AN ABSTRACT OF THE THESIS OF

John H. Nguyen for the degree of Honors Baccalaureate of Science in Computer Engineering presented on March 16th, 2007. Title: Autonomous Helicopter Systems: Design and Implementation Solutions

Abstract Approved: ______________________________

Mario E. Magaña

Creating intelligent robotics is at the cutting edge of technology. Unmanned aerial vehicles (UAV) are of particular interest for surveillance applications. UAVs have the potential to provide quick and cheap aerial data that would be difficult to obtain by conventional means. UAVs also have the ability to navigate without being hampered by ground terrain, which presents an enormous advantage over land-based, autonomous vehicles. Achieving autonomy, especially in small-scale aircraft, continues to be a difficult task.

A basic design for an autonomous helicopter based on the DraganFly Innovations DraganFlyer V Ti is explored. The design, implementation, and results of the undergraduate engineering project, called the AeroBot, will be analyzed with the purpose of providing a foundation for future projects relating to this topic. The concept behind achieving autonomy through sensor integration is explored, followed by a breakdown of the AeroBot design, and finally an investigation into the results of the project. The information should provide practical insight into the construction of an autonomous, aerial vehicle for future endeavors in this field of engineering.

Key Words: Autonomous Helicopter, Artificial Intelligence, Unmanned Aerial Vehicle (UAV), Robotics, Autonomous Vehicles

Corresponding E-mail Address: nguyenjo@engr.oregonstate.edu

Autonomous Helicopter Systems: Design and Implementation Solutions

by

John H. Nguyen

A PROJECT

submitted to

Oregon State University

University Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Computer Engineering (Honors Scholar)

Presented March 16th, 2007

Commencement June 2007

Honors Baccalaureate of Science in Computer Engineering project of John H. Nguyen

presented on March 16th, 2007

APPROVED:

____

Mentor, representing Electrical & Computer Engineering

____

Committee Member, representing Electrical & Computer Engineering

____

Committee Member, representing Mechanical Engineering

____

Dean, University Honors College

____

John H. Nguyen, Author

AKNOWLEDGEMENTS

First of all, I would like to thank Tripod Data Systems (TDS) in Corvallis for their generous support. Their financial backing of the project made it possible. I would especially like to thank Paul Harmon of TDS for his interest and immediate support for this project. His guidance was priceless. Secondly, I would like to thank Mike from Spectrolutions for dedicating hours of his time to help decode and redesign the helicopter. He never hesitated to help regardless of time constraints. Third, I would like to thank my project partners Hai Yue Han and Jason George. This project was realized through our collective effort, talent, and sacrifice. Fourth, I would like to thank Dr. Mario E. Magaña for his mentorship while writing my thesis. His enthusiasm to innovate and to go beyond “cookbook solutions” is infectious. Without professors like him, few students would have the desire or confidence to engineer dreams into reality. Finally, because behind every good man is an even better woman, I would like to thank Amanda. Her never-ending understanding of the countless hours of work, late nights, and missed dates necessary for this project is more than appreciated. Thank you.

TABLE OF CONTENTS

LIST OF FIGURES

1. Introduction

The idea of blending automation with aeronautics has always been on the minds of those interested in flight. Even the earliest pioneers of flight, such as the Wright Brothers and Sir Hiram Maxim in the early 20th century, endeavored to make much of flying autonomous due to the inherent instability and difficulty of manual flight [4]. Today, the evolution and advancement of technology has made many aspects of flight autonomous to assist a pilot in his or her mission. The next logical step is to create completely autonomous aircraft. With applications such as research, exploration, and reconnaissance, the possibilities for these devices are endless.

Unfortunately, completely autonomous flight faces several obstacles especially in the realm of small-scale helicopter systems. Difficulties in achieving autonomy usually stem from the inability to implement autonomous stabilization and accurate movement tracking. Helicopter-based, autonomous vehicles tend to drift in orientation and are very sensitive to wind forces. As a result, tracking techniques such as dead reckoning require an abundance of sensors to help compensate for drift to keep the vehicle on course. Nevertheless, the creation of autonomous helicopters is possible as long as an array of redundant, internal and external sensors is utilized. The goal of the AeroBot 2006 design project was to create a unmanned aerial vehicle (UAV) by enhancing a radio controlled (RC) helicopter to demonstrate this feasibility.

The 2006 AeroBot project involved integrating and programming the necessary components to convert an RC helicopter to be completely autonomous. The objective was to create an end product that could fly to a programmed location, capture image data, return to the initial location and land without human intervention. To create the autonomous helicopter system, the DraganFlyer V Ti by DraganFly Innovations was chosen as the platform for the design. This is an RC helicopter system with four individually controlled motors for stability. Creating autonomous behavior required the addition of redundant sensors to the helicopter to achieve autonomous stabilization by simulating a sense of kinesthesia. A microcontroller was added to manage the sensors, the acquisition of sensor data and the control systems.

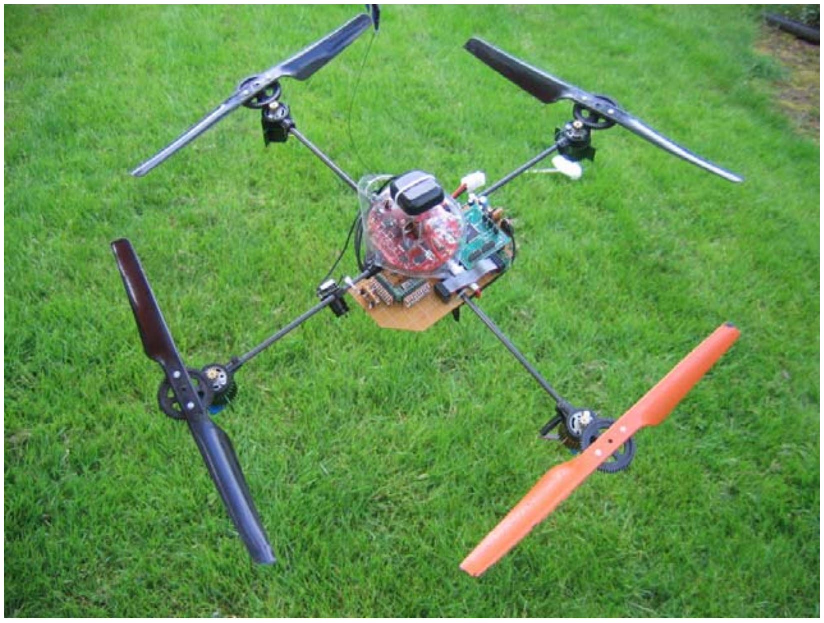


Figure 1: AeroBot

Although the AeroBot project did not achieve all of the initial goals, it still demonstrates the plausibility of creating autonomous helicopter systems. For now, the AeroBot serves as a model for small-scale autonomous vehicle designs. Both the successes and mistakes of the AeroBot project provide designers with insight into the construction and design of such vehicles.

2. Autonomy: Fundamentals and Design Considerations

An essential aspect of an effective UAV is its ability to navigate and interact with its environment. This means that it can perform all of its normal function independent of its conditions. To achieve this level of intelligence, one must provide a means by which the UAV can acquire data about its environment as well as the UAV's state in that environment. This data enables the UAV to make decisions on how to adapt to its surroundings to complete all of its tasks autonomously.

By starting with a fully functioning, manual vehicle, the creation of an autonomous vehicle simply entails the addition of autonomous management of the controls. Managing the controls requires the acquisition of data that gives "feedback to the robot containing information about the robot's action (internal sensors) or about its environment (external sensors)" [10]. Therefore, with an array of sensors, a robot can make informed decisions based on the data provided and the mission of the robot.

Having sensors is vital, and there are many key design decisions regarding the sensors that are paramount to the success or failure of robotic systems. Aspects such as the number, placement, and type of sensors, whether internal or external, need to be considered during the design process. The management of the sensor behavior and collected data also plays a critical role in the performance of the system as certain techniques only exhibit advantages in very specific applications. This section outlines the fundamental considerations that must be accounted for when designing an autonomous system.

2.1 Degree of Freedom and Degree of Mobility

The first items to consider when designing a navigation system for a UAV are its degrees of freedom and degrees of mobility which determine the number of sensors and the complexity of characterizing the robotic system [10]. The location of an object can be defined by three Cartesian coordinates: x, y, and z, which translates to three degrees of freedom in which it can move. Still, knowing the exact position of an object is insufficient in completely describing the state of a flying object because “there are infinite possible ways to orientate the object about the selected point. To fully specify the object in space, in addition to the location of a selected point on it, one needs to specify the orientation of the object” [9]. Moreover, one needs to add three additional degrees of freedom to describe an object’s orientation at a point, which translates to roll, pitch, and yaw in aeronautics. Thus, a total of six degrees of freedom is required for an effective UAV system. Less than this would be a limitation of the aerial robot as it would indicate the robots inability to navigate in a certain direction or to change its orientation [9]. Each of these degrees of freedom must be measured by sensors for the robot to navigate intelligently.

