

call ADPoll ;waits for conversion to end

banksel ADRESH
movff ADRESH, POSTINC0 ;puts high byte of gyro 3 in buffer
banksel ADRESL
movff ADRESL, POSTINC0 ;puts low byte of gyro 3 in buffer

banksel INTCON
bsf INTCON, GIE ;enables interrupts

banksel FSR0
lfsr FSR0, 0x10 ;address of unformatted data buffer
```

```
;******************** Formatted Data Buffer Load *************
BufferLoad
movff POSTINC0, TEMP
call ASCII ;converts data to ASCII format
movff HI, POSTINC1 ;puts converted dat in buffer
movff LO, POSTINC1

movlw 0x22
cpfseq FSR0L ;has all data been converted?
goto BufferLoad ;no, go convert more data
lfsr FSR1, 0x30 ;yes, reset buffer pointer

;*****************************************************************
; This section of code transmits the data readings in ASCII format.
;*****************************************************************
Transmit
banksel INTCON
bcf INTCON, GIE ;disables interrupts for transmit

banksel TXSTA
bsf TXSTA, TXEN ;enables transmit

movf POSTINC1, W ;ASCII value of 1st number
movwf TXREG
```

```
call  TransPoll

movf  POSTINC1, W ;ASCII value of 2nd number
movwf TXREG
call  TransPoll

movf POSTINC1, W ;ASCII value of 3rd number
movwf TXREG
call  TransPoll

movf POSTINC1, W ;ASCII value of 3rd number
movwf TXREG
call  TransPoll

movlw a',';ASCII character for a comma
movwf TXREG
call TransPoll

movlw 0x50
cpfseq FSR1L ;tests if all data been sent
goto Transmit

movf POSTINC1, W ;sends 1st number of sonar
movwf TXREG
```

```
call TransPoll

movf POSTINC1, W ;sends 2nd number of sonar
movwf TXREG
call TransPoll

movf POSTINC1, W ;sends 3rd number of sonar
movwf TXREG
call TransPoll

movf POSTINC1, W ;sends 4th number of sonar
movwf TXREG
call TransPoll

movlw a';';ASCII character for a semicolon
movwf TXREG
call TransPoll

movlw 0x0D ;ASCII value for a carriage return
movwf TXREG
call TransPoll

movlw 0x0A ;ASCII value for a line feed
movwf TXREG
```

```
call TransPoll

banksel TXSTA
bcf TXSTA, TXEN ;disables transmit

banksel INTCON
bsf INTCON, GIE ;enables interrupts

goto Main

;*********************************************************
; This function polls Timer 0 to see if it has overflowed.
;*********************************************************
TMR0Poll
banksel INTCON
btfss INTCON, TMR0IF ;has Timer 0 overflowed?
goto TMR0Poll ;no, test again
bcf INTCON, TMR0IF ;yes, clear Timer 0 flag
bcf T0CON, TMR0ON ;disable Timer 0

return

;************************************************************
; This routine test to see if data is still being transmitted
```

```
; over the IIC bus. No interrupt is generated, but the IIC
; flag bit is tested.
;*****************************************************************
IICPoll
banksel PIR1
btfss PIR1, SSPIF ;has the transmission ended?
goto IICPoll ;no, test again
bcf PIR1, SSPIF ;yes, clear flag and return

return

;*****************************************************************
; This routine polls to see if an AD conversion is complete.
;*****************************************************************
ADPoll
banksel PIR1
btfss PIR1, ADIF ;is conversion complete?
goto ADPoll ;no, test again
bcf PIR1, ADIF ;yes, clear flag bit

return

;*****************************************************************
; This routine polls to see if data transmission over the
```

```
; serial port is complete. The transmit flag is continuously
; polled.
;*************************************************************
TransPoll
banksel TXSTA
btfss TXSTA, TRMT ;is data being transmitted
goto TransPoll ;yes, test again

return

;*************************************************************
; This routine waits for the data line from the x-axis
; accelerometer to go low if it started in a high state.
;*************************************************************
HighDutyX
banksel PORTA
btfsc PORTA, RA5
goto HighDutyX
return

;*************************************************************
; This routine waits for the data line from the y-axis
; accelerometer to go low if it started in a high state.
;*************************************************************
```

```
HighDutyY
banksel PORTE
btfsc   PORTE, RE0
goto   HighDutyY
return

;*****************************************************************
; This routine waits for the data line from the z-axis
; accelerometer to go low if it started in a high state.
;*****************************************************************
HighDutyZ
banksel PORTE
btfsc PORTE, RE1
goto HighDutyZ
return

;*****************************************************************
; This routine finds the period of the PWM waveform of the
; x/y accel.
;*****************************************************************
Period1
banksel PORTA
btfsc PORTA, RA5 ;tests state of input
call HighDutyX ;if high, wait for it to go low
```

```
LowTest
btfss PORTA, RA5 ;tests state of input
goto LowTest ;if low, wait for it to go high

banksel T0CON
bsf T0CON, TMR0ON ;enables timer 0

call HighDutyX ;waits for input to go low
LowTest2
btfss PORTA, RA5 ;tests state of input
goto LowTest2 ;if low, wait for it to go high

banksel T0CON ;otherwise, disable timer0
bcf T0CON, TMR0ON

movf TMR0L, W

movff TMR0H, POSTINC0 ;puts high byte of period length in reg
movff TMR0L, POSTINC0 ;puts low byte of period length in reg

clrf TMR0H ;resets timer0 for next pass
clrf TMR0L

return
```

```
;*********************************************************
; This routine finds the period of the PWM waveform of
; the z accel.
;*********************************************************
Period2
banksel PORTE
btfsc PORTE, RE1 ;tests state of input
call HighDutyZ ;if high, wait for it to go low
LowTest3
btfss PORTE, RE1 ;tests state of input
goto LowTest3 ;if low, wait for it to go high

