THE UNIVERSITY OF BRITISH COLUMBIA

Department of Electrical and Computer Engineering

Final Report

EZ-FLY RC Helicopter

7/24/2007

EECE474 Team 3
Scott Davis
Roy Ho
Henry Kwan
Louis Liang
Sijia Wang

ABSTRACT

This document gives details on the EZ-Fly four rotor radio controlled helicopter. While they may seem simple, helicopters are naturally unstable aircraft with many subtleties. Unless designed very carefully, they are subject to disturbances in the air, rendering them very difficult to control and stabilize. The electronics need to be carefully managed, since the current drawn by the four motors is on the order of 8 Amps, an amount which may damage sensitive components. The most challenging aspect of designing an auto-stabilizing helicopter of this nature is the software – programming a microcontroller to deal with the constant tilt corrections, combined with the input and output port processing, is a time-consuming, complicated task. The task is further compounded by difficulties introduced by noise in the signal processing, by the difficulties in getting accurate readings from the sensors due to the dynamic and always moving system, which changes even readings of gravity, due to acceleration. Despite the myriad problems involved in designing a stable helicopter, it is not impossible. It cannot be done quickly, nor cheaply, but it can be done.

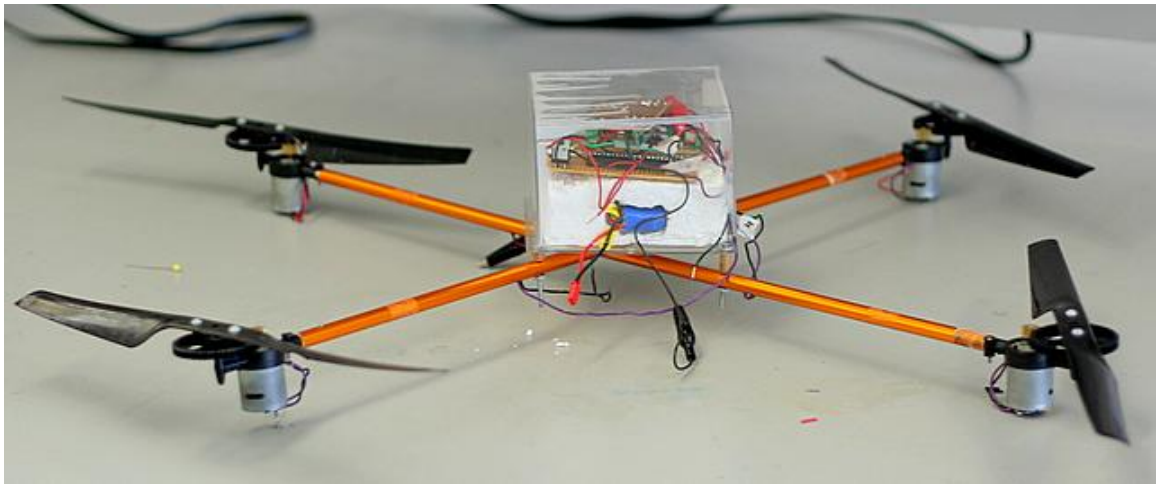

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

Radio Controlled helicopters are wonderfully entertaining devices that can provide hours of entertainment. However, they are also very costly, and have a steep learning curve. RC helicopters come in a variety of sizes, from micro helicopters that fit in the palm of your hand, to gas powered models which can be up to about half the size of a small car. The larger the model, the more expensive it is, and the more expert you need to be to control it. Even with a beginner's model, it can take weeks of practice for a novice user to learn how to even get the aircraft to hover with any degree of stability. Due to the complex nature of helicopter aviation, the radio transmitters often need more than 6 frequency channels which the pilot needs to use to control all the different axes of movement.

In standard RC helicopters, there often is no dedicated micro-controller. There are simply mechanical and electrical controls which are all controlled indirectly by the pilot through the RF link. There are controls for adjusting the aircraft's yaw, pitch, roll, and altitude, which each need to be adjusted in real-time for stable flight. And when the pilot slips up and makes a small mistake, gravity, as it always does, eventually wins the argument and a costly crash occurs.

As a result, people who don't have the required time or money to invest in this hobby are unable to participate in it. For our EECE474 design project, we seek to design and build a radio controlled helicopter that is easy to fly, and which can be built with a modest budget – in our case, under $400.

This report contains details on efforts made by others in this area, then goes on to explain our goals for the project, and how we managed to implement them, with differing degrees of success.

PROJECT BACKGROUND

FIGURE 1 – TYPICAL BEGINNER'S RC HELICOPTER (CRAZYABOUTGADGETS, 2007)

To penetrate the beginner's market in the RC helicopter hobby industry, there are products which purport to be easy to fly right out of the box. These beginner's helicopters often use fixed pitch rotors, meaning that the device cannot in practice be steered and directed all that effectively, since all it can do is move up and down, and (in some cases) change direction. We seek to build a helicopter that can be steered in any direction, easily.

Researchers at MIT have been using commercially available RC helicopters and modifying them to be controlled by computer algorithms, complex enough that multiple aircraft can be controlled to work together to perform various UAV tasks, such as search and rescue, or vehicle tracking. While this approach is novel, and significant progress has been made, that type of application is useful more for military or law-enforcement applications, instead of the average consumer. (MIT, 2007)

A company called SilverLit has released a product that is basically a commercially available version of what we propose to build. It is a four rotor gyro stabilized toy helicopter that claims to be very easy to fly, with controls for up/down, left/right and forward/back. It is also not overly expensive, coming in at about the $200 range (Silverlit Toys Manufactory Ltd., 2007). However, upon reading user's comments and reviews, the reliability of this product seems to be in question, with reports of fragile components, and denials of its ease of use (DiBona, 2006).

FIGURE 2 – SILVERLIT X-UFO R/C HELICOPTER (R/C AIRPLANE WORLD, 2007)

FIGURE 3 – DRAGANFLYER SAVS STABILIZED AERIAL VIDEO SYSTEM GYRO STABILIZED RC HELICOPTER (DRAGANFLY INNOVATIONS INC., 2007)

Also available are more complete, thoroughly built machines that are full of features and abilities, but cost significantly more than what we're striving to do. An example is the DraganFlyer® brand of helicopters (see Figure 3), which can cost in excess of $2000.

The steep learning curve of piloting RC helicopters is for the most part taken as a given, as part of the territory of the hobby. We seek to change that.

DESIGN REQUIREMENTS AND GOALS

HELICOPTER FRAME DESIGN

In line with the goal of providing a stable, easy-to-fly helicopter, we will design it to be a four-rotor helicopter. A four rotor helicopter is the simplest design mechanically, since the four rotors automatically balance out the torque of the motors, something which needs to be dealt with in single-rotor designs with a tail rotor.

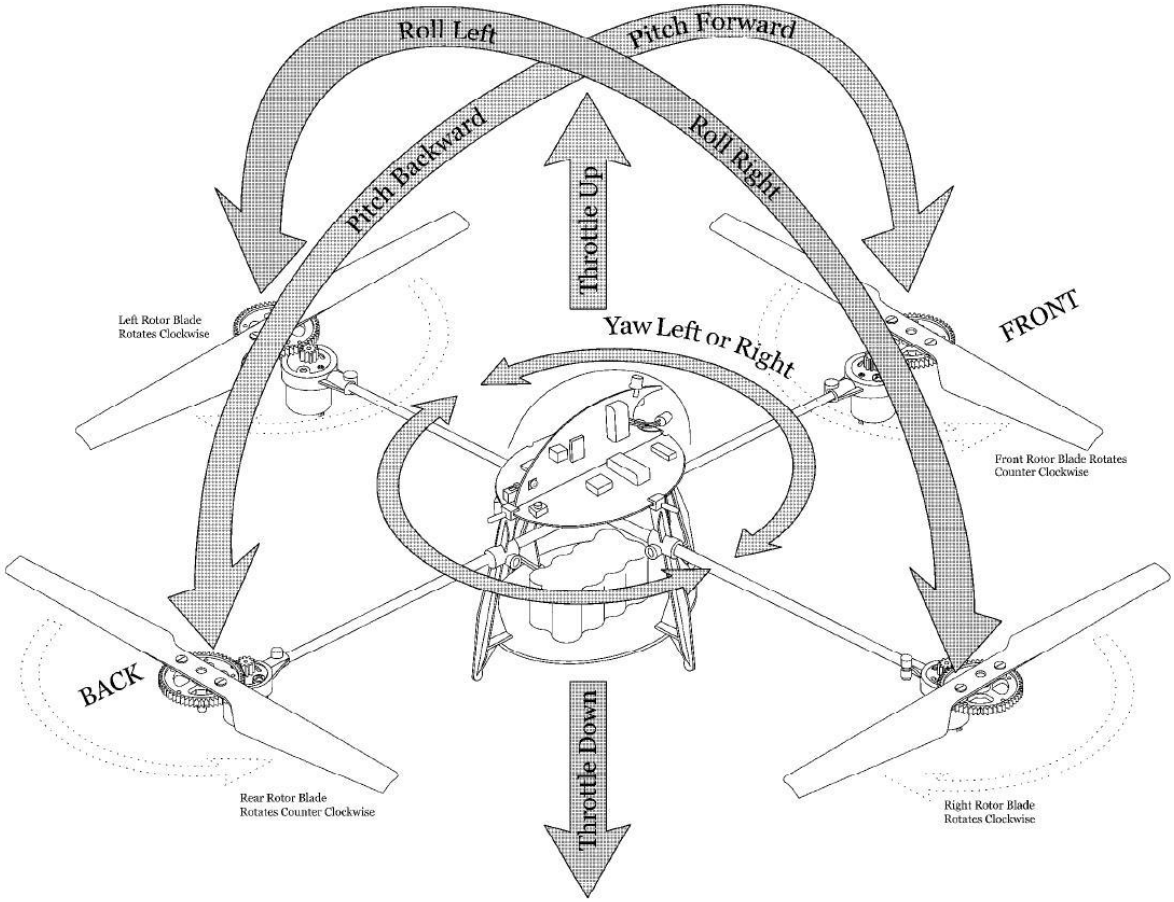

FIGURE 4 – AXES OF MOVEMENT & EXAMPLE OF FRAME DESIGN

Figure 4 above shows the general design of a four-rotor helicopter. It will consist of a main body in the center which houses the electronics; four booms extend out from this main body at 90 degrees to each other, and at the end of each lies a motor mount, a DC motor, and a gear for the rotor. As can be seen from the figure, each opposite pair of rotors rotates in opposite directions to provide the torque balance. This design simplifies things, since there is no need to implement individual rotor tilting, and is also inherently more stable than a single or even dual-bladed design, due to the spread-out nature of the rotors. Figure 4 also highlights the dynamics of helicopter flight – it shows the axes of movement which we will need to control. Because the only source of energy to the system comes from the four sets of rotors, all movement must come from modifying the level of downward thrust provided by each rotor. For

example, to turn the aircraft left, we would increase the thrust provided by the right rotor blade while simultaneously decreasing the thrust of the left rotor blade. This would cause the aircraft to roll to the left, causing the net downward force to be at a slight angle to the Normal, introducing a lateral force component, which would cause the aircraft to move to the left. The same principle applies when moving in any of the other directions.

FLIGHT CONTROL

To differentiate the EZ-Fly helicopter from other commercially available helicopters, we wanted to give as much control of the helicopter to the on-board microcontroller as possible, i.e. we wanted the input provided by the user to be as simple as possible. To this end, we plan on developing a control method that will allow the user to provide control to the following directions of flight for the helicopter:

- Up/Down
- Left/Right
- Forward/Back

There are different ways of accomplishing this that would provide the same result. One possible method is to use a lever/switch system, where each direction pair (up/down, right/left, forward/back) would get its own lever with up to 255 levels of accuracy (i.e. for an 8 bit digital signal). Thus, the more you move the lever, the more the helicopter would move in that direction.

An alternative way would be to use simple pushbuttons, with a specific amount of direction change per button push.

Both of these alternatives would require encoding the information into a format easily transmittable over RF to the helicopter, where the signal would be decoded. Either way, the user would not have to deal with stabilizing the helicopter, since that would be taken care of by the onboard electronics. Due to the time constraints of this project, we have decided to simplify our goal for the end of the project, and not implement any direction changing method. Rather, we will require the helicopter to simply respond to a “Go” signal from the RF handheld transmitter, at which point the helicopter will execute a simple takeoff-up-hover-down-land flight pattern. However, to fully implement the ability to turn and move forward at the user’s discretion, the only changes necessary would be both in the software, dealing with the user-provided course changes, as well as in the RF transmitter circuit.

MOTOR CONTROL

In order to provide stable flight, we need to be able to control each motor independently, with a significant degree of accuracy. Since the DC motors each draw a lot of current when in operation, we need a way of controlling their speed with the microcontroller that would not hurt the sensitive IC’s due to the large current draw. We propose to do this with an H-bridge, setup with optocouplers to completely isolate the microcontroller circuit from the motor circuit. Each motor would get its own optocoupler and H-bridge, which receives as input a Pulse-Width-Modulated (PWM) signal from the microcontroller. It will be the Duty Cycle of the PWM signal that will dictate the speed at which the motors run. The optocouplers prevent any back EMF or noise to disrupt the PWM signal coming from the microcontroller.

HELICOPTER STABILITY

If it is to fly by itself with minimal user intervention, there needs to be a system to automatically stabilize the helicopter while in flight. Otherwise, the slightest gust of wind will drive it unstable and cause a most unfortunate crash.

To accomplish this, we need a device capable of measuring the tilt of the helicopter, or in other words, we need to know when the angle of the helicopter changes from 0 degrees to anything else. If the microcontroller can be told instantly when the angle of flight changes, it can make minute changes, many times per second, to the duty cycle of each individual motor's control signal, compensating for the angle. For example, if the helicopter starts to tilt to the right, the sensor would detect the change in angle, notify the microcontroller, which would then slightly increase the power to the right motor, while slightly decreasing power to the left motor, bringing the helicopter back into level flight. This process would need to be performed many times per second, and continuously during flight, to ensure stability.

There are two types of sensors which could help do this: gyroscopes and accelerometers. A gyroscope would tend to be more accurate and less affected by gravity, but costs significantly more than an accelerometer. Thus for budget concerns, we will use a multi-axis accelerometer to monitor the X and Y axes of tilt. The outputs from the accelerometer are analog signals which can be tied directly to the microcontroller's Analog to Digital ports.