The degree of mobility adds enormous complexity to a robotic system. The degree of mobility is determined by the joints in a robot that add degrees of freedom to the overall system without adding degrees of freedom to the end-effectors [10]. End effectors are the devices with which the robot interacts with the environment such as tools or grippers. Having joints in a system requires the use of kinematics to determine end-effector position from joint coordinates and inverse kinematics to determine the joint positions given the end-effector coordinates. While jointed robots are useful in manufacturing environments, the added complexity required to characterize jointed systems makes the addition of degrees of mobility impractical for UAV applications.

2.2 Tracking Techniques: Dead Reckoning and GPS

There are many options to tracking the movement of a robot. Two of the most common methods include global positioning systems (GPS) and dead reckoning. Dead reckoning determines a vehicle's position based on odometry. This can be done by keeping track of the rotation of motors or wheels through shaft encoders or by using inertial navigation systems (INS). Shaft encoders are more commonly used because of their simplicity and because INS generally have demanding power requirements and higher costs associated with it [3]. While tracking motor movement is reliable for position determination in robotic arms, the application to mobile systems is misleading due to factors such as wheel slippage or sliding that cause immeasurable movement. Also, errors in calculating position propagate to future calculations causing a progressive drift. Therefore, a more appropriate choice for tracking UAV movement is GPS. Through satellite triangulation, GPS allows for the computation of the longitude, latitude, and altitude of a system. The only challenge with using GPS is that the signal integrity can be compromised by surroundings such as structures or trees. Overall, GPS is the better candidate for UAV navigation and tracking systems.

2.3 Internal and External Sensors

While tracking systems such as dead reckoning and GPS can give information on the location of an object, an array of internal and external sensors is required to determine the state of an object in its environment accurately. After all, tracking systems only account for three of the proposed degrees of freedom, namely the x-, y-, and z-coordinates. Internal sensors allow robots to measure velocity, acceleration, and force, which can be applied to track the roll, pitch, and yaw degrees of freedom. External sensors allow the robot to be aware of its immediate surroundings.

2.3.1 Internal State Sensors

The goal of internal sensors is to provide information on velocity, position and force of the system. This internal state sensing simulates the biological sense of proprioception, i.e. the sense of movement and spatial positioning [10]. Gyroscopes are an example of internal sensors that can be applied to robotics. These are angle rate sensors that can be used to measure angular changes of a system. By having gyroscopes measure angular changes around the x-, y-, and z-axis, a level of proprioception can be achieved. In addition, accelerometers can be added to measure acceleration along any of the Cartesian axes. Monitoring rotation and acceleration will permit the aerial robot to achieve autonomous stability by detecting and compensating for unwanted changes in position and orientation.

2.3.2 External Sensors

External sensors are required to provide information to the robot about its environment. The external sensors allow the robot to achieve exteroception, or the ability to perceive external objects through simulated, biological senses such as touch, vision, and hearing. Exteroception can be achieved with contact or noncontact sensors. Contact sensors are inexpensive and provide straightforward data, but they have the disadvantage of requiring contact to detect an object and are blind to objects in close proximity. In some applications, an array of contact sensors may be used because of the ability to acquire two-dimensional data. In the realm of UAVs, contact sensors are impractical and noncontact sensors should be used exclusively. Using noncontact sensors will aid in averting any damage from collisions by detecting any close objects.

Two of the most common proximity detectors are infrared and ultrasonic systems. Ultrasonic systems are advantageous because they are robust, inexpensive and consume relatively low power, but they are susceptible to background noise and false readings. In addition, ultrasonic systems are only effective at short ranges (typically less than 10 meters) and can miss objects that absorb too much of the incident wave energy such as porous objects [10]. Infrared systems, on the other hand, have a high angular resolution but are more difficult to use and are considerably more expensive. Although both systems have their advantages, ultrasonic ranging is the favored mechanism for proximity sensing because the availability and the accuracy of readings makes it more practical as long as the range limitations are accounted for.

2.4 Control Method: Deliberative vs. Reactive Systems

The method of robotic control is one final critical design decision when creating an autonomous robot. Two camps of thought exist in this realm of design: deliberative and reactive. Deliberative control is an approach that relies on making the robot more intelligent while the reactive approach is behavior based, relying on very little intelligence.

2.4.1 Deliberative Control

Deliberative, or hierarchical control, relies on a robot's ability to reason out situations to make decisions. A hierarchical structure of decisions is created based on the work of J. Albus of the National Institute of Standards and Technology who bases intelligence on sensory processing, world modeling, task decomposition, and value judgment [3]. Tasks are decomposed and then analyzed for decisions on a low intelligence level at the bottom of the hierarchy structure. The low-level decisions then propagate up the hierarchy to controls that make more involved and abstract decisions. Theoretically, this hierarchy will allow the robot to exhibit intelligent behavior by making an informed and well "thought out" decision, but this control system is not without its drawbacks. Ronald C. Arkin asserts why deliberative systems are not practical to use:

> Reactive systems, however, were developed in response to apparent drawbacks associated with the hierarchical design paradigm including a perceived lack of responsiveness in unstructured and uncertain environments due both to the requirements of world modeling and the limited communication pathways; and the difficulty in engineering complete systems as incremental competency proved difficult to achieve, that is, virtually the entire system needed to be built before testing was feasible. [3]

While the deliberative control would be an ideal system for robotics, the difficulty of creating a system makes it extremely impractical. One must then consider the use the alternative approach of reactive systems.

2.4.2 Reactive systems

Reactive systems are based on reflexive behavior. This means tightly coupling the robot's perception to its action so that it can quickly respond to a dynamic and indefinable environment. For example, the AeroBot's software uses interrupts and a tight polling structure to minimize the time between receiving measured data and the response to that data. This provides a significant advantage over deliberative models because the "more the world changes during execution, the more the resulting value of any plan generated a priori decreases, and the more unstable any representational knowledge stored ahead of time or gathered during execution and remembered becomes" [3]. So although the decisions of deliberative systems are "thought out," the lag in deriving the solution diminishes the value of that data. As a result, reactive systems serve as a more reliable and practical approach as modeling the world for deliberative control is too cumbersome to be useful.

3. Overview of the Initial Hardware

The platform chosen for creating the AeroBot was the DraganFlyer V Ti by DraganFly Innovations. By starting with an established system, the effort of the project could be focused solely on achieving autonomy, making the project more practical. Also, much of the default hardware on the DraganFlyer makes it an ideal candidate for an autonomous vehicle, such as the built in infrared sensors and gyroscopes. This section outlines the basics of the hardware that accompanies the standard DraganFlyer helicopter.

3.1 DraganFlyer V Ti Overview

The AeroBot platform is based on DraganFly Innovations' DraganFlyer V Ti Pro available at www.rctoys.com. This is an RC helicopter with four, individually controlled propellers for added stability. The complete system weighs approximately 450 grams, depending on the battery used.

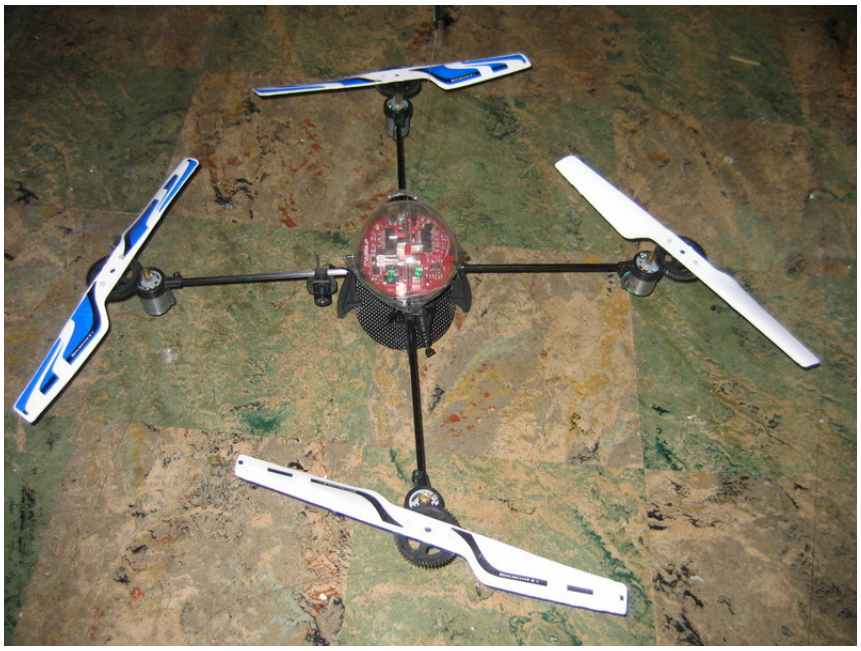


Figure 2: Default Hardware Configuration

Figure 3 depicts the factory configuration of the hardware, with the front of the aircraft pointed downward. Most of the frame, including the vertical risers and tubing, is comprised of carbon fiber. Carbon fiber is a composite material produced from carbon filament. It has a high strength to weight ratio, making it ideal for small-scale aircraft. With high tensile and compression strength, the only drawback for this material is the relatively weak resistance to bending.

