University of Florida

EEL 4665C Intelligent Machine Design Laboratory

Final Report

Autonomous Quadcopter Robot

Robot Name: Horus

Student Name: Miles Moody

Instructors: Dr. A. Antonio Arroyo and Dr. Eric Schwartz

TA's: Mike Pridgen, Ryan Stevens, Tim Martin, Thomas Vermeer, Devin Hughes

Date: 12/7/10

Website: http://randomhacksofboredom.blogspot.com/p/quadcopter-robot-project.html

Table of Contents

Abstract

Horus is a quadcopter robot, meaning it flies in the air using the power of four motor/propeller combinations. It is able to rotate itself in the three directions pitch, roll, and yaw by differential thrust of opposing motors. It is also able to hover and translate itself in any direction. The robot is able to fly itself fully autonomously, but a manual override of the robot is always possible for safety concerns. Horus stabilizes itself by estimating its orientation using sensor data from its inertial measurement unit. The robot is able to hold a specified altitude both close to the ground, using a downward facing sonar, and at higher altitudes using a barometric pressure sensor. It can also navigate and perform patterned flights through GPS waypoints.

Introduction

Unmanned aerial vehicles (UAVs) are increasingly becoming a substitute for manned aerial systems, especially when it means keeping a pilot from dealing with overly dangerous or tedious situations. However, many of today's UAVs still require an operator at all times and for the most part are large, fixed wing platforms. The need for small, hover capable aerial robots is becoming evident to perform tasks their larger fixed wing counterparts are unsuited for. One such type of aerial platform that could be suitable is the quadcopter. Able to achieve good stability performance with relatively low cost controllers and sensors, the hovering quadcopter could become an excellent platform for navigation, mapping, and remote sensing in smaller spaces and closer to the ground than fixed wing UAVs are able to fly.

Integrated System

Horus was built to be very modular; all parts can be rapidly replaced or upgraded without disrupting other parts. The robot can be divided into three main parts: the mobile platform, the sensors, and the controller. A hardware flow diagram of how each part interconnects with one another is shown in figure 1 below.