MICROCONTROLLER

The EZ-Fly helicopter will need a brain if it is to operate as desired. Requirements for the microcontroller include:

- Small, easy to program
- On-chip memory to hold the program
- Minimum 3 A/D ports for sensor inputs
- Minimum 4 output ports for the PWM signal to the motors
- Input port for RF signal

We also need the microcontroller to have a setup such that it is easy and quick to download the program to its memory.

RF LINK

To control the helicopter remotely, we will need some form of RF communication. The more complicated we make the handheld controller, the more complicated the scheme necessary for RF transmission. Professional RC helicopters and planes are equipped with multi-channel transmitters, since the user needs to control several aspects of the flight simultaneously. Due to our limited goal of a "Go" signal for an on-board flight path, our RF transmitter can stay very simple. We need to transmit a signal, any signal, which the microcontroller on the helicopter can recognize. To this end, we will use a simple 555 timer IC to produce a square wave pulse which will be transmitted whenever the push-button is pushed down. The frequency and duty cycle of this pulse are not important, since it is not the nature of the signal, but merely its existence, that the microcontroller will read. A 9V battery will be sufficient to power the circuit.

POWER

As mentioned above, the DC motors require a significant amount of current. To provide this current, we will need a battery pack which has sufficient voltage to enable lift-off, but one that is also lightweight, and can easily handle at least 8A of current draw. Lithium-Polymer batteries are the most commonly used for this type of application; they come in 3.4V per cell, often tied in multi-cell packs. Since they are also fairly expensive, a 2 cell 7.4V pack will be all we need for this project.

To power the microcontroller circuit, RF receiver, and accelerometer, we will need at least two different voltages: a +5V for the microcontroller circuit, and a +3.3V Vdd source for the accelerometer. A small 9V will do the trick, combined with two voltage regulators to dial down the voltage for the microcontroller and for the accelerometer. The RF receiver can be powered by up to 12V, so the 9V from the battery will work just fine.

OVERALL DESIGN GOAL

For the EZ-Fly helicopter, the end goal of our project is quite ambitious. The original goal of having a fully functional helicopter which can be flown in any direction is unrealistic in the given time frame. So the goal for this time frame is to have the helicopter be able to lift off the ground without any assistance, and to visibly remain stable in the air for several seconds before landing back onto the ground.

RESULTS

MECHANICAL DESIGN

The first design for our helicopter frame involved using a Styrofoam block as the material for the main body. We then had 9mm diameter aluminum tent poles coming out of the block, acting as the 4 booms of the helicopter. Figure 5 shows this first design.

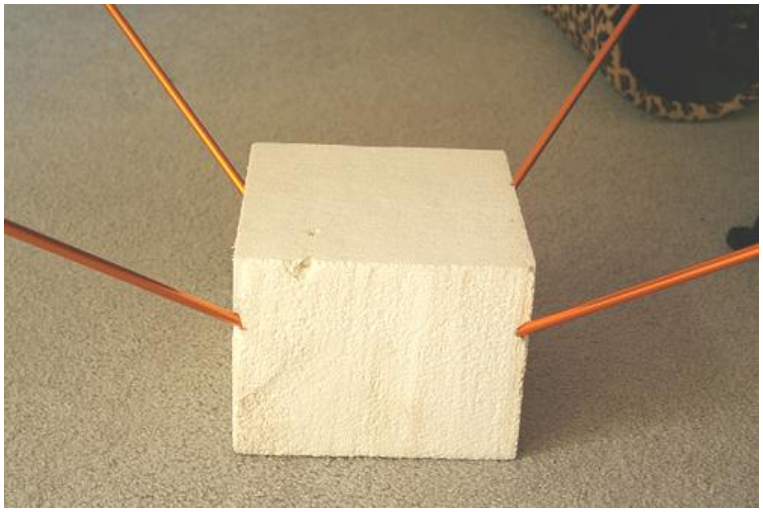

FIGURE 5 – FIRST HELICOPTER FRAME DESIGN

The poles entered into the foam block 2/3 of the way from the bottom, at an angle of 36.5 degrees, all converging at the bottom of the block in the center. The plan was to cut a hole in the top middle of the block for the electronics, feeding the wires through a small channel underneath the foam to the open ends of the four poles at the bottom. This design was originally conceived as being more stable, since the thought was that the center of gravity would be far below the sources of lift, increasing stability.

We soon realized that the motor/rotor system at each apex would be a significant weight factor, making it more top-heavy than originally thought. A redesign was made, and Figure 6 shows this replacement design. We returned to the design implemented by the Draganflyer brand of helicopters, with the booms extending out from the body parallel to the ground. A polycarbonate plastic sheet was used to bind the poles together in the center (with screws going through the plastic into the aluminum), as well as to provide a base upon which we could place the electronics of the system. This design turned out to be far more stable, and more firm and secure than the original design would have been.

FIGURE 6 – CURRENT HELICOPTER FRAME DESIGN

The motor mounts are attached to the aluminum poles by means of small aluminum dowels machined by the ECE machine shop, with small

machine screws placed to as to prevent movement. The two wires connected to each DC motor are fed through the hollow aluminum tubes and pulled out of small wires drilled in the tubes near the center, so as to be fed up into the electronics.

To house the electronics, we decided to use some of the foam from the original frame idea, cut into a small block that would fit on the polycarbonate base, and stick the circuit boards to it. This foam block can be seen in Figure 7.

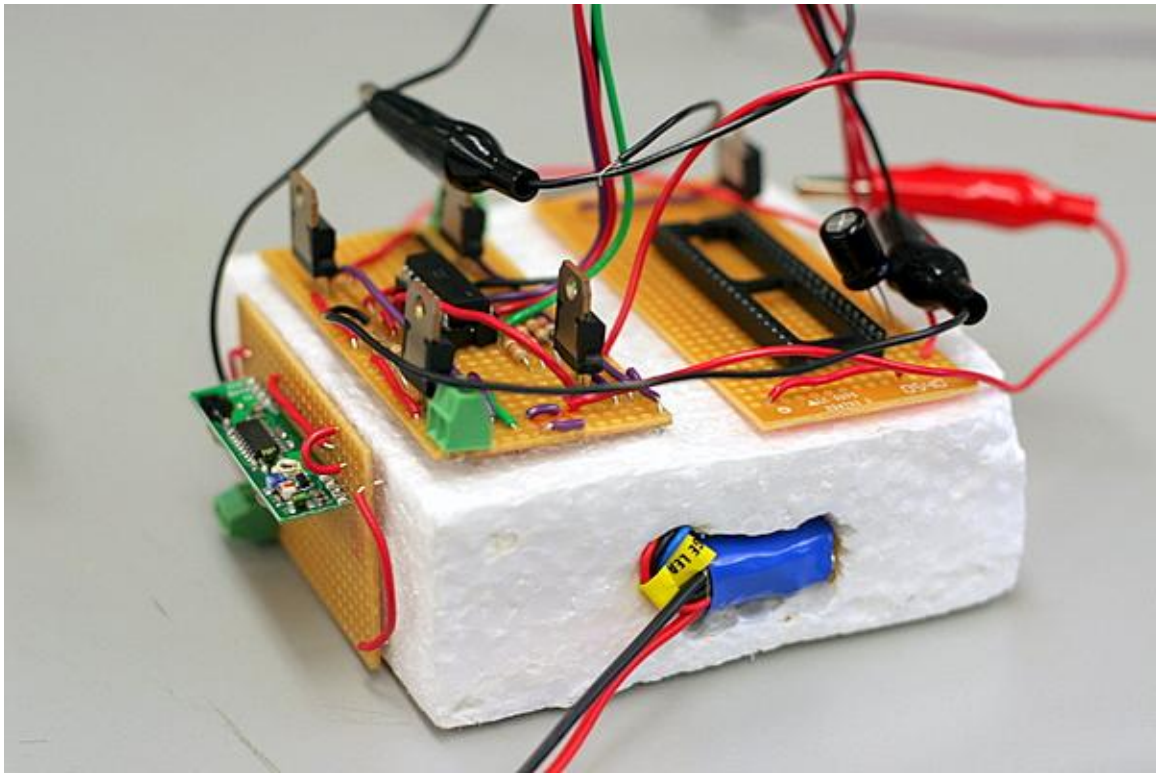

FIGURE 7 – FOAM BLOCK FOR HOUSING HELICOPTER ELECTRONICS

The 7.4V Lithium-Polymer battery is inserted into a hole melted out of the interior of the block. There is a similar hole cut out on the other side of the block for the 9V battery. This foam block is wedged in-between two plastic risers screwed into the polycarbonate base, so as to be securely, yet not permanently, installed onto the helicopter frame.

During testing with the lab's DC power supplies, it was found that each pair of DC motors drew the power supply's maximum current, around 4 amps. So a rough estimate of the total current draw from the 4 motors is about 8-9 Amps. This works out to be just about right, because the E-flite 7.4V 800mAh 2-Cell LiPo battery that we purchased has a maximum continuous current draw of 8A, with burst discharges of up to 12.5A(Hobby Zone, 2007).

Testing of the stability of the frame design went well – with the motors connected to either the DC power supply or the battery, the motors provided plenty of lift to enable take-off, but the system, such as we were testing it without the microcontroller, was inherently unstable. Because all 4 motors were being powered by the same signal, no auto-stabilizing was occurring. Thus, while holding the frame with

our hands, any slight movement caused the system to go unstable and veer to one side, requiring a manual correction with our hand. This aside, the frame itself seemed reasonably well balanced. For although it was impossible to ensure that each boom was exactly the same length, and that the 4 motors were exactly equidistant from each other, the differences due to these imperfections are small enough so as to be easily compensated for electronically.

It was found that without the electronics on board (a not insignificant amount of weight), a voltage of about 4.7V applied to all 4 motors equally was sufficient to produce take-off. With the electronics block on board, it was closer to 6V where lift-off occurred.

The aluminum poles used are strong enough that even when tested at full length (on the helicopter, they're used at half length), it took almost all the force a human could apply to bend the pole any significant amount. The forces the poles are subjected to during flight are nowhere near the forces applied by two hands trying to bend the pole, and are thus deemed quite strong enough for stable flight.

ELECTRONIC CONTROL

SCHEMATICS

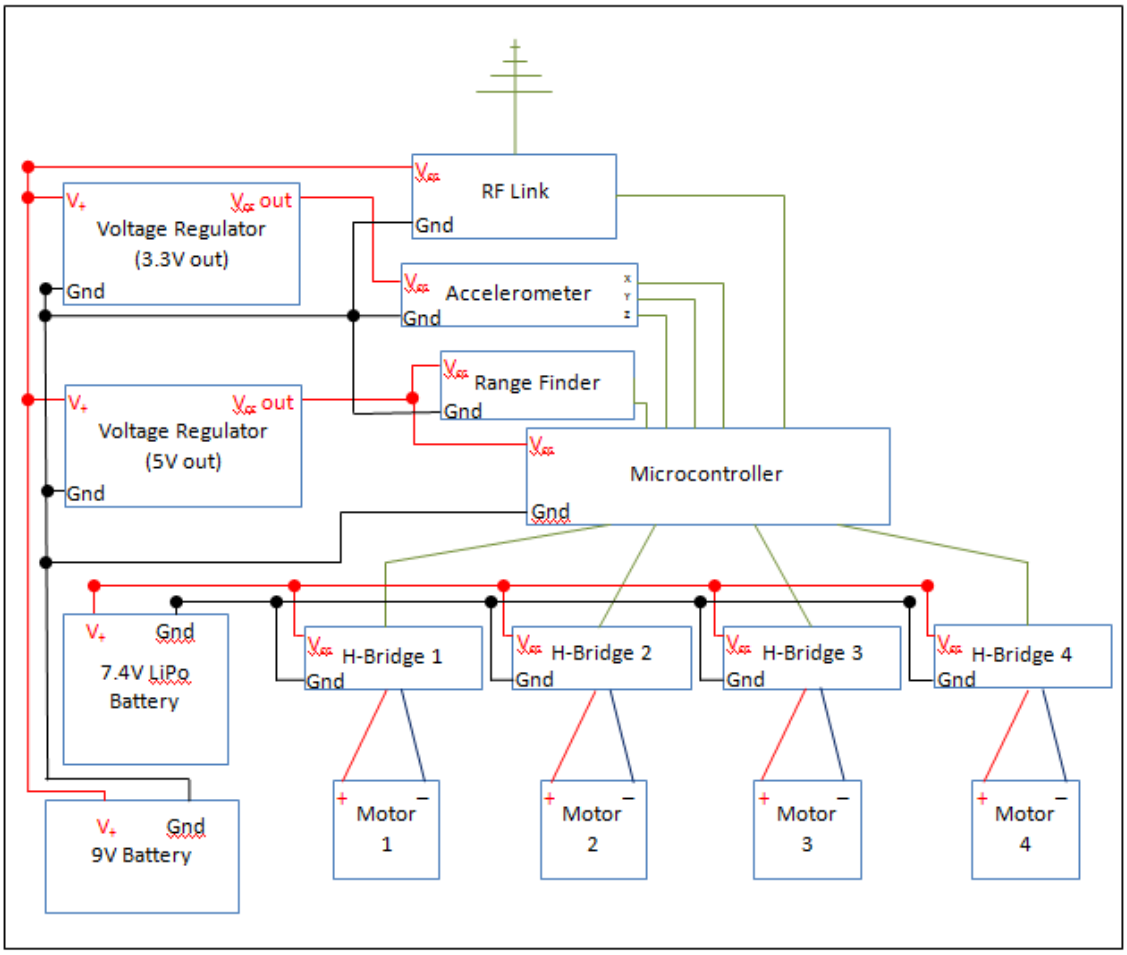

FIGURE 8 – BLOCK DIAGRAM OF HELICOPTER CIRCUITRY

ELECTRONIC SPEED CONTROL

In our project, an electronic speed change circuit is required to control the speed of the motor, which should change depending on the output signal of the Microcontroller. The circuit that would satisfy this goal is an H-bridge, an electronic circuit which enables smooth control of a DC motor with the use of electronic components such as MOSFETS and Optocouplers.

An H-bridge circuit that would be normally used in a robotic project contains 4 MOSFETS and a single Optocoupler and could control the DC motor in both directions. For the EZ-Fly helicopter, there are four DC motors and each one of them needs to be controlled by an H-bridge circuit. However, only 1 MOSFET and 1 Optocoupler is required for each H-bridge, since the motors only spin in one direction. With such a modification to the H-bridge, the electronic components required to build the circuits are significantly reduced and using less components also benefits our project since we are on a tight budget.

