SELF-AWARE UNMANNED AERIAL VEHICLE

COMPUTER ENGINEERING SENIOR PROJECT

2010

http://pisco.flux.utah.edu/uav

GRANT E. AYERS

grant.ayers@utah.edu

NICHOLAS G. MCDONALD

nic.mcdonald@utah.edu

DECEMBER 23, 2010

TABLE OF CONTENTS

TABLE OF FIGURES

Figure 1 - UAV Helicopter

1.0 ABSTRACT

Unmanned Aerial Vehicles (UAVs) are used for many purposes when employing a pilot is impossible, dangerous, or too expensive. They rely on an array of environmental sensors to replace a pilot's awareness of the world and time-critical analysis of sensor data to make complex decisions in real-time. Without this capability, no UAV could remain stable, safe, or airborne. We present a sensor and analysis system to enable self-aware, sensor-assisted unmanned flight on a UAV using only commodity hardware.

2.0 INTRODUCTION

Unmanned Aerial Vehicles are used in a variety of applications spanning large-scale military grade hunter-killer surveillance vehicles to smaller personal and industrial vehicles. Small-scale UAVs with limited range are sometimes controlled through direct commands by a person on the ground within a line of sight. However, many applications exist which prevent the use of a human pilot in the air or on the ground. Any UAV that requires traveling more than a few hundred meters, with negative visibility, or under conditions where human presence is dangerous or impossible must be able to sense its own environment and make time-critical flight decisions unassisted. Accomplishing this requires enough sensors and computational capability to fully substitute a human pilot. We have achieved this using common and low-cost hardware.

In typical flight scenarios, a pilot gathers environmental information through his or her own senses with supplemental instrumentation data. Our system supplies all environmental information through the use of three accelerometers, three gyroscopes, three magnetometers, a GPS receiver, and an ultrasonic range finder. This data is sent periodically to a flight control system, which analyzes it and replaces a pilot's decision making with its own software routines. These routines include a flight stabilization unit, hardware control, and an interface to accept commands from and send status messages to a ground station. With the proper synchronization between all of these systems stable, safe, and continuous flight is possible.

3.0 OVERVIEW

Our vehicle is a small helicopter with a length of 3.8 feet and a rotor diameter of 4.4 feet [1]. It can carry a payload of over five pounds, which includes all of the sensor and computational hardware. The helicopter carries eleven sensors and four independent microcontrollers. All sensor data is brought into the onboard flight control system either directly or through these intermediary microcontrollers. The flight control system is responsible for controlling the servos of the helicopter based off of sensor readings, ground commands, and parameters sent from a ground station. The ground station is a standard computer running software which connects to the helicopter through a wireless link. The system also supports a safety mode which overrides the autonomous flight control system and allows the pilot to directly control the helicopter through a handheld transmitter. The helicopter is capable of stabilizing any combination of roll, pitch, yaw, heading, ground distance, rate of ascent, and absolute position. When all of the stabilization modes are enabled, the helicopter is able to move to and remain in specific 3D locations relative to earth.

4.0 SYSTEM DESIGN

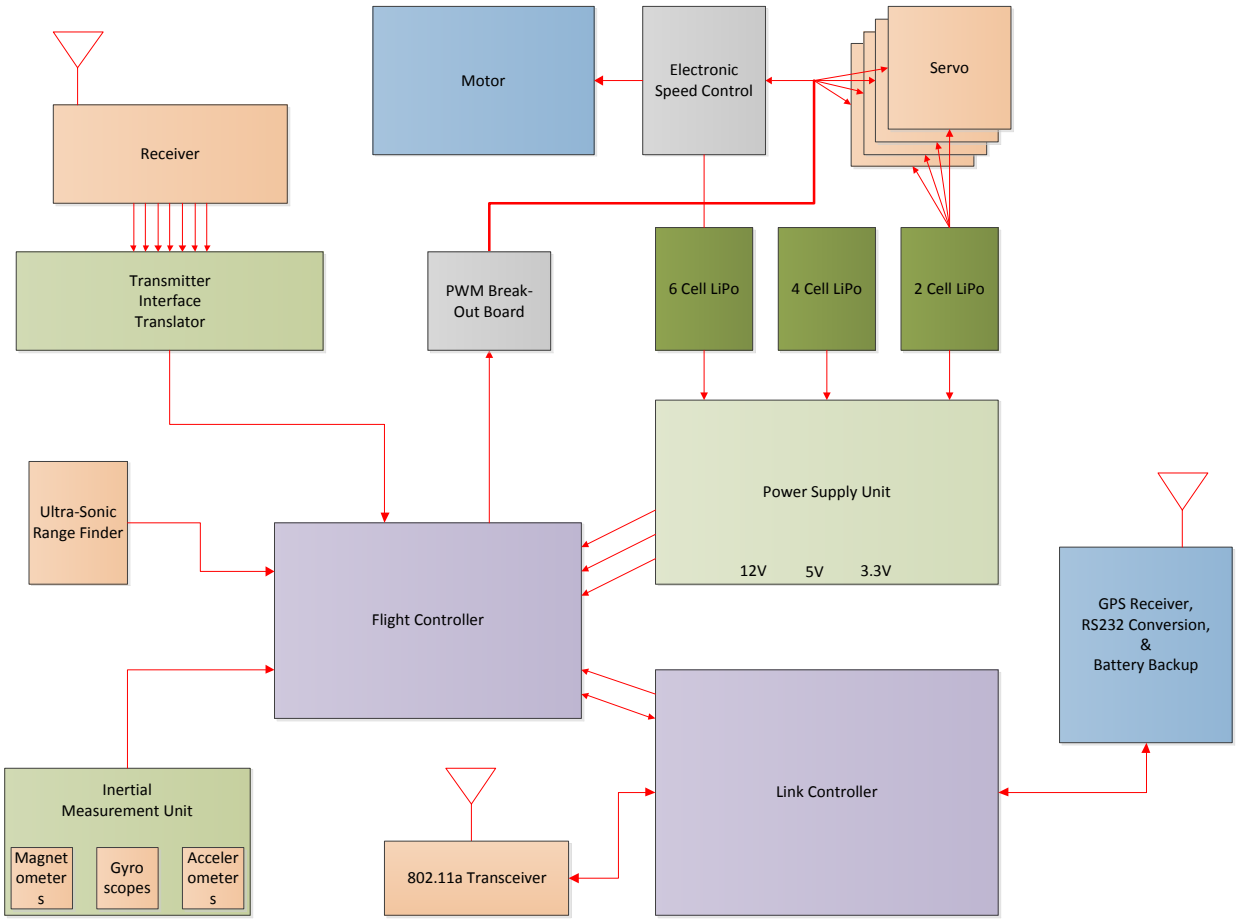

Figure 2 - Hardware Flow Diagram

Our system hardware design utilizes several processing devices that run parallel to each other. The basic flow of the aircraft hardware is a central processing unit with numerous coprocessors assigned

to specific tasks. All sensor data is attached to one of the processing units. Many complex data protocols are used to synchronize the timing of these processors and transfer data from one device to another.