3.2 DraganFlyer Motors

The helicopter is equipped with four, individually controlled Mubachi RC-280SA motors. These are high-torque 14 point gear motors with a diameter of approximately 27mm. The motors operate at approximately 12 VDC from the unregulated power supply, even though the factory specified operating voltage range is only rated at

4.5-9.0 Volts. Overdriving the motors with the higher voltage paired with an additional payload leads to higher heat generation during use, so heat sinks were attached directly to each motor.

The four motors of the helicopter are individually controlled by a motor driver and four power MOSFETs on the horizontal helicopter board. The motor driver controls the speed of the motors by varying the effective voltage sent to the motors through pulse width modulation (PWM). The average output voltage through PWM is given by:

$$V_{out} = V_{cc}(\frac{t_1}{t})$$

where t_1 is the rise time and t is the period of the modulated signal. The direction of the motor spin is arranged so that there are equal rotational forces in all axes (roll, pitch, and yaw). The motors on the sides of the DraganFlyer rotate clockwise and the front and rear motors will rotate counter-clockwise. The helicopter's microcontroller manages the motor rotation so that the helicopter is relatively stable with no control input by balancing the contributions in both rotational directions using built-in sensors.

3.3 DraganFlyer Sensors

An advantage of using the DraganFlyer V Ti is that it is equipped with four thermal sensors for added stability. These sensors detect a heat differential to help stabilize the helicopter by allowing it to level itself to the heat gradient that it detects. Unfortunately, these readings are impaired by cloud coverage and by any warm objects in close proximity including people, animals, vehicles and buildings. Although the performance of the thermal senor readings can be compromised, this auto-stabilization feature aids in achieving autonomy by helping the aircraft to stabilize itself during flight and hovering.

The DraganFlyer V models are also equipped with onboard gyroscopes for stabilization. These compensate for movements that are not the result of input from the controller. Changes on all three axes (roll, pitch, and yaw) are corrected using the on-board gyroscopes. Unfortunately, these gyroscopes only compensate for some quick movements and do not correct for gradual angular changes. Overall, the base hardware of the DraganFlyer makes it a good candidate for creating a UAV because the existing sensors make it an inherently stable system when compared to other RC helicopters.

4. Achieving Autonomy

With an understanding of autonomous design principles and the foundational hardware upon which this project is based, the process of achieving autonomy in this project can now be tackled. This section overviews the hardware and software additions to the DraganFlyer V system to achieve the desired functionality of the AeroBot.

4.1 Power Management

The first step in adding hardware to achieve autonomy is to create a power source for the hardware. Usually the power available on a system cannot be used without some kind of conversion. For example, the power from the DraganFlyer battery will range from 11-14 Volts depending on the battery used, while the sensors and microcontroller to be added operate at only 5 Volts. In addition to modifying the voltage level available for the new hardware, the reduction of noise from the source and the integration of protection mechanisms is crucial for success.

4.1.2 Voltage Regulators, Circuit Isolation and Noise Management

A voltage regulator allows for the use of 5 Volt circuits on an unregulated power supply of about 7 Volts or greater. In the AeroBot, this allows the battery to power the circuitry added for autonomous behavior. In addition to modifying the voltage level, a voltage regulator can contribute numerous benefits to a circuit. First, the voltage regulator creates a clean signal to supply to the additional circuitry by isolating the circuits on the regulator output from the input power. This is important because the motors generate noise on the unregulated power supply that render sensor readings useless if the noise is present on the sensor power supply. Second, the isolation of the robotic circuitry prevents it from competing with the motors for power. Third, when properly configured with capacitors, a voltage regulator circuit further reduces noise on circuitry by compensating for power sags and spikes. Figure 3 depicts a typical configuration for a 7805 voltage regulator suggested by manufacturers.

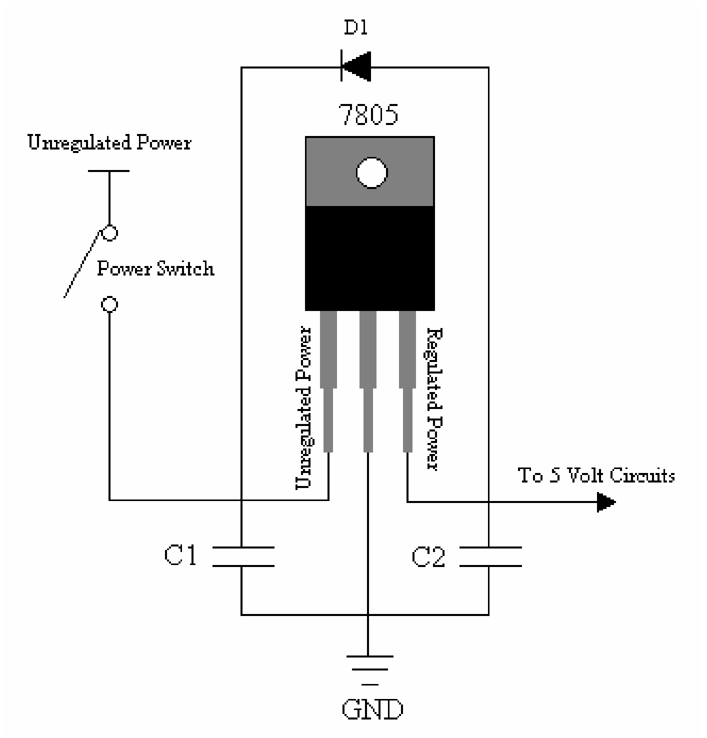


Figure 3: 7805 Voltage Regulator Circuit Setup

The capacitors C1 and C2 charge during voltage spikes and discharge during voltage sags, allowing the robot circuitry to see less noise on their supply. Since C1 is connected to the unregulated supply, it is important to note that voltage rating for the capacitor should be able to handle voltages slightly higher than the maximum unregulated voltage. As for C2, which is acting as backup for power sags, a large capacitance value is necessary for adequate performance.

4.1.3 Circuit protection

When making modifications and additions to power supplies, it is important to consider mechanisms that will protect the robotic circuitry. For example, referring back to Figure 3, the recommended placement for diode D1 protects the voltage regulator from damage from a reverse current, but provides a path for unwanted current to the robotic circuits if the battery were accidentally placed with the polarity reversed. A better configuration is presented in Figure 4, where D1 protects the circuitry and voltage regulator by cutting off current flow from an incorrectly installed battery. Fuses also provide a cheap and effective means to prevent high currents from flowing through certain components. Installing fuses is a simple task and their availability, affordability, and small size make them practical for use in robotic circuitry.

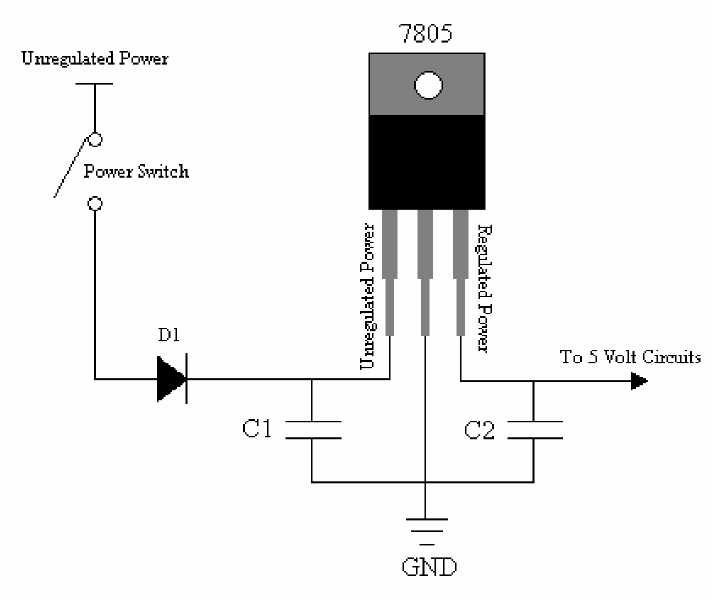


Figure 4: 7805 Voltage Regulator with Reverse Battery Protection

One final device to consider for circuit protection is a zener diode. This is similar to a normal diode but allows a reverse current at a certain breakdown voltage. Figure 5 depicts the characteristic of a typical zener diode. At voltages above 5.4 Volts, this zener diode allows the flow of reverse current. When applied to the regulated power supply and ground (with the cathode connected to the power supply), the diode will prevent any damage to the circuit connected to the regulated power supply by conducting when overvoltages occur.