A single H-bridge circuit consists of one MOSFET, 3 resistors and one Optocoupler. The Optocoupler is powered by a single output pin of a microcontroller. In the case of our project, the output from the microcontroller is a Pulse Width Modulated signal. When the Optocoupler is powered, it would then trigger the MOSFETS. Once the MOSFETS triode mode condition is satisfied, the MOSFET would be turned on and allow current through the DC motor. However, the MOSFETS would operate at cutoff mode when the optocoupler is not powered. And therefore the DC motor would stop running.

OPTOCOUPLER

An Optocoupler can be represented graphically as a Light emitting diode and a Transistor in parallel. However, this transistor has only two pins: a collector pin and an emitter pin. When the voltage across the LED is 5V, the transistor is triggered and current will flow from the collector to the emitter.

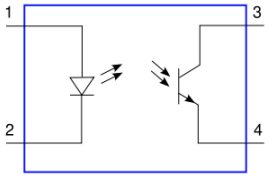

FIGURE 9 – OPTOCOUPLER (WIKIPEDIA, 2007)

The configuration of the Optocoupler in our project is very simple. The positive pin of the LED is directly connected to the 5 volt battery through a 19 kΩ resistor and the negative pin is connected to the output of the microcontroller. When the microcontroller outputs 5V, the voltage difference across the LED is zero and the Optocoupler is turned off. When the microcontroller outputs 0V, the voltage difference across the LED is 5V which powers on the Optocoupler.

The emitter of the transistor is connected to ground while the collector is connected both to a 100 kΩ resistor and to the gate pin of the MOSFET. The other end of the 100 kΩ resistor is connected to the Lithium-Polymer battery. When the Optocoupler is powered, the gate voltage would be at approximately 0V and the MOSFETS would be turned on.

The model of Optocoupler we used in this project is LTV847 IC. The other model that is also available are the QTC4N35—a 6-pin single Optocoupler. This model was used during testing of the H-bridges because the LTV847 wasn't available for a long period of time. However, the nature of operation between QTC 4N35 and the LTV 847 are the same except the fact that one is a 6-pin single Optocoupler IC while the other is a 16-pin four Optocoupler IC.

MOSFET

The other component of the H-bridge circuit is a MOSFET, used as a switch. A MOSFET has 3 pin: Gate, Source and Drain. Voltage change across these pins would change the mode of operation of a MOSFET. The Gate pin is connected to the collector of the Optocoupler through a 100 Ω resistor, the Source pin is connected directly to the 7 volt battery and the Drain pin is connected to the DC motor.

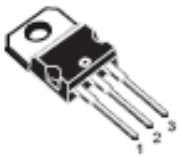

FIGURE 10 – MOSFET CHIP (STMICROELECTRONICS, 2007)

In an H-bridge, MOSFET functions as a switch and therefore triode mode is preferred. When two of the following conditions are satisfied: voltage across the gate and source (V_{GS}) is larger than the threshold voltage (V_{th}), and the difference of V_{GS} and V_{th} is larger than the voltage across drain and source(V_{DS}). On the datasheet of P12PF06 MOSFET, the maximum threshold voltage is 4 volt. When the Optocoupler is powered, it would satisfy the first condition since the V_{GS} would be at approximately 7V which is larger than maximum V_{th}. The difference of the V_{GS} and V_{th} is now 3Vand it must be larger than V_{DS} to satisfy the second condition. The resistor across Drain and Source (R_{DS}) is less than 0.2 Ωand the current across the Drain and Source (IDS) would be 2 A at maximum. Therefore VDS is approximately 0.4 V using Ohm's law. Both conditions are satisfied when the Optocoupler is turned on and therefore the MOSFET would allow current flow through the motor.

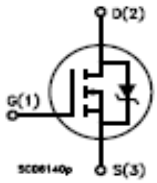

FIGURE 11 – MOSFET GATES (STMICROELECTRONICS, 2007)

H-BRIDGE

When all the components are connected together and powered, the H-bridge would work as following. Whenever the microcontroller outputs 0V, the motor is turned on at full strength. Since the microcontroller outputs PWM, the motor would be running when the pulses are at 0V. The longer the signal stays at 0V the faster the motor would spin, up to its maximum. Therefore we can control the motor speed by changing the duty cycle of the PWM signal.

At a duty cycle of 0%, the motor, for example, would be running at full speed while at duty cycle of 100%, the motor would stop completely.

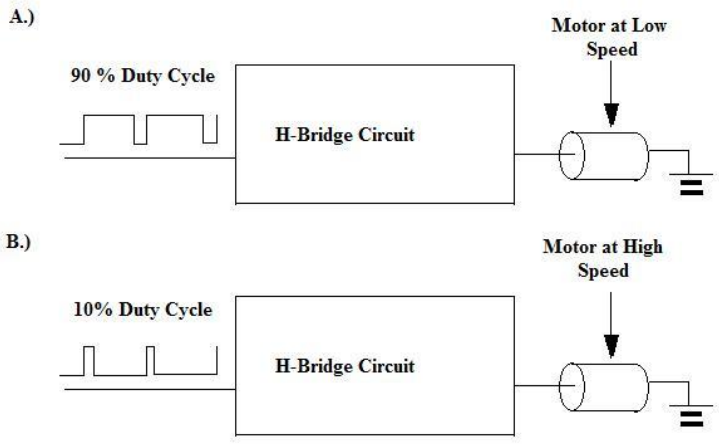

FIGURE 12 – DUTY CYCLE CONTROLLED MOTORS

TESTING

The H-bridge circuit was first built on the breadboard for testing purposes. After it was constructed, the circuit response was observed on the display of the oscilloscope. The H-bridge behaves as expected: a shorter duty cycle square wave signal would produce a faster motor speed while a longer one would slow the motor.

At one point during testing, we used the signal generator to produce the PWM signal to the H-bridge. This worked quite well, and we were able to get the helicopter to hover, if somewhat unstably. At one point the testing wire we used to connect the signal generator to the helicopter became accidentally disconnected, causing an instantaneous 0V on the PWM input, causing an instantaneous jump up to full motor speed. This sudden jump in current draw shorted out the MOSFETS, and cause a small crash, since we were unprepared for the sudden increase in thrust, and the helicopter jumped up into the air in a very unstable fashion.

A PCB was produced for the microcontroller + H-bridge circuit, but a design flaw which had the microcontroller and the motors power by the same battery caused us to split up the circuit into 3 parts, on separate boards: microcontroller + voltage regulator, RF circuitry, and the H-bridge. Figure 13 below shows the H-bridge setup for our helicopter.

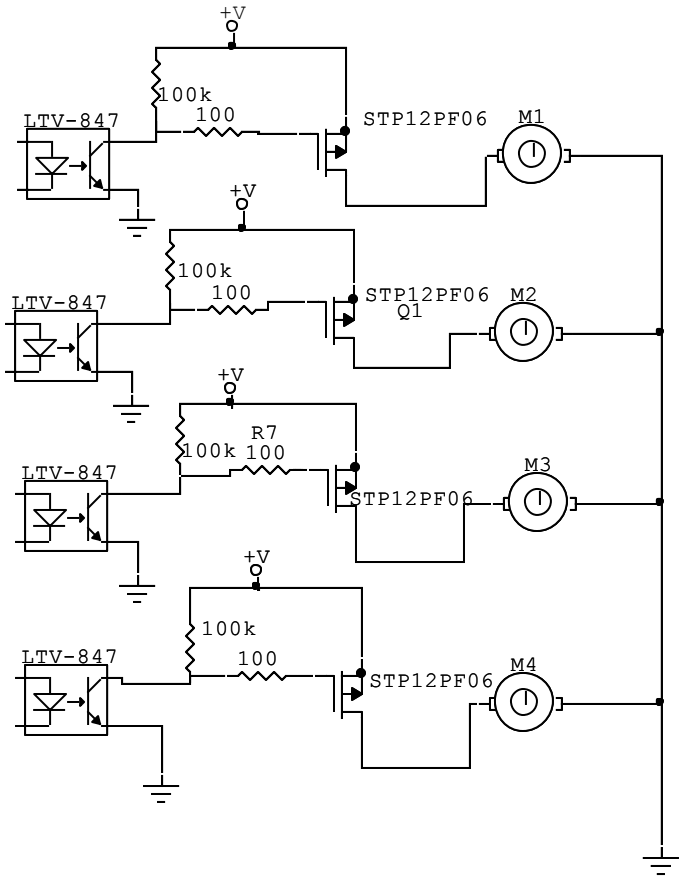

FIGURE 13 – H-BRDIGE CONNECTED TO MOTORS

RF COMMUNICATION

After debating on multiple RF TX/RX schemes, our group decided on using the TWS-434A and TWS-434 RF TX/RX ready-made chips for our project. The reason behind the decision is that these chips are easy to configure, consume very little power, and were readily available in the ECE library.

The transmitter consists of 4 pins: positive power supply pin, ground pin, input pin and antenna pin. To power the transmitter circuit, the positive power supply and ground must be connected accordingly. The input signal is connected to the input pin. For the purpose of our report, an extended antenna is not required since we are testing the project within a very short range.

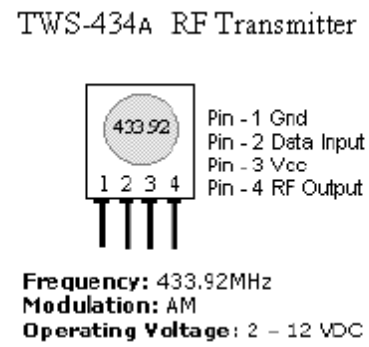

FIGURE 14 – RF TRANSMITTER CHIP(RENTRON)

The receiver consists of 8 pins: three ground pins, two positive power supply pin, one linear output pin, one digital output pin and one antenna pin. The digital output pin is becomes the input to the microcontroller. An extended antenna is also not used for the same reason stated above.

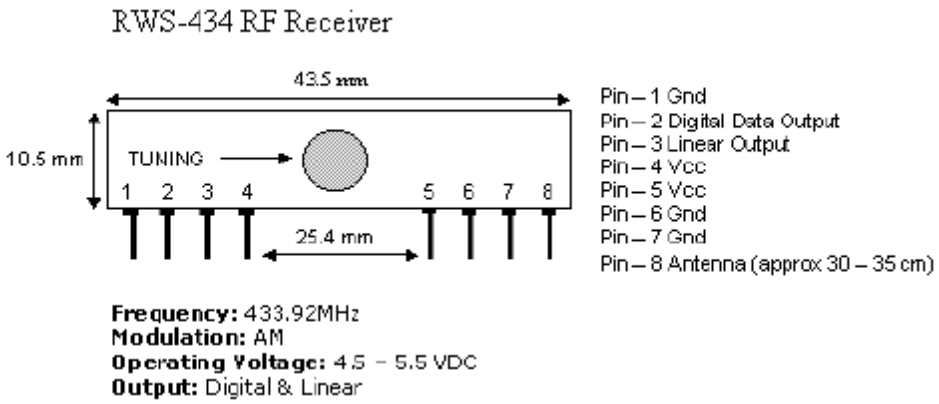

FIGURE 15 – RF RECEIVER CHIP (RENTRON)

The input to the transmitter needed to be a signal the microcontroller could easily recognize. The easiest way we found of implementing this was with a 555 timer producing a square wave pulse. This square wave pulse was fed into the transmitters input, and came out the receiver's output when activated.

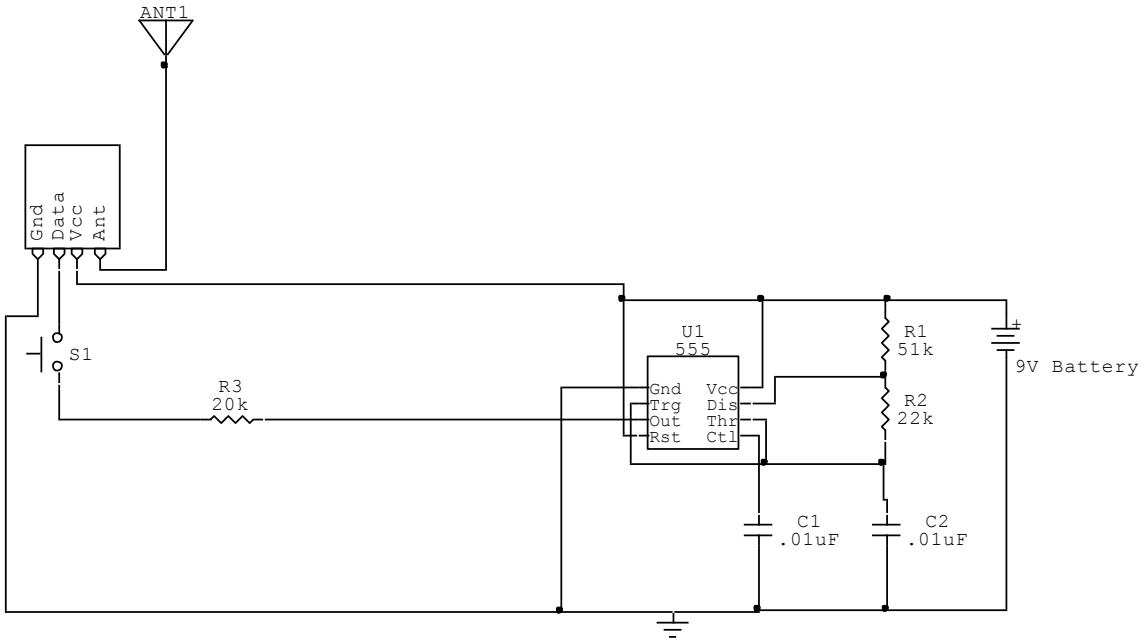

FIGURE 16 – 555 TIMER CIRCUIT ATTACHED TO RF TRANSMITTER

The switch is our pushbutton activator, which when pushed, sends the "Go" signal to the receiver, which then is tied into an input port on the microcontroller.

Testing of the Rx/Tx circuit shows that there is some degree of noise in the system, simply due to all the electromagnetic interference travelling through the air, but in general, when activated, the signal was sent through the air nice and strong, with very little attenuation, and only a slight time delay.

MEASURING TILT

ADXL330 ACCELEROMETER

To measure the tilt in the helicopter, an accelerometer was used. Traditionally, accelerometers are used to measure the acceleration of moving objects, but it can also be used to measure the tilt of an object if the acceleration due to gravity and acceleration due to other movements can be read separately. This will be mounted in the center of the helicopter, underneath the foam block, oriented such that each of the four booms are running along the sensor's X or Y axes, and where it will receive the most balanced noise distribution. Figure 17 shows the wiring schematic for our accelerometer, the ADXL330 by Analog Devices.