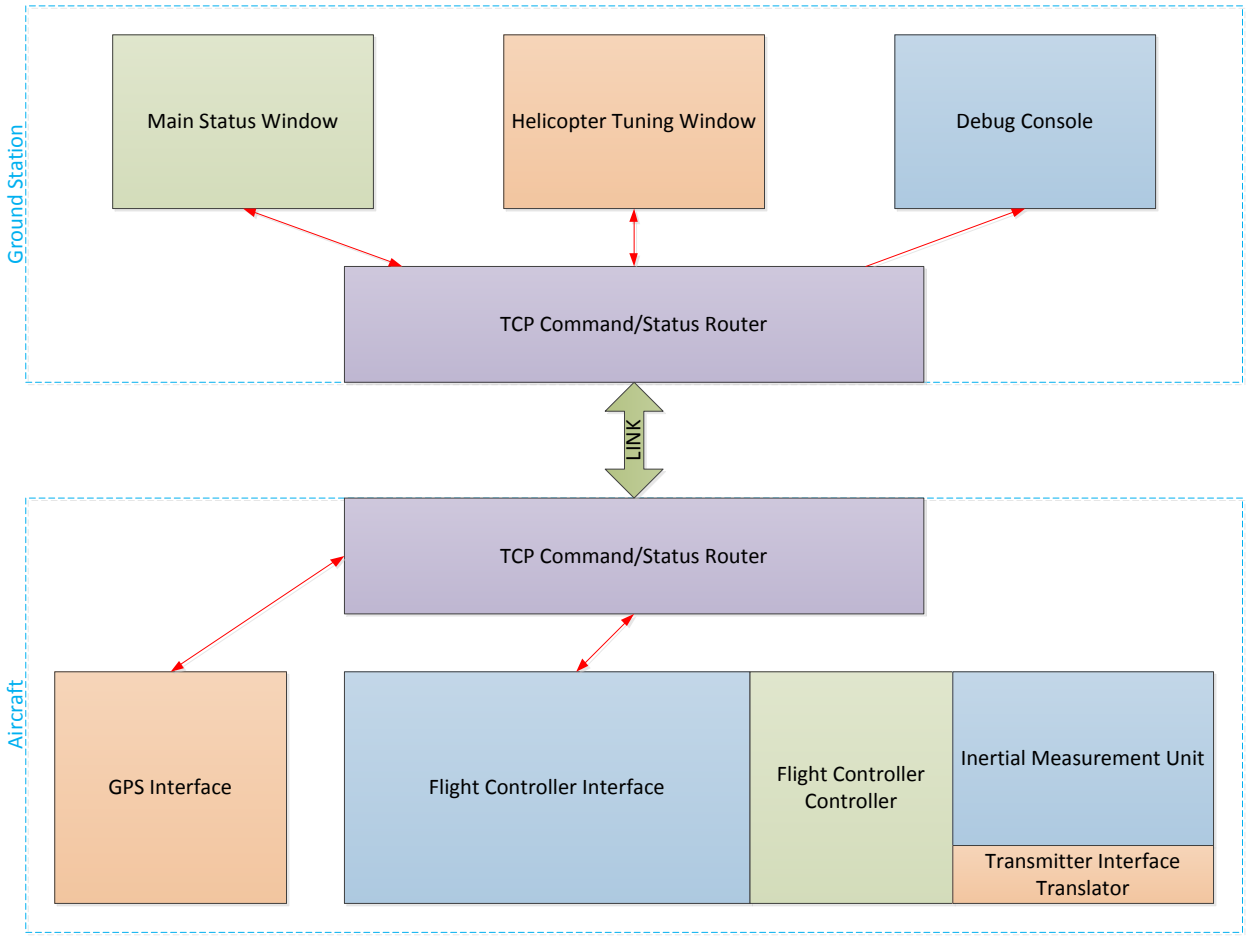

Figure 3 - Software Flow Diagram

Data originates on the ground in the Ground Station software. The Ground Station software is a high level graphical user interface that allows a pilot to control the helicopter. Data is then transferred wirelessly from the ground to the air. On the helicopter, the data is captured in a high-level software suite that processes the data before it reaches critical helicopter components. Once the data is validated, it is forwarded on to the lower level processors that control the aircraft. Data also flows in the exact opposite direction using the same methodology. The lower-level software on the helicopter is spread across several processing devices. Each low-level processing device contributes to the overall stabilized control of the helicopter.

5.0 THREE MODES OF FLIGHT

The pilot can choose one of three modes of flight using a 3-way switch found on the transmitter. Each mode of flight is controlled in the software that resides on the aircraft. The three modes of flight are as follows:

5.1. MODE 1 - MANUAL FLIGHT

Flight mode 1 is a very basic flight mode which resembles the flight style of a typical RC helicopter. The cyclic control (similar to an airplane's aileron and elevator controls) are set manually by the pilot. The tail is auto-stabilized such that the transmitter tail control sets a desired yaw rate and the helicopter's actual yaw rate matches it. To fly in this mode the pilot must have a lot of training and experience driving RC helicopters. Helicopters are inherently unstable and this causes the pilot to continuously make corrections to the aircraft's attitude using visual feedback.

5.2. MODE 2 - STABILIZED FLIGHT

Flight mode 2 is a highly stabilized flight mode typically only found in military helicopters. The pilot sets the desired aircraft attitude using the transmitter. The aircraft's Flight Controller translates the raw transmitter signals into desired aircraft attitude angles. The flight control system then sets the aircraft at the desired angle. Piloting in this mode is much easier because the transmitter sticks always reposition themselves to the center when not being physically moved. This means that the pilot can simply let go of the cyclic control stick which will make the aircraft level with the horizon and stop any rotational velocity. The only manual flight control needed to fly in mode 2 is the collective pitch. This is the control that sets the aircraft's ascent/descent rate. Given that the rest of the system is fully stabilized, controlling the collective pitch is very easy.