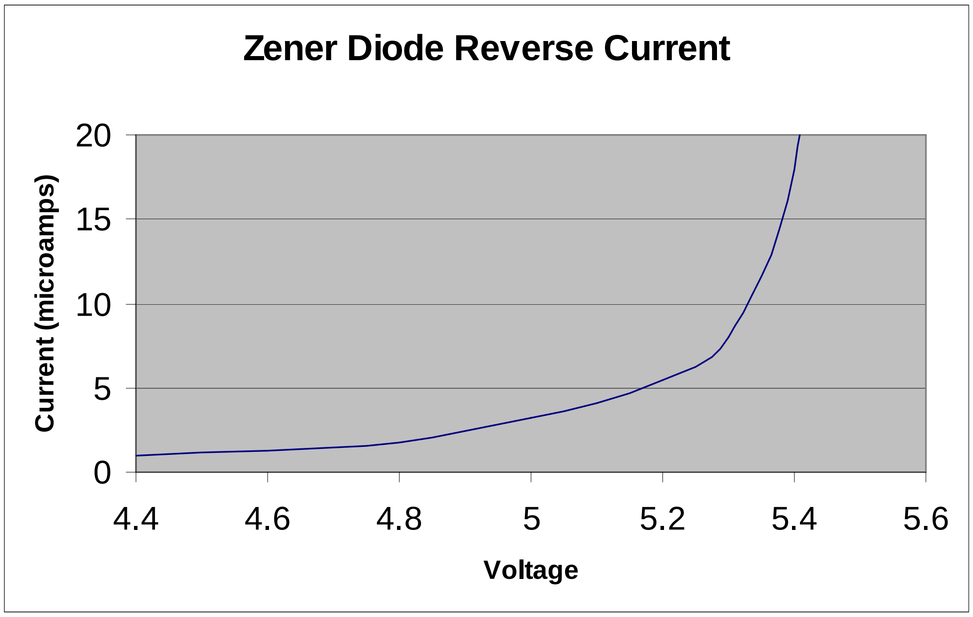


Figure 5: Zener Diode Reverse Current

4.2 Hardware Additions

Below is a block diagram of the hardware structure for the autonomous system being added to the DraganFlyer. An array of sensors including gyroscopes, accelerometers, sonar, and GPS were added to the helicopter. The sensor array is attached to an on-board microcontroller that manages data collection as well as the data processing. Using this information, the microcontroller sends the appropriate control signals to the helicopter as defined by the control algorithm. This section overviews the hardware added to the DraganFlyer.

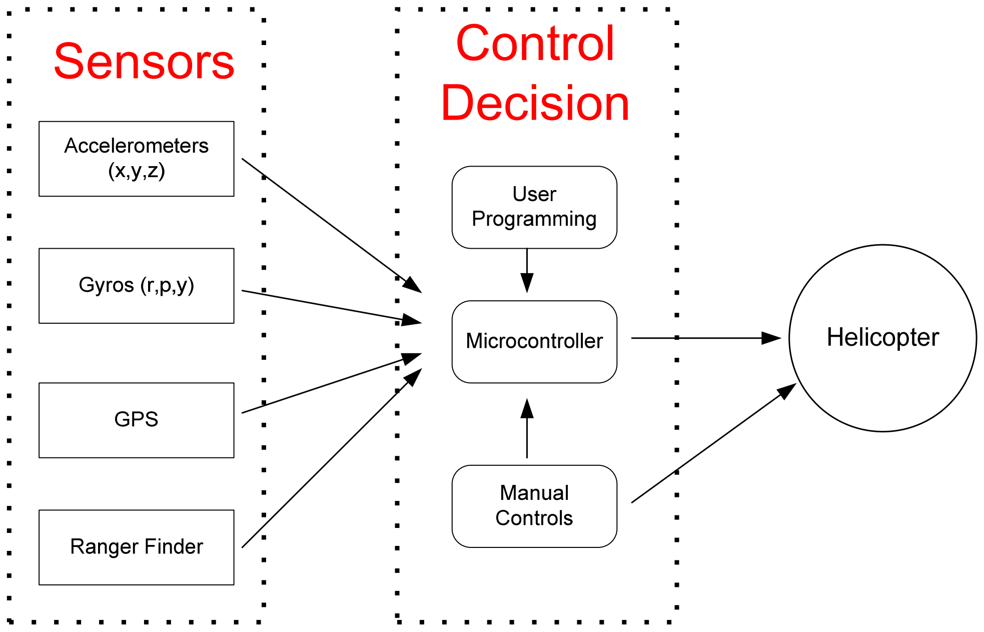


Figure 6: Hardware Block Diagram

4.2.1 The Brain

Autonomous vehicles need a “brain” to make decisions and control behavior. The brain for an autonomous vehicle can range from a logic chip to a microcontroller. Logic chips have the advantage of being much more inexpensive, faster in response time, and versatile in accepting wider ranges of voltages and currents. A microcontroller, on the other hand, has the advantage of being programmable, making it capable of doing more complicated tasks. In general, any robot with nontrivial functionality should have a microcontroller.

To achieve intelligent functionality, the AeroBot was equipped with an ATmega128-16AI AVR microcontroller. This microcontroller was chosen for many reasons. First, it maintains a diverse list of compatible communication protocols including RS-232, SPI, I2C. Second, the multitude of general purpose ports allow for a myriad of applications. Third, the ATmega128 has several built-in functions such as ADC and DAC, timer/counters for time management and interrupts, and two programmable USARTS. Finally, the familiarity of the device at Oregon State University provided considerable incentive to use the chipset since it is the focus of several courses at the university.

4.2.2 Ultrasonic Rangefinder

Mounted on the bottom of the AeroBot is a Devantech SRF08 ultrasonic rangefinder for measuring relative altitude, depicted below. An ultrasonic rangefinder was chosen for several reasons.

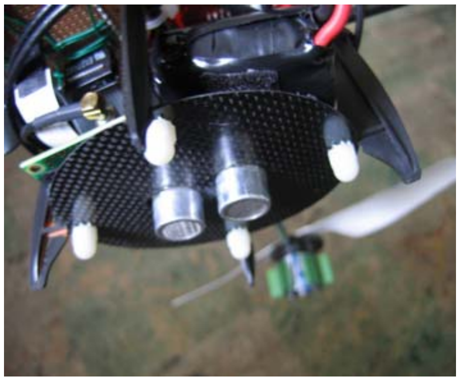


Figure 7: Rangefinder Placement on AeroBot

To understand why it is advantageous to use sonar as the primary means of extracting ranging, one must first understand how it works. Active sonar works by using a projector to create a pulse that will be reflected back once it hits an object. This echo is then detected by a transducer and the system records the time from the pulse generation to the time of the received echo. The speed of sound is highly predictable so the distance to the object that created the echo can be calculated using the following equation:

$$D = \frac{v \times t}{2}$$

where *D* is the distance traveled, *v* is the velocity of sound, and *t* is the time from pulse creation to echo detection. There are two important considerations when using sonar, which are the pulse duration and pulse repetition frequency. Pulse duration will determine resolution of the readings according to the equation:

$$D = v \times L$$

where *D* is the discrimination between echoes, *v* is the velocity of the waves and L is the pulse length [13]. Although pulse frequency may increase resolution, one must take into account that high frequencies "attenuate much faster than the lower frequency signals, which severely limits their range" and that "lower frequency transducers have wide beam angles and a severely deteriorated lateral resolution," [9]. The pulse repetition frequency, on the other hand, determines the maximum range of a sonar system with the following equation:

$$MaximumRange = \frac{v \times t}{2}$$

where *v* is the velocity of sound and *t* is the time between pulses [13].

With the understanding of the mechanics of sonar, we can now evaluate it as a ranging tool. First, sonar is completely independent of light levels, unlike its optical counterpart, which is the second most common approach to navigation. Again, it is important to emphasize that an autonomous robot must be robust and able to operate independent of the condition of its environment in order to be successful. If varying light levels adversely affected a robot's ability to navigate, then it would be a poor design because the robot's effectiveness would be limited to a very few environments. Additionally, sonar is almost immune to background noise if there are proper drive signals and filtering [13]. Finally, an advantage to using sonar is that it is relatively inexpensive. This makes it practical for robot designers to use multiple sensors at a time, leading better measurements.