banksel T0CON
bsf T0CON, TMR0ON ;enables timer 0

call HighDutyZ ;waits for input to go low
LowTest4
btfss PORTE, RE1 ;tests state of input
goto LowTest4 ;if low, wait for it to go high

banksel T0CON ;otherwise, disable timer0
bcf T0CON, TMR0ON
```

```
movf TMR0L, W

movff TMR0H, POSTINC0 ;puts low byte of per len in reg
movff TMR0L, POSTINC0 ;puts high byte of per len in reg

clrf TMR0H ;resets timer0 for next pass
clrf TMR0L

return

;*****************************************************************
; This routine waits for a new period of the PWM input
; from the x-axis accelerometer to begin. When the data line
; goes high, timer 0 begins counting a determines the high
; time of the line.
;*****************************************************************
LowDutyX
banksel PORTA
btfss PORTA, RA5 ;tests state of input
goto LowDutyX ;input low, wait until high

banksel T0CON
bsf T0CON, TMR0ON ;enables timer 0
```

```
call HighDutyX ;waits for signal to come low
;again
banksel T0CON
bcf T0CON, TMR0ON ;disables timer 0

movf TMR0L, W

movff TMR0H, POSTINC0 ;puts high byte in accel reg
movff TMR0L, POSTINC0 ;puts low byte in accel reg

clrf TMR0H ;resets timer0 for next pass
clrf TMR0L

return

;*********************************************************
; This routine waits for a new period of the PWM input
; from the y-axis accelerometer to begin. When the data
; line goes high, timer 0 begins counting a determines
; the high time of the line.
;*********************************************************
LowDutyY
banksel PORTE
btfss PORTE, RE0 ;tests state of input
```

```
goto LowDutyY ;input low, wait until high

banksel T0CON
bsf T0CON, TMR0ON ;enables timer 0

call HighDutyY ;waits for signal to come low
;again
banksel T0CON
bcf T0CON, TMR0ON ;disables timer 0

movf TMR0L, W

movff TMR0H, POSTINC0 ;puts low byte in accel reg
movff TMR0L, POSTINC0 ;puts high byte in accel reg

clrf TMR0H ;resets timer0 for next pass
clrf TMR0L

return

;*************************************************************
; This routine waits for a new period of the PWM input
; from the x-axis accelerometer to begin. When the data
; line goes high, timer 0 begins counting a determines
```

```
; the high time of the line.
;*************************************************************
LowDutyZ
banksel PORTE
btfss PORTE, RE1 ;tests state of input
goto LowDutyZ ;input low, wait until high

banksel T0CON
bsf T0CON, TMR0ON ;enables timer 0

call HighDutyZ ;waits for signal to come low
;again
banksel T0CON
bcf T0CON, TMR0ON ;disables timer 0

movf TMR0L, W

movff TMR0H, POSTINC0 ;puts low byte in acceleration reg
movff TMR0L, POSTINC0 ;puts high byte in acceleration reg

clrf TMR0H ;resets timer0 for next pass
clrf TMR0L

return
```

```
;*******************************************************
; This routine takes numbers that represent data
; and reformats it to ASCII standards. It is assumed
; that only numbers will be passed to this routine.
; The code below works as it was tested with the file
; ASCII.asm.
;*******************************************************
ASCII
movff TEMP, SWAP ;puts a value in SWAP
swapf SWAP ;switches nibbles of SWAP
Pass
movlw 0x0F
andwf SWAP ;isolates a nibble
movlw 0x0A
subwf SWAP, W ;subtracts 0x0A from nibble

banksel STATUS
btfss STATUS, C ;tests if nibble is letr or num
goto Number
goto  Letter ;no borrow operation so is a letr

Number
banksel SWAP
```

```
movlw 0x30 ;ASCII code for zero
addwf SWAP
goto Next

Letter
banksel SWAP
movlw 0x37 ;ASCII code added to get a letr
addwf SWAP

Next
btfsc FLAG, 1 ;tests if high or low nibble
goto Finish
movff SWAP, HI ;puts result in a register

bsf FLAG, 1
movff TEMP, SWAP
goto  Pass
Finish
movff SWAP, LO
bcf FLAG, 1
return

END
```

Appendix B

Simulation Code

In this appendix we present the Matlab code used to simulate the blimp. We begin in Section B.1 by presenting the code used to simulate the blimp with perfectly known states. Then in Section B.2 we present the code used to simulate the blimp when the states are measured by an IMU. Finally, in Section B.3 we present the code used to simulate the blimp when the states are measured by an IMU and a camera system.

B.1 Code for Perfectly Known States Simulations

A Simulink block diagram of a simulation for perfectly known states is shown in Figure B.1. Noise can be added to the model by disconnecting the constant zero value from the summing junction and connecting the noise block. In Figure B.1 the blocks labeled blimpGuide (the guidance law), blimpCntr (the control law), blimpModel (the blimp model), and blimpObs (the system observer) are Matlab s-functions. The code for blimpGuide follows.