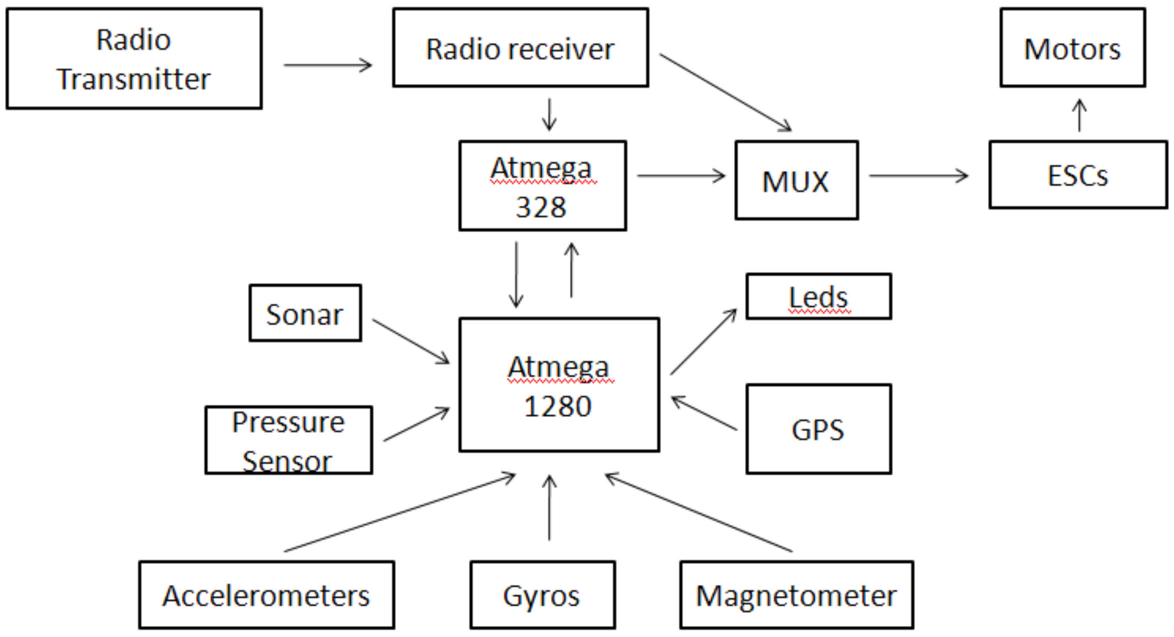

Figure 1 – Hardware Flow Diagram

Mobile Platform

1st revision

The quadcopter's mobile platform was all hand built from scratch. A majority of components were simply cut or shaped with nothing more than a drill, Dremel tool, and hacksaw. For this reason, precision of the frame could have been better. However, the frame stood up to a not insignificant amount of abuse and in addition to being very low cost, it also was able to be easily built and rebuilt without having to wait for shipping of parts. Having said that, in the future I would choose to begin with an off-the-shelf frame and then modify to my requirements.

The frame initially was built by cutting four arms of Home Depot "shower bar stock" to 11" and sandwiching two 1/8"x5 ½"x5 ½ " wooden plates over the bars. The bars were then secured by machine screws, washers, and locking nuts. Holes for motor mounts were drilled into the ends of the arms and motors were mounted.

The motors chosen for the mobile platform were brushless outrunners made for remote control planes. They were sized appropriately for the expected thrust needed with a generous factor of safety afforded. Initially, three bladed 9050x3 propellers were bought because they were readily available, even knowing that in general three bladed propellers are less efficient than two bladed ones. Propellers were difficult to find for this project because the blades must be bought in both clockwise and counterclockwise rotation configurations. Simply turning a propeller upside down does not work. Clockwise (also called pusher or reverse) props are not made in very many sizes and are difficult to source. Both counterclockwise and clockwise spinning props are necessary so that the platform can exhibit yaw control.

Brushless electronic speed controllers were also purchased as standard RC parts and a large factor of safety was also used to determine the max amperage of the speed controllers. Not only did this allow for manufacturer inaccuracy in specs (as is common with cheaper manufacturers) but also allowed for possible switching out of motors for larger ones down the road if deemed necessary.

Batteries were known to be a major source of weight in quadcopters. To save cost however, I reused a 11.1V 5000mah lithium polymer(Lipo) pack that I had several of from previous RC projects. After many frustrating problems including a random motor stopping in mid flight, I found these batteries to either be incorrectly spec'd or too old to supply the current required current. I replaced them with a new 11.1V 5400mah battery which could theoretically constantly produce 100 amps (my design never needed more than 40 amps).

Lastly, landing gear was needed. Instead of making my own, I found it easier to adapt a set of skids made for small RC helicopters to be used on my robot. This required drilling new holes in the landing gear and adding spacers but resulted in a very firm product.

Additionally, a testing rig was constructed from PVC pipe and some frame extensions. This testing rig can be seen in figure 2 below. This rig allowed me to tune the pitch and roll axes independently of each other without having to worry about crashing the quadcopter.