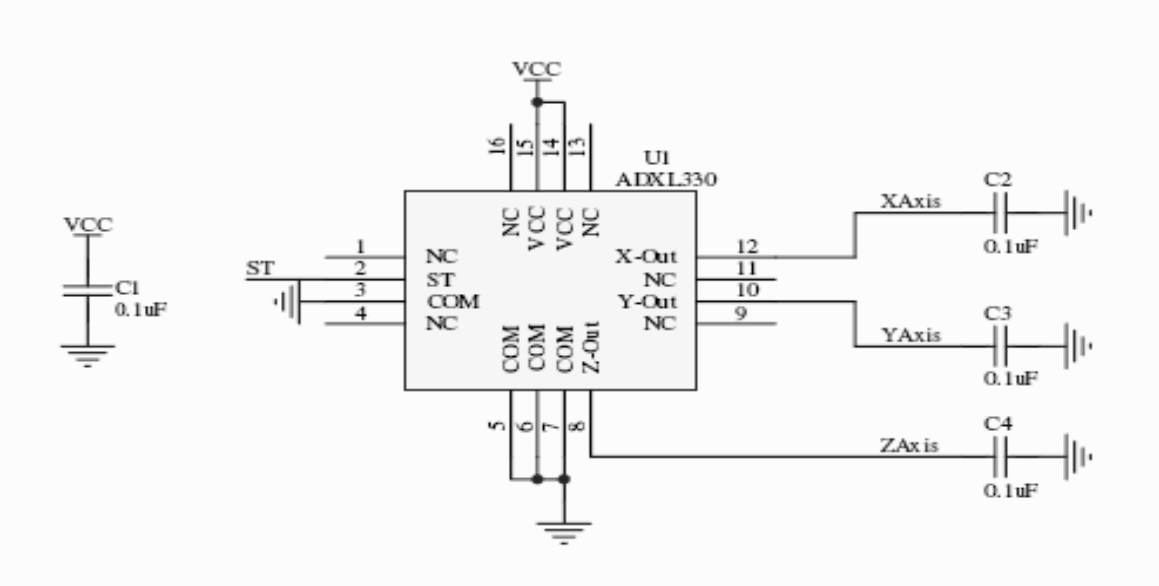

FIGURE 17 – ADXL330 ACCELEROMETER CIRCUIT

The ADXL330 is a three axis accelerometer capable of measuring ±3 *g* of movement, which is enough to measure the tilts we are expecting, with a sensitivity of 300mV/g. It's powered by 3.3V and only uses 320µA of current. When the accelerometer is perpendicular to gravity, it will have an output in the range of 1.1V to 1.3V. The output of the sensor is an analog voltage output proportional to the acceleration. The bandwidth is set to 50Hz by default, but can be changed up to 1,600Hz by adding capacitors at the outputs. The output of the ADXL330 will go directly into one of the microcontroller's built in A/D ports. Because the rest of the electronics require +5V Vcc, a voltage regulator was required to bring down the 5V to 3.3V: we used the LM317 adjustable voltage regulator.

The microcontroller used for this project, ATMEGA644V, is 8-bit. This means at 3.3v operating voltage, 255 samples of the input voltage can be obtained for each step. This gives the controller a resolution of 12.9mV.

To obtain tilt from acceleration, the following equation is used:

$$V_{out} = V_{offset} + \left(\frac{\Delta V}{\Delta g} \times 1g \times \sin\theta \right)$$

where

V_{out} = Acceleration Output in volts

V_{offset} = Accelerometer 0g offset (reference point)

$\frac{\Delta V}{\Delta g}$ = sensitivity (300mV/g for ADXL330)

1g = Earth's gravity (9.81m/s^2)

θ = angel of tilt

$$\therefore \theta = \sin^{-1}\left(\frac{V_{out} - V_{offset}}{\frac{\Delta V}{\Delta g}} \right)$$

Using the above equation, it can be seen that as the angle of tilt increases, the accelerometer loses accuracy. Figure 18 shows a graph of V_{out} vs. Degrees of tilt. As can be seen, the readings become non-linear as the level of tilt increases to perpendicular.

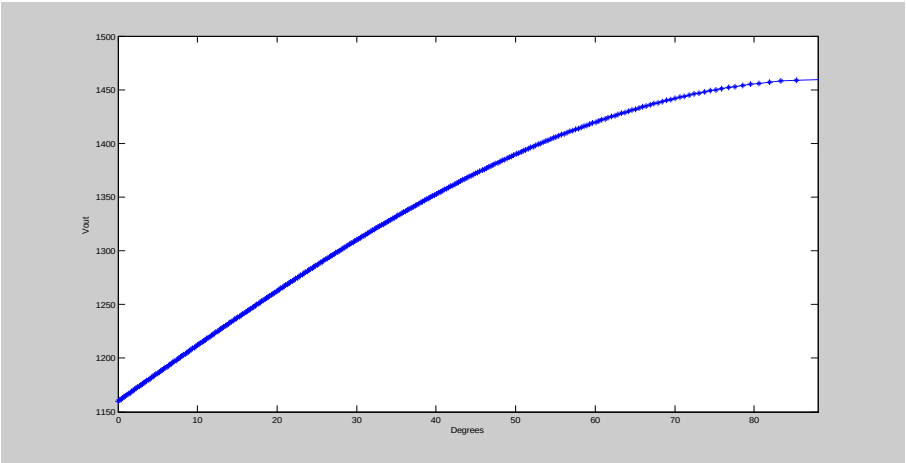

FIGURE 18 – ADXL330 ACCURACY

It can be calculated that the resolution of the ADXL330 is less than one degree near the zero degree mark, and more than six degrees near the 90 degree mark. For our purpose, the helicopter will experience more tilt near the zero degree mark rather than the 90 degree mark. It is safer to slow down or land the helicopter when it is near 90 degrees rather than try to control it because the helicopter can get out of control quite quickly when it is severely tilted. According to Analog devices, the z axis, which we're not using, is less sensitive.

	Nominal Output (V)	**Max(V)**	**Min(V)**
X axis	**1.16**	**1.56**	**0.74**
Y axis	**1.24**	**1.70**	**0.87**
Z axis	**1.19**	**1.45**	**0.97**

TABLE 1 – 3 AXES ACCELEROMETER OUTPUT

ULTRASONIC RANGE FINDER

It was determined that we needed a way for the helicopter to know by itself how high off the ground it is. This is necessary to enable automatic soft landing. When landing the helicopter without the range finder, it is easy to shut off the motors late/early. If the motors are shut off late, the helicopter will land at a high velocity which can cause a crash, leading to undesirable damages. If the motors are shut off early, it gives the same effect of dropping the helicopter from a distance, which also causes damages to the helicopter. With the range finder, the landing can be automated using the microcontroller. The range finders will also used to determine if the helicopter has reached a desired height.

We chose to use the Maxbotix LV-MaxSonar-EZ1 Sonar Module for our range finder. The most attractive feature on the EZ-1 is that it can output the range in three different ways: analog, PWM, and ASCII. This gave us more flexibility in terms of programming and reading the values. We chose the PWM output because we can check the values on the oscilloscope to test/debug the circuit as well as easy and

accurate readings for the microcontroller. The EZ-1 operates at 5V with 42 kHz ultrasonic ping. It only has one sonar module; therefore it doesn't have a central blind spot when there are two sonar modules. It has a 2.5 cm resolution and can measure from 15.24 cm to 645 cm. To read the PWM, every 147µs can be interpreted as one inch (2.5 cm), while the PWM has a period of 50ms. We mounted the range finder on the bottom of the polycarbonate plate. Since all it needs is a clear line of sight with the ground, it was not important that it be placed in the middle of the helicopter.

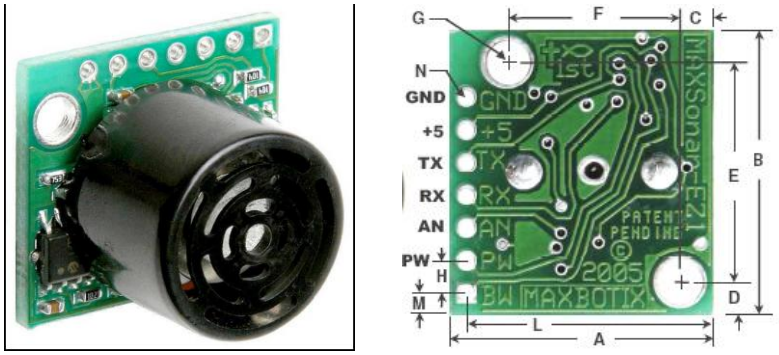

FIGURE 19 – EZ1 SONAR MODULE

The EZ-1 needs to calibrate the distance at every startup, and needs a clearance of at least seven inches, preferably closer to fourteen. The module will be calibrated after the first cycle. Because our helicopter itself has a height less than seven inches, we'll need to manually give the helicopter a seven inch clearance each time. The actual position of the EZ-1 is about 3 cm off the ground.

During testing, we encountered a lot of noise as the distance increased. The sensor becomes more sensitive to the surroundings. However, this is expected because as the distance increases, the sound wave from the EZ-1 will grow in size, which will bounce off surrounding objects causing the reading of the actual distance to be less accurate.

SYSTEM CONTROL

A linearized four rotor helicopter can be simulated with Matlab. We will use this as a guideline for our controller.

The dynamics of the helicopter can be described by the following equations (x and y only):

$$m\ddot{x} = -(r_1 + mg)\frac{\tan\theta}{\cos\phi}$$

$$m\ddot{y} = (r_1 + mg)\tan\phi \qquad \ddot{y} \approx g\phi$$

$$\ddot{z} = \frac{1}{m}(-a_{Z1}z - a_{Z2}(z - z_d))$$

$$\ddot{\psi} = -a_{\psi 1}\dot{\psi} - a_{\psi 2}(\psi - \psi_d)$$

$$\dot{\phi} = \ddot{\tau}_{\phi}$$

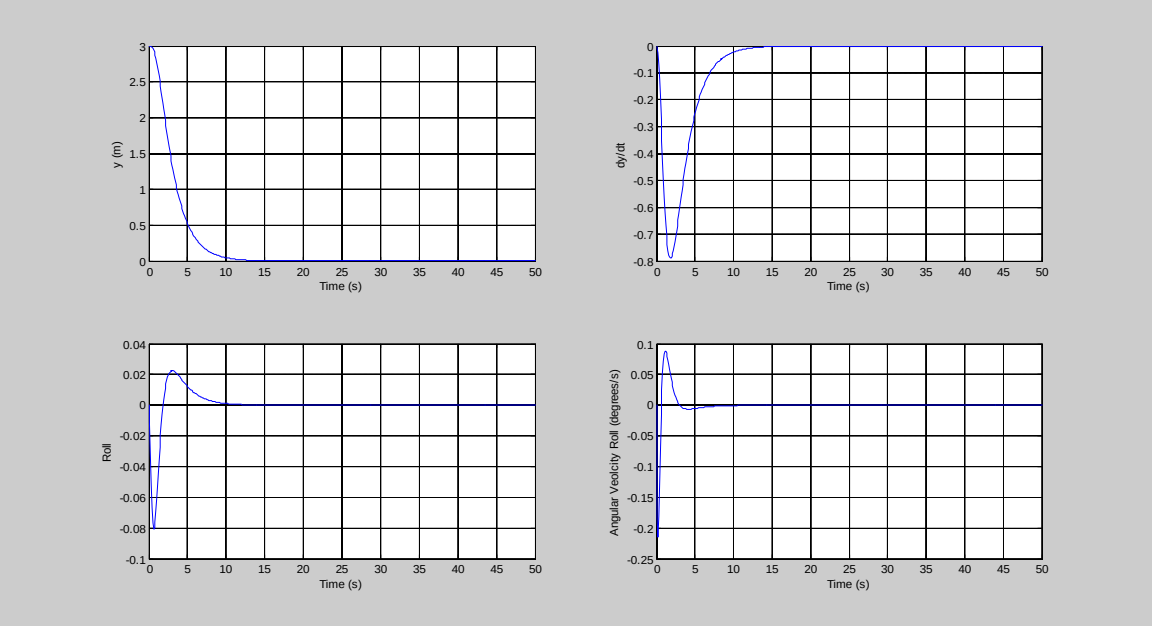

FIGURE 20 – MATLAB SIMULATIONS FOR CONTROLLER RESPONSE TIME

The calculations have initial conditions of three meters and zero tilt. As seen from the above graphs, the helicopter will take about 10 seconds to reach steady state while the velocity and angular momentum will experience spikes initially. Though the graph only shows the y values, we expect the x values to behave very similarly.

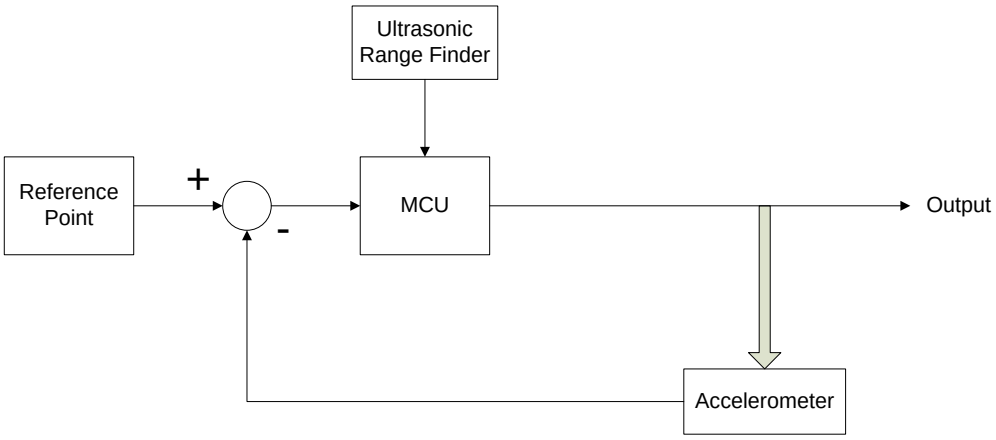

FIGURE 21 – FEEDBACK CONTROL LOOP OF HELICOPTER

The control loop can be summarized as follows:

- Reference point obtained by sampling the accelerometer outputs 20 times and finding an average when helicopter is first turned on and is on a flat surface.
- Current accelerometer outputs are compared with the reference point to determine the error/tilt. Will allow a 10% steady state error.
- MCU will adjust the Motor speed accordingly

- Although there are no inputs going into the accelerometer, tilt measurements by the accelerometer is directly proportional to the output of the MCU, i.e. The speed of the motors.