```
function [sys,x0,str,ts] = blimpGuide(t,x,u,flag)
% inputs (11):
%   blimp state (8)
%   blimp destination waypoint (x,y,z) (3)
% states (0):
% outputs (3): thrust command (for attitude control/fan control);
%   forces to apply to blimp in x-y-z coordinates (inertial frame; z up)

switch flag,
  case 0, [sys,x0,str,ts]=mdlInitializeSizes;
```

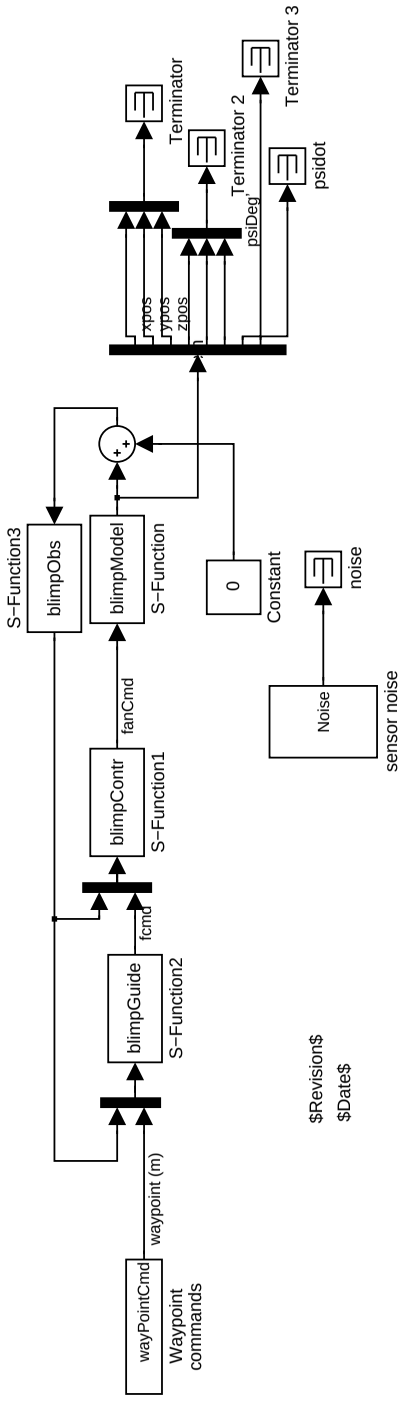


Figure B.1: Block Diagram of Simulation with Perfectly Known States

```
  case 1, sys=mdlDerivatives(t,x,u);
  case 2, sys=mdlUpdate(t,x,u);
  case 3, sys=mdlOutputs(t,x,u);
  case 4, sys=mdlGetTimeOfNextVarHit(t,x,u);
  case 9, sys=mdlTerminate(t,x,u);
  otherwise error(['Unhandled flag = ',num2str(flag)]);
end

% ----- All functions below this line are "local" ----

function [sys,x0,str,ts]=mdlInitializeSizes(Ts)
  sizes = simsizes;
  sizes.NumContStates  = 0; % see comments above
  sizes.NumDiscStates  = 0;
  sizes.NumOutputs     = 3;
  sizes.NumInputs      = 11;
  sizes.DirFeedthrough = 1;
  sizes.NumSampleTimes = 1;
  sys = simsizes(sizes);

  x0  = [];
  str = [];
  ts  = [0.1 0];  % 10 Hz
```

```
return


%=========================================
% mdlDerivatives
function xDot=mdlDerivatives(t,x,u)
  xDot = [];
return


%=========================================
% mdlUpdate
function xNext=mdlUpdate(t,x,u);
  xNext = [];
return


%=========================================
% mdlOutputs
function yy=mdlOutputs(t,x,u,Cd)
  %   blimp state (8)
  %   blimp destination waypoint (x,y,z) (3)
  bPos = u(1:3,1);
  bVel = u(4:6,1);
  wPos = u(9:11,1);  % destination waypoint
  psiRad = u(7)*pi/180;
```

```
  % control law parameters
  c1 = -4/20;  % 20 sec settling  time in position
  c2 = -4/5;  % 5 sec settling time in velocity
  a = [-4/30; -4/5; -4/5]; % blimp natural friction coefficient
  bt = 1; % blimp force-> acceleration coefficient

  xe = wPos - bPos;      % compute position error xe
  vcmd = -c1*xe;

  ve = vcmd - bVel;      % compute velocity error ve

  % get air friction forces
  cc = cos(psiRad);
  ss = sin(psiRad);
  i2b = [ cc,ss,0; -ss,cc,0; 0,0,1 ];
  fCmd = (-c2*ve - ((i2b')*diag(a)*i2b - c1*eye(3))* bVel)/bt;

  yy = fCmd;   % output is force command
return; % end mdlOutputs

%=============================================
% mdlGetTimeOfNextVarHit
function sys=mdlGetTimeOfNextVarHit(t,x,u)
  % not used
```

```
  sampleTime = 1;
  sys = t + sampleTime;
return; % end mdlGetTimeOfNextVarHit

%=============================================
% mdlTerminate
function sys=mdlTerminate(t,x,u)
  sys = [];
return; % end mdlTerminate

function y = fixAngle(x);
% normalize an angle between +/- 180 degrees
xrad = x*pi/180;
y = 180*atan2(sin(xrad),cos(xrad))/pi;
return
```

The s-function for the block blimpCntr is

```
function [sys,x0,str,ts] = blimpControl(t,x,u,flag)
% inputs (11):
%   blimp state (8)
%   blimp force command (x,y,z) (3) [from guidance law]
% states (0):
% outputs (3): commands to the blimp
```

```
switch flag,
  case 0, [sys,x0,str,ts]=mdlInitializeSizes;
  case 1, sys=mdlDerivatives(t,x,u);
  case 2, sys=mdlUpdate(t,x,u);
  case 3, sys=mdlOutputs(t,x,u);
  case 4, sys=mdlGetTimeOfNextVarHit(t,x,u);
  case 9, sys=mdlTerminate(t,x,u);
  otherwise error(['Unhandled flag = ',num2str(flag)]);
end

% ----- All functions below this line are "local" ----

function [sys,x0,str,ts]=mdlInitializeSizes(Ts)
  sizes = simsizes;
  sizes.NumContStates  = 0;
  sizes.NumDiscStates  = 0;
  sizes.NumOutputs     = 3;
  sizes.NumInputs      = 11;
  sizes.DirFeedthrough = 1;
  sizes.NumSampleTimes = 1;
  sys = simsizes(sizes);

  x0  = [];
```

```
  str = [];
  ts  = [0.1 0];  % 10 Hz

return

%=====================================
% mdlDerivatives
function xDot=mdlDerivatives(t,x,u)
  xDot = [];
return

