VCU SCHOOL OF ENGINEERING
Virginia Commonwealth University

Improvements to an Autonomous Unmanned Aerial Reconnaissance System

Andrew Bourassa, Marcus Jones, James Roberson, Matthew Ryan, and Andrew Spear
Faculty Advisor: Dr. Robert Klenke

June 1, 2006

Electrical and Computer Engineering Dept.
601 W. Main St.
Richmond, VA 23284

Abstract – *Previously the VCU Skyline teams have built and improved an autonomous aerial vehicle system from the ground up. This system included an onboard flight computer, all the sensors necessary to monitor flight, as well as a ground control station Windows application. With the basic system in place, over the past year, the program has expanded and more VCU students have sought to improve and develop this system even further. These improvements include implementing the autonomous vehicle system on a rotary-wing aircraft and a ground vehicle. Additionally, electronics on the fixed-wing aircraft have been upgraded to provide more functionality, sensing capabilities, and smaller size and lower weight in the form of an Angular Position Measurement System. With this ongoing innovation and development, the VCU autonomous vehicle system will be capable of coordinated operation between fixed and rotary winged aircraft in the air, as well as vehicles on the ground.*

TABLE OF CONTENTS

1. INTRODUCTION.....2

A. PROJECT GOALS 2
B. BACKGROUND 2

2. FLIGHT CONTROL SYSTEM 2

A. THE VEHICLES 2
B. HARDWARE 3
C. FLIGHT CONTROL SOFTWARE 4

3. GROUND CONTROL STATION.....6

4. ANGULAR POSITION MEASUREMENT SYSTEM.....6

A. HARDWARE 7
B. SOFTWARE 7
C. SYSTEM DESIGN 8
D. TESTING 10

5. CONCLUSIONS 10

1. INTRODUCTION

Three groups of Senior Electrical and Computer Engineering students from Virginia Commonwealth University, for senior design projects, have set out to improve and add-on to the VCU UAV Skyline system. The three projects include, a UAV – Helicopter, an Unmanned Ground Vehicle (UGV) and an Angular Position Measurements System (AMS). This paper details the three sub-projects of these groups that have come together to compete at the 4th Annual AUVSI Student UAV Competition.

A. Project Goals

The goal of the UAV Helicopter team was to create an autonomous rotary wing vehicle based on the existing VCU UAV system for surveillance purposes. The helicopter should have the ability to receive GPS latitude and longitude, a compass heading, and an altitude. A ground control station is also implemented to control the aircraft as well as collect and display data in a user-friendly manner. The helicopter is to be able to take a target latitude, longitude, compass bearing, and altitude, received from ground control station input, then autonomously travel to the target area and hover as it waits for the next desired input from the ground control station.

Similarly, the unmanned ground vehicle's goal was to successfully travel to waypoints assigned by a control station, while avoiding obstacles in its path. Additionally, the ground vehicle will have an onboard wireless video system with pan and tilt functionality to provide reconnaissance throughout the mission.

The goal of the angular position measurement system was to maintain a reasonable estimate of the angular orientation of an aircraft in flight. This system was designed at VCU in order to avoid the high cost and poor performance of commercial systems with similar functionality.

B. Background

For the 3rd AUVSI Student Competition, the VCU Skyline Team designed an Autonomous Unmanned Aerial Reconnaissance Vehicle and Ground Control Station. Previously the VCU autonomous vehicle system used an Atmel FPSLIC microcontroller and Trimble Lassen-LP GPS unit. Over the past year several improvements have been made to the systems hardware. This year, a smaller, more robust Suzaku-S microcontroller and a faster U-Blox GPS unit are being used. These hardware upgrades are outlined further in subsequent sections of this paper.

The previous UAV system was capable of supporting multiple UAVs as well as flexible mission planning. The VCU UAV was able to fly to a given set of waypoints capture still images and video at these waypoints and transfer them back to the ground station for analysis.

The VCU UAV system works on a client-server model. The client-server model allows for a great deal of expandability and flexibility. The server is the central hub of the system as it communicates with the vehicle client application. The UAV helicopter and UGV are each equipped with an autonomous Flight/Vehicle Control System and paired with a Ground Control Station that displays data collected from the on-board sensors and communicates mission parameters.

2. FLIGHT CONTROL SYSTEM

A. The Vehicles

The AUVSI competition aircraft are based on surplus FQM-117B "Mig-27" Army target drones. The frames of these aircraft are made of foam. This makes them light and sturdy but with enough room for on-board electronics. A Saito 1.25 CI four-stroke glow fuel engine powers one of the Migs and a Zenoah 20 cc two-stroke gas engine powers the other. Equipped with 24 oz. tanks, the aircraft are capable of approximately 30 minutes of flight. A larger wing is used to increase payload and decrease stall speed. Examples of the VCU Mig aircraft are shown in Figure 1.