The ultrasonic range finder will interrupt the MCU when it reaches the targeted height or when landing height has reached.

MICROCONTROLLER

The motions of the helicopter are manipulated by a single microcontroller – the Atmel Atmega644. It is a high-performance, low-power AVR® 8-bit Microcontroller. The microcontroller chosen has 40 pins with DIP packaging. It is capable of operating at 8MHz and upon stabilizing the default internal clock frequency is adjusted to 1MHz. We selected this chip for our helicopter because it consists of 3 timers/counters, of which there are two 8-bit and one 16-bit timer with separate pre-scalers. Each timer/counter can generate 2 individual modulated waveforms allowing a maximum of 6 channels of pulse to be sent at designated outputs. We decided to use the 4 channels from the two 8-bit timers for motor speed controls. The 16-bit timer is used to measure the pulse width of input signals captured from the ultrasonic sensor which will be further explained in later sections. Moreover, the microcontroller has 8 analog to digital input channels that allows analog signals from the designated accelerometer – the AXDL330, to be converted into discrete digital data for further processing. Figure 22 below shows the general configuration diagram of the microcontroller:

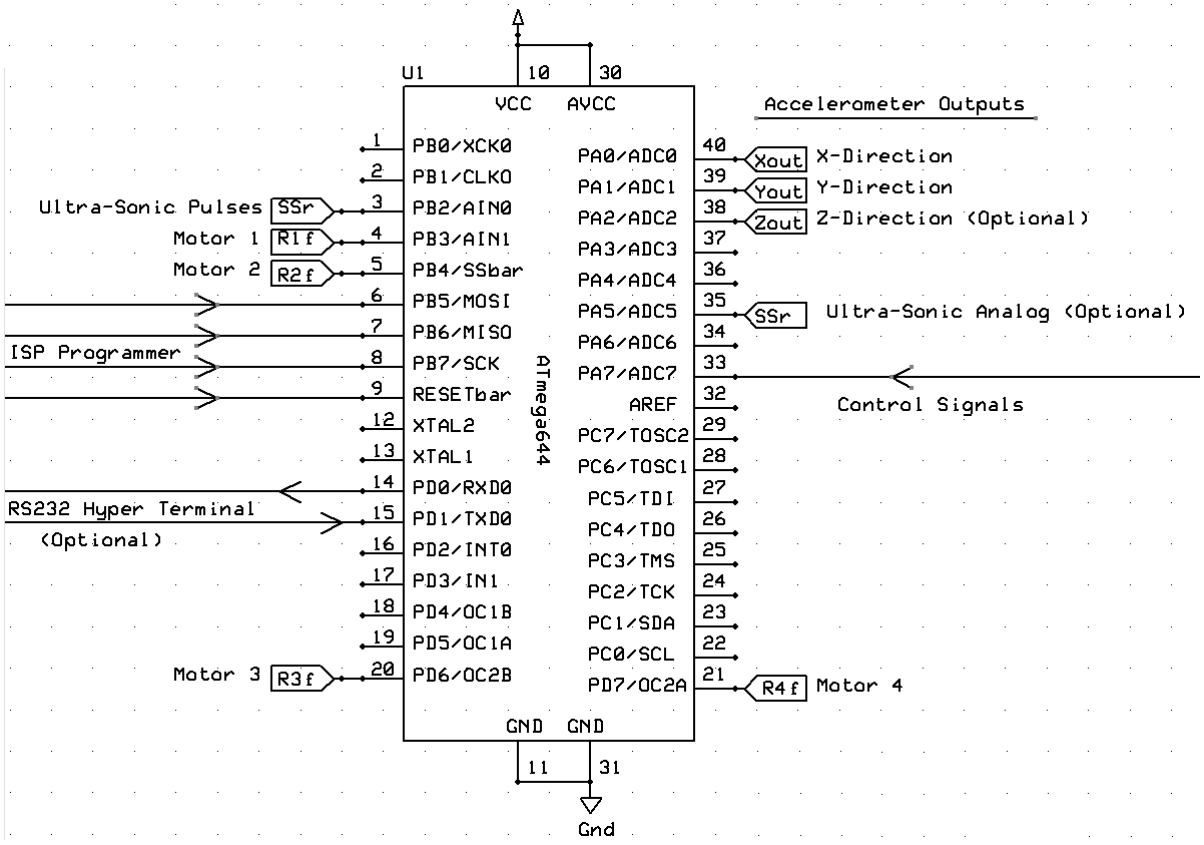

FIGURE 22 – HARDWARE CONFIGURATION OF MICROCONTROLLER

ANALOG TO DIGITAL CONVERSION(ADC)

The Atmega644 microcontroller has 8 ADC channels in Port A, and each of them can be used to convert analog signals into 10-bits digital data into the ADC data register. While in normal mode, Port A is simply an I/O port and its ADC function is disabled. To enable the ADC mode, special registers need to be configured and the reference voltage has to be set to properly convert the analog signals.

ADC SETUP

To set Port A into ADC mode, special registers need to be configured. There are 3 main registers that we need to get the conversion to start properly: ADMUX, ADCSRA, and ADCSRB. Figure 23 below shows the ADMUX register and its corresponding selection bits.

ADMUX – ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

FIGURE 23 – ADMUX REGISTERS FOR ADC

The two higher bits, bit 6 and 7, are used to configured the voltage reference for the conversion. If the analog voltage ranges from 0V to 1.1V or 2.56V, then the two bits should be set 10 or 11, respectively. If the analog voltage achieve as high as the power supply voltage (V_{cc}), then the two bits should be set to 01, otherwise 00 for a designated voltage. If a designated voltage level is required for the conversion, the voltage should be applied to the AREF pin.

The analog signals will be divided into a 10-bit digital data, that is, 1024 levels into a special register called the ADC register. In our case, the accelerometer outputs voltages in the range between 0 to 2.6V, so it is sufficient to use the internal voltage reference mode for the conversion. Bit-5 of the ADMUX register determines the precision of the conversion. Realistically, the converted digital data is stored in two registers in the ADC register, the ADCH and ADCL registers, of which 2 of the 10 bits are separated from the remaining bits. Here bit-5 in the ADMUX register determines whether the data should be left or right adjusted, that is, whether the higher 8 bits of data should be together or vice versa. For the best accuracy of the measurement result, we used the full 10 bit data for further calculations. The remaining five bits in the ADMUX register determines the channel that performs the A-to-D conversion since only one channel can be converted at a time. Since our accelerometer outputs to more than one channel, we need to switch channels continuously (the X, Y and Z analog signal inputs) to receive digital data from each of them. Channels can be switched by adjusting the lower five bits in the ADMUX register.

Next, we have to configure the ACDSRA and ACDSRB registers. The two registers are shown below in Figure 24 and Figure 25 respectively.

ADCSRA – ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

FIGURE 24 – ADCSRA - ADC CONTROL AND STATUS REGISTER A

ADCSRB – ADC Control and Status Register B

Bit	7	6	5	4	3	2	1	0	
(0x7B)	–	ACME	–	–	–	ADTS2	ADTS1	ADTS0	ADCSRB
Read/Write	R	R/W	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

FIGURE 25 – ADCSRB - ADC CONTROL AND STATUS REGISTER B

To enable the analog-digital conversion function, the ADEN bit in the ADCSRA register must be set all the time. The ADSC bit can be set to start a conversion. Once the conversion is completed, the ADSC bit is cleared by the hardware automatically. If the ADC interrupt is enabled, that is, the ADIE bit is set prior to the conversion, then the ADIF bit will be set along with ADSC bit when a conversion is completed. Auto-trigger is not required for our helicopter so its corresponding bit (ADATE) remains clear all the time. The lower 3 bits in the ADCSRA register sets the clock rate of the conversion. It is important to initialize this clock bit to 001 (clock speed without pre-scaling), because the conversion would be begin until the clock rate is set. The time of a single conversion usually takes about 12 system clock cycles, and with pre-scaling, the conversion takes approximately multiples of the scaling factor and the 12 cycles. At the end, there's no external triggering source and we want the A-to-D conversion to be performed continuously, so we cleared all the ACDSRB bits to set our conversion mode to free-running mode.

Ultrasonic Range Finder Measurement

The ultra-sonic sensor is used to measure the height between the ground and the helicopter during its flight. The output signals from the sensor can either be an analog or digital signal, depending on our preference for detection. The changes in height detected by the sensor can be measured by either the analog signal (with changes equal to 10mV per inch), or digital signal (changes in width of pulses at 147us per inch). For comparison, we have tested both methods of detections and evaluated their outcomes for the best method. If the analog signal is used, we need to perform an A-to-D conversion similar to the conversions of accelerometer signals mentioned in earlier sections. If the digital signal is used, then we need to measure the time elapses of the pulses to determine the height. The following section will explain the methods of measuring the width of pulses in greater details.

Pulse Width Measurement

We can measure the width of a pulse by making use of the timer1's special feature – Input capture function. Basically we use the input capturing feature in timer1 to detect the rising and falling edge of pulses, and measure the width in between to obtain the time elapsed. Since the ultrasonic sensor changes its pulse width when it detects a different height, we could obtain the approximate height if we measured the length of the pulse correctly. It works by synchronizing a timer that counts forever with the

input capture functions. We first record the counted value starting at the rising edge then the falling edge, and calculate their differences to obtain the number of counts in between.

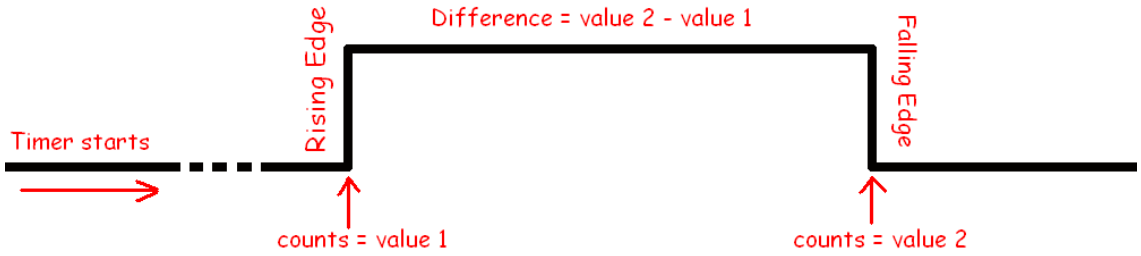

FIGURE 26 – PULSE WIDTH MEASUREMENT

Knowing the system's clock speed, we can then calculate the period of time at which the counting lasts. At the end, we can divide the time by our sensor resolution (147us/inch) to find the distance that the pulse width represents. If there is pre-scaling of the timer's counting frequency, the formula can be adjusted by multiplying the scaling factor into the counted period. The formulas for the calculations are as follows:

$$Distance = (Rising\ edge\ count - Falling\ edge\ count)\left(\frac{Prescale\ factor}{Clock\ Frequency}\right)\left(\frac{1\ inch}{147\mu s}\right)$$

SETUP FOR INPUT CAPTURE MODE

Unlike timer 0 and timer 2, timer 1 now needs to operate in normal mode, instead of PWM mode. This can be done by setting the COM1A and COM1B pins in TCCR1A register all to 0, and WGM11 and WGM10 to 0 as well. Further, WGM13 and WGM12 in TCCR1B have to be cleared as well in order to set the operating mode to normal mode. The two registers are shown below in figure 7 and 8.

TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

FIGURE 27 – TCCR1A TIMER 1 CONTROL REGISTER A

TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

FIGURE 28 – TCCR1B TIMER 1 CONTROL REGISTER B

Next the ICNC1 bit in TCCR1B register needs to be set to activate the noise canceller which reduces noise captured by the microcontroller. The lower 3 bits in Register B is the clock selection bit, we selected a scaling factor of 8 for moderate counting speed. Bit-6 in Register B is the edge selection bit for

which there would be an interrupt signal when the designated edge is detected. In the software, we wrote our program such that an interrupt is triggered when the signal toggles. Further, we find the number of counts by subtracting the rising edge count from the falling edge count, which yields the number of counts during the high time. Eventually, we are able to measure the width of pulses by using the formula shown above.

PULSE WIDTH MODULATION

As previously mentioned, the motor speed control is accomplished by utilizing a pulse width modulation (PWM) scheme together with the H-bridge circuit.

PWM is accomplished by using the 16-bit Timer/Counter 1 built-in in the ATmega644V chip. Timer 1 was chosen for this purpose since it is the only Timer module that allows for both frequency and phase correction.

To activate the PWM generation mode specific registers in Timer/Counter 1 needs to be set.

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

FIGURE 29 – TCCR1A TIMER/COUNTER1 CONTROL REGISTER A

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

FIGURE 30 – TCCR1B TIMER/COUNTER1 CONTROL REGISTER B

Specifically, we need Timer/Counter1 to operate in mode 9. This means setting the WGM1n bits in both TCCR1A and TCCR1B needs to be set. The COM1An and COM1Bn bit in TCCR1A define the output compare modes. In our case, we would like the output compare to match the value in the Output Compare Register and toggle OCnA on compare match. The CS bits in TCCR1B select the clock source to be used by the Timer/Counter. We also disabled the Input Capture Edge Select as well as Input Capture Noise Canceller since we're not doing any input capturing or comparing. Lastly and very importantly, the PRTIM1 bit in the Power Reduction Register must be written to zero to enable Timer/Counter1 Module.

In the end we have both registers set up as follows:

COM1A1	COM1A0	COM1B1	COM1B0	-	-	WGM11	WGM10
1	0	1	0	0	0	0	0

ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10
0	0	0	1	0	0	1	0

TABLE 2 – REGISTER SETUP FOR PWM

The Frequency and Phase Correct mode works in dual slope mode meaning the counter module counts continuously up till it matches the value in the Register and then counts back down.