%=====================================
% mdlUpdate
function xNext=mdlUpdate(t,x,u);
  xNext = [];
return

%=====================================
% mdlOutputs
function yy=mdlOutputs(t,x,u,Cd)
  %   blimp state (8)
  %   blimp destination waypoint (x,y,z) (3)
  bPsiDeg = fixAngle(u(7));
  bRdegs = u(8);
```

```
fCmd = u(9:11,1);


% controller parameters
c3 = -4/20;
c4 = -4/5;
a2 = -4/10;
J = 1;


% rotate force command into yaw=0 body frame
psiRad = bPsiDeg*pi/180;
fCmd = fCmd .* [-1;1;-1];
cc = cos(psiRad);  ss = sin(psiRad);
fbCmd = [cc,ss,0; -ss,cc,0; 0,0,1] * fCmd;


% select yaw command
psiCmd = (180/pi)*atan2(fCmd(2),fCmd(1));
psiErr = fixAngle(psiCmd - bPsiDeg);


% select rCmd -> psi settles in 30 sec
rDegCmd = -c3*psiErr;
re = rDegCmd - bRdegs;


yawTrq = J*( -(c3 + a2)*bRdegs - c4*re);
```

```
  % choose fan angle to match angle of force command
  fAngRad = atan2(-fbCmd(3),fbCmd(1));
  fAngDeg = fAngRad*180/pi;

  % now choose fan thrust by least squares:
  % min | fThr | subject to
  %  fDir*fThr = fCmd - yawF(y)
  cc = cos(fAngRad);
  ss = sin(fAngRad);
  fth = [cc;0;-ss];
  ell = min(1,max(0,fth'*fbCmd));

  yy = [yawTrq; ell*100; fAngDeg];

return; % end mdlOutputs

%=============================================
% mdlGetTimeOfNextVarHit
function sys=mdlGetTimeOfNextVarHit(t,x,u)
  sampleTime = 1;
  sys = t + sampleTime;
return; % end mdlGetTimeOfNextVarHit

%=============================================
```

```
% mdlTerminate
function sys=mdlTerminate(t,x,u)
  sys = [];
return; % end mdlTerminate


function y = fixAngle(x);
% normalize an angle between +/- 180 degrees
xrad = x*pi/180;
y = 180*atan2(sin(xrad),cos(xrad))/pi;
return
```

The code for blimpModel is

```
function [sys,x0,str,ts] = blimpModel(t,x,u,flag,myX0)
% inputs (3):
%   yaw torque (N-m)
%   propulsion fan throttle (percent)
%   propulsion fan angle (degrees)
% states (8):
%   position (x,y,z) (inertial frame) (meters)
%   velocity (x,y,z) (m/s)
%   yaw angle (degrees)
%   yaw rate (deg/sec)
% outputs (11): all of the states + acceleration.
```

```
switch flag,
  case 0, [sys,x0,str,ts]=mdlInitializeSizes(myX0);
  case 1, sys=mdlDerivatives(t,x,u);
  case 2, sys=mdlUpdate(t,x,u);
  case 3, sys=mdlOutputs(t,x,u);
  case 4, sys=mdlGetTimeOfNextVarHit(t,x,u);
  case 9, sys=mdlTerminate(t,x,u);
  otherwise error(['Unhandled flag = ',num2str(flag)]);
end

% ----- All functions below this line are "local" ----
% Only blimpModel() can see them and use them.  They are called
% by Simulink when it selects the right value for -flag-.

function [sys,x0,str,ts]=mdlInitializeSizes(myX0)
  sizes = simsizes;
  sizes.NumContStates  = 8;
  sizes.NumDiscStates  = 0;
  sizes.NumOutputs     = 8;
  sizes.NumInputs      = 3;
  sizes.DirFeedthrough = 0;
  sizes.NumSampleTimes = 1;
  sys = simsizes(sizes);
```

```
  x0  = myX0;  % start at rest at the origin
  str = [];
  ts  = [0 0];  % continuous time system

return

%=============================================================
% mdlDerivatives
function xDot=mdlDerivatives(t,x,u)

  % model parameters
  m = 1;  % mass
  J = 1;  % moment of inertia
  rLen = 3; % radius from cg to yaw fan

  a1 = -4/30;  % 30 sec settling time friction in body x-axis
  a2 = -4/5;   % 5 sec settling time friction in body y-axis and z-axis
  a3 = -4/10;  % 10 sec settling time friction in yaw rotation axis.

  % extract inputs and position variables
  pos = x(1:3,1); % position
  vel = x(4:6,1);  % velocity
  psiRad = x(7)*pi/180;  % angular position - in radians
```

```
omRad = x(8)*pi/180;  % om -> omega, angular velocity



% limit inputs
u = min(max(u,[-0.5;0;-180]),[0.5;100;180]);
yawTorque = u(1);
fpct = u(2);  % fan power level in percent
fangDeg = u(3); % fan angle in degrees.