5.3. MODE 3 - NAVIGATED FLIGHT

Flight mode 3 is a fully autonomous flight mode that has the ability to navigate to any 3D coordinate. The only use of the transmitter during this mode is to set the mode. All other controls are set using the custom ground station software. In this mode the aircraft accepts positional commands from the Ground Station. The position control system is based around global positioning system (GPS) satellites and an onboard GPS receiver.

The navigation software onboard the flight control system uses a distance magnitude crossover system. When the aircraft is within the distance threshold set by the Ground Station, it uses a directional navigation algorithm that controls the aircraft's roll and pitch to navigate to the desired location. In this navigation mode the tail control is always zero and the helicopter has no rotational velocity. When the aircraft is above the threshold, it uses a pointing navigation algorithm that controls the aircraft's pitch and yaw rate but keeps the roll at zero. In this mode the helicopter points in the direction it needs to go and uses its pitch to drive forward. Once the helicopter is below the threshold again it switches over to the directional navigation mode.

6.0 COMPONENTS

6.1. POWER SYSTEM & POWER SUPPLY UNIT

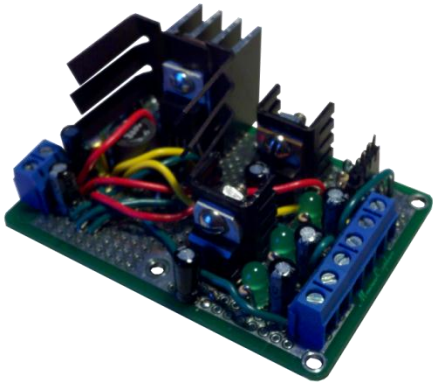

Figure 4 - Power Supply Unit (PSU)

Our UAV design utilizes Lithium-Polymer (LiPo) batteries for powering the various parts of the aircraft. LiPo batteries can store an extremely large amount of energy while having a relatively low weight. This large amount of energy comes at a cost, namely, LiPo batteries have a very strict voltage range of operation. Allowing a LiPo battery to become overcharged will result in an explosion or fire. Allowing a LiPo to become undercharged will result in permanent damage in which the battery will no longer hold charge. The voltage of any cell of a LiPo battery must be within 3.0V to 4.23V.

Our design uses three LiPo batteries: 2100mAh 20C 2-Cell, 2500mAh 30C 4-Cell, and 5000mAh 30C 6-Cell. The 2-Cell battery is used only for powering the servos and electronic speed control (ESC). The 4-Cell battery is used to power all other onboard electronics. The 6-Cell battery is used only for powering the motor.

Due to the strict voltage requirements of LiPo batteries, our Power Supply Unit (PSU) needed to supply voltage status signals as well as the different voltage sources for the system. The only battery that needed further regulation is the 4-Cell battery. Our power distribution system on the PSU takes the varying 4-Cell voltage of 12V to 17.2V and regulates it to 3 supply voltages: 3.3V, 5V, and 12V. The PSU also takes the 3 battery voltages and uses a voltage divider to create a reference voltage signal between 0V and 5V. These voltages can then be monitored by the flight control system and be sent to the Ground Station.

6.2. TRANSMITTER INTERFACE TRANSLATOR

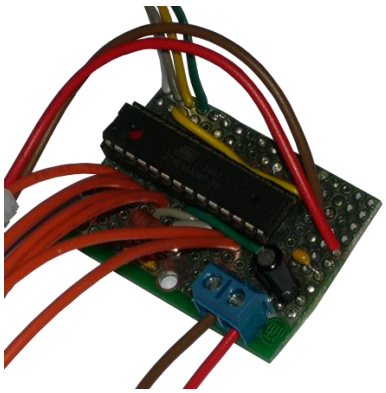

Figure 5 - Transmitter Interface Translator (TIT)

The Transmitter Interface Translator (TIT) is a custom microcontroller circuit designed specifically to capture the seven pulse width modulation (PWM) channels that the standard receiver outputs. For this design we chose the 8-bit ATmega328P AVR microcontroller [2] for its unique ability to simultaneously capture pin change interrupts across an entire port.

This component is necessary because capturing seven channels of PWM across 20 milliseconds uses a substantial amount of processing time in which interrupts are used. We decided to have this processing done separate from the main Flight Controller (FC) in order to parallelize the computation.

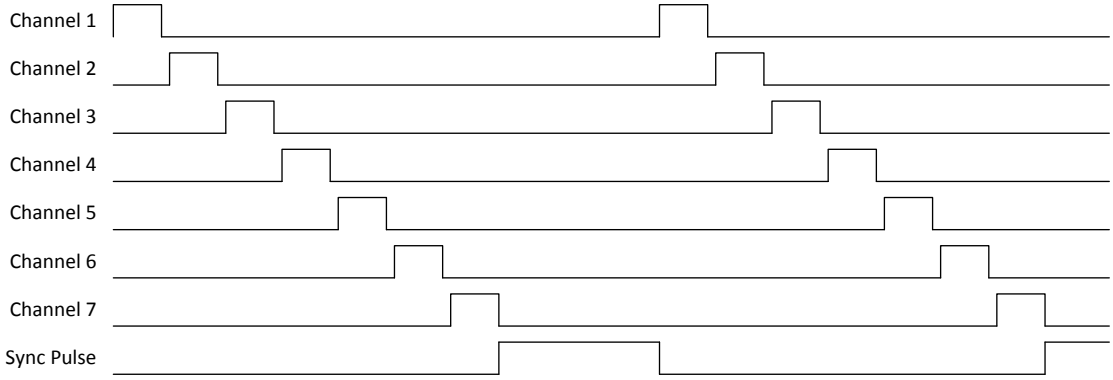

Figure 6 - TIT Timing Diagram

After a bit of testing, we found that the receiver always outputs the seven channels in the same order and always separates the beginning of each frame by 20 milliseconds. The pin change interrupt system of the AVR microcontroller has the ability to execute the interrupt code for any pin that changes logic levels on a given port. We designed an 8-stage finite state machine (FSM) for determining the pulse width of each channel. We made the FSM self-synchronizing by resetting it anytime the width was above a certain threshold.

After gathering an entire frame of data (seven channels), the TIT asserts a control signal that tells the FC that a packet is available. The result of this control signal becomes a 50 Hz synchronizing pulse. We decided to synchronize the entire system to this pulse. This strategy allows us to have a highly parallel system while not worrying about timing constraints.