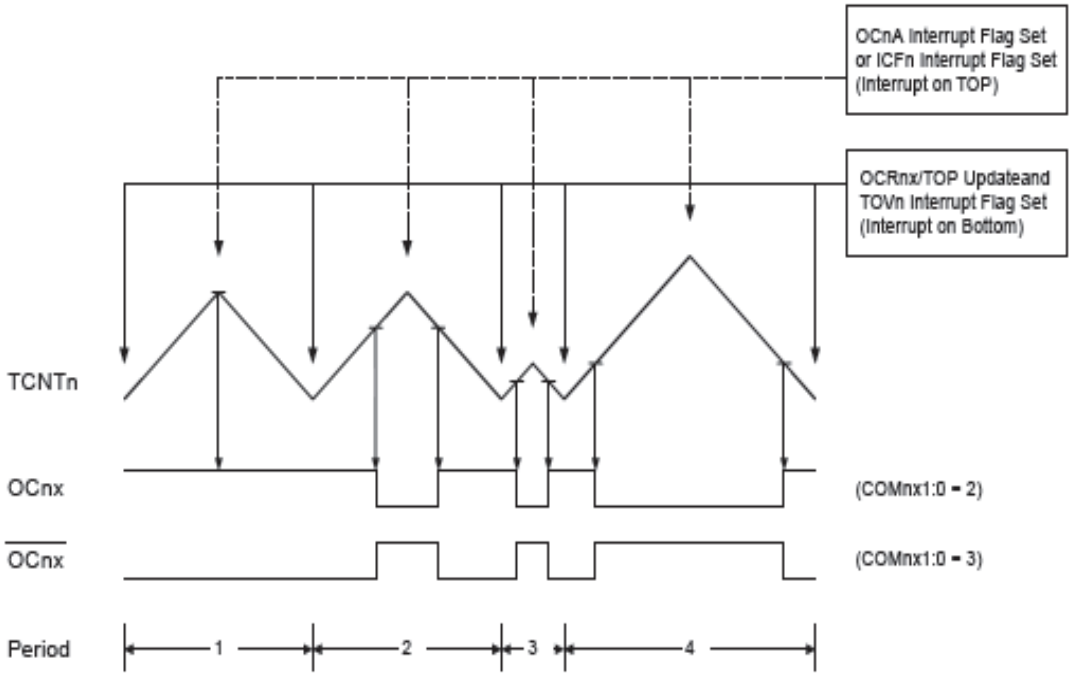

FIGURE 31 – PHASE & FREQUENCY CORRECT PWM TIMING DIAGRAM

By setting the OCR1x Register we can control when the OC1x pin toggles, generating a pulse at variable duty cycle.

MOTOR SPEED CONTROL

With the data obtained from the accelerometer through ADC and the distance obtained from timer 1, we adjust the speed of our motors based on the changes of those parameters. As mentioned in earlier sections, we control the motor's speed by adjusting the duty cycles send to the H-bridge. In our case, the motors rotate when the signals are active low. As a result, we increase the speed of motors by sending more "low" signals and decrease the speed of the motors by sending more "high" signals. The signals sent to each motor are different and they vary by the data received from the accelerometer and sonic sensor.

In order for the helicopter to maintain its stability during flight, the microcontroller needs to attempt to balance the helicopter in the flat position all the time. For example, if the helicopter is tilted to one of its side, the motor on the lower side needs to increase its rotation speed to lift its side up, while the opposite side tends to lower its rotation speed and allow the helicopter to be rotated. We set the duty cycles to range from 0 to 255, with 0 to be all low and 255 to be all high. While maintaining a stable flight in horizontal, we expected that the duty cycle to each motor should be nearly the same. Unfortunately each poles and gears do not have the exact same length and weight, and so slightly larger rotation speed on one side than its opposite side is required for the helicopter to remain balanced. A series of test data through observation in the oscilloscope and trial and error is shown here.

Height = 1m	Duty Cycles Approximation in Percentages (%)			
Angles Rotated To Front	Motor 1 Front	Motor 2 Back	Motor 3 Left	Motor 4 Right
0°	45%	45%	40%	45%
0°	42%	45%	40%	45%
0°	45%	43%	42%	43%
0°	47%	45%	40%	40%
30°	32%	63%	30%	30%
30°	27%	70%	30%	25%
30°	25%	70%	35%	33%
30°	30%	65%	30%	30%
60°	12%	80%	20%	25%
60°	15%	70%	30%	20%
60°	20%	85%	25%	20%
60°	10%	80%	25%	25%

TABLE 3 – OBSERVATION OF DUTY CYCLE CHANGES FROM TURNING HELICOPTER

From testing, we observed that the accelerometer is very sensitive to even small tilt, and the tilt could be tracked accurately by the microcontroller. One difficulty arises when balancing the helicopter and it is the amount of adjustment need to be made to each motor. It seems logical to increase the thrust of one motor by the same amount as the thrust that its opposite one had decreased by. The aerodynamics associated with balancing an object in the air becomes tedious because the helicopter has trouble recognizing the balanced position.

While balancing the helicopter in the steady horizontal position is challenging, we tried to at least maintain the helicopter at a constant height to avoid unexpected crashes. As mentioned earlier, we don't use the Z-direction signal from the accelerometer to balance the vertical position, but instead we use the ultrasonic sensor to measure the distance between ground and sensor. Table 4 below is shown to illustrate the sensitivity of the ultrasonic sensor.

Pulse Width Changes in milliseconds (Note: 0.147ms/inch)					
0 inch	Minimal	**12 inches**	+0.18ms	**18 inches**	+0.01ms
2 inches	+0.29ms	**13 inches**	+0.15ms	**19 inches**	+0.00ms
4 inches	+0.29ms	**14 inches**	+0.12ms	**20 inches**	+0.00ms
6 inches	+0.25ms	**15 inches**	+0.10ms	**21 inches**	+0.00ms
8 inches	+0.23ms	**16 inches**	+0.05ms	**22 inches**	+0.00ms
10 inches	+0.20ms	**17 inches**	+0.02ms	**23 inches**	+0.00ms

TABLE 4 – ULTRASONIC SENSOR SENSITIVITY

This sensor was tested in open fields, meaning it was pointed towards an open area with a person walking further from the sensor all the while observing the changes. We realized that the sensor sense changes in distance more accurately when the target is a large plain surface, instead of an open area with lots of movements near it. We could still make use of the sensor however, because we are only pointing the sensor towards the floor, which is just a flat and clear surface. The sensor becomes insensitive when the distance between object and itself is further apart to approximately 20 inches.

According to the datasheet of the sensor, it should be able to sense changes within 600 inches! We supposed that there were lots of interferences nearby that had caused the inaccuracy of the measurements.

PCB LAYOUT

We implemented the circuit shown in Figure 32 onto a printed circuit board (which can be seen in Figure 33). It's important to note that the PCB does not contain the circuit for the accelerometer and the ISP programmer circuit for the microcontroller. Those missing circuits are actually external components on separate boards or devices. On the PCB, there are drill holes for all external components to be soldered together with the main PCB.

After various tests and verifications, we found that the circuit would be more stable if the control systems have different groundings than the motors, because both were controlled by separate voltage sources, so it would be better if they are separated to avoid different grounding references.

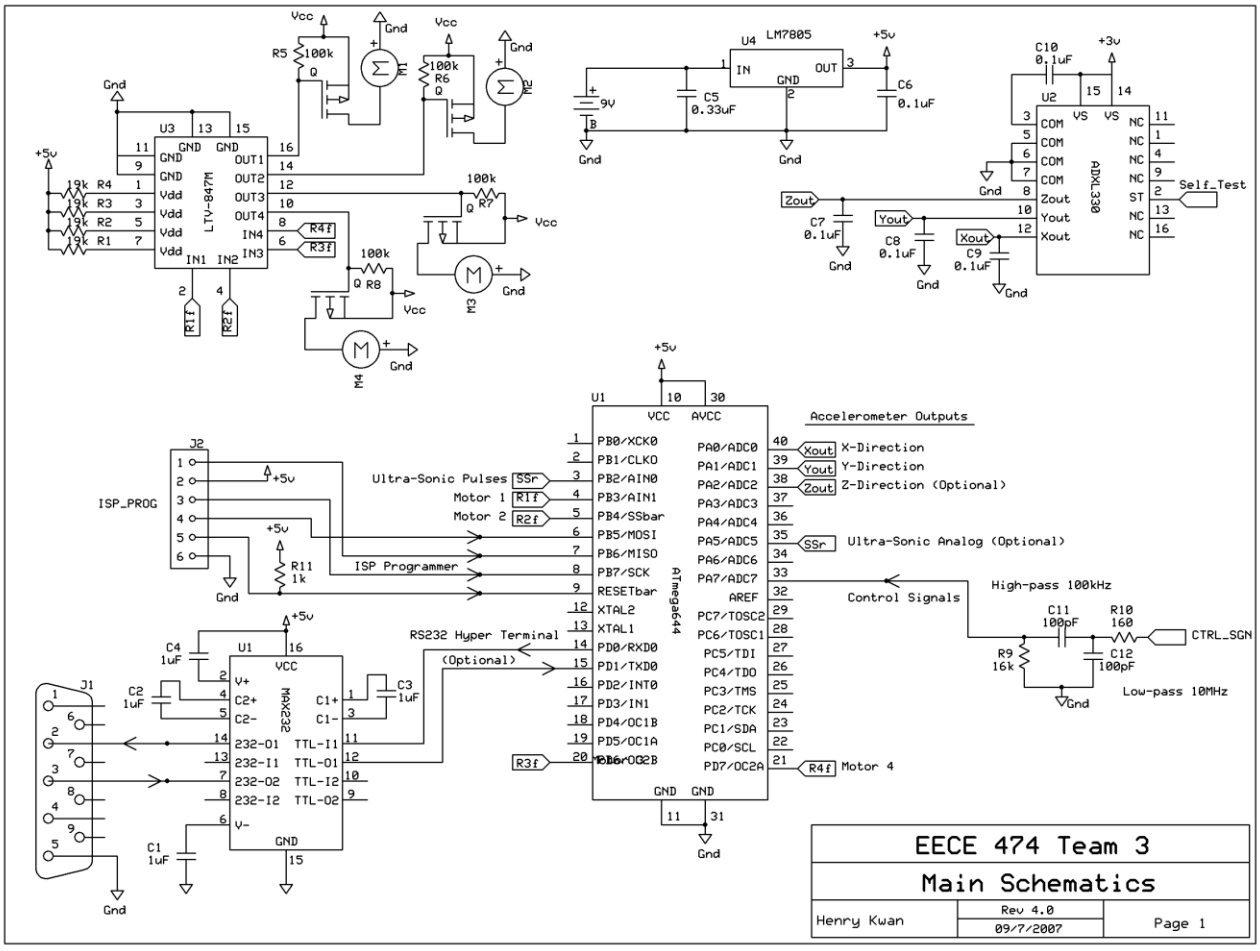

FIGURE 32 – CIRCUIT DIAGRAM FOR WHOLE SYSTEM

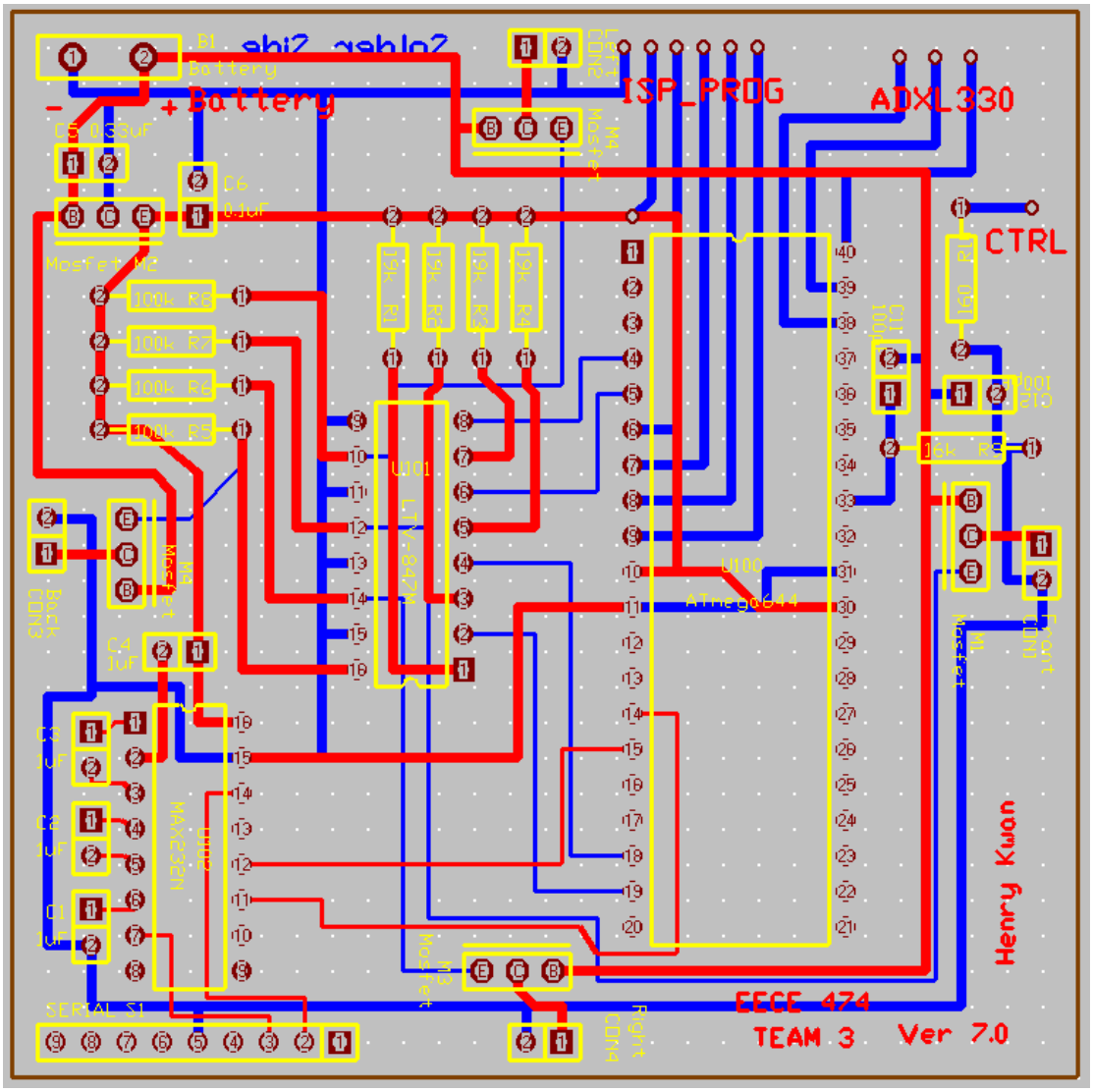

FIGURE 33 – PCB SUBMITTED

As mentioned earlier, it was only after this PCB was printed that we realized that we had designed it for one single power source, when in fact we needed two – one for the motors, one for the circuit. Thus, this PCB was not actually used in the helicopter, though the same circuit (with modifications made to the power, of course, to implement two separate power sources) was used.