The SRF08 rangefinder operates by sending 8 consecutive 120 dB pings at 40 kHz and averaging those values. The 40 kHz frequency is based on the optimal operation of the 400ST air ultrasonic ceramic transducers on the SRF08. These transducers are the most sensitive at a frequency of about 40 kHz. This yields an approximate sensing range of 3cm – 11m. Care must be taken in analyzing the range measurements returned, however, because reflections and other sources of noise can lead to misleading data. Therefore, “sanity checks” should be embedded in the code that handle the range readings to filter out erroneous readings. In addition, the construction and environment of the rangefinder must be taken into account when analyzing data because of the wide beam width of the rangefinder, which is approximately 55°.

4.2.3 GPS (Global Positioning System)

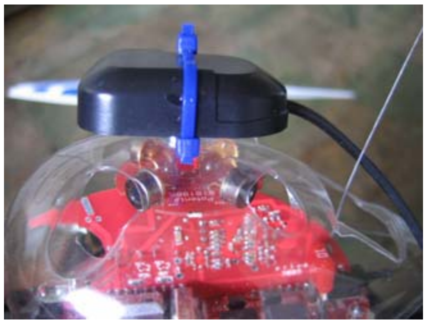


Figure 8: GPS Antenna

The AeroBot is equipped with a SBR-LS sensor-based GPS receiver board, manufactured by UBlox. This GPS unit uses the ANTARIS® GPS positioning engine that gives relatively high performance in areas typically difficult for GPS receivers. The accuracy of the GPS coordinates provided by the unit is accurate within a 6 meter diameter window. When the GPS loses a GPS lock, it compensates by utilizing built-in dead reckoning functions. These functions use gyroscopes and odometer pulses to approximate the movement experienced during signal loss. In addition, these sensors are used for next-position approximations using weighted averages of the measurements taken. This robust GPS design makes it a prime choice for the navigational system. Unfortunately, the coordinates are updated slowly (approximately 1Hz) but the accuracy is enhanced by the active antenna mounted on the top of the AeroBot.

4.2.4 Gyroscopes

To measure angular movement, the AeroBot was equipped with three, single-axis gyroscopes assigned to measure yaw, roll, and pitch. Gyroscopes use the principle of conservation of momentum to measure rotation about an axis. These components output analog signals that reflect movement around the measured axis and the microcontroller decodes this information to determine the actual movement of the AeroBot.

Gyroscope selection is tricky because there are a vast assortment of gyroscopes available. Designers need to be aware of the pitfalls of mechanical gyroscopes in particular. Ball bearing noise inside mechanical gyroscopes can cause random drift that peaks at certain frequencies. This in-run noise is manifested on the output and is proportional to the ball size, number of balls, operating speed, and any asymmetry between bearings at the end of the gyroscope spindle [7]. Mechanical gyroscopes also tend to exhibit a day-to-day uncertainty up to an order of magnitude larger than in-run noise because of “aging internal magnets and bearings, laser gas contamination, or from mirror or optical fiber aging” [7]. Finally, designers must also be aware that although gyroscopes are designed to measure velocity, they also respond to acceleration. With these factors in mind, an appropriate gyroscope can be selected, and the aforementioned problems can be handled through compensating software algorithms.

The device chosen for the AeroBot was the Analog Devices ADXR300 gyroscope. This is a microelectromechanical system (MEMS) device, which includes all of the electronic and mechanical systems on a single die. This makes the device highly affordable, compact, lightweight, and sensitive (5.0 ± 8% mV/°/s). In addition, MEMS devices have been proven to be reliable in several applications including automotive safety, indicating that the in-run errors are negligible. This particular device also has the ability to run a self-calibration routine at startup, which is an enormous advantage over typical mechanical gyroscopes, as it eliminates day-to-day errors.

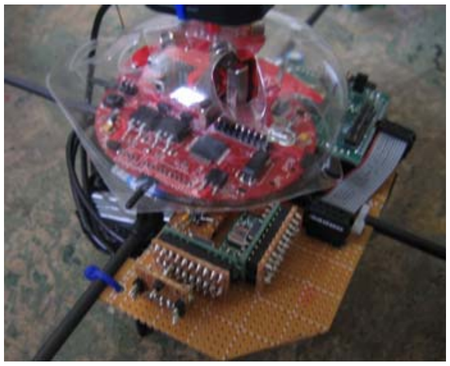


Figure 9: Gyroscope Placement on the AeroBot

4.2.5 Accelerometers

Measuring the forces on the three axes of the helicopter (roll, pitch, and yaw) required the addition of two, 2-axis accelerometers. When choosing accelerometers, one must consider its sensitivity to anisoelasticity, which is a problem for many accelerometers. Anisolascticity is a perturbation of accelerometers by rotational motion from “mismatches in the stiffness of the members supporting the sensing element” [7]. The ADXL203 analog accelerometers from Analog Devices were used for this project because of the precision guaranteed with these devices. These MEMS components have all of the same benefits of the MEMS gyroscopes along with a high sensitivity (±4 mG).

4.3 Software Overview

A step-by-step analysis of the software written for the AeroBot would not be very beneficial for discussing UAV design. Appendix E gives a full breakdown of the organization of the software as well as a functional analysis if there is interest in the details of AeroBot software. A more useful discussion concerns the software techniques that were vital to the success of the robot.

Without question, the most important software technique applied to the AeroBot project was the reduction of noise through software. First, the integration of “sanity checks” throughout the software was important. This helped prevent the AeroBot from exhibiting erratic behavior because of erroneous sensor when encountering noise or interference. Even with hardware mechanisms to compensate for sensor noise, software checks prove to be a necessary supplement for the best results. These sanity checks can simply be checks on sensor data to see if the read value is within a user-defined bound. This can be done by defining an absolute bound that determines whether or not a value is acceptable or by defining an acceptable change in values, because a drastic change in sensor readings could also identify bad data points. The second method of using software to reduce noise is through averaging. Noise is random over time, so averaging will allow the extraction of a signal because it would “have some nonrandom properties that will not average out to a zero value” [6]. This adds a level of noise reduction that would be difficult to achieve through other means such as hardware filters, especially when there is no definite noise signature, such as a frequency to target for filtering.

5. Results Analysis

The AeroBot did not achieve much of the autonomous functionality that was originally intended. The final product could only compensate for yaw and altitude changes, allowing it to take off and hover for only a few seconds without the pitch and roll adjustments. Time and money constraints along with some key design flaws hampered the project’s success. However, by highlighting the weaknesses of the project, the goal of this document can be achieved, which is to allow similar projects to leverage the work and experience from the AeroBot.

Beginning with a discussion of the pitfalls of the project, one tremendous design flaw was the goal of the project. The original goal was to create a system that could fly to a location, take a picture and land at the starting location autonomously. This objective was far too ambitious given the time and resource constraints on the project. In addition, a lot of time was invested into modifying the DraganFlyer helicopter to be able to manage the autonomous tasks it would be programmed to accomplish. This included upgrading the helicopter by redesigning and machining body parts and replacing the motors and power systems. The attempt at upgrading the helicopter was an ambitious project on its own and consumed several weeks of work without yielding any benefits, as the upgrades made the fragile system even more unstable. The correct approach would have been to base the project around a practical goal, such as creating a helicopter that could autonomously hover. Then, if time permitted, resources could be invested to add new functionality, which would avoid divesting resources from essential aspects of the project.

Another fault in the AeroBot design was the placement of electronic components. From Figure 9, one can see that the gyroscope and accelerometers were placed away from the center of the helicopter, which is the pivot point of the system. This caused a critical error in calculating position changes with these sensors. These MEMS sensors are very accurate, but placing them away from the axis that they were measuring introduced anisoelasticity errors to the accelerometers and in-run errors to the gyroscopes, as discussed in sections 4.2.4 and 4.2.5. In other words, the gyroscopes became hyper sensitive to accelerations and the accelerometers became hypersensitive to rotations. The only usable data from those sensors was from the z-axis accelerometer and the yaw rate gyroscope because the other sensors exhibited unrecoverable drift in measurement immediately after take-off. Without the other sensors, the helicopter could not detect and compensate for changes in roll and pitch. This failure to achieve proprioception was the main reason for the helicopter’s inability to hover for long periods of time.