Body Material:	Foam
Wingspan:	66" stock, 85" expanded
Fuselage Length:	68"
Engines:	1.2 ci 4 stroke glow and 20 cc gas
Fuel:	15% Nitro Glow Fuel or Gasoline
Prop Size:	15" x 6"

Figure 1: FQM-117B with stock wing

The rotary-wing aircraft used is a Miniature Aircraft X-Cell Ion-X Electric Helicopter shown in Figure 1. The full frame is made from graphite. This makes for a study frame and fairly lightweight for its size. An undercarriage compartment was fabricated to enclose the on-board electronics. The helicopter makes use of two Li-Polymer 8000 mAh/ 18.5V batteries to power the electric motor. The two batteries allow for a flight time of approximately 20 minutes.

Body Material:	Graphite
Main Rotor Diameter:	1640 mm max
Tail Rotor Diameter:	262 mm
Overall Length:	1346 mm
Overall Height:	432 mm
Weight:	~10.5 lbs w/o undercarriage

Figure 2: Miniature Aircraft X-Cell Ion-X Helicopter

A Traxxas E-Maxx "off-the-shelf" electric remote controlled truck is the chassis of the UGV. This vehicle provides excellent suspension, handling, four-wheel drive, torque and power necessary to navigate fairly rough terrain. This vehicle's use can be further justified by its present-day service with the US Armed Forces in Operation Iraqi Freedom. A box mounted on top of the vehicle was fabricated to house the on board electronics. The vehicle's powertrain uses twin 14V Titan 550 Electric motors powered by two 7.2V 3000mAh batteries. This gives the vehicle a run time of about 30 minutes depending on speed of operation.

Overall Length:	486 mm
Overall Height:	267 mm
Wheelbase:	305 mm
Drive System:	Shaft Driven 4WD
Chassis Type:	Molded composite nylon tub
Top Speed:	~23 mph

Figure 3: VCU Ground Vehicle in the field

B. Hardware

The main on-board computer system used by the UAV Helicopter and UGV is the Suzaku-S Microcontroller. The on-board sensors, GPS and wireless modems are interfaced to the Suzaku in order to perform data collection and autonomous flight. Each component of the FCS is explained in more detail below.

Suzaku-S Microcontroller

The Atmark-Techno Suzaku Microcontroller uses a Xilinix Spartan-3 FPGA with an embedded Microblaze Softcore processor. This allows software to be developed for the 32-bit processor and VHDL hardware descriptions to handle time sensitive tasks. Applications are interfaced using predefined and custom logic blocks or "cores." The Suzaku runs a version of the Linux operating system called μCLinux or Micro-Linux. This allows applications to be written in C versus assembly language. The development environment that was used for the FPGA is the Xilinx Embedded Development Kit v7.1 and GCC was used to compile the C software.

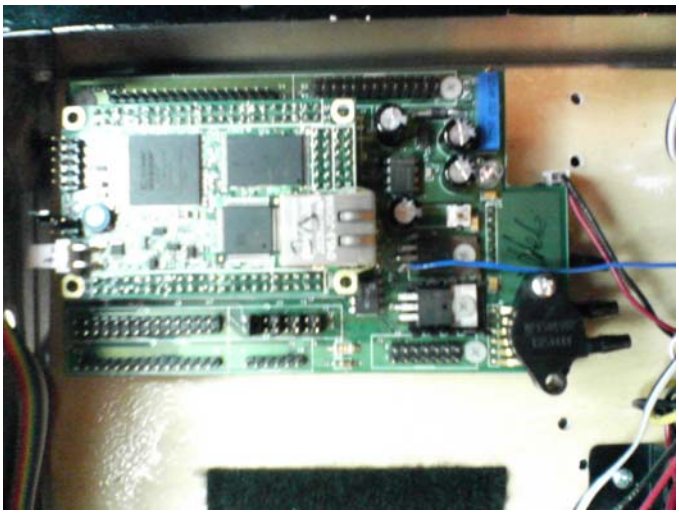

Figure 4: Suzaku Microcontroller mounted on Suzaku-EX board with Barometric pressure sensor visible

Devantech Compass

The Devantech CMPS03 compass is a two-axis magnetometer. This device serves as the main heading sensor. The compass is connected to the Suzaku via the industry standard Philips I^2C (Inter Integrated Circuit) Bus. The FCS of the helicopter and VCS of the UGV read this sensor at a rate of 4 Hz.