%implement differential equations
posDot = vel;
psiRadDot = omRad;

% put velocity vector into blimp frame of reference

cc = cos(psiRad);
ss = sin(psiRad);

% rotate velocity vector to blimp frame of reference
i2b = [cc,ss,0; -ss,cc,0; 0,0,1]; % inertial to body rotation
vb = i2b*(vel .*[-1;1;-1]);    % rotate in horizontal plane

% fvb: force from velocity in body frame
fvb = diag([a1,a2,a2])*vb;
```

```
  % tvb: torque from velocity in body frame
  tvb = a3*vb(2);

  % now rotate body frame force back into inertial frame
  fvi = (i2b'*fvb) .* [-1;1;-1];

  fAR = fangDeg*pi/180;
  fthrustx  = cos(fAR)*fpct/600;  % force in kg.
  fthrusty = yawTorque/rLen;
  fthrustz = - sin(fAR)*fpct/600;  % force in kg. (z is down!)
  fbf = [fthrustx; fthrusty; fthrustz];  % fan force - body frame
  fif = (i2b'*fbf).*[-1;1;-1];  % fan force - inertial frame

  velDot = (fif + fvi)/m;
  omRadDot = (tvb + omRad*a3 + yawTorque)/J;  % fixed lack of
   % damping here


  % change units on angle
  psiDegDot = psiRadDot*180/pi;
  omDegDot = omRadDot*180/pi;
  xDot = [posDot; velDot; psiDegDot; omDegDot];
return
```

```
%=============================================

% mdlUpdate

function xNext=mdlUpdate(t,x,u);

  xNext = [];

return


%=============================================

% mdlOutputs

function yy=mdlOutputs(t,x,u,Cd)

  yy = [x];

return; % end mdlOutputs


%=============================================

% mdlGetTimeOfNextVarHit

function sys=mdlGetTimeOfNextVarHit(t,x,u)

  sampleTime = 1;

  sys = t + sampleTime;

return; % end mdlGetTimeOfNextVarHit


%=============================================

% mdlTerminate

function sys=mdlTerminate(t,x,u)

  sys = [];
```

```
return; % end mdlTerminate
```

The code for the block blimpObs is

```
function [sys,x0,str,ts] = blimpObs(t,x,u,flag)
% inputs (8):
%   blimp state (8)
% states (24):  % record of the last three measurements
% outputs (8): estimate of state vector

switch flag,
  case 0, [sys,x0,str,ts]=mdlInitializeSizes;
  case 1, sys=mdlDerivatives(t,x,u);
  case 2, sys=mdlUpdate(t,x,u);
  case 3, sys=mdlOutputs(t,x,u);
  case 4, sys=mdlGetTimeOfNextVarHit(t,x,u);
  case 9, sys=mdlTerminate(t,x,u);
  otherwise error(['Unhandled flag = ',num2str(flag)]);
end

% ----- All functions below this line are "local" ----

function [sys,x0,str,ts]=mdlInitializeSizes(Ts)
  sizes = simsizes;
```

```
  sizes.NumContStates  = 0;
  sizes.NumDiscStates  = 24;
  sizes.NumOutputs     = 8;
  sizes.NumInputs      = 8;
  sizes.DirFeedthrough = 1;
  sizes.NumSampleTimes = 1;
  sys = simsizes(sizes);

  x0  = zeros(24,1);
  str = [];
  ts  = [0.1 0];  % 10 Hz

return

%=============================================
% mdlDerivatives
function xDot=mdlDerivatives(t,x,u)
  xDot = [];
return

%=============================================
% mdlUpdate
function xNext=mdlUpdate(t,x,u);
  % record the last three measurements of the state x.
```

```
  xk = u;                  % current state measurement
  xk1 = x(1:8,1);
  xk2 = x(8+(1:8),1);
  xk3 = x(16+(1:8),1);



  xNext = [xk; xk1; xk2];
return

%=============================================
% mdlOutputs
function yy=mdlOutputs(t,x,u,Cd)
  %   blimp state (8)
  %   blimp destination waypoint (x,y,z) (3)

  xk = u;                  % current state measurement
  xk1 = x(1:8,1);
  xk2 = x(8+(1:8),1);
  xk3 = x(16+(1:8),1);
  yy = mean([xk,xk1,xk2,xk3]');   % FIR filter of state information

return; % end mdlOutputs

%=============================================
```

```
% mdlGetTimeOfNextVarHit
function sys=mdlGetTimeOfNextVarHit(t,x,u)
  sampleTime = 1;
  sys = t + sampleTime;
return; % end mdlGetTimeOfNextVarHit

%=============================================
% mdlTerminate
function sys=mdlTerminate(t,x,u)
  sys = [];
return; % end mdlTerminate

function y = fixAngle(x);
% normalize an angle between +/- 180 degrees
xrad = x*pi/180;
y = 180*atan2(sin(xrad),cos(xrad))/pi;
return
```

The waypoint commands given by the block wayPointCmd in Figure B.1 are generated by code called blimpMain that runs the entire simulation. This code generates a movie of the blimp's behavior and plots the blimp's state information. The code is as follows:

```
clc
tic; % start the timer running
```

```
if( ~exist('x0') )
  x0 = zeros(8,1);  % need this to run the simulation
end
% select the waypoint
tEnd = 800.0;
%tEnd = 30;
dt = tEnd/(15*25);  % 25 sec @ 30 frames persecond
% sequence of waypoints and times; each row is [tk, x(tk), y(tk), z(tk)]
pWaypoint = [0, 0,0,0; ...
             10, 0,0,50; ...
             160, 75,0,50; ...
             310, 75,75,50; ...
             460, 0,0,50; ...
             610, 0,0,0; ...
             1000, 0,0,0];
[ts,xx,observed,actual] = sim('blimpSim1',0:dt:tEnd);

% compute commands from guidance and control
for nn=1:length(ts)
  xn = xx(nn,:)';
  tWayp = pWaypoint(min(find(pWaypoint(:,1)>=ts(nn))),:);
  gc = blimpGuide(ts,[],[xn;tWayp(2:4)'],3);
  cc = blimpContr(ts,[],[xn;gc],3);
  xx(nn,8+(1:3)) = gc';
```

```
  xx(nn,11+(1:3)) = cc';
end


fn = 0;  % figure number
statenames = { 'posx', 'posy', 'posz', 'velx',...
  'vely', 'velz', 'psiDeg', 'psiDotDeg/s', 'fcx',...
  'fcy,' 'fcz', 'yawTrq', 'fanThrPct', 'fAngDeg' };
for ii=1:length(statenames)
  figure(ii);
  if( ii<9)
    plot(ts,xx(:,ii),'-',ts,observed(:,ii));
    legend('Actual State', 'Measured State');
  else
    plot(ts,xx(:,ii),'-');
  end
  title(sprintf('state %d %s',ii,statenames{ii}));
  grid on
  xlabel('time (s)');
  fn = fn+1; eval(sprintf('print -depsc ...
   blimpMainIMU-%.2d.eps',fn));
end
 blimpSim1
 print -depsc -s'blimpSim1' blimpSim1.eps
```