Another drawback in the design was in the management of the microcontroller because the microcontroller became overburdened with the tasks assigned to it. Having only one microcontroller introduced a few subtle timing errors because the multitude of tasks handled in the polling loop of the software was constantly being disturbed by interrupts. These interrupts were usually time-critical tasks essential for sensor management and could not be ignored. However, the numerous interrupts left the microcontroller stuck in interrupt service routines for the majority of execution time, leaving other tasks unmanaged and leaving little time for calculations. The project would have been more successful if the work assigned to the single ATmega128 was partitioned and assigned to two microcontrollers. A more effective design would have been to have a microcontroller dedicated to sensor management and a microcontroller dedicated to task management. The sensor managing microcontroller would handle the scheduling of data acquisition so that the other microcontroller could focus its processing power and time on data management and task execution.

Lastly, a flaw in the AeroBot project was the lack noise management systems and a lack of redundancy in fail-safe mechanisms. The noise manifested in the sensor data made it difficult even to determine the utility of the data from the gyroscopes and accelerometers, much less to use them. It was not until late into the project that software averaging techniques were applied to the noisy signals. This filtered out the noise that could not be isolated through physical means such as filters. The helicopter's vibrations were causing erroneous readings from all of the MEMS sensors and averaging the sensor data extracted the useable data. As for fail-safe mechanisms, the project had too little emphasis on redundancy. Adding modifications to the power supply of the helicopter was very risky and lead to the destruction of several components and delays to the project. In addition, most of the circuitry was connected without any levels of isolation to minimize damage. Therefore, a short circuit or reverse current in one part of the circuit usually propagated to the rest of the circuit. The use of diodes, zener diodes, fuses and voltage regulators would have saved a lot of time spent on repairs and debugging.

6. Conclusion

Today it seems as though all of the electronics around us are becoming more and more intelligent, such as cars, cell phones, and even vacuums. It would be difficult to argue that the consistent advances in the technological complexity and designs of such devices have not benefited humankind. Therefore, it is reasonable to conclude that advancements in the field of UAVs and other autonomous robots would also improve the quality of life. With applications such as research, exploration, and reconnaissance, the possibilities for these devices are endless. However, to broaden the scope of robotic applications, intelligent designs for these robots must be developed before applications can be considered. This must be taken one step at a time, and hopefully this project resulted in a step forward for these ideas.

Even though the AeroBot project yielded fewer positive results than originally hoped, the project was still considered a success. As mentioned earlier, the overall aim of the project was too ambitious, so what little autonomous functionality actually achieved by the system was still satisfying. In fact, many people, including the sponsoring company was skeptical that the AeroBot would achieve that much. Hopefully, providing this document helps to yield more fruitful projects in the future by allowing other engineers to leverage the work put into the AeroBot. After all, the suggestions put forth by this document are the result of experience. Encountering the same difficulties that the AeroBot project experienced would be detrimental, but many of these problems are difficult to foresee. Therefore, chronicling the creation of the AeroBot has much more of an impact than the AeroBot itself.

Bibliography

[1] A. Elfes. Sonar based real world mapping and navigation. *IEEE journal of Robotics and Automation*, (1987):233-247.

[2] Albus, J. Outline for a Theory of Intelligence. *IEEE Transactions on Systems, Man and Cybernetics*, Vol.21, No.3, May-June: 473-509.

[3] Arkin, Ronald C. Behavior-Based Robotics.Cambridge: The MIT Press, 1998.

[4] Billings, Charles E. Aviation Automation: The Search for A Human-Centered Approach. Mahwah: Lawrence Erlbau Associates, Inc. 1997.

[5] Cook, David. Intermediate Robot Building. Berkley: Apress, 2004.

[6] Hunt, V. Daniel. Understanding Robotics. San Diego: Harcourt Brace Jovanovich, Publishers, 1990.

[7] Lawrence, Anthony. Modern Inertial Technology: Navigation, Guidance, and Control. New York: Springer-Verlag, 1993.

[8] Leonard, John J. and Hugh F. Durrant-Whyte. Directed Sonar Sensing for Mobile Robot Navigation. Norwell: Kluwer Academic Publishers Group, 1992.

[9] Niku, Saeed B. Introduction to Robotics Analysis, Systems, Applications. Upper Saddle River: Prentice Hall, 2001.

[10] Poole, Harry H. Fundamentals of Robotics Engineering. New York: Van Nostrand Reinhold, 1989.

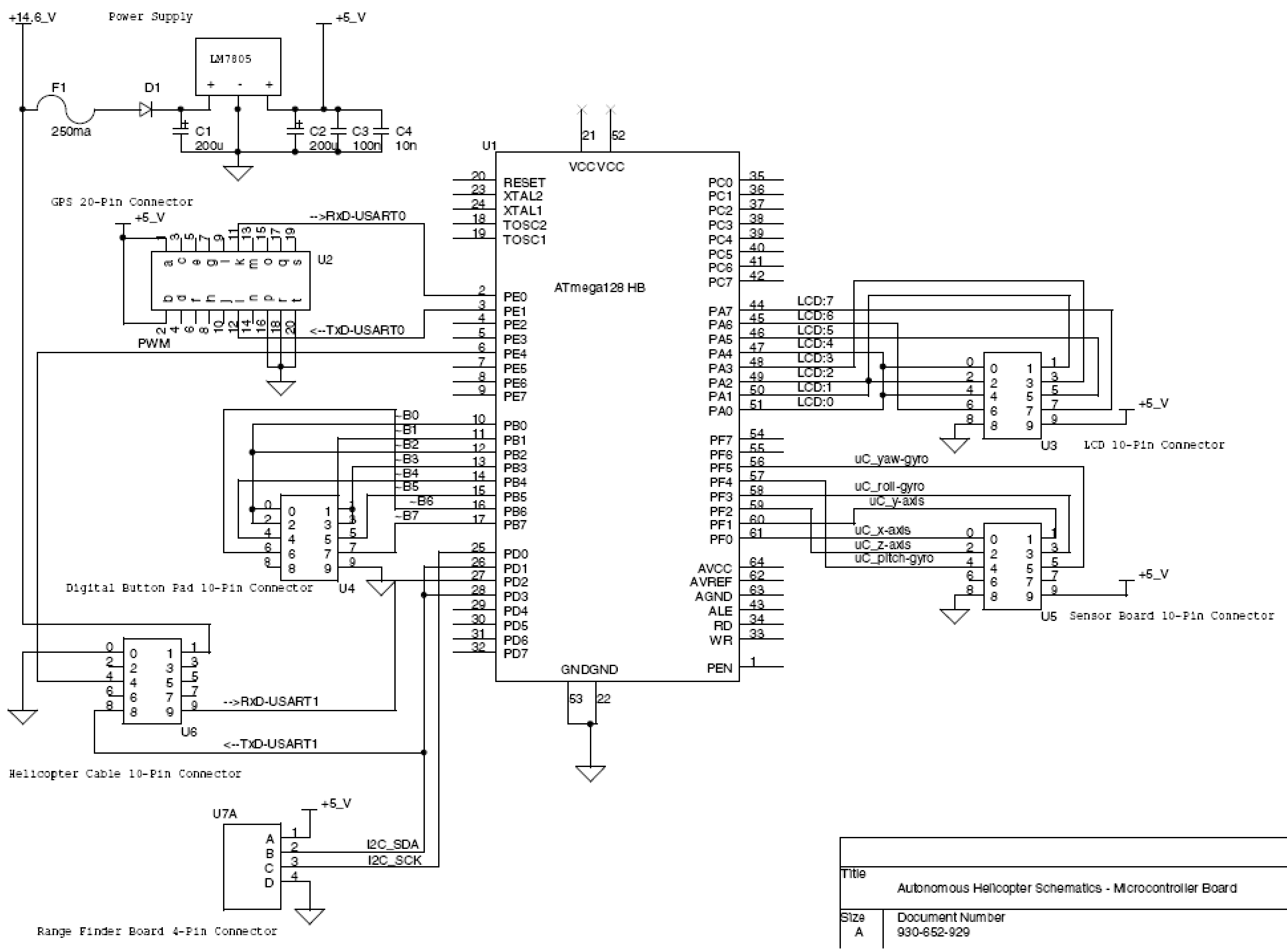
[11] Ricker, Dennis W. Echo Signal Processing. Norwell: Kluwer Academic Publishers Group, 2003.

[12] Smith, Penelope. Active Sensors for Local Planning in Mobile Robotics. River Edge: World Scientific Publishing Co. Pte. Ltd., 2001.

[13] Tetley, L. and D. Calcutt. Electronic Aids to Navigation. London: Edward Arnold, 1991.