Barometric Pressure Sensors

The barometric pressure sensors serve as the main altitude sensors. They are connected to the Suzaku through the Analog-Digital Converter (ADC). The ADC is then connected to the Serial Peripheral Interface (SPI) Bus on the microcontroller. The FCS of both the Mig and helicopter reads these sensors at 4 Hz. The UGV does not utilize the pressure sensors.

Wireless Modems

A pair of 900MHz Maxstream XStream wireless modems are used to communicate between the FCS and the Ground Control Station. Both modems are interfaced via serial port – one connected to the Suzaku microcontroller (or FCS) the other connected to the GCS. A data rate of 9600 bps is used to communicate between the GCS and FCS/VCS.

GPS Receiver

The u-Blox Antaris GPS Receiver is used as the main positioning sensor for the air vehicles. In addition to providing three-dimensional global positioning (latitude, longitude and altitude), it also provides current velocities while the helicopter is in motion. The u-Blox is connected to the Suzaku through another RS-232 serial port. The GPS antenna is mounted down the tail-boom of the helicopter for maximum reception. The antenna for the plane is mounted towards the tail of aircraft on a small aluminum square that has been proven to increase reception.

The UGV uses the slower Trimble Lassen GPS receiver as its positioning sensor. The Trimble is connected to the Suzaku through an RS-232 serial port just as the u-Blox is for the air vehicles. The Trimble gives the same functionality as the u-Blox but at a slower rate of 1Hz.

Figure 5: UGV Vehicle Control System, w/ Trimble GPS, Maxstream Modem and Suzaku visible

C. Flight Control Software

The flight control software run on the Suzaku microcontroller is written in the C programming language. The software is compiled on a standard Linux box using the GCC Compiler. Since the Suzaku runs μCLinux, C was a natural choice of programming language because of its familiarity. The software runs three main threads. One thread handles I/O on the serial ports. This includes reading from the GPS and reading and writing to the GCS. The next thread is responsible for transferring the telemetry data of the system. The data from the on board sensors is read and sent down to the GCS in a VACS packet. The VACS protocol will further detailed shortly. The final thread is responsible for running the PID control loops.

VACS Protocol (VCU Aerial Communications Protocol)

The lowest level of the VACS communications standard is the serial packet format. Each VACS packet includes two sync bytes, a set of header fields, a variable length data payload, and a two-byte checksum.

Byte	Field
0	Sync1 Byte (0x76, 'v')

1	Sync2 Byte (0x63, ‘c’)
2	Destination Address
3	Source Address (address of the sending host)
4	Message ID (Low Byte) (Defines type of message)
5	Message ID (High Byte)
6	Length N (low byte) (Gives length, in bytes, of the data payload)
7	Length N (high byte)
…	Data payload (N bytes of data)
N+8	Checksum A
N+9	Checksum B

Table 1: VACS serial packet format

Checksum Calculation

The checksum is calculated using a 16-bit Fletcher algorithm, similar to the TCP protocol error detection scheme. The checksum calculation includes the data bytes as well as the destination address, source address, message ID, and length. It does not include the Sync bytes. The two bytes of the checksum, ChkA and ChkB, and calculated as follows:

```
ChkA = 0;
ChkB = 0;
// Array bytes contains the data to which
the
// check sum is being applied
for(int i=0; i<num_bytes; i++)
{
      ChkA = ChkA + bytes[i];
      ChkB = ChkB + ChkA;
}
```

It is important that this operation is done using 8-bit operations. If the operation is being implemented with larger numbers, the resulting checksum bytes should be logically AND’ed with 0xFF to clear all but the lower 8 bits.

Addresses

The current VACS protocol defines only an 8-bit address space, giving a total of 255 possible addresses. An implementation is free to assign addresses as desired, however the address 255 is recommended to be reserved for a broadcast address and address 0 to be reserved for a Ground Control Station, in such situations where there is a single controlling GCS.

Reserved Addresses

GCS – 0x0
Broadcast – 0xFF

Message Types

VACS defines the serial packet structure used to communicate, however the actual data that is communicated over the link depends on the implementation of the vehicle and ground station using it. A message vocabulary is defined as a set of messages (each with a unique message ID) with defined data payload formats. Each message type can be either a **report** type or a **command** type.

Reports

Report messages are sent by the UAV to give information about its status; they can be sent periodically or only when requested. A report message can be requested from a UAV by sending an empty (no data payload) message with the message ID of the report desired, addressed to the UAV.