```
% generate AVI movie
makeMovie = 0;
if(makeMovie)
  blimpData = [ts,xx(:,[1:3,7])];
  visBlimp
else
  !touch blimMovie.avi
end
```

B.2 Code for IMU Measured States Simulations

The Simulink block diagram for simulating the blimp using IMU measured states is shown in Figure B.2. The blocks with the same names as the blocks in Figure B.1 have the same code as presented in Section B.1. The block labeled sensorModel contains the following code that models the IMU's sensor behavior.

```
function [sys,x0,str,ts] = sensorModel(t,x,u,flag)
% inputs (11):
% states (8):
%   position (x,y,z) (inertial frame) (meters)
%   velocity (x,y,z) (m/s)
%   yaw angle (degrees)
%   yaw rate (deg/sec)
% outputs (5): sensor measurements
```

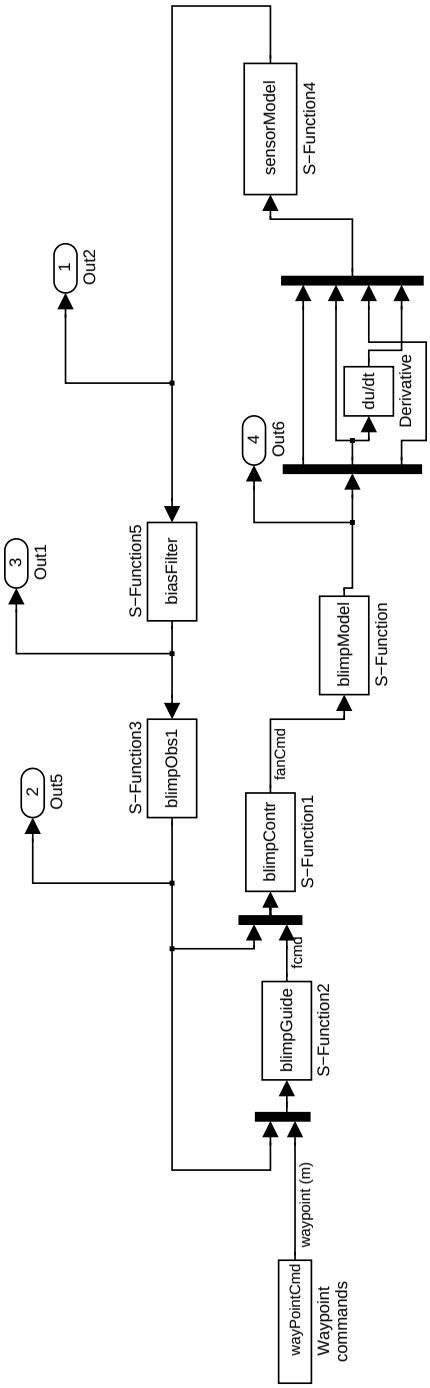


Figure B.2: Block Diagram of Simulation with IMU Measured States

```
%   body frame acceleration (m/s^2)
%   altitude (m)
%   yaw rate (deg/sec)

switch flag,
  case 0, [sys,x0,str,ts]=mdlInitializeSizes;
  case 1, sys=mdlDerivatives(t,x,u);
  case 2, sys=mdlUpdate(t,x,u);
  case 3, sys=mdlOutputs(t,x,u);
  case 4, sys=mdlGetTimeOfNextVarHit(t,x,u);
  case 9, sys=mdlTerminate(t,x,u);
  otherwise error(['Unhandled flag = ',num2str(flag)]);
end

function [sys,x0,str,ts]=mdlInitializeSizes(Ts)
  sizes = simsizes;
  sizes.NumContStates  = 0; e
  sizes.NumDiscStates  = 0;
  sizes.NumOutputs     = 5;
  sizes.NumInputs      = 11;
  sizes.DirFeedthrough = 1;
  sizes.NumSampleTimes = 1;
  sys = simsizes(sizes);
```

```
  x0  = [];  % start at rest at the origin
  str = [];
  ts  = [1/8.4 0];  % sampling time of 8 Hz

return

%=============================================================
% mdlDerivatives
function xDot=mdlDerivatives(t,x,u)
  xDot = [ ];
return

%=============================================================
% mdlUpdate
function xNext=mdlUpdate(t,x,u);

  xNext = [ ];

return

%=====================================================================
% mdlOutputs
function yy=mdlOutputs(t,x,u,Cd)
  psiDeg = u(7);
```

```
  psiRad = pi*psiDeg/180;
  cc = cos(psiRad); ss = sin(psiRad);
  i2b = [cc,ss,0; -ss,cc,0; 0,0,1];
  accelBody = i2b*u(9:11)/9.8;   %body frame acceleration in g's