VOLTAGE REGULATOR

We required a voltage regulator to power our microcontroller, since power was coming from a 9V battery, and the microcontroller operates at 5V. As shown above in, the regulator we used is the LM7805. It is a common series of three-terminal positive voltage regulators which employ built-in current limiting, thermal shutdown, and safe-operating area protection which makes them virtually immune to damage from output overloads. It delivers 5V at its output and we have two capacitors connected to maintain constant voltage input and noise-free voltage output.

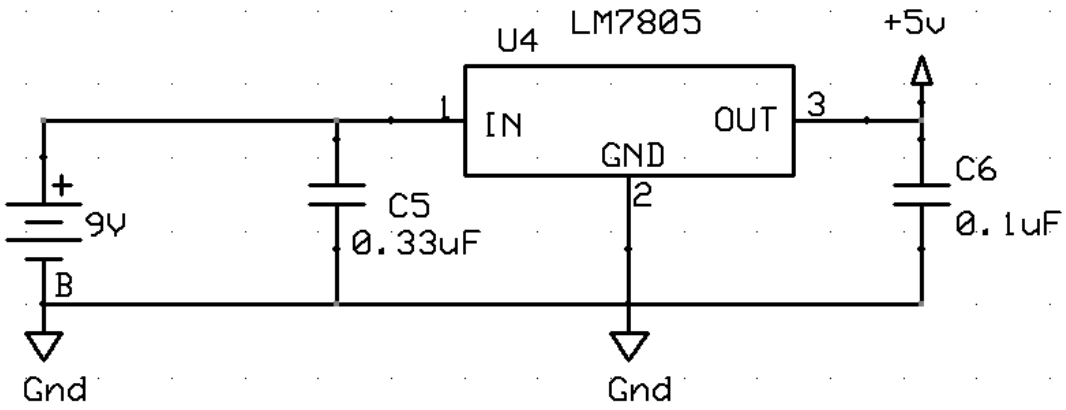

FIGURE 34 – SCHEMATIC FOR 5V VOLTAGE REGULATOR

DATA TRANSFER

The connection interface to our computer for programming is shown below in Figure 35. It requires a communication device named MAX232. It is a dual driver/receiver that includes a capacitive voltage generator to supply EIA-232 voltage levels from a single 5-V supply. Each receiver converts EIA-232 inputs to 5-V TTL/CMOS levels. The MAX232 is commonly used by programmers to configure microcontrollers through serial ports. This is an optional feature that we decided not to implement onto our circuit board.

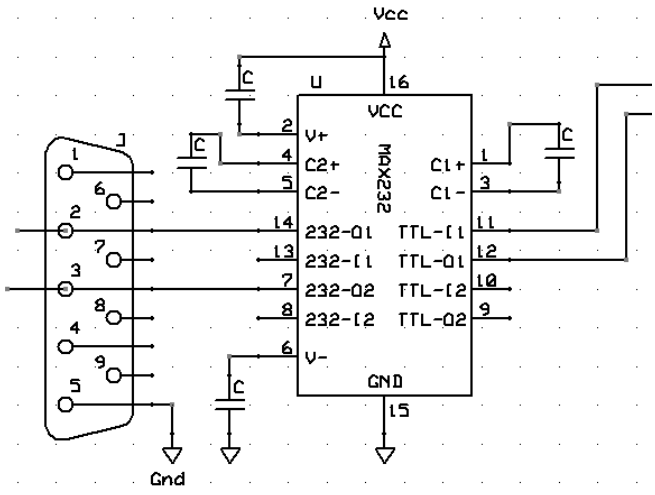

FIGURE 35 – SCHEMATIC OF RS232 CONNECTION TO MICROCONTROLLER

ASSESSMENT & ANALYSIS

The final helicopter we produced at the end of the project was no where near as fully functional as we had set out to build at the beginning of term. Several problems were encountered which made us simplify the helicopter.

PARTS

Too much time was spent during the time of this project waiting for parts to be shipped. The motors and associated hardware are examples of this. Though we new of a Canadian online source where we could obtain these parts, we attempted to find them through other sources in an effort to minimize cost. Not only was the shipment of those parts delayed, but it was discovered, through testing of the one sample motor we had obtained, that the teeth of the pinions attached to the motor shafts were too small for the gears we had. Thus poor planning on our part led to us not having the motors and the motor mounts until ¾ of the way through the project.

PCB

Miscommunication between group members led to the PCB being designed with a single power source – the 7.4V LiPo battery – being used for both the motors and the microcontroller. We decided, too late, that it would not be a good idea to do this, so we were forced to solder some proto boards manually, resulting in a much less attractive circuit, split up into different parts.

Despite this, having the circuit split up into different proto boards allowed us greater flexibility to make changes and additions. For example, we were unable to obtain a dedicated 3.3V voltage regulator for the Vdd source for the accelerometer, and so ended up creating a circuit with a variable regulator and a potentiometer. This would have been much more difficult had everything been done on a printed circuit board.

H-BRIDGE

Twice during testing of the helicopter, we shorted out the MOSFET's of the H-bridge. The first time it happened, we assumed it was due to the sudden current surge they underwent due to a wire accidentally disconnecting. But the second time, the cause of the short was unknown. It was speculated that the Styrofoam the electronics were mounted on built up a static charge that caused the short, but this was not confirmed. Some way of ensuring that the H-bridge was more stable and robust should have been implemented.

FRAME

Because of the limited time we had to assemble everything once parts arrived, as mentioned above, the mounting of the electronics onto the helicopter had to be done in a less than desired way. In other words, it ended up looking 'messier' than it should have.

The frame itself ended up being very strong, stronger than it needed to be. Despite the difficulty in making it perfectly balanced, it ended up being quite close to balanced.

FIRMWARE

The most difficult part of our project was designing the software to run it. Because none of the group members had any real experience programming such a device in the C language, we had to learn from scratch. It was in an effort to simplify this programming experience that we reduced the end goal of a fully functional helicopter that could be flown in any direction to one that simply executes a flight plan of take-off, hover, landing. Despite this, we were still able to enable the helicopter to fly by itself, not tethered to any external device.

Given more time, the hardware is there to be taken advantage of, and it would not be a big stretch to implement the more advanced functionality. A couple more wires soldered on the circuit board, a more advanced RF transmitter, and a firmware upgrade is all that would be required for the fully-functional flying helicopter originally envisioned.

IN GENERAL

The largest problem that plagued Team 3 during this term of ECE474 was the slow progress made on the individual components. We kept telling each other that such and such module would get done by 'next week', and that continued on through the weeks. Also, waiting for parts led some team members to be more idle than they ought to have been. To have made this helicopter better, team 3 ought to have stuck more rigidly to the timeline, enforcing intra-week goals of having various sections complete, ready for testing and amalgamation with the other parts.

Despite the problems, we believe that there is nothing fundamentally wrong with the helicopter. It was not too heavy, it was quite well balanced. It was able to get off the ground successfully. It goes to show that with more time and effort, UAV's in the form of four –rotor helicopters are perfectly feasible, and can be designed to be easily flown.

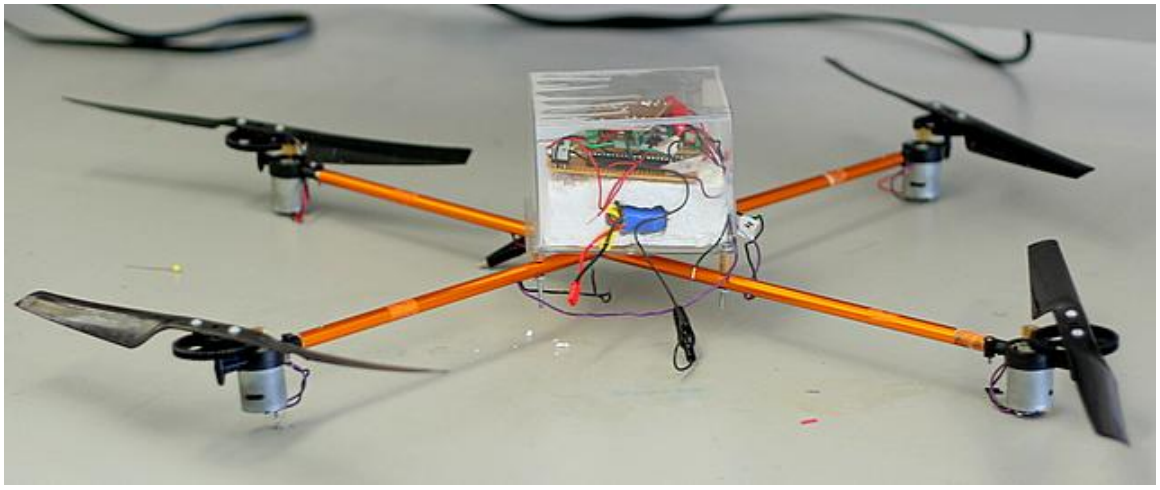

FIGURE 36 – EZ-FLY HELICOPTER

REFERENCES

#1 Nikko Radio Remote Control Vehicles. (2006). *Dragan Flyer V - New Version!* Retrieved May 25, 2007, from #1 Radio Remote Control Vehicles: http://store.1-nikkoradioremotecontrolvehicles.com/drfl4dehewim.html

Blair Lee, J. M. (1999). *RC Car Controller*. Retrieved 6 17, 2007, from http://instruct1.cit.cornell.edu/courses/ee476/FinalProjects/s1999/blair/RCcar.html

Chen, Crystal; Novick, Greg; Shimano, Kirk. (n.d.). *RISC Architecture*. (Stanford University) Retrieved June 17, 2007, from Sophomore College 2000 Class: http://cse.stanford.edu/class/sophomore-college/projects-00/risc/

CrazyAboutGadgets. (2007). *Bladerunner Helicopter.* Retrieved May 24, 2007, from CrazyAboutGadgets Gadget listings: http://www.crazyaboutgadgets.com/detail.asp?ID=242

DC-DC Converter Tutorial. (n.d.). Retrieved 6 17, 2007, from http://www.maxim-ic.com.cn/appnotes.cfm/appnote_number/2031/

DiBona, C. (2006, February 22). *Silverlit X-UFO's Gyro 'issues'*. Retrieved May 24, 2007, from Egofood Blog: http://egofood.blogspot.com/2006/02/silverlit-x-ufos-gyro-issues.html

DPRG: Low Cost Gyro-Accelerometer Combo Sensor. (2003, January). Retrieved May 25, 2007, from http://www.dprg.org/projects/2003-01a/

Draganfly Innovations Inc. (n.d.). Retrieved 6 17, 2007, from RC Toys: http://www.rctoys.com/rc-toys-and-parts/DF-VTIPRO/RC-HELICOPTERS-DRAGANFLYER-VTI-PRO.html

DraganFly Innovations Inc. (2007, May). *The Future of RC, UAVs and Robotics*. Retrieved May 25, 2007, from DraganFlyer SAVS: http://www.rctoys.com/rc-toys-and-parts/DF-SAVS/RC-HELICOPTERS-DRAGANFLYER-SAVS.html

Helpful RC Transmitter Information. (2007, May). Retrieved May 25, 2007, from Remote-Control-RC-Hobby: http://www.remote-control-rc-hobby.com/rc-transmitter.html

Hobby Zone. (2007). *Hobby Zone - Batteries, LiPo, Battery, 7.4V 800maH 2-Cell LiPo, JST/Balance*. Retrieved July 23, 2007, from Hobby Zone RC helicopter and plane parts: http://secure.hobbyzone.com/catalog/HZ/catalog/catalog_batterieslipo/EFLB0990.html

K'NEX Standard Light Grey Rod - 7 1/2 in. (2007). Retrieved May 24, 2007, from http://www.knex.com/building_toys/stard_light_gray_rod_7_12_in.php

MIT. (2007). *UAV SWARM Health Management Projeft.* Retrieved May 2007, from Aerospace Controls Laboratory at MIT: http://vertol.mit.edu/index.html

R/C Airplane World. (2007). *RC UFO(Silverlit X).* Retrieved May 24, 2007, from R/C Airplane World: http://www.rc-airplane-world.com/rc-ufo.html

RadioShack.com. (n.d.). Retrieved 6 17, 2007, from http://www.radioshack.com/product/index.jsp?productId=2476767&cp=2032062&pg=2&f=Taxonomy%2FRSK%2F2032062&categoryId=2032062&kw=controller&kwCatId=2032062&parentPage=search

Rentron. (n.d.). *TWS-434/RWS-434 Data Sheet.* Retrieved July 24, 2007, from Reynolds Electroncis: http://www.rentron.com/files/NEW_TWS_RWS_DOC.pdf

Robotics, Wright Hobbies. (n.d.). *Wright Hobbies Robotics*. Retrieved 6 17, 2007, from ATmega644: http://www.wrighthobbies.net/catalog/product_info.php?cPath=22&products_id=111

Seddon, J. (1990). *Basic Helicopter Aerodynamics.* Lodon: BSP Professional Books.