The communication between the TIT and FC is an I^2C bus in which the TIT is a slave at a fixed address and the FC is the master. The TIT simply reports the pulse width for each of the 7 channels.

6.3. Inertial Measurement Unit

Figure 7 - Inertial Measurement Unit (IMU)

To accomplish any kind of stabilization, the aircraft must know its current attitude. We decided to use an Inertial Measurement Unit (IMU) for determining aircraft attitude. We purchased a board that has an 8-bit ATmega328P AVR microcontroller, triple-axis accelerometer, dual-axis gyro, and a single-axis gyro [6]. We also purchased a triple-axis magnetometer and attached it to the top of the IMU board.

Figure 8 - 3-Axis Magnetometer

An accelerometer measures static acceleration typically due to gravity. Actual aircraft acceleration causes the readings to be inaccurate. A gyro measures angular velocity on a rotational axis. Gyros cannot give an absolute attitude because they measure a rate rather than an angle. A magnetometer measures the magnetic field on one axis. Our IMU system measures three axes of gyroscopic velocity, three axes of acceleration, and three axes of magnetic field.

No sensor alone can provide enough information to determine the current aircraft attitude. We used the direction cosine matrix (DCM) algorithm [7] for fusing the 3 sensors together to produce aircraft attitude. The DCM algorithm works by using gyroscopic magnitudes to determine the priority of each sensor. As the gyro reading increases in velocity, the DCM algorithm increases the gyro priority and decreases the accelerometer priority. This makes intuitive sense because accelerometers are good for reading static acceleration and gyros are good at reading objects in motion. On all the axes of measurement, the gyro is used to estimate the position by integrating the gyro output. When the aircraft is not in rapid motion, the accelerometer and magnetometer are used to correct the gyro estimation error.

Our IMU system (IMU board with the attached magnetometer) was designed to be a coprocessor to the FC due to the high computational effort needed to compute the DCM algorithm. The onboard microcontroller gathers the sensor data, computes the DCM output, and reports the aircraft attitude to the FC. The values reported are: roll, pitch, yaw, and heading. This computation is indirectly synced with the 50 Hz signal generated by the TIT.

6.4. FLIGHT CONTROLLER

Figure 9 - Flight Controller (FC)

The Flight Controller (FC) is the central control processing unit for the aircraft. All sensor data converges on this unit. The hardware chosen for its implementation is an 8-bit ATmega1280 AVR microcontroller development board [2]. This microcontroller is well suited to be the central processor because it has many serial I/O ports. Each coprocessor of our system communicates to the FC using some serial communication protocol.

Besides serial protocol interrupts, this microcontroller is running only one thread for all its processing. This control thread self-synchronizes with the 50 Hz signal generated by the TIT. The result of this is that the control functions generated by the FC update the system at 50 Hz.

The first task of the FC is to gather all the data from the coprocessors. It first receives the transmitter data from the TIT and then receives the attitude data from the IMU. It then decides which stabilization features to implement based on the selected flight mode. For example, if the pilot has selected flight mode 1, the FC only computes the stabilized output for the tail signal. The rest of the signals are output in the same way the pilot set them using the transmitter. If the pilot has selected flight mode 2, the FC computes the stabilized output for roll, pitch, and yaw signals based on angles set by the transmitter controls and settings on the Ground Station. If the pilot has selected flight mode 3, the FC computes the stabilized outputs based on GPS position and altitude rather than transmitter input. The desired 3D position is selected on the Ground Station.

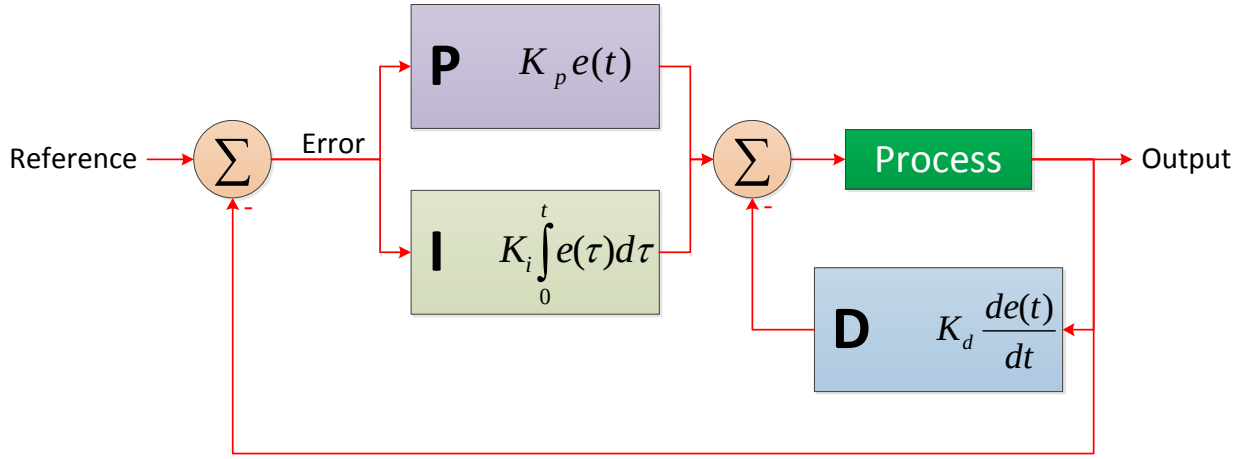

Figure 10 - Modified PID Feedback Diagram

The stabilization system of the FC is based around modified proportional-integral-derivative (PID) feedback controllers. For each axis of stabilization needed, there is a PID controller that can be utilized to control the aircraft. The proportional, integral, and derivative gains are set using the Ground Station software. These settings are stored in EEPROM so that the optimized values are not lost after each flight.

After all the controls have been decided and computed, they must pass through a mixing algorithm tailored to our helicopter's cyclic/collective swash plate. This algorithm is called cyclic collective pitch mixing (CCPM). The swash plate on the T-Rex 600 ESP helicopter has 3 points of connection separated by 120° each. The aileron, elevator, and pitch controls all get put into this algorithm. The purpose of the algorithm is to translate these three signals into three signals that will directly control the three servos connected to the swash plate. We designed our algorithm to work only with this type of swash plate. The output of the CCPM algorithm was sent directly to the servos using pulse width modulation (PWM).

The FC has numerous settings that the pilot can set from the Ground Station software. Many of these are stored in EEPROM so that the settings will not be lost between flights. The FC implements a custom command/status protocol over a standard UART. The Link Controller sets the interface between

the FC and the Ground Station. The FC was set up to be able to handle 35 command or status packets per second because much of the data collected by the FC would be requested frequently by the Ground Station for the user interface. The pilot could then see what the aircraft is doing in real-time.