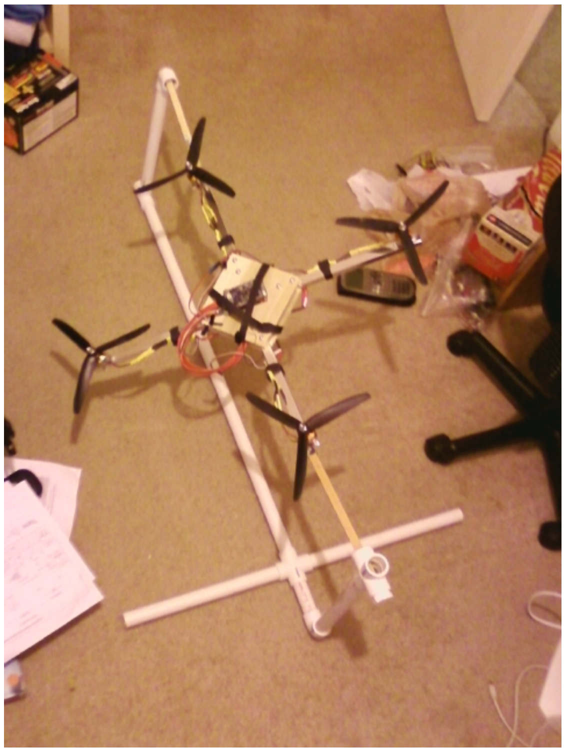

Figure 2 – Pitch and Roll Testing Rig with Revision 1 platform

Revision 2

Later on in the project, when a large variety of props were able to be sourced, I performed some basic prop thrust tests on five sets of propellers using a simple testing rig. The test rig consisted of a digital kitchen scale, a Tupperware container filled with ~2lbs of weight, and a spare motor mount. The scale was zeroed with the tupperware full and the motor and prop mounted. The motor was then turned on so that it attempted to lift the tupperware container. The amount of static thrust could then be directly measured from the scale. To measure the current being consumed by the motor during operation, a "Watt-Meter" was used in between the speed controller and battery. The results of these tests are shown below in figures 3 and 4. By analyzing this data, the most efficient props were determined to be the 10x4.7 and 10x3.8 props.

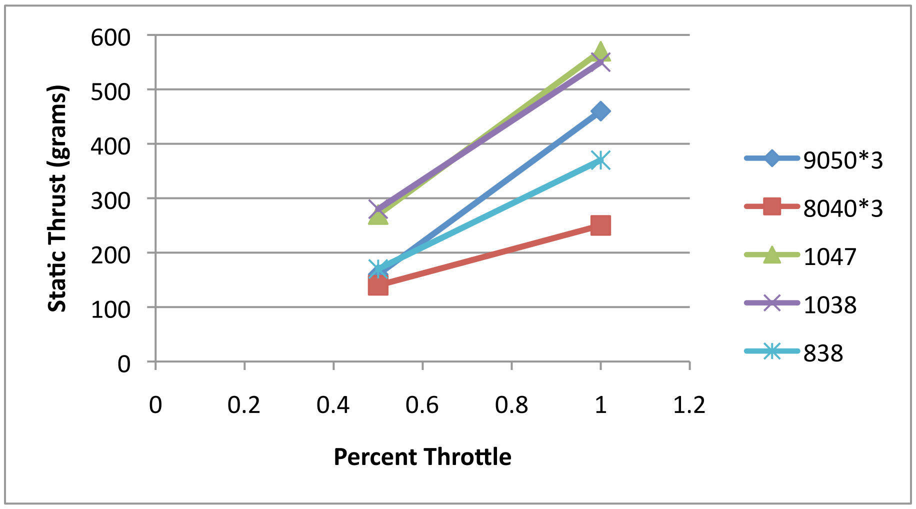

Figure 3 – Thrust versus Throttle Position for Various Propeller Types

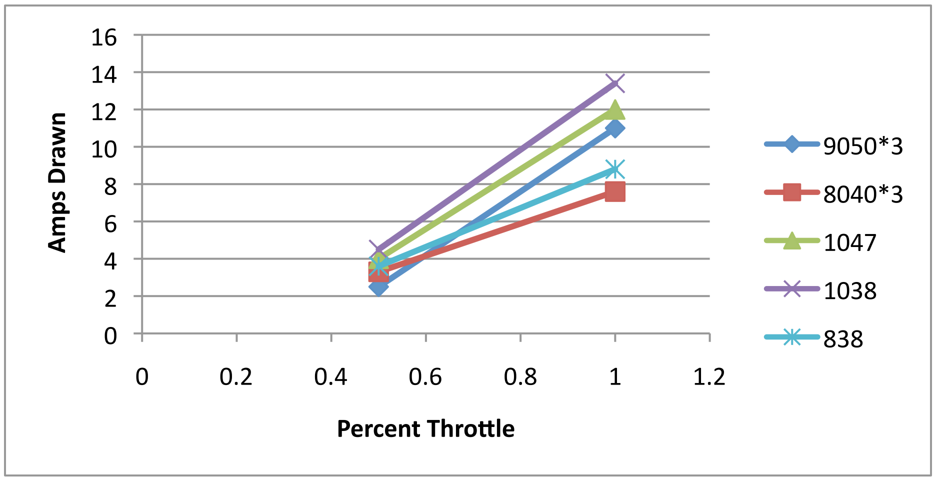

Figure 4 – Current Drawn versus Throttle Position for Various Propeller Types

A major revision was needed to be done on the frame once the number of sensors increased beyond the space that was available on the first platform. To add more space, a second level was constructed from thin wooden sheets and wooden dowel rods. In addition, as testing became more intense and more autonomy was given to the robot, my concerns rose for the safety of the electronics housed on the frame. To keep the electronics safe in the event of a crash upside down, a durable removable locking tupperware container was added to the frame design. Holes were cut in the top of the container to allow an unobstructed gps view of the sky, the side of the container to allow the Xbee radio antenna to extend outside of the robot, and slots were cut to allow wires to go in and out. A hot knife greatly facilitated this cutting. The current frame design can be seen below in fig 5.