Silverlit Toys Manufactory Ltd. (2007, May). *SilverLit Flying Club*. Retrieved May 24, 2007, from X-UFO Highest Tech R/C Flying Toy: http://www.silverlit-flyingclub.com/xufo.htm

STMicroelectronics. (2007). *Datasheet for STP12PF06.* Retrieved July 24, 2007, from STMicroelectronics: http://www.st.com/stonline/books/pdf/docs/11236.pdf

Titan PS2 Wire Gamepad User Manual. (n.d.). Retrieved 6 17, 2007, from http://rsk.imageg.net/graphics/uc/rsk/Support/ProductManuals/2600239_PM_EN.pdf

Wikipedia. (2007). *Optocoupler*. Retrieved July 24, 2007, from Wikipedia: http://en.wikipedia.org/wiki/Optocoupler

APPENDIX A – C CODE FOR MICROCONTROLLER

Definition.h

```
#define DEFINITION

#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/iom644.h>

// General Functions
#define BIT(x) (1 << (x))
#define SETBITS(x,y) ((x) |= (y))
#define CLEARBITS(x,y) ((x) &= (~(y)))

#define SETBIT(x,y) SETBITS((x), (BIT((y))))
#define CLEARBIT(x,y) CLEARBITS((x), (BIT((y))))
#define BITSET(x,y) ((x) & (BIT(y)))
#define BITCLEAR(x,y) !BITSET((x), (y))
#define BITSSET(x,y) (((x) & (y)) == (y))
#define BITSCLEAR(x,y) (((x) & (y)) == 0)
#define BITVAL(x,y) (((x)>>(y)) & 1)

// Global Variables


// ADC Functions
#define Clear_ADC_Int_Flag()            (ADCSRA |= _BV(ADIF)) // Clear Interrupt Flag
#define ADC_Int_Enable()                (ADCSRA |= _BV(ADIE)) // Interrupt Enable
#define ADC_Int_Disable()               (ADCSRA &= ~_BV(ADIE))        // Interrupt Disable

void ADC_initial(void);
void ADC_Channel(int channel);
unsigned short a2dConvert10bit(unsigned int ADC_channel);
void delay(unsigned char time);
ISR(PCINT0_vect);
ISR(PCINT1_vect);
ISR(PCINT2_vect);

int X_direction_Set(void);
int Y_direction_Set(void);

// PWM Functions
// Timer/Counter 0
void Timer0_initial(void);
ISR(TIMER0_COMPA_vect);
```

```
ISR(TIMER0_COMPB_vect);
void Set_Timer0_DutyCycle_A(int dutyA);
void Set_Timer0_DutyCycle_B(int dutyB);

// Timer/Counter 1

#define Num_Sonars              4

#define Set_Input_AIN0_Rising()             (TCCR1B |= _BV(ICES1))      // Set capture rising edge
#define Is_Input_AIN0_Rising()      (TCCR1B & _BV(ICES1))       // Is capture rising edge?
#define Set_Input_AIN0_Falling()    (TCCR1B &= ~_BV(ICES1))             // Set capture falling edge
#define Is_Input_AIN0_Falling()     ~(TCCR1B & _BV(ICES1))      // Is capture falling edge?

#define Clear_IC_Flag()                     TIFR1 |= _BV(ICF1)                  // Clear input capture flag

#define US_PER_INCH         149.28

#define ENABLE_SONAR()      SETBIT(PORTB,2)
#define DISABLE_SONARS()      CLEARBIT(PORTB,2)

#define IC1_Enable()    (TIMSK1 |= _BV(ICIE1))                          // Input capture enable
#define IC1_Disable()   (TIMSK1 &= ~_BV(ICIE1))                         // Input capture disable

uint32_t sonar_get_dist(void);
void sonar_start_reading(void);

void Timer1_initial(void);
ISR(TIMER1_CAPT_vect);

// Timer/Counter 2
void Timer2_initial(void);
ISR(TIMER2_COMPA_vect);
ISR(TIMER2_COMPB_vect);
void Set_Timer2_DutyCycle_A(int dutyA);
void Set_Timer2_DutyCycle_B(int dutyB);
```

---

adc.c

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/iom644.h>
#include <D:\474Project\definition.h>

volatile unsigned int ADC_Complete_Flag;

void ADC_initial(void)          // ADC initialization
        {
                ADMUX = 0xC0;           // ADC0, Right adjust,Internal 2.56 Voltage Reference
```

```
                                        // Enable ADC and set prescaler to 1/2*1.00MHz = 500 KHz

        ADCSRA = 0x88;//8F; //88;       // Enable ADC, ADC interrupt, prescaler 128
        ADCSRB = 0x00;                  // Free Running Mode
    }

void ADC_Channel(int channel)    // Switching channels between PCINT0, PCINT1, PCINT2
    {
        if(channel == 0)
            {
            ADMUX = 0xC0;
            }
        if(channel == 1)
            {
            ADMUX = 0xC1;
            }
        if(channel == 2)
            {
            ADMUX = 0xC2;
            }
        if(channel == 5)
            {
            ADMUX = 0xC5;
            }
    }

int X_direction_Set(void)
    {
        return a2dConvert10bit(0);
    }

int Y_direction_Set(void)
    {
        return a2dConvert10bit(1);
    }

unsigned short a2dConvert10bit(unsigned int ADC_channel)
    {
        int ADCHL;
        int ADCHLfake;

        ADC_Complete_Flag = 0;                          // clear conversion complete flag
        ADC_Channel(ADC_channel);                       // set channel
        SETBIT(ADCSRA, ADIF);               // clear hardware "conversion complete" flag
        SETBIT(ADCSRA, ADSC);               // start conversion

        while( bit_is_set(ADCSRA, ADSC) );          // wait until conversion complete
```

```
//                      ADCHL = ADC;
                        ADCHL = ADCL;
                        ADCHLfake = ADCH;
                        // CAUTION: MUST READ ADCL BEFORE ADCH!!!

                        return ADCHL-30;                                        // Change the subtraction value
to change set point;
        }

void delay(unsigned char time)          //Function to generate time delay
        {
                int l;
                int m;

                for(l=0; l<time; l++)
                        {
                                for(m=0;m<10000;m++)
                                {};
                        }
        }

ISR(PCINT0_vect)                        // ADC interrupt routine for PA0
        {
                PORTD = 0xFF;
        }

ISR(PCINT1_vect)                        // ADC interrupt routine for PA1
        {
                PORTD = 0xFF;
        }

ISR(PCINT2_vect)                        // ADC interrupt routine for PA2
        {
                PORTD = 0xFF;
        }

ISR(ADC_vect)                                   // ADC Complete Interrupt
        {
                ADC_Complete_Flag = 1;
        }
```

Pwm.c

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/iom644.h>
#include <D:\474Project\definition.h>

static uint16_t time_rising_edge;
```

```
static uint16_t time_falling_edge;
uint16_t ov_counter;
uint32_t count;

void Timer0_initial(void)                    // Counter 0 Initialization
        {
                TCCR0A = 241;                        // Set to Phase correct PWM mode
                                                     // Set OC0A to 1 when up-counting match
                                                     // Set OC0A to 0 when down-counting match

                TCCR0B = 1;                          // Start clock with no prescaling: factor = 1

                TIMSK0 = 0;//6;                      // Compare match A & B Interrupt Enable

                OCR0A = 80;                          // Match value for register A
                OCR0B = 80;                          // Match value for register B
        }

ISR(TIMER0_COMPA_vect)                       // Timer 0 Output A Interrupt Routine
        {
                Set_Timer0_DutyCycle_A(duty);
                SETBIT(PORTB,0);
        }

ISR(TIMER0_COMPB_vect)                       // Timer 0 Output B Interrupt Routine
        {
                Set_Timer0_DutyCycle_B(duty);
                SETBIT(PORTB,1);
        }

void Set_Timer0_DutyCycle_A(int dutyA)       // Set duty cycle for output A
        {
                OCR0A = dutyA;
        }

void Set_Timer0_DutyCycle_B(int dutyB)       // Set duty cycle for output B
        {
                OCR0B = dutyB;
        }

////////////////////////////////////////////////////////////////////

void Timer1_initial(void)
        {
                // Analog Comparator Unit
                ADCSRB = 0x00;
                ACSR = 4;//36;   //4; //12;//44;                    // Analog Comparator ACO and Input
Capture Enable
```

```
                                                                    // Interrupt Enable
        TCCR1A = 0;                                 // Normal Mode

        TCCR1B = 130;//132;//129;//133;//131;//130;//129;                   // No clock,
Prescaling Clk/1
                                                                    // Noise canceller Enable
                                                                    // Positive edge trigger
    }

ISR(TIMER1_OVF_vect)
    {
     ov_counter++;                                  // Increment overflow counter when overflow occurs
    }

uint32_t sonar_get_dist()
    {
            count = (uint32_t)time_falling_edge - (uint32_t)time_rising_edge + (uint32_t)ov_counter;

    // Number of counts between edges


    // Each count is 1/1MHz seconds = 0.000001s
            uint32_t count_time = count * 0.000001*8;           // Length of Time = # of counts
x time per count

    // Counted Time in Seconds
            uint32_t count_time_micro = count_time * 1000000;   // Converted counted time to micro
seconds

    // 147us per inch

//          distance = (uint8_t)(difference / (int16_t)(US_PER_INCH));
//          uint32_t distance = (uint32_t)(count_time_micro/(uint32_t)US_PER_INCH);
            uint32_t distance = count_time_micro/US_PER_INCH;
            if (distance > 5)
            {
            SETBIT(PORTB,0);
            }
            else
            {
            //CLEARBIT(PORTB,0);
            }

        return distance;
    }

void sonar_start_reading()
    {
```

```
                Set_Input_AIN0_Rising();        // Set input trigger rising edge
                Clear_IC_Flag();                // Clear input capture flag
                IC1_Enable();                           // Input Capture Interrupt Enable
                SETBIT(TIMSK1,0);                       // Overflow Interrupt Enable
        }

ISR(TIMER1_CAPT_vect)                   // Timer 1 Input Capture Interrupt
        {
                if (Is_Input_AIN0_Rising())
                {
                        time_rising_edge = ICR1;

                        Set_Input_AIN0_Falling();
                        Clear_IC_Flag();

                        PORTD = _BV(PD0);

                        ov_counter = 0;
                }
                else
                {
                        time_falling_edge = ICR1;

                        Set_Input_AIN0_Rising();
                        Clear_IC_Flag();

                        sonar_get_dist();

                        PORTD = _BV(PD1);
                }
        }

//////////////////////////////////////////////////////////////////////

void Timer2_initial(void)               // Counter 2 Initialization
        {
                TCCR2A = 241;                   // Set to Phase correct PWM mode
                                                        // Set OC2A to 1 when up-counting match
                                                        // Set OC2A to 0 when down-counting match

                TCCR2B = 1;                             // Start clock with no prescaling: factor = 1

                TIMSK2 = 0;//6;                         // Compare match A & B Interrupt Enable

                OCR2A = 80;                     // Match value for register A
                OCR2B = 80;                     // Match value for register B
        }
```

```
ISR(TIMER2_COMPA_vect)                  // Timer 2 Output A Interrupt Routine
        {
                Set_Timer2_DutyCycle_A(duty);
                SETBIT(PORTB,0);
        }

ISR(TIMER2_COMPB_vect)                  // Timer 2 Output B Interrupt Routine
        {
                Set_Timer2_DutyCycle_B(duty);
                SETBIT(PORTB,1);
        }

void Set_Timer2_DutyCycle_A(int dutyA)  // Set duty cycle for output A
        {
                OCR2A = dutyA;
        }

void Set_Timer2_DutyCycle_B(int dutyB)  // Set duty cycle for output B
        {
                OCR2B = dutyB;
        }
```

Main.c

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/iom644.h>
#include <D:\474Project\definition.h>

void Global_initial(void)
        {
                DDRA = 0x00;            // All pins in Port A are inputs
                DDRB = 0x19;//18;               // PB3 & PB4 of Port B are outputs
                DDRC = 0xFF;            // All pins in Port C are outputs (Not Used)
                DDRD = 0xFF;            // PD4 & PD5 of Port D are outputs
        }

int main(void)
{
        Global_initial();
        ADC_initial();

        Timer0_initial();
        Timer1_initial();
        Timer2_initial();
        sonar_start_reading();

        sei();                                          // DO NOT FORGET THIS or I register won't be on!
                                                        // Enable Interrupt
```

```
int ADCHZ;

while(1)
{
        ADCHZ = a2dConvert10bit(7)+30;
        if(ADCHZ > 100)
                {
                PORTD = 0x0F;
                }
                else
                {
                PORTD = 0x00;
                }

}

int X_set = X_direction_Set();                  // Initialize X setpoint
int Y_set = X_direction_Set();                  // Initialize Y setpoint

int ADCHX;                                      // Initialize Motor speed variables
int ADCHY;

uint8_t Motor_1 = 120;                          // Beginning motor duty out of 255
uint8_t Motor_2 = 120;
uint8_t Motor_3 = 120;
uint8_t Motor_4 = 120;

while(1)
{
        if(ADCHX > 145)                                 // (650/1024)*(2.56v) ~ 1.625v
        {
        PORTD = 0x0F;
        }
        else
        {
        PORTD = 0x00;
        }

        ADCHX = a2dConvert10bit(0);                     // X direction Status

                if(Motor_1 == 255)
                        {
                                Motor_1 = 254;
                        }
                else if(Motor_1 == 0)
                        {
                                Motor_1 = 1;
                        }
```

```
		if(ADCHX < X_set-5)                                    // MOTOR 1
			{
				Motor_1 = Motor_1 + 1;// Changing the  Incremental value changes the setpoint
				Set_Timer0_DutyCycle_A(Motor_1);      // Higher it is the more HIGH time
			}
		else if(ADCHX > X_set+5)
			{
				Motor_1 = Motor_1 - 1;
				Set_Timer0_DutyCycle_A(Motor_1);
			}

		if(Motor_2 == 255)
			{
				Motor_2 = 254;
			}
		else if(Motor_2 == 0)
			{
				Motor_2 = 1;
			}
		if(ADCHX < X_set-5)                                    // MOTOR 2
			{
				Motor_2 = Motor_2 - 1; // Changing the  Incremental value changes the setpoint
				Set_Timer0_DutyCycle_B(Motor_2);      // Higher it is the more HIGH time
			}
		else if(ADCHX > X_set+5)
			{
				Motor_2 = Motor_2 + 1;
				Set_Timer0_DutyCycle_B(Motor_2);
			}

ADCHY = a2dConvert10bit(0);                                    // Y direction Status

		if(Motor_3 == 255)
			{
				Motor_3 = 254;
			}
		else if(Motor_3 == 0)
			{
				Motor_3 = 1;
			}
		if(ADCHY < Y_set-5)                                    // MOTOR 3
			{
				Motor_3 = Motor_3 + 1;// Changing the  Incremental value changes the setpoint
				Set_Timer2_DutyCycle_A(Motor_3);      // Higher it is the more HIGH time
			}
		else if(ADCHY > Y_set+5)
			{
				Motor_3 = Motor_3 - 1;
```

```
                        Set_Timer2_DutyCycle_A(Motor_3);
                }

                if(Motor_4 == 255)
                {
                        Motor_4 = 254;
                }
                else if(Motor_4 == 0)
                {
                        Motor_4 = 1;
                }
                if(ADCHY < Y_set-5)                                     // MOTOR 4
                {
                        Motor_4 = Motor_4 - 1; // Changing the  Incremental value changes the setpoint
                        Set_Timer2_DutyCycle_B(Motor_4);       // Higher it is the more HIGH time
                }
                else if(ADCHY > Y_set+5)
                {
                        Motor_4 = Motor_4 + 1;
                        Set_Timer2_DutyCycle_B(Motor_4);
                }
        }
        return 0;
}
```