Figure 11 - Ultra-Sonic Range Finder

The FC also has the ability to interface with stand-alone sensors such as ultra-sonic range finders, temperature sensors, and other such devices. Our system uses an ultra-sonic range finder underneath the aircraft to measure the distance to ground. This is useful for autonomous take-off and landing. More range finders can be attached to create a collision detection system.

6.5. LINK CONTROLLER

Figure 12 - Link Controller

The Link Controller provides data communication between the Ground Station and the onboard GPS receiver and Flight Controller. It's a VIA ARTiGO A1000 microcomputer with a 1GHz VIA C7 processor and 512MB of memory [3]. This is essentially a small embedded version of an x86 system you might find on your desk. This unit is lightweight and includes four USB ports. While running, the ARTiGO draws approximately 15 watts.

We originally chose this unit for its ability to support 802.11 WIFI and GSM/CDMA 3G data links between the ground and air. Many USB adapters exist which support these types of data links. However, the majority of them have closed-source drivers that only operate in specific environments. Due to this limitation, we chose to install Windows XP on the Link Controller because of its nearly universal driver support. This allowed us to focus on other project goals instead of reverse engineering binary drivers for USB devices. Because of the lack of stability and timing guarantees in a Windows XP system, we designed all of functionality of the Link Controller to be as fast as possible but not time-critical. We also allowed for the Link Controller to crash without destroying the flight control system of the helicopter. Despite the disadvantages of using a non-real-time operating system (not to mention Windows), the speed offered by the 1GHz CPU allowed us to be confident about its computational capabilities.

The Link Controller's functionality is contained within a single air support program which runs continuously. At a high level, the program is a three-point data router with the Ground Station, GPS receiver, and Flight Controller as the endpoints. The implementation is accomplished with approximately 3,000 lines of C# source code which define the GPS, Flight Controller, and Router interfaces using over ten synchronizing threads. The Router interface contains a connection handler which maintains contact with the Ground Station over TCP, and takes the appropriate action when this connection is interrupted or lost. A standardized data packet can be sent between the ground and air through the Router. This data contains a GET or SET command and any applicable data. The Router forwards the data to either the GPS or Flight Controller interface. Likewise, these two can send a data packet to the Router which will automatically be sent to the Ground Station.

The design of the Link Controller software was methodical and tedious, but provided a very stable system upon which we could confidently rely. The hardware was equally dependable.

6.6. GLOBAL POSITIONING SYSTEM

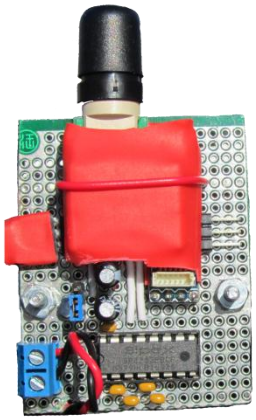

Figure 13 - GPS Receiver

The majority of the sensors on the helicopter report information about relative motion, but provide no reference for absolute position. Even with the aid of the magnetometer and range finder, which provide limited absolute heading and distance-to-ground, the combined collection of sensors would be too inaccurate to enable the helicopter to hold a position or fly to a specific location without a three-dimensional absolute coordinate position system. The Global Positioning System (GPS) allows for this.

Our approach to incorporating GPS into the flight system is based on relative referencing. The GPS interface first records an absolute GPS position as the origin when a valid lock becomes available. From there the GPS interface can calculate its position relative to the origin based on current GPS readings. Calculating the distance between two GPS coordinates, however, is non-trivial. One must take into account that the earth is neither flat nor spherical, but rather an approximate oblate spheroid. The coordinate system is most definitely not a simple 2D or 3D Cartesian space. There are several formulae which attempt to simplify this problem by making gross mathematical assumptions, but many of these lack real accuracy. We decided to use the Bowring Distance Formula which enables distance calculation to within a millimeter accuracy anywhere on earth for distances less than 150km [4]. It greatly simplifies a full conversion by reducing the required operations by more than half. The GPS Interface uses this formula to then keep track of its origin and current position North, East, and "Above" relative to the origin in units of feet.

Our design uses a u-blox 5H-based GS407 GPS receiver with the Sarantel SL1206 active antenna [5]. This module normally runs on 3.3V and draws approximately 70mA. It is excellent at finding a 3D

lock within less than a minute from a cold start, and even detects a strong signal indoors. All commodity GPS receivers support the Coarse Acquisition (CA) L1 frequencies, which only enable position pinning to within several meters at best. This module has support for the Wide Area Augmentation System (WAAS) as well, which enables it to receive correctional data from specific North American satellites (currently PRN #135 and PRN #138) yielding a position pinning accuracy within two meters based on testing with this device. The u-blox module is very configurable, and includes support for its own binary data format instead of the ASCII-based NMEA output format. We set the device to report a valid lock only when 3D and within 10 meter accuracy estimates. We also filtered its drift with a static hold threshold of 0.1 meters/second.

The u-blox GPS module is incorporated into the flight system of the helicopter through an RS-232 serial port into the Link Controller. The GPS Interface module of the air software system is in charge of continuously reading current GPS information, computing relative distances, and presenting all of this data to the rest of the flight system. This data includes but is not limited to the following: GPS current and origin coordinates in DMS format, relative distances North, East, and Above the origin, velocity in three and combined dimensions, atomic date and time, accuracy estimates for horizontal, vertical, and 3D position, the number of satellites currently being used in the computation, and other status information such as the type of GPS fix and whether the WAAS is being utilized. The Ground Station can request some or all of it, or initiate automatic updates from the GPS interface at specified rates. The Flight Controller interface continuously polls portions of the GPS data and can use it with the stabilization system to hold still or seek specific points. This is only possible when the other stabilization systems are calibrated and functional.