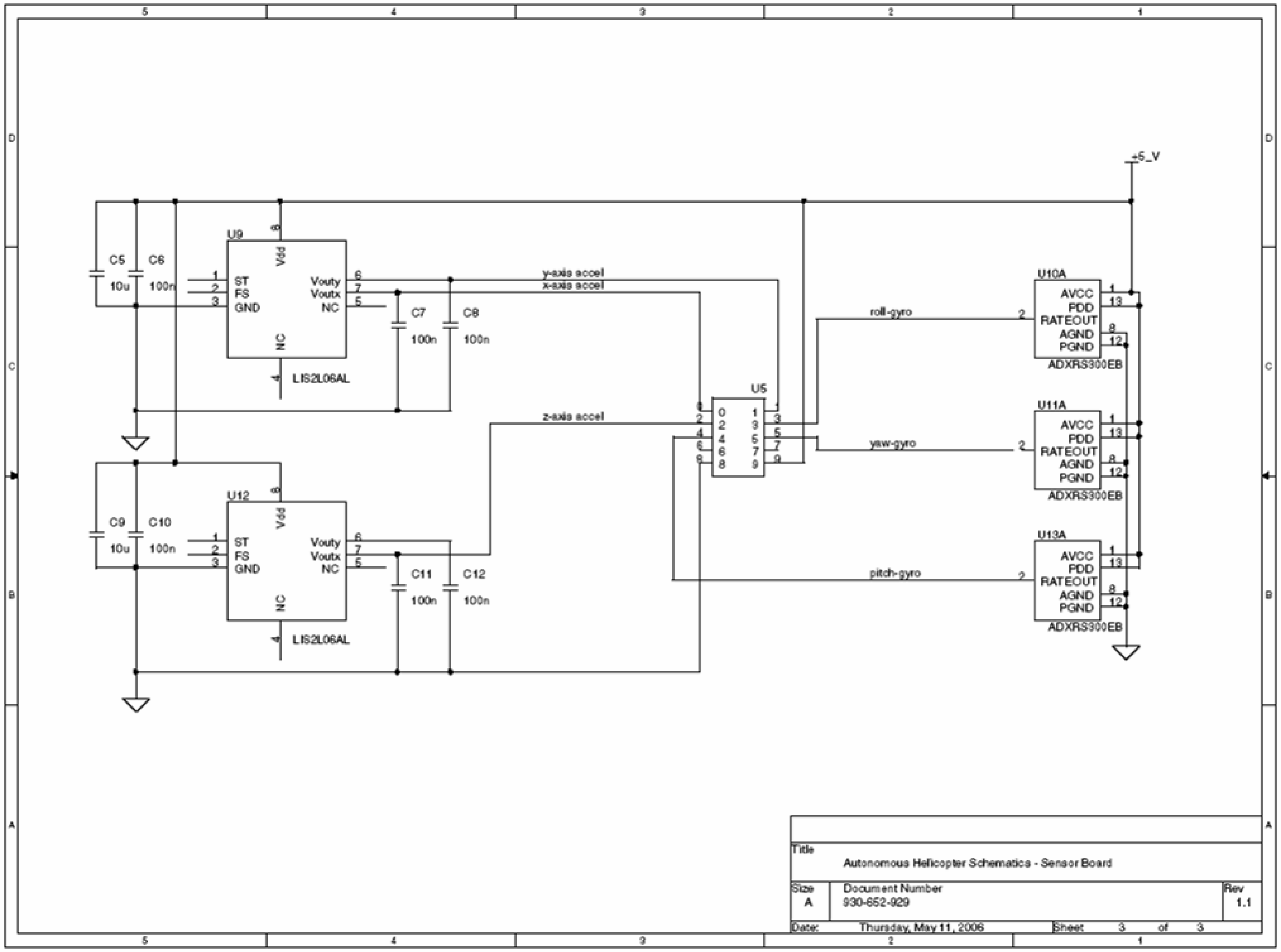
[14] Waite, Ashley. Sonar for Practising Engineers. West Sussex: John Wiley & Sons Ltd, 2002.

Appendix A: Microcontroller Schematic



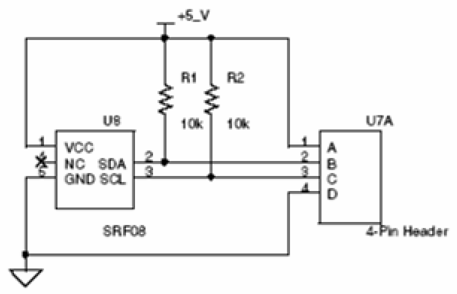
Title	Autonomous Helicopter Schematics - Microcontroller Board	
Size A	Document Number 930-652-929	Rev 1.1

Appendix B: Sensor Board Schematic



Title		
Autonomous Helicopter Schematics - Sensor Board		
Size A	Document Number 930-652-929	Rev 1.1
Date: Thursday, May 11, 2006	Sheet 3 of 3	

Appendix C: Rangefinder Schematic



Title	Autonomous Helicopter Schematics - Range Finder Board	
Size A	Document Number 930-652-929	Rev 1.1
Date: Thursday, May 11, 2006	Sheet 2 of 3	

Appendix D: Port Assignments

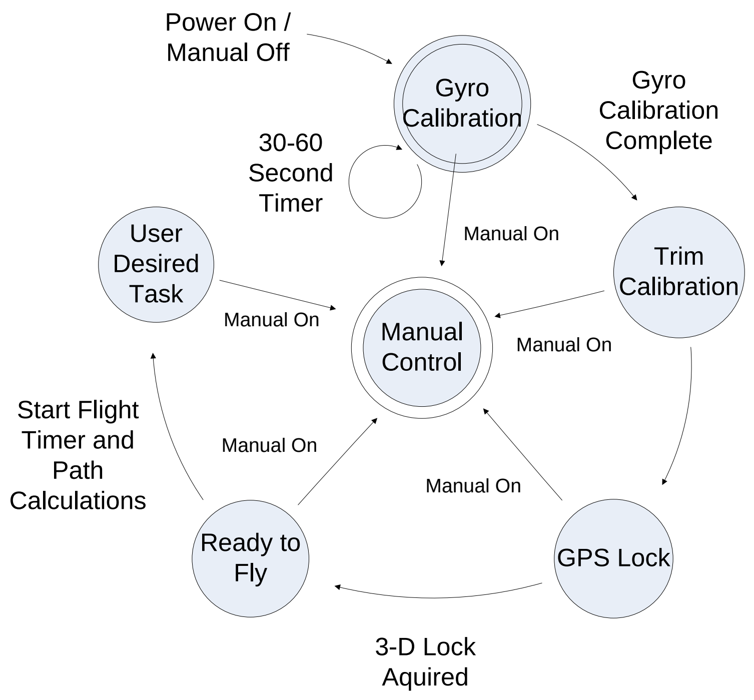
The following is a list of the port/pin assignments of the helicopter. These port/pin assignments are important for understanding how the microcontroller communicates with the rest of the devices.

Module	**Pins on uC**	**PORT**
LCD	51-44	A0-A7
RangeFinder CLK (scl)	25	D0
RangeFinder Data (sda)	26	D1
Helicopter Transmit (uC RxD)	27	D2
Helicopter Receive(uC TxD)	28	D3
GPS Transmit	02	E0
GPS Receive	03	E1
Helicopter PWM (ex.int4)	06	E4
X-Axis Accelerometer	61	F0
Y-Axis Accelerometer	60	F1
Z-Axis Accelerometer	59	F2
Gyro Roll Rate	58	F3
Gyro Pitch Rate	57	F4
Gyro Yaw Rate	56	F5
OPEN	55	F6
OPEN	54	F7