%Since theblimp is modeled as neutrally bouyant,
%gravity has no effect.
  xAccel = accelBody(1) - 0.0048906 + 0.00044971*randn(1);
  yAccel = accelBody(2) + 7.34250e-4 + 0.00034547*randn(1);
  zAccel = accelBody(3) + 0.032194 + 0.0037683*randn(1);

  accel = [xAccel; yAccel; zAccel]*9.8; %convert from g's to m/s^2

  sonar = u(3)*100 - 0.1404 + 0.7336*randn(1); %sonar output in cm
  sonar = sonar/100;    %sonar output in m;

  yawRate = u(8) - 9.79251 + 0.39243*randn(1); %output in deg/s

%Limit the sensor outputs to appropriate range
  gLimit = 11.76*ones(3,1); %accels have limit of +/- 1.2g
  accelModel = max(min(accel,gLimit),-gLimit);

  sonarModel = max(min(sonar,6),0); %sonar has a range of 0-6m
```

```
  gyroModel = max(min(yawRate,150),-150);   %gyro has a range of +/- 150


  yy = [accelModel; sonarModel; gyroModel];
return; % end mdlOutputs


%=============================================
% mdlGetTimeOfNextVarHit
function sys=mdlGetTimeOfNextVarHit(t,x,u)
  sampleTime = 1;
  sys = t + sampleTime;
return; % end mdlGetTimeOfNextVarHit


%=============================================
% mdlTerminate
function sys=mdlTerminate(t,x,u)
  sys = [];
return; % end mdlTerminate


function y = fixAngle(x);
% normalize an angle between +/- 180 degrees
xrad = x*pi/180;
y = 180*atan2(sin(xrad),cos(xrad))/pi;
return
```

The block biasFilter of Figure B.2 removes the constant bias term present in the sensors as follows.

```
function [sys,x0,str,ts] = biasFilter(t,x,u,flag)
% inputs (8):
%   sensor measurements
%   accelerometers (x,y,z) (body frame) (m/s^2)
%   sonar (m)
%   gyro (yaw rate) (deg/sec)
% states (0):
% outputs (5): sensor measurements filtered

switch flag,
  case 0, [sys,x0,str,ts]=mdlInitializeSizes;
  case 1, sys=mdlDerivatives(t,x,u);
  case 2, sys=mdlUpdate(t,x,u);
  case 3, sys=mdlOutputs(t,x,u);
  case 4, sys=mdlGetTimeOfNextVarHit(t,x,u);
  case 9, sys=mdlTerminate(t,x,u);
  otherwise error(['Unhandled flag = ',num2str(flag)]);
end

% ----- All functions below this line are "local" ----
```

```
function [sys,x0,str,ts]=mdlInitializeSizes(Ts)

  sizes = simsizes;

  sizes.NumContStates  = 0; % see comments above

  sizes.NumDiscStates  = 0;

  sizes.NumOutputs     = 5;

  sizes.NumInputs      = 5;  % see comments above

  sizes.DirFeedthrough = 1;

  sizes.NumSampleTimes = 1;   % at least one sample time is needed

  sys = simsizes(sizes);


  x0  = zeros(1,50);

  str = [];

  ts  = [-1 0];  % 8.4 Hz


return


%=============================================================

% mdlDerivatives

function xDot=mdlDerivatives(t,x,u)

  xDot = [];

return


%=============================================================
```

```
% mdlUpdate
function xNext=mdlUpdate(t,x,u);

  xNext = [];

return

%=============================================================
% mdlOutputs
function yy=mdlOutputs(t,x,u,Cd)

  bias = [-0.0048906; 7.34250e-4; 0.032194; -0.1404; -9.79251];
  xk = u - bias;                  % current state measurement

  yy = [xk];

return; % end mdlOutputs

%=============================================
% mdlGetTimeOfNextVarHit
function sys=mdlGetTimeOfNextVarHit(t,x,u)
  sampleTime = 1;
  sys = t + sampleTime;
return; % end mdlGetTimeOfNextVarHit
```

```
%================================================
% mdlTerminate
function sys=mdlTerminate(t,x,u)
  sys = [];
return; % end mdlTerminate
```

The block blimpObs1 of Figure B.2 converts unbiased sensor measurements into a state vector as shown in the code that follows.

```
function [sys,x0,str,ts] = blimpObs1(t,x,u,flag)
% inputs (5):
%   sensor measurements
%   accelerometers (x,y,z) (body frame) (m/s^2)
%   sonar (m)
%   gyro (yaw rate) (deg/sec)
% states (7):
% outputs (8): estimate of state vector

switch flag,
  case 0, [sys,x0,str,ts]=mdlInitializeSizes;
  case 1, sys=mdlDerivatives(t,x,u);
  case 2, sys=mdlUpdate(t,x,u);
  case 3, sys=mdlOutputs(t,x,u);
```

```
  case 4, sys=mdlGetTimeOfNextVarHit(t,x,u);
  case 9, sys=mdlTerminate(t,x,u);
  otherwise error(['Unhandled flag = ',num2str(flag)]);
end

% ----- All functions below this line are "local" ----

function [sys,x0,str,ts]=mdlInitializeSizes(Ts)
  sizes = simsizes;
  sizes.NumContStates  = 0;
  sizes.NumDiscStates  = 7;
  sizes.NumOutputs     = 8;
  sizes.NumInputs      = 5;
  sizes.DirFeedthrough = 1;
  sizes.NumSampleTimes = 1;
  sys = simsizes(sizes);

  x0  = zeros(1,7);
  str = [];
  ts  = [1/8.4 0];  % 100 Hz

return

%=============================================
```

```
% mdlDerivatives
function xDot=mdlDerivatives(t,x,u)
  xDot = [];
return

%=============================================
% mdlUpdate
function xNext=mdlUpdate(t,x,u);

  yawDeg = fixAngle(x(7));
  yawRad = pi*yawDeg/180;

  cc = cos(yawRad); ss = sin(yawRad);
  i2b = [cc,ss,0; -ss,cc,0; 0,0,1];
  accelInert = i2b'*u(1:3);

  vel = accelInert/8.4 + x(1:3);

  yawAngle = u(5)/8.4 + x(7);

  position = x(1:3)/8.4 + x(4:6);

  xNext = [ vel; position; yawAngle];
return
```

```
%================================================
% mdlOutputs
function yy=mdlOutputs(t,x,u,Cd)

  pos_a = x(4:6);
  vel = x(1:3);

  if( (u(4) < 6) )
      height = u(4);
  else
      height = pos_a(3);
  end

  pos = [pos_a(1); pos_a(2); height];

  yy = [pos; vel; x(7); u(5)];

return; % end mdlOutputs

%================================================
% mdlGetTimeOfNextVarHit
function sys=mdlGetTimeOfNextVarHit(t,x,u)
  sampleTime = 1;
```