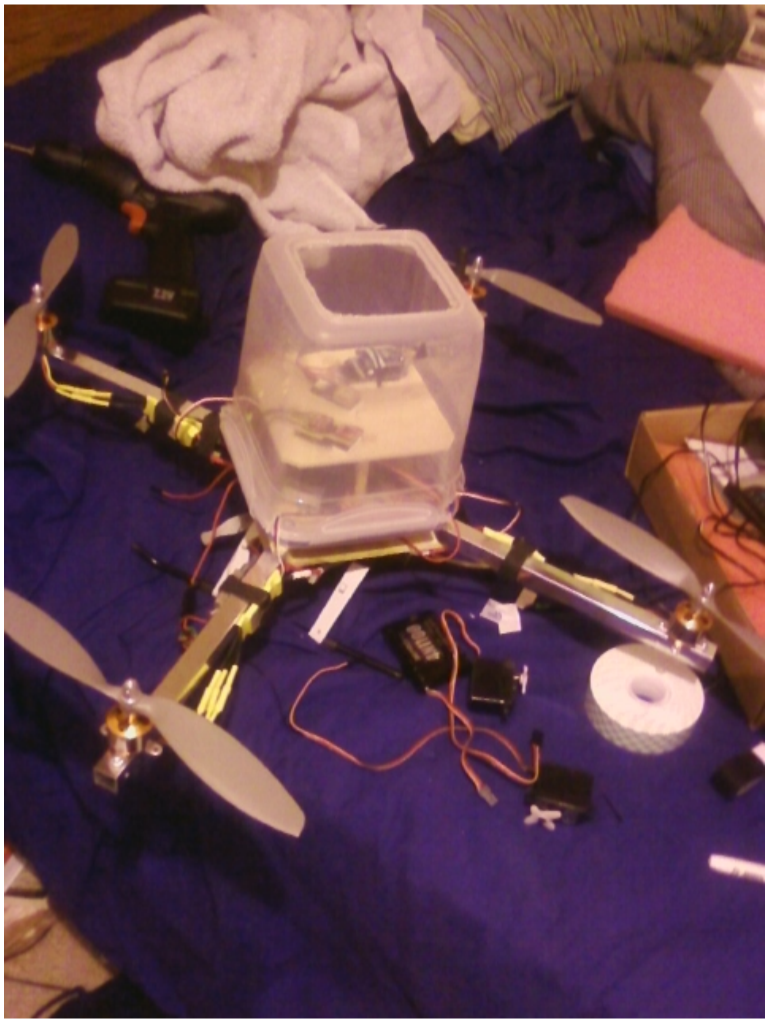

Figure 5 – Revision 2 of Airframe

Controller

The controller chosen for this project is the Atmega1280 based Ardupilot Mega (APM) board. This controller was chosen for several reasons. The first reason was due to my own experience dealing with the arduino development environment, to which the APM is based around. In addition, the APM board is specifically designed for autopilot functions, and was developed with the use of a mating inertial measurement board in mind.

The board contains several pieces of hardware, the Atmega1280 which is the main processor. This is the processor upon which all of my code was written and stored. In addition, there is an Atmega328 processor which serves as a slave whose function is to decode signals received from the radio receiver, encode signals to pulse width modulation (PWM) values, and in fixed wing autopilots, provide failsafe control. There also exists a multiplexer chip which also provides failsafe functions for fixed wing autopilots allowing the user to manually override the autopilot in case of autopilot failure. These failsafe functions were not used in this project because they do not apply to rotary aircrafts.