Commands

Commands are sent by the ground station to a specific UAV or broadcasted to all UAVs on the channel (by addressing the message to the broadcast address). When a command is received, the receiving UAV should send back an acknowledgement. The acknowledgement message is an empty message with the message ID of the command being acknowledged.

Inter-vehicle Communications

The VACS system is built to support communication among vehicles in the air without the involvement of a GCS. This can be used in the test/implementation of cooperative mission algorithms, and may also prove useful for long-range communications using multi-hop networking. This issue needs to be looked at, and may require redesigning the command/report method. Or, it may not. It is possible for a plane to send a command addressed to another plane, or to request a report from another plane.

Special Messages

All vehicles using the VACS protocol should implement certain messages. For now, one message is required to identify what type of plane is transmitting. More special messages may be defined in the future.

Identify Message (0x00)

If a single ground station is going to support different vehicles (or two similar vehicles with different software versions), with potentially different message vocabularies, there must be a common way to identify what type of vehicle a message comes from. The VACS system provides two values to identify a vehicle and the types of messages it supports: The **vehicle type** and a **version**. Vehicle type is a 16-bit integer, version is an 8-bit integer (Vehicle type could potentially be represented as a string, and this is being considered). All vehicles should implement a common “identify” message,

with message ID 0. This message should be broadcast when the vehicle is turned on, and sent in response to a request.

Byte	Description
0	Vehicle type low byte
1	Vehicle type high byte
2	Version
…	Optional: Additional data bytes specific to an implementation

Table 2: Data payload format for identify message (ID: 0x0)

Ground Station Implementation

The communications system discussed here is intended for a system with one GCS controlling multiple UAVs. Some redesigning of the basic communication and data storage system in the GCS is required in order to support multiple UAVs of different types and to allow quick run-time configuration of the message protocols used by the planes. Two primary new features are planned for the GCS: The use of a "property tree" for storing all the system state information, and the definition of a generic interface for plane communications.

Property Tree

The property tree is a concept borrowed from the open source flight simulator FlightGear. A property tree is a collection of nodes arranged in tree form, with each node storing a value of one of several types, such as double, int, or string. A property location is represented by a string such as "/uav[0]/gps/longitude", where longitude is a child node of gps, which is a child node of uav[0]. The property tree allows multiple nodes of the same name to be referenced by index, as done with the node named 'uav'. Nodes do not have to be pre-defined, one code module can add a new node and it will become available immediately to others, however in practice some nodes (such as for longitude, latitude) will be standard for all planes.

The property tree (or branches of the tree) can be easily represented as XML documents for communication or storage. Logging of a mission can be achieved by writing out the entire tree, selected properties, or perhaps changes in properties to a file. Additionally, logging can be easily configured by a file that specifies what properties are to be logged and how frequently. As an example, suppose a new sensor is added to a plane: once the sensor value is assigned to a property (details on how this assignment can be configured will be presented further on) it can automatically be logged to a file or printed to a Windows form just by referencing it's property value; no GCS re-compile necessary

3. GROUND CONTROL STATION

The GCS system serves as the human control interface for the UAVs. As such, it must provide the user with two basic functions: control over the flight of the UAVs and easy access to data collected by the UAVs. A client/server model is used for the ground control station software. The central hub of the system is the GCS. The server communicates with the UAVs via point-to-point radio links and collects images from the planes camera via wireless video links. The client applications connect to the server over a network and communicate using the VACS protocol, transmitting information in the form of XML messages. Additionally, the server acts as a central database, maintaining mission data and saving it for later review.

The GCS for the helicopter, shown in Figure 6, can transmit and receive data from the helicopter during flight. The GCS receives compass yaw, latitude, longitude, ground speed, altitude, ground track and barometric altitude from the helicopter's onboard sensors. There are two gauges on the ground station used to show some of this information. The top gauge gives both the yaw and ground track to track the helicopters movement. The lower gauge acts as an altimeter, based on data from the barometric pressure sensors. The map display can load any map in the map inventory, and provides visual assistance in tracking the helicopter. The helicopter is represented on the map by a red triangle and is chased by a red line representing the helicopter's previous positions.