```
  sys = t + sampleTime;
return; % end mdlGetTimeOfNextVarHit

%=============================================
% mdlTerminate
function sys=mdlTerminate(t,x,u)
  sys = [];
return; % end mdlTerminate

function y = fixAngle(x);
% normalize an angle between +/- 180 degrees
xrad = x*pi/180;
y = 180*atan2(sin(xrad),cos(xrad))/pi;
return
```

B.3 Code for Camera and IMU Measured States Simulations

In Figure B.3 we show the block diagram used to simulate the blimp when its states are estimated by a camera system and an IMU. The block labeled camera1 models the output of a camera system and is implemented with the following code.

```
function [sys,x0,str,ts] = camera1(t,x,u,flag)
% inputs (5):
%   sensor measurements
```

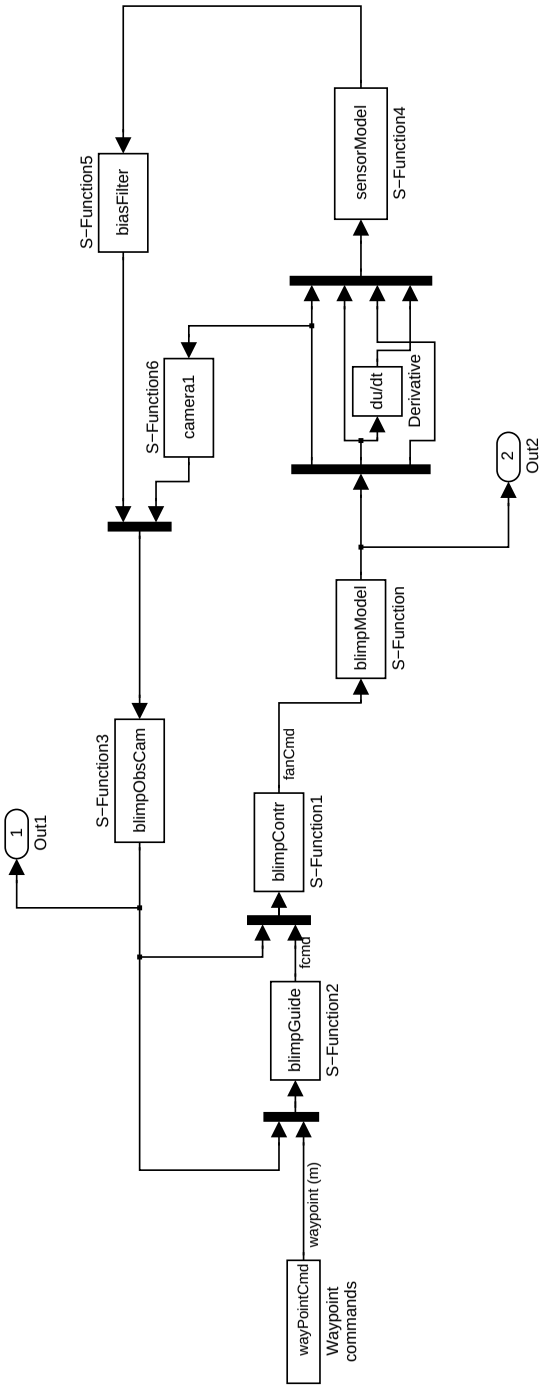


Figure B.3: Block Diagram of Simulation with States Measured by an IMU and a Camera System

```
%   accelerometers (x,y,z) (body frame) (m/s^2)
%   sonar (m)
%   gyro (yaw rate) (deg/sec)
% outputs (3): estimate of position

switch flag,
  case 0, [sys,x0,str,ts]=mdlInitializeSizes;
  case 1, sys=mdlDerivatives(t,x,u);
  case 2, sys=mdlUpdate(t,x,u);
  case 3, sys=mdlOutputs(t,x,u);
  case 4, sys=mdlGetTimeOfNextVarHit(t,x,u);
  case 9, sys=mdlTerminate(t,x,u);
  otherwise error(['Unhandled flag = ',num2str(flag)]);
end

% ----- All functions below this line are "local" ----

function [sys,x0,str,ts]=mdlInitializeSizes(Ts)
  sizes = simsizes;
  sizes.NumContStates  = 0;
  sizes.NumDiscStates  = 0;
  sizes.NumOutputs     = 3;
  sizes.NumInputs      = 3;
  sizes.DirFeedthrough = 1;
```

```
  sizes.NumSampleTimes = 1;
  sys = simsizes(sizes);

  x0  = [];
  str = [];
  ts  = [0 0];

return

%=============================================
% mdlDerivatives
function xDot=mdlDerivatives(t,x,u)
  xDot = [];
return

%=============================================
% mdlUpdate
function xNext=mdlUpdate(t,x,u);
  xNext = [ ];
return

%=============================================
% mdlOutputs
function yy=mdlOutputs(t,x,u,Cd)
```

```
  xCam = u(1)+sqrt(3)*randn(1);
  yCam = u(2)+sqrt(3)*randn(1);
  zCam = u(3)+sqrt(3)*randn(1);

  yy = [xCam; yCam; zCam];

return; % end mdlOutputs

%=============================================
% mdlGetTimeOfNextVarHit
function sys=mdlGetTimeOfNextVarHit(t,x,u)
  sampleTime = 1;
  sys = t + sampleTime;
return; % end mdlGetTimeOfNextVarHit

%=============================================
% mdlTerminate
function sys=mdlTerminate(t,x,u)
  sys = [];
return; % end mdlTerminate
```