6.7. GROUND STATION

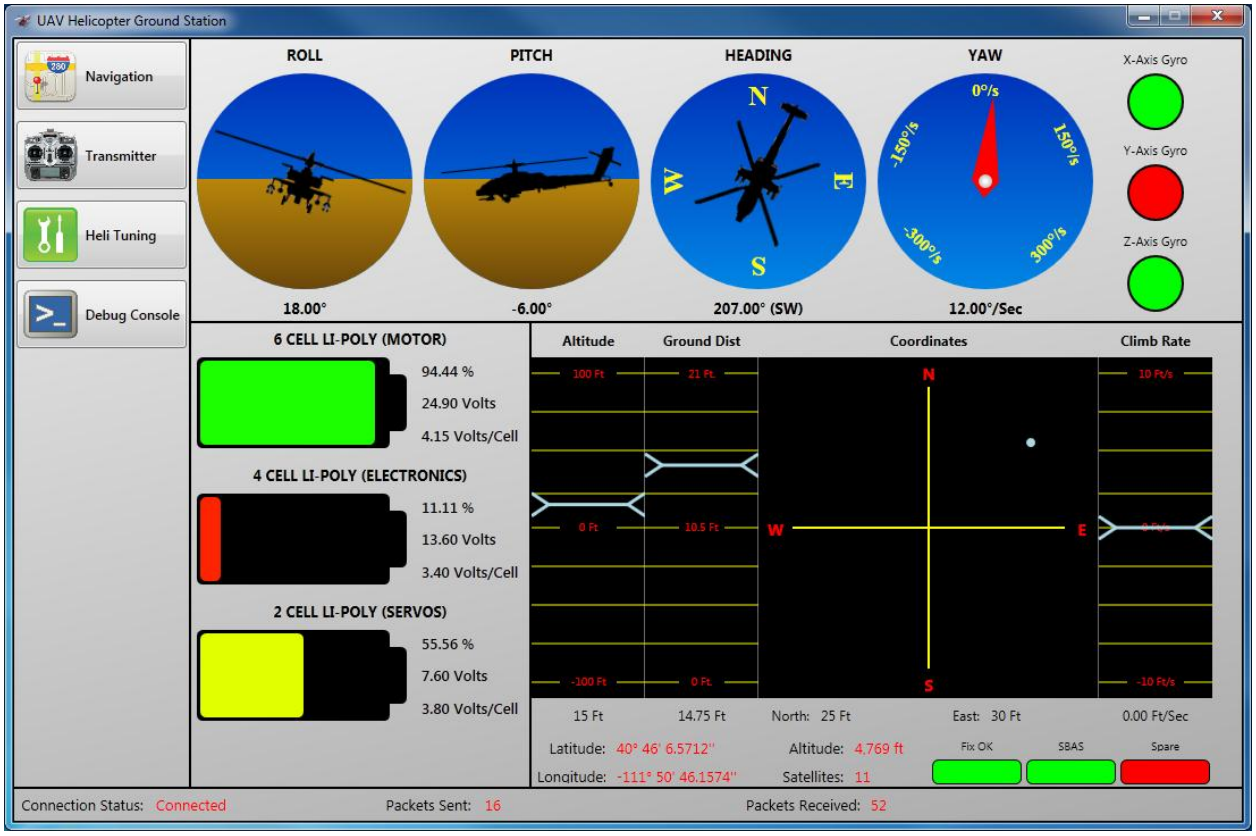

Figure 14 - Ground Station Main Window

The Ground Station provides a wireless communication link to the helicopter. It enables commands to be sent to the helicopter and receives flight information and other status messages. The Ground Station consists of a WIFI router and laptop. The router was chosen to extend the wireless range and signal strength when compared to an ad-hoc wireless configuration. We used a laptop for portability but were otherwise unconcerned with the hardware specifications of the Ground Station.

We wrote the Ground Station software in C# using the Windows Presentation Foundation (WPF) platform, which we found to have a high visual appeal to development time ratio. The purpose of this software is twofold: first, to allow command and status information to be transmitted between the ground and air, and second, to provide a visually-appealing presentation of the status data from the air. We accomplished this with over 4,500 raw lines of code, or 6,200 including the XAML GUI support code. However, as with most GUIs, much of the code is support for UI controls and is repeated throughout the program.