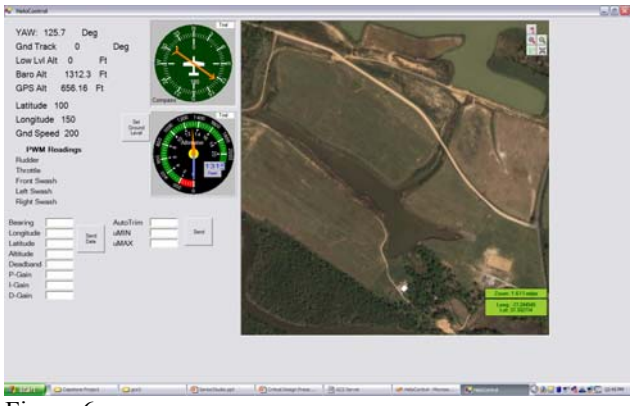

Figure 6

A similar GCS is used for the UGV. It has the capabilities to control the direction in which the UGV is traveling as well as monitor the on board video system. Additionally, it will display telemetry data sent from the Vehicle Control System on board the vehicle.

4. ANGULAR POSITION MEASUREMENT SYSTEM

The Angular Position Measurement System uses the output from two different sensors to calculate the angular position of the system in terms of pitch and roll. The angular position is calculated from a three axis accelerometer and two gyroscopes mounted perpendicularly, creating a two axis gyroscope. Optimization equations take the calculated angle

from each sensor and determine the best estimated angle depending on the current movement of the system. A microcontroller with a multiplexed seven-input analog-to-digital converter polls the sensors and maintains a data output rate of 20 hertz. The microcontroller outputs the digital value of each sensor to the user display on a PC using a RS-232 port (Figure 7). All angle calculations are performed on the PC, which are then shown in real time on the user display at 20 hertz.

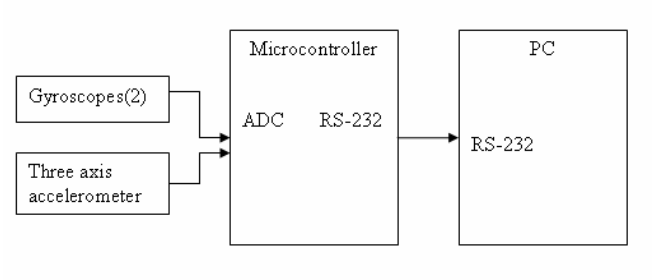

Figure 7

A. Hardware

The hardware for the system consists of three main components: the gyroscopes, accelerometer, and microcontroller (Figure 8).

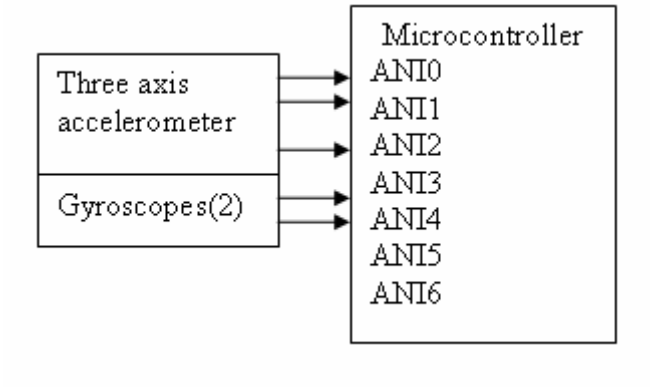

Figure 8

1. Gyroscopes

Each of the gyroscopes is part number ADXRS300 from Analog Devices. The range of these gyroscopes is 300°/sec with a sensitivity of 5 mV/°/sec. The gyroscopes generate a voltage proportional to rotational rate about an axis normal to the top surface of each chip. The output has a range between 0 volts and the input voltage, using a bias voltage near half the input voltage. Subtracting the bias voltage from the output provides the angular rate of the gyroscope, a positive result representing one direction of rotation and a negative result representing the other direction. A 3300 pF capacitor is placed between the gyroscope output and ground to reduce high frequency noise. Because of the nature of the gyroscopes, each has a unique bias voltage. Also, a 2000Ω resister is placed between two of the gyroscope pins to increase the sensitivity by decreasing the range of the gyroscope to approximately 180 °/sec. Since each resister has a slightly different value, the sensitivity of each gyroscope is also unique. Therefore each gyroscope's sensitivity (mV/°/sec) must be independently characterized for accurate angle calculation.

2. Accelerometer

The accelerometer is part number MMA7261Q from Freescale. The accelerometer generates voltages proportional to acceleration along three separate axes. The operation is similar to the gyroscopes, with a bias voltage about half of the input voltage. The range of the accelerometer is adjustable, but set at ±2.5g's. When gravity is the only force on the device, the tilt of the accelerometer can be calculated based on the distribution of the gravity vector among the three axes. However, this is less useful when the device is accelerating in any direction.