Appendix E: Software Functional Description

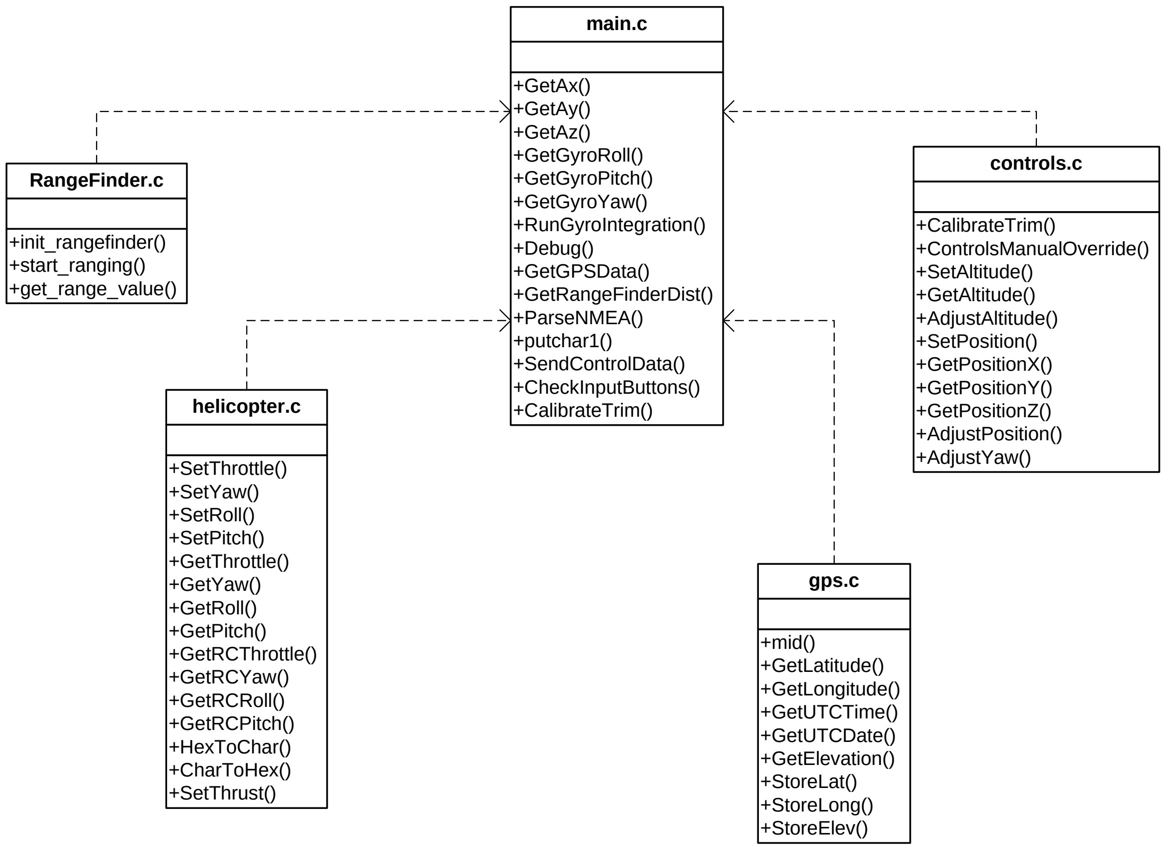
Software State and Block Diagrams

Below is a state diagram of the behavior of the system. When system is in autonomous mode, it performs a user defined task after it has completed gyroscope calibrations, trim calibrations, and has acquired a GPS lock. If the AeroBot is in manual mode, it does not need to do any calibrations besides the usual trim settings before flying. An important note to keep in mind is that if the system is in manual mode, it should not be switched to autonomous mode.



The figure below shows the general organization of the software. The software is split into five separate C files:

1. main.c
2. RangeFinder.c
3. helicopter.c
4. gps.c
5. controls.c.



As depicted above, the functions of the AeroBot code are split into 5 different files, with main.c tying all of the other files together. Each of the other 4 files have a set of related modules that abstract the users from having to understand specific hardware details. The functions therefore provide a high level interface to the hardware modules such as the sensors. main.c should provide ways to tell the AeroBot what actions to perform and when. The other four modules must provide the hardware components with instructions on how to perform the task.

main.c

main.c ties all of the modules of the AeroBot together. It schedules the data acquisition from the sensors, analyzes the data, and sends the appropriate instruction to the helicopter controls.

interrupt [TIM0_OVF] void timer0_ovf_isr(void)
Description: Timer 0 overflow interrupt service routine. This is a hybrid round-robin scheduler for tasks.

interrupt [ADC_INT] void adc_isr(void)
Description: ADC interrupt service routine with auto input scanning

interrupt [USART0_RXC] void usart0_rx_isr(void)
Description: USART0 Receiver interrupt service routine

interrupt [USART1_RXC] void usart1_rx_isr(void)
Description: USART1 Receiver interrupt service routine

interrupt [USART1_TXC] void usart1_tx_isr(void)
Description: USART1 Transmitter interrupt service routine

void putchar1(char c)
Description: Write a character to the USART1 Transmitter buffer

void main(void)
Description: Encapsulates all of the initialization and general functionality of the AeroBot

float GetAx(void)
Description: Returns the acceleration in the X-axis in G’s

float GetAy(void)
Description: Returns the acceleration in the Y-axis in G’s

float GetAz(void)
Description: Returns the acceleration in the Z-axis in G’s

float GetGyroRoll(void)
Description: Returns the position of the gyro's roll

float GetGyroPitch(void)
Description: Returns the position of the gyro's pitch

float GetGyroYaw(void)
Description: Returns the position of the gyro's yaw

void RunGyroIntegration()
Description: Runs all the calculations for gyro data, loads into an array for later reading

int GetRangeFinderDist(void)
Description: Returns the smallest distance detected by the rangefinder

void Debug (void)
Description: Encapsulates all of the troubleshooting functionality such as displaying information to the LCD

helicopter.c

This module encapsulates the functions for communicating with the helicopter.

void SetThrottle(int throttle)
Desciption: Sets the throttle of the helicopter. Notes: 0 is no power, 100 is full throttle

void SetYaw(int Yaw)
Description: Sets the yaw of the helicopter. Notes: -50 is full left, 50 is full right

void SetRoll(int Roll)
Description: Sets the Roll of the helicopter. Notes: -50 is full left, 50 is full right

void SetPitch(int Pitch)
Description: Sets the Pitch of the helicopter. Notes: -50 is full pitch up (backwards), 50 is full pitch down (forwards)

int GetThrottle (void)
Description: Get the master throttle of the helicopter. Notes: 0 is no power, 100 is full throttle

int GetYaw (void)
Description: Gets the master yaw of the helicopter. Notes: -50 is full left, 50 is full right

int GetRoll (void)
Description: Gets the master roll of the helicopter. Notes:-50 is full left, 50 is full right

int GetPitch (void)
Description: Gets the master pitch of the helicopter. Notes: -50 is full backwards, 50 is full forwards

int GetRCThrottle (void)
Description: Gets the RC controller throttle value of the helicopter. Notes: 0 is no power, 100 is full throttle

int GetRCRoll (void)
Description: Gets the RC controller roll value of the helicopter. Notes: -50 is full left, 50 is full right

int GetRCPitch (void)
Description: Gets the RC controller pitch value of the helicopter. Notes: -50 is full back, 50 is full forwards

int GetRCYaw (void)
Description: Gets the RC controller yaw value of the helicopter. Notes: -50 is full left, 50 is full right

unsigned char CharToHex (char c)
Description: Converts a char value to hex value ('A' to 0xA)

void SetThrust (float x)
Description: Sets the thrust given a thrust in Newton

controls.c

This module manages the controls of the helicopter.

void CalibrateTrim (void)
Decription: Call this function to calibrate the trim if needed (assumes a level platfrom)

void ControlsManualOverride (void)
Description: This function is called to perform manual override of 1 or more axes while flying autonomously

void SetAltitude (int alt)
Description: This function is called to set the altitude valid range (0-400)

int GetAltitude (void)
Description: This function is called to get the altitude

void AdjustAltitude (int newRange)
Description: This function is called to adjust and maintain the altitude (hover) pass in the newest range value

void SetPosition (float x, float y, float z)
Description: This function is called to set the new position coordinate

float GetPositionX (void)
Description: This function is called to get the last set x coordinate

float GetPositionY (void)
Description: This function is called to get the last set y coordinate

float GetPositionZ (void)
Description: This function is called to get the last set z coordinate

void AdjustPosition (void)
Description: This function is called to adjust and/or maintain the MasterPosition

void AdjustYaw (float YawInput)
Description: This function is called to maintain the current yaw position setting

gps.c

*char *mid (char *str, int start, int len)*
Description: Extracts a string segment

float GetLatitude(vo id)
Description: Returns the latitude

float GetLongitude(void)
Description: Returns the longitude

float GetLongitude(void)
Description: Returns the longitude

*char *GetUTCDate(void)*
Description: Returns the UTC date

*char *GetUTCTime(void)*
Description: Returns the UTC time

float GetElevation(void)
Description: Returns the elevation

int GetFix(void)
Description: Returns the GPS fix state (1 = No Fix, 2 = 2-D fix, 3 = 3-D fix)

*void ParseNMEA(char *Data)*
Description: Extracts the proper data from the NMEA string

RangeFinder.c

void init_rangefinder()

Initializes the rangefinder unit. Holds the bus busy until initialization is complete

void start_ranging()
Description: Starts the ranging operation. Holds the bus and microcontroller busy until the command is successful.

unsigned int get_range_value(void)
Description: Gets the last value found by the rangefinder. Returns an unsigned int. Must be called at least 65ms after start_ranging().