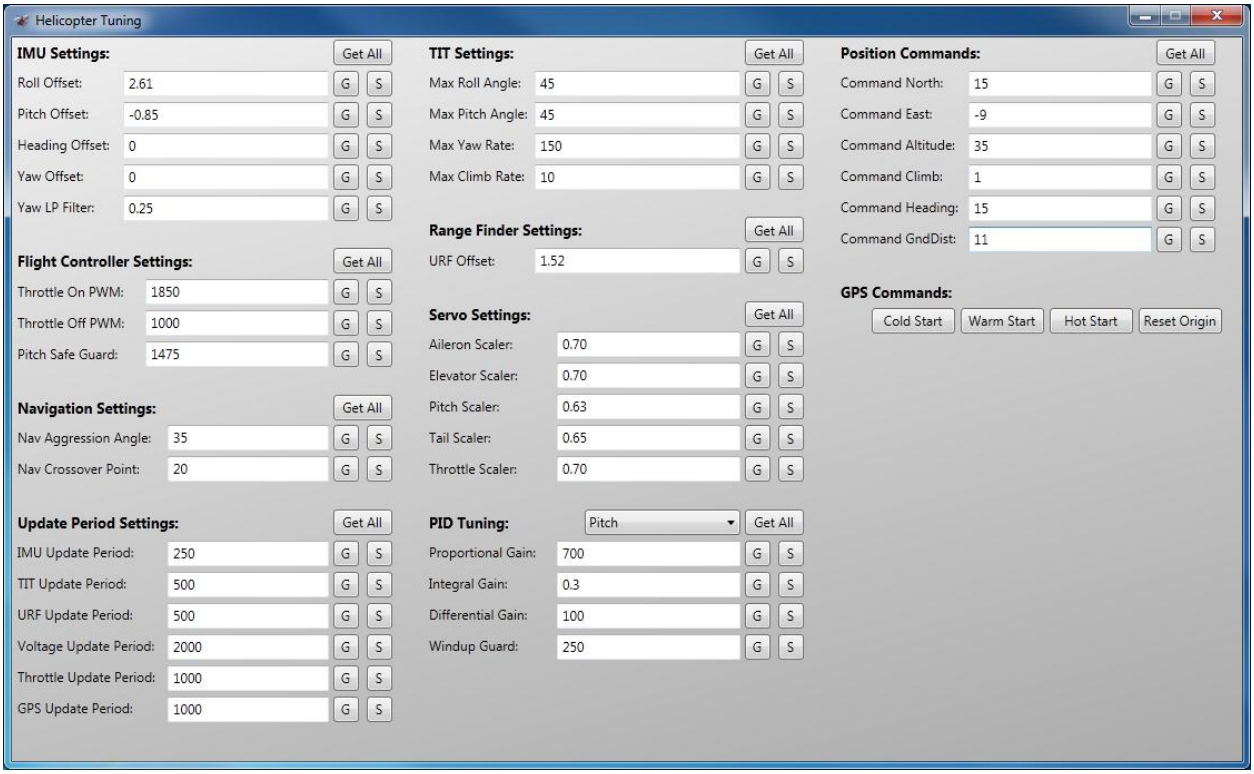

Figure 15 - HeliTuner Window

The main window of the Ground Station software shows in-air flight information from the helicopter, including roll, pitch, heading, yaw, altitude, ground distance, relative location, GPS location, rate of ascent, and battery status. The “Heli Tuning” window allows the setting and polling of over 60 settings and tuning parameters of the helicopter such as PID gains, position offsets, update rates, and filters. It also allows for positional commands when all stabilization units are active. The tuning window was crucial to stabilizing the helicopter, since PID values, offsets, and other settings had to be determined during flight. The “Debug Console” provides a way to look into the network traffic between the Ground Station and Link Controller on the helicopter for debugging purposes.

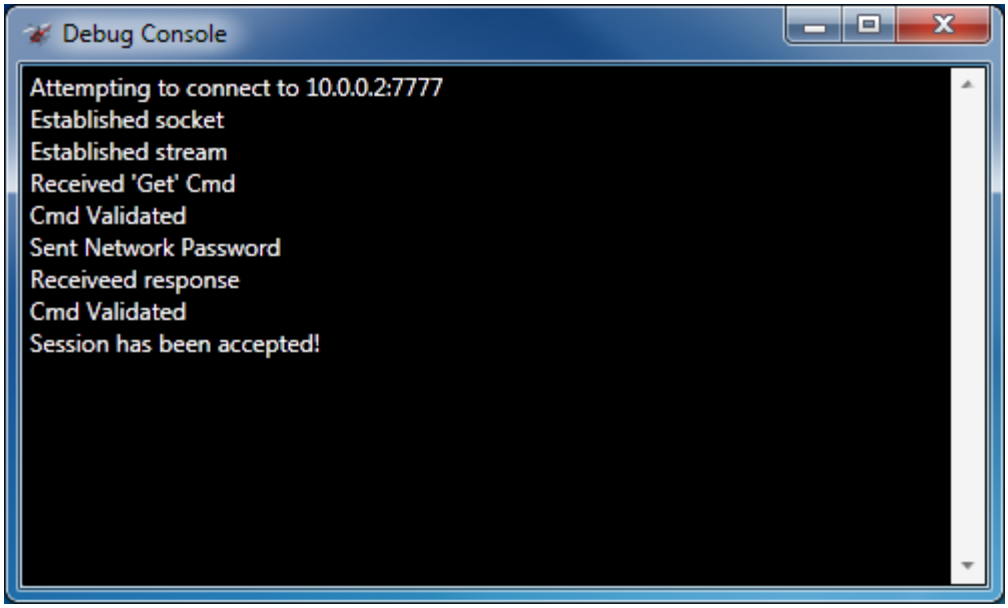

Figure 16 - Debug Console

While the Ground Station software provides control and a thorough view of the helicopter, it is not a necessary component of flight. We designed the flight system to operate with or without a link from the Ground Station. In the event that a link is disrupted while in autonomous mode, the helicopter will either hover or return to the origin depending on the configuration. While in manual mode, a disrupted link does not affect the helicopter at all. This design approach allowed for great robustness and flexibility while testing the system.

7.0 DEVELOPMENT DIFFICULTIES

After our UAV was designed and implemented, we ran into several problems. The major problems we encountered during development are as follows:

7.1. RF SPECTRUM INTERFERENCE

The transmitter we have for the helicopter is a Spektrum Dx7 which operates on the 2.4GHz band. The WIFI module we begun using was an 802.11g USB modem. Due to both wireless technologies utilizing the 2.4GHz band, we had huge interference issues. Luckily the Dx7 transmitter transmits at a higher power than the WIFI module. Typically the interference caused the Ground Station to drop packets or become completely disconnected. A few unfortunate cases caused the RC receiver to pick up corrupted packets which in turn caused extreme out-of-control behavior of the aircraft. Luckily, it didn't last long, and we never crashed.

We corrected our design by switching from 802.11g to 802.11a which utilizes the 5GHz band. This was a great switch because most modern wireless technologies use the 2.4GHz band, and thus the 5GHz band is less congested.

7.2. VOLTAGE PLANE DISRUPTION

Figure 17 - Static Suppressor

After the system was set up, we found that the power system suffered significantly from static electricity and other voltage noise sources. We created a chassis ground plane using all the large metal sources found on the helicopter. This helped a little bit but the system was still vulnerable. We decided to add a large capacitor bank as a static suppressor to the two main power supply outputs: 5V and 12V. This device adds a 4700µF capacitor and a 1000µF to each voltage source. After adding the static suppressor, the system became much more tolerant to static and other noise sources.

7.3. VIBRATION INDUCED SENSOR DATA CORRUPTION

The first time we spun up the rotors with all our electronics connected we noticed that the IMU data was completely corrupted. At first we assumed the motor was creating a large magnetic field around it. We tried moving the IMU to different mounting places on the helicopter but nothing seemed to help. We then put the IMU on the ground next to the helicopter but with no direct connection. This test resulted with uncorrupted data. We then realized that the data corruption was not coming from magnetic fields, but rather it was coming from high frequency vibration that resonates through the helicopter chassis.

Along with destroying sensor data the vibration induced noise destroyed all communication with the hard drive in the Link Controller. Due to this issue, we had to trick Windows into booting from a USB thumb drive. After we figured out how to pull this off, we discovered that even though Windows can boot from a USB device it still requires that at least one regular hard drive be attached, even if it isn't being used.

We attempted many mounting systems to rid the IMU of vibration induced noise. The strategy that worked best was mounting the IMU inside upholstery foam. The best location for mounting the IMU inside the foam was beneath the Link Controller. This is due to its relatively large mass and surface area. Mounting in this location allowed us to use a lot of foam and keep the IMU centered and level. This mounting location was great for vibration but introduced another data corruption issue.

7.4. MAGNETIC FIELD INDUCED SENSOR DATA CORRUPTION

After we found the ideal location to rid the IMU of vibration induced noise, we discovered that the magnetometer was reporting a false heading. By moving the IMU around, we discovered that the hard drive within the Link Controller was inducing a magnetic field on the magnetometer. Typical hard drives have an electromagnet that moves along a magnetic field created by a rare-earth magnet. This was the source of our magnetic field induced sensor data corruption.