3. Microcontroller

The microcontroller used in the system is a NEC µPD78F9418A 8-bit microcontroller. This particular microcontroller is implemented because of the seven multiplexed analog-to-digital converter (ADC) inputs for the gyroscope and accelerometer rate outputs. The microcontroller's 16-bit timer is used to maintain a constant output rate of 20 hertz using the timer's compare interrupt. The UART of the microcontroller sends the packets of sensor data serially to the PC at 38400 baud.

B. Software

The system software is contained in two different places, the microcontroller and the PC. The microcontroller was programmed in C and contains the software to poll the sensors through the ADC and to send the sensor data to the PC at 20 hertz. The PC contains the software to convert the sensor data into angles, and the optimization equations to determine the best angle estimate and was programmed using C#. Also, the PC contains the user display that represents the angles in numerical form along with an artificial horizon display.

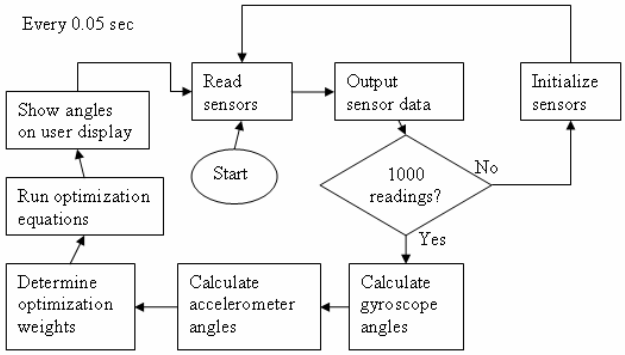

Figure 9

1. Microcontroller

Upon power-up, before any data is sent to the PC, the microcontroller's control registers are initialized. The UART registers are set to transmit only at 38400 baud, at 8 bits, with one stop bit and no parity bit or handshaking. The timer register is initialized to run at 156.3 kHz and the timer compare interrupt register is set at 0.05 seconds to control the output rate. After the timer and UART initialization, the microcontroller program is sent to an infinite loop. During the first interrupt, each ADC input is read and placed in the output buffer along with the packet number and the timer interval, or 0.05 seconds. The output buffer is sent to the PC through the UART using a VACS (VCU Aerial Communication Standard) packet. This interrupt service routine occurs at 20 hertz until the system is powered down.

2. PC

Before any angles can be calculated, the sensors' bias values have to be initialized. With the system level and static, 1000 samples are taken from each sensor axis and averaged. This value becomes the sensor bias value for the entirety of the program. Each time the program is restarted, this initialization is performed. During initialization, the user display informs the user that the system is initializing and the sensor values are shown, with the angles remaining at the initialized values of zero.

After initialization, the system runs in its regular loop. Upon receiving the data packet from the microcontroller every 0.05 seconds, the program subtracts the previously established bias value from the sensor value for each sensor reading. The bias-adjusted sensor values from the gyroscopes are divided by a characterization constant to translate the unitless sensor values into degrees per second. The degrees per second values are integrated using the timer interval from the microcontroller output packet and the previous iteration's optimized angle to obtain the gyroscope angles. The previous optimized angles are fed back into the gyroscope integration to remove accumulating integration error. After the gyroscope pitch and roll angles have been calculated, the bias adjusted sensor values from the accelerometers are used to calculate the accelerometer angles using trigonometric equations.

After the individual gyroscope and accelerometer angles have been calculated, the optimization weight is determined by comparing an approximate z axis value calculated from the accelerometer pitch and roll to the measured z axis value. If the difference between the approximate z axis and the measured z axis is too great, then the weight is shifted toward the gyroscope angles because the accelerometer is responding to acceleration rather than gravity. Otherwise, the weight is shifted towards the accelerometer angles, except when the previous angle was greater than 70 degrees.

Once the optimization weight is determined, the gyroscope and accelerometer angles are placed in the optimization equations to calculate the final pitch and roll angles. These pitch and roll angles are displayed on the user display along with the accelerometer angles and the sensor data values from the microcontroller (see Figure 9 for flow chart).

C. System Design

The system was designed as three separate subsystems: gyroscope angles, accelerometer angles, and optimization. The gyroscope and accelerometer subsystems were developed independently and in parallel, whereas the optimization's final design was limited until the two other subsystems had completed full development and testing. After the optimization was finalized, the complete system was assembled and tested. The subsystems are described in detail below.