Sensors

The sensor suite for Horus is quite extensive. First and foremost is the Inertial Measurement Unit (IMU). The IMU selected is the codenamed “Oilpan” made by DIYdrones.com. This IMU was made to fit physically and electrically to the APM board. It contains the a 3 axis accelerometer (ADXL335), a 2 axis gyro (IDG500), another single axis gyro (ISZ500),and a barometer (SCP-1000). In addition, the Oilpan board contains an ftdi usb to serial chip, a 16Mb dataflash memory chip, a 12-bit ADC, a relay, and several switches and LEDs. All of these parts are contained on one self contained board which snaps directly over (or under) the APM board.

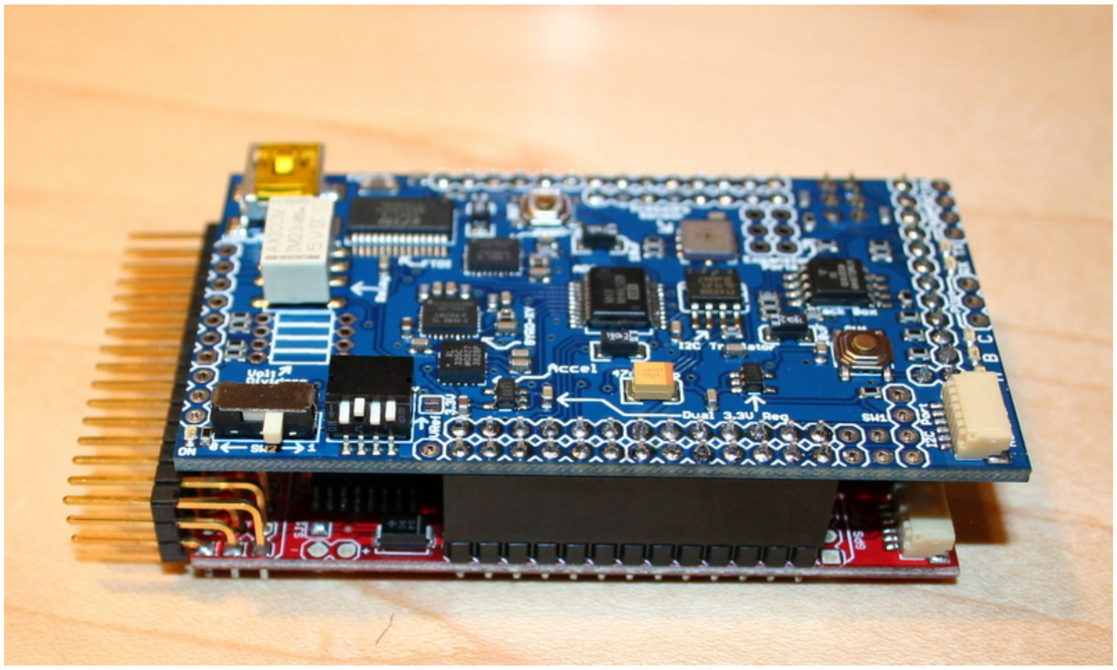

Figure 6 – APM board (Red) and Oilpan board (Blue)

In addition to the Oilpan board, several otehr sensors were used on the robot. A three axis magnetometer (HMC5843) was added as a compass sensor of sorts. This gave the robot an absolute yaw reference to compensate for the yaw gyro's drift. A downward facing sonar (Maxbotix EZ-1) was installed on the robot for altitude hold and an obstacle avoidance mechanism. The sonar sensor was tested to determine experimentally a correlation between the analog value outputted and the actual distance to a planar object (the ground). This was necessary because the documentation included was meant for small, non planar objects, and its validity had been called into question by several colleagues. The results for the testing can be seen below in figure 7. The data determined from the tests shows the sensor to be very linear within the testing range, but the sensitivity taken from the linear fit did not match the datasheet value. It is probably wise to determine a calibration value through experimental methods rather than use the value in the datasheet.