At this point we knew that we couldn't mount the IMU in the ideal location for vibration, and we had a trade-off decision to make between vibration induced noise and magnetic field noise. We found a mounting position for the IMU in a location with almost no magnetic noise. The location produced

some vibration induced noise but we were able to filter it out using a digital low-pass filter on our sensor readings. We made the low-pass filter crossover point selectable via the Ground Station software so that we could optimize the frequency response during flight.

8.0 RESULTS

Figure 18 - Flying 'No Hands'

Our system design and implementation was a huge success. Flying an unstabilized helicopter is near impossible even for expert pilots. Having just one axis of stabilization, typically the yaw axis, results in a much easier flight. Having several axes of stabilization causes the flight to seem near autonomous.

Having a complete ground station with an exhaustive set of features was invaluable during the testing phase. Even though implementing these features took literally thousands of lines of code, we would not have been able accomplish all that we did without the ability to change so many settings during flight. Another priceless feature was the automatic connecting and negotiating data link between the air and the ground. Our system allowed us to break any connection without causing errors or restarts to initiate more connections. The link design was created to be extremely tolerant to low latency connections and spurious connection drops. Had we not had this foresight, we would have spent a lot of time in frustration.

Figure 19 - Flying Indoors

We were able to implement the full UAV functionality of the helicopter, however, we were not able to calibrate and test all the methods of stabilization and navigation. An under sight on our part was not knowing the vast amount of time it takes to calibrate each stabilization function. All of the stabilization and navigation functions are implemented in the embedded software but without sufficient time, we were unable to calibrate and test all of them. This was also due in part to the non-ideal mounting location for the IMU which did not hold the IMU secure enough to keep consistent angular offsets. We designed the software system to be ready to go only after sufficient calibration, and found that our system worked exactly as planned once the proper calibration settings were made.

9.0 CONCLUSION

Modifying an aircraft to support sensor-assisted unmanned flight requires rapid sensor data acquisition, frequent calibration, and time-critical computation. We developed a flight control system which provides in-air stabilization and navigation to a helicopter using eleven environmental sensors and four onboard microcontrollers. We implemented this system using only common inexpensive parts. The result of our project is a fully-enabled UAV which implements control of roll, pitch, yaw, heading, ground distance, rate of ascent, and absolute global positioning with or without support from a ground station. It is capable of maintaining stable and directed flight between any set of geodetic coordinates and provides a communication link through which status information and control commands may be sent. While unmanned aerial flight is not a new concept, it has nevertheless been largely inaccessible to the public due to cost and lack of implementation. Our project has broken these barriers by successfully implementing a low-cost UAV described in full in this report.

10.0 ACKNOWLEDGEMENTS

Figure 20 - L3 Communications

We would like to thank L-3 Communications for generously lending us the T-Rex 600 ESP RC Helicopter and Spektrum Dx7 Transmitter/Receiver pair. Without a high quality RC platform, we could not have accomplished our project goals. We would also like to thank them for extending their array of batteries to support our project. Working with only one battery would have made development near impossible.

Figure 21 - VIA Technologies

We would also like to thank VIA technologies for donating the VIA ARTiGO A1000 embedded x86 system. Having this unit helped us focus on aircraft-specific engineering without wasting time on driver development issues. A small embedded computer that is capable of running operating systems with large amounts of support was a very valuable.

11.0 SOURCE CODE

All of our source code is rooted at http://pisco.flux.utah.edu/uav/code. A map to each specific section is provided below:

11.1. GROUND STATION

The Ground Station is a single Visual Studio 2010 project. All code is written in C# with supporting XAML for the Visual Presentation Foundation GUI support. "Settings.cfg" is the configuration file for the software and provides configuration information for the TCP/IP connection settings.

http://pisco.flux.utah.edu/uav/code/HeliGroundSoftware

11.2. LINK CONTROLLER

The Link Controller is also a Visual Studio 2010 project. "VIA.cs" is the top-level source file. It links in the code for the Router, Flight Controller and GPS interfaces. "Settings.cfg" is the configuration file for the software and provides configuration information such as serial port names and TCP/IP connection settings.

http://pisco.flux.utah.edu/uav/code/Heli_VIA_Software

11.3. FLIGHT CONTROLLER

The Flight Controller software was written in the Arduino IDE. The environment is technically C++ but we wrote all of our software to be C-compliant for use in other compilers. The project, or "sketch" root file is "FlightController.pde," and everything else comprises code and header files.

http://pisco.flux.utah.edu/uav/code/FlightController

11.4. INERTIAL MEASUREMENT UNIT

The Inertial Measurement software was also written within the Arduino IDE. This directory contains "sketches" for the full IMU and individual components such as the ADC, compass, DCM algorithm, and other supporting code.

http://pisco.flux.utah.edu/uav/code/InertialMeasurementUnit

11.5. NETWORK COMMANDS GENERATOR

The Network Commands Generator is the system we used while developing the communication protocol between the ground and air. It allowed us to modify commands and automatically update the Ground Station, Link Controller, and Flight Controller software packages.

http://pisco.flux.utah.edu/uav/code/NetworkCommandsGenerator

12.0 REFERENCES

[1] Assurance RC. 2010. *T-REX ESP Superior KX016013A* [Online]. Available: http://www.alignrcusa.com/index.php?main_page=product_info&cPath=3&products_id=1491

[2] Arduino. 2010. *Arduino* [Online]. Available: http://www.arduino.cc/

[3] VIA Technologies. 2010. *ARTiGO* [Online]. Available: http://www.via.com.tw/en/products/embedded/artigo/a1000/index.jsp

[4] Surveying Engineering Department, Ferris State University. *Direct and Inverse Geodetic Problem* [Online]. Available: http://www.ferris.edu/faculty/burtchr/sure452/notes/direct-inverse.pdf

[5] u-blox AG. 2010. *u-blox 5 GPS modules* [Online]. Available: http://www.u-blox.com/en/gps-modules/pvt-modules/previous-generations.html

[6] 3D Robotics LLC. 2010. *ArduIMU+ V2 (Flat)* [Online]. Available: http://store.diydrones.com/ArduIMU_V2_Flat_p/kt-arduimu-20.htm

[7] Google Project Hosting. 2010. *ArduIMU+ V2: Arduino based IMU & AHRS* [Online]. Available: http://code.google.com/p/ardu-imu/downloads/list