1. Gyroscope Angles

Since the gyroscopes output angular rate, the output needs to be integrated to calculate the angular position. Before integration is performed, the digital gyroscope value must be converted into usable units, such as degrees per second. The degree per second conversion constant was calculated by characterizing the sensitivity of each gyroscope, because each gyroscope has different sensitivity due to the added resistor. The sensitivity was calculated by using a turntable with a known angular rate (Figure 10). The design of the turntable includes a dc motor to power the platform, an optical encoder to accurately determine the angular rate, and a Lazy Susan bearing to provide stability to the platform.

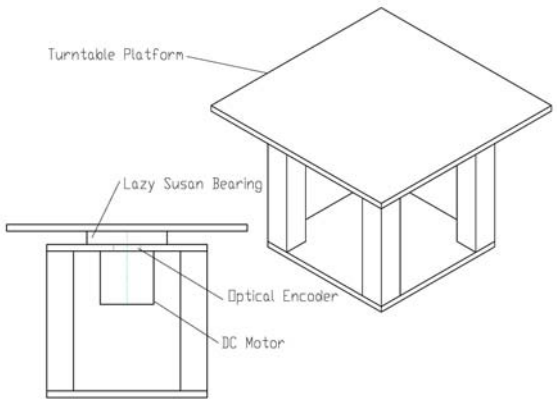

Figure 10

The angular rate of the turntable was calculated using an optical encoder and the edge triggered interrupt of the 16-bit timer on the NEC microcontroller. The optical encoder contains a photoresistor, led, and a round Mylar plate with 400 tick marks. Each time the led passes a tick mark, a pulse is generated on the encoder's output. Thus, given 400 pulses, the Mylar plate has completed a single rotation or between two tick marks represents 0.9 degrees. Each time the optical encoder generated a pulse, the microcontroller's timer would start and continue until another pulse was generated. The time

between pulses was then used to calculate the degrees per second of the turntable.

By dividing the characterization constants from the digital gyroscope value for the ADC, the degrees per second for each gyroscope are calculated. Using the degrees per second and the timer interval of the microcontroller, the gyroscope angles are integrated as shown in (1).

$$Gyro\_Angle = Gyro\_Angle + Gyro\_DPS * Timer\_Interval \quad (1)$$

2. Accelerometer Angles

Since the accelerometer senses only the gravity vector when at constant velocity, the degree of tilt of the system can be calculated based on the distribution of acceleration due to gravity on the three axes. The accelerometer angles can be determined using (2) and (3).

$$X\_Accel\_Angle = -\sin^{-1}\left(\frac{y_{axis}}{g}\right) \quad (2)$$

$$Y\_Accel\_Angle = \sin^{-1}\left(\frac{x_{axis}}{g}\right) \quad (3)$$

The opposite axis is used in the arcsine function because when one axis tilts, it affects the other's angle as shown in Figure 11. Once the angles are calculated, they just need to be converted from radians to degrees by multiplying the angles by 180/pi.

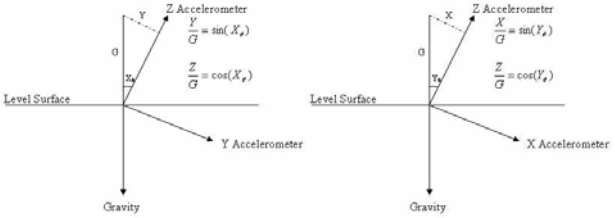

Figure 11

3. Optimization

The optimization equations directly take advantage of the strengths of each measurement device. The estimate of angular position is obtained from a weighted sum of the gyroscope measurement and the accelerometer measurement. The optimization equation is shown below in (4).

$$Opt\_Angle = Accel\_Angle(weight) + Gyro\_Angle(1 - weight) \quad (4)$$

The result of the optimization equation is fed back into the gyroscope calculation, so that the result of each iteration is added to the best estimate rather than to the sum of previous gyroscope iterations. This allows the accelerometer measurement to eliminate the error due to integration and noise in the gyroscope signal.

The measurement weights change according to the type of motion of the system. When the system is not under acceleration, the accelerometer provides a better measurement of angular position than the gyroscopes. Conversely, when the system is accelerating, the gyroscopes provide a better estimate than the accelerometer. Because of this effect, the gyroscopes will be given a higher measurement weight when the system is under acceleration, and the accelerometer will have more weight when the system is under constant velocity. The shifting of measurement weights is illustrated in Figure 12.