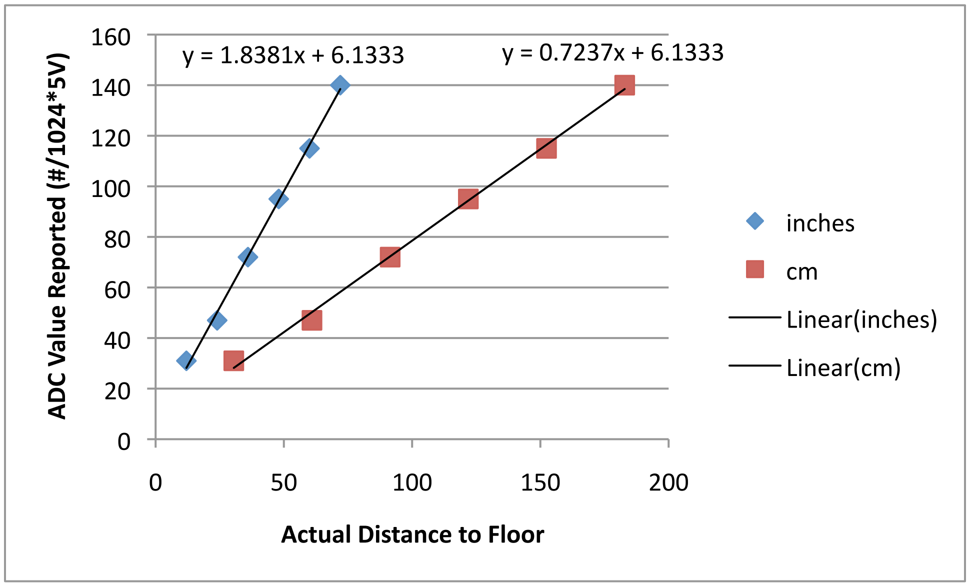

Figure 7 – Experimental Characterization of Sonar Sensor

A small GPS module(MT3329) was installed onto the robot to allow for absolute position determination as long as an accurate GPS signal could be obtained. This model had no problems getting a 3D lock even under clouds or partial tree cover (sometimes inside too) but when using either the included software or the premade GPS parsing library, a constant GPS offset was seen when compared to latitude/longitude taken from google maps.

Several other sensors were purchased but as of yet have not been installed. These include four more sonar sensors meant to provide obstacle information for indoor flight and outdoor flight with obstructions. Additionally, force sensing resistors were meant to be installed into the landing gear in support of automatic takeoff. These are still ongoing goals.

Lastly, though not technically a sensor, Xbee modules provided a wireless communications link between the robot and a computer. This was critical for debugging purposes.

Behaviors

Behaviors for the robot were slowly added in a sequential fashion. First, the robot needed to be able to respond to inputs given from the remote control transmitter. Next, the robot needed to send the correct signals to the speed controllers to tell the motors what velocity to spin at. To accomplish these tasks, a supplied library made for the APM board was used. Because of the slave processor on the APM board, very little computational overhead was used on the main processor. This behavior was fairly trivial to implement.

Next, the robot needed to process the IMU data to obtain an idea of its current orientation in 3d space. There are many ways this could have been done. One method commonly used is to employ a kalman filter to fuse, interpret, and filter the accelerometer, gyro, and magnetometer data to obtain orientation. This method is very resource intensive on an 8-bit 16mhz microcontroller so another method was used. This method is called the Direction Cosine Matrix (DCM) algorithm, developed by Robert Mahony specifically for use in small aerial vehicles. The DCM algorithm is much less computationally intensive and relative to the kalman method, much easier to understand. Once an estimate of orientation has been developed, the quadcopter can begin stabilization.

Stabilization is achieved simply enough through a set of independent proportional integral (PID) controllers. Each axis is stabilized independently by a PID loop, each tuned to that specific axis. Due to symmetry, the roll and pitch axes have (almost) identical PID constants. The yaw dynamics are very different however. Motor calibration constants were also used to account for the slight but noticeable differences in motor power at the same control signal. Stabilization was the most difficult behavior to achieve; constant tuning and retuning was necessary. Every shift of weight or crash created need for either retuning of the PID constants or of separate motor calibration constants.

After satisfactory stabilization had been established, the next step was altitude hold. In a way, altitude hold was considered an obstacle avoidance behavior where the obstacle is simply the ground. In practice it worked very similar to a wall following routine flipped on its side. At first, only a barometer was used to calculate differences in absolute pressure and from that, altitude. This worked to some degree but was not an ideal solution. Using the downward facing sonar, a more accurate measurement of distance from the ground was received as long as the robot was within about 8 feet of the ground. To actually control the altitude, another independent PID loop was created and tuned. This loop was found to be particularly sensitive to integral gain, so after the loop was tuned it is very close to a PD controller instead. To account for using two different sensors for altitude measurement, the sonar is used if the altitude is less than 90% of its max range and beyond that, the barometer is used until the robot is brought lower.

The last behavior added up to this point is waypoint pattern flight. Using a copy of the fixed wing navigation software, which is what the APM was initially built for, it was fairly easy to develop a scheme to navigate from one waypoint to another. The challenge was that because the GPS module had an offset, and the fact that I would be demoing the robot in multiple unknown locations, it would be difficult to preload in absolute waypoints. To get around both of these problems, I decided to make the preprogrammed waypoints relative to the starting position of the robot. This allowed me to program in more of a “pattern” than a set list of waypoints.

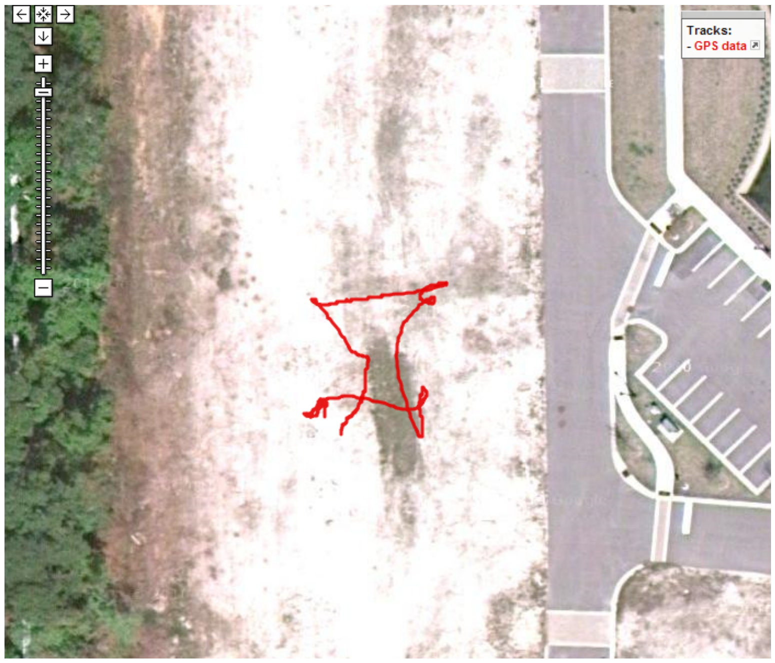

Figure 8 - GPS Path Followed During Initial Testing of Waypoint Navigation – Square Pattern

Conclusions and Future Plans

I am very happy with the way this robot turned out. It has accomplished my main goals and some of my “reach” goals. In the future, I would like to continue to develop this platform as well as apply the knowledge and experience gained to other platforms. I feel a small lightweight quadcopter suitable for indoor flight would be a good followup to this project. Additionally, I would like to modify the current software for use in small model helicopters. The standard helicopter would be able to achieve much better runtime and efficiency but admittedly would be harder to control.

Some improvements for the current project include the following of a land based object. This would be accomplished by having a beacon consisting of another GPS module attached to an Xbee, which transmits the beacon’s location to the robot. The robot can then follow the object carrying the beacon from overhead by using the beacon’s location as a sort of moving waypoint. Additionally, I would like to increase the autonomity of the robot by allowing it to perform automatic takeoff and landing.

References

APM board http://www.sparkfun.com/products/9710

Oilpan Sensor Board http://store.diydrones.com/product_p/br-0013-01.htm

Sonar information http://www.maxbotix.com/MB1010___LV-MaxSonar-EZ1.html

GPS Module http://store.diydrones.com/MediaTek_MT3329_GPS_10Hz_Adapter_Basic_p/mt3329-02.htm

DCM algorithm http://diydrones.com/profiles/blogs/dcm-imu-theory-first-draft?xg_source=activity

Code

The final code, including libraries, used at the end of this class can be downloaded in zip format from my website.

Additionally, my ongoing development code can found in my branch of the subversion repository for the *Ardupirates* project here: http://code.google.com/p/ardupirates/