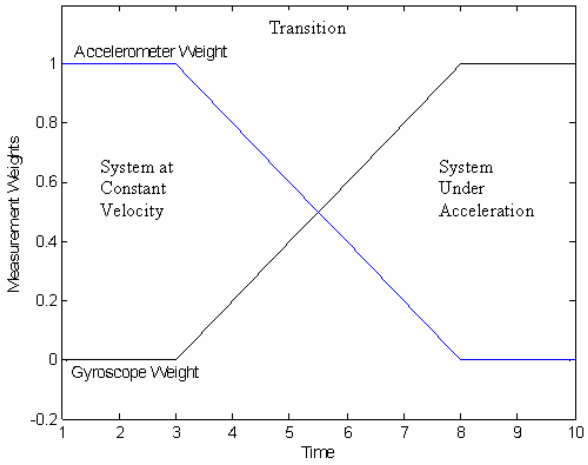

Figure 12

The accelerometer will be used to determine whether or not the system is under acceleration. This can be determined by calculating what the z axis should be based on the x and y accelerometer angles given that gravity accounts for all acceleration. Since the x and y axis are perpendicular planes, their acceleration vectors are multiplied. When multiplied by g and added to the z axis it will equal one g under constant velocity as shown in (5).

$$z + g\cos(|X\_Accel\_Angle|) * \cos(|Y\_Accel\_Angle|) = g \quad (5)$$

To determine if the system is under acceleration, the z axis is calculated under the assumption the system is under constant velocity and then compared the measured z axis to obtain z_{diff} as shown in (6).

$$z_{diff} = g(1 - \cos(|X\_Accel\_Angle|) * \cos(|Y\_Accel\_Angle|)) + z_{axis} \quad (6)$$

If z_{diff} is above some threshold (around 0.02g) then the system is assumed to be under acceleration, and the weight shifts toward the right-hand side of Figure 12. Also, when the system is upside down, the accelerometer angles cannot distinguish between up or down, so the weight cannot be increased to the accelerometer.

As the system state changes, the measurement weights must be changed gradually to prevent sharp changes in the estimation of angular position. Such changes in the output of the system could have negative effects on the control system using the measurement. The appropriate rate of change in the measurement weights has been determined that

a shift from full accelerometer to full gyroscope should take no less than one second.

D. Testing

Testing was completed in the similar order as the design. The gyroscope and accelerometer subsystems were tested in parallel, and after they were fully functional, the complete system was tested with the optimization equations combining the gyroscope and accelerometer angles.

1. Gyroscope Angles

The gyroscopes were tested using the 2-axis tilt table, microcontroller code, and output to a PC user display. The PC user display was used only to display real time angle measurements from the microcontroller. All the gyroscope angle calculations were performed on the microcontroller using methods described in the design. The calibration constants were fine tuned using the tilt table to ensure the angle shown on the user display was equal to the actual angle of the tilt table.

2. Accelerometer Angles

The accelerometer was tested using the same methods as the gyroscopes, but with different microcontroller code to implement the accelerometer equations and a modified user interface to reflect the accelerometer angles. During testing, the gravity constant was adjusted until the calculated accelerometer angles matched the actual tilt table angles. During testing, it was discovered the accelerometer angles are less accurate above 50 degrees and essentially worthless above 70 degrees. The optimization weight adjustment took this limitation into account. Both the Crossbow and Freescale accelerometers were tested in the subsystem and they both had comparable angle accuracy.

3. Optimization/Complete System

During testing it was discovered that while the microcontroller was capable of processing the gyroscope and accelerometer equation separately, when combined into a single program for optimization, the microcontroller could not handle the load. Therefore the system was tested using the microcontroller functions of analog-to-digital conversion and timing with all the calculations performed on the PC behind the user display. As testing proved, this was immediately effective as very little fine tuning was necessary. Similar to the gyroscope and accelerometer testing, the compete system with the optimization equations was testing on the tilt table, so the calculated angles could be compared to the actual angles. Subtracting 360 degrees after the system had made a complete rotation was the only real change made to the system after testing.

It should also be noted that the Crossbow accelerometer was mainly used for complete system testing due to the different input voltage requirements of the Freescale and Crossbow (3.3V and 5V respectively). As previous accelerometer testing proved, this was deemed acceptable because of the similarities between the two accelerometers.

5. CONCLUSIONS

In past years, the VCU Skyline team has done very well at the AUVSI competition. The basic system was initially setup to support multiple UAVs. The main goal of the Skyline team this year was to expand and improve the system from last year. The three projects that we described in this paper are to be part of a larger system that includes fixed-wing aircraft, rotary-wing aircraft as well as an unmanned ground vehicle. The Angular Measurement System improves upon the current system by providing more accurate attitude measurements. Other modifications, such as the new FCS platform and new GPS system have reduced the weight and size of the system from previous years. By adding these improvements, the VCU UAV System has become more robust and capable of accomplishing a wide array of tasks..