AR.UAV AutoFlight

Final report

tWP7

John Brock

David Dornier

Alex Perovich

Department of Computer Science and Engineering

Texas A&M University

May 7, 2012

Table of Contents

1 Executive summary

Large parking areas need regular monitoring, patrol, and enforcement to ensure thefts, break-ins, and parking violations are minimized. However, monitoring large parking lots is very expensive in terms of human resources and operational resources. Additionally, a large number of obstacles and environmental conditions only add to the challenge of monitoring parking areas. Parking areas are being larger, spread further apart, and parking space is only being reduced and thus the need for reliable, robust monitoring is crucial.

The tWP7, AR.UAV team decided the best approach to this pressing need was an unmanned, teleoperated quadcopter. The copter will be controlled via a Windows Phone cellular device through a wireless connection. The advantage of this arrangement was to allow operation of the aerial craft from any location with access to a network connection. Remote operation removes the human form being exposed to the environmental elements and adds to the safety of the operator and the pedestrians in the parking area under patrol. Windows Phone was chosen as the platform due to the performance of the operating system and the relative inexpensive cost of the phones due to the large subsidies granted by Microsoft in efforts to promote their new phone platform.

The quadcopter aerial vehicle was selected since it provides a very mobile, yet stable platform. The copter allows payload to easily be carried and transported which is crucial since we will be carrying a camera that provides live video streams back to the operator through the phone. The small form factor of the quadcopter allows the craft to navigate small areas and take the most direct routes possible between two points of interest. Also, the copter was very easily controllable and the susceptibility to winds is minimized since the design of the quadcopter includes a 4 propeller, horizontal propulsion mechanism. The copter was not the most durable platform that could be chosen but by using protective bumpers and lowered landing devices, the craft and any other items (vehicles) are protected in the event of a potential crash or malfunction.

Teleoperation is traditionally thought to include very fine-grained controls, but we have leveraged automated flight functionality and allow the user to simply select GPS locations / waypoints rather than require delicate, precise operation. Additionally, the design and primary idea behind the craft is that in operation it will have the ability to automatically traverse parking rows to patrol for potential parking violations. As the craft is traveling down the row of vehicles, the live video will be transmitted to the operator using the phone controller and the operator will be able to spot any parking infractions; the live video feed was not implemented this semester, but it is scheduled as future work. The operator does have the ability to maneuver the craft manually to gain a better perspective of the vehicle and license plate to proceed with the parking infraction ticket. Once the video steam is implemented, all video and photos taken will be saved to serve as documentation for future reference and legal documentation. A potential improvement that cannot be performed by the AR.Drone is independent camera controls, but different platforms could implement this additional functionality.

Currently, the operations of our solution allow the users to control the drone using static controls implemented with intuitive buttons – movement direction controls allow the user to move the drone forward, backward, strafe right and left, altitude adjustments, and automated takeoff and landing. Additionally, the user can press and hold the accelerometer control button and the movement of the drone is influenced by the physical movement of the phone – this method seems to be the preferred method of operation from observations of user during user trials. Additionally, our app allows the user to place waypoints on an interactive map and select either in-order or shortest path traversal; the drone automatically navigates between the specified waypoints allowing the user to take a hands-off approach when flying the drone between long distances or relocating / dispatching the drone to a new operational area. Flight settings are fully customizable to tune the drone's flying characteristics and documentation is available for the user to learn more about the specific components or entire solution implementation.

The design of the system was performed in a parallel but in a very coordinated and collaborative manner within our team. Team members took ownership of the phone development, craft assembly and programming, testing framework, and the navigational algorithms to ensure the system was completed in the most time effective manner while also ensuring the independent systems were able to interface together upon completion of each stage. We used multiple check-points and sub-tests to ensure our solution was converging to the objectives we set at the beginning of the project and subsequently modified as the project progressed. Technical issues were encountered, but momentum was maintained throughout the project. The timelines that were set proved to be good baselines and guidelines; however, we did lag on the deadlines, which may be due to overestimation of ability and underestimation of complexity. In the end, our team worked effectively together to accomplish the goals that we set together.

We are excited to have provided a system that can serve as a baseline, prototype solution to the pressing parking monitoring problems. The AR.UAV team was committed to quality and safety throughout the duration of this project and we hope all future developments maintain these pillars as top priorities.

2 Project background

Two general problems our team is addressing are teleoperation using a Windows Phone cellular device and the monitoring of large parking areas.

Smartphones are becoming much more prevalent by the day and they are being used to leverage technology in aspects that are improving efficiency, reducing cost, and connecting every individual on an even more personal level to the internet and, in turn, the rest of the world. Windows Phone is attempting to gain market share from Apple iOS and Android devices by providing an easy-to-use, interesting, and "different" user experience than the two other main competitors. Microsoft is encouraging development around their phone OS in a number of different ways – one of these ways is university relations and the desire to have teleoperation functionality implementations for common, valuable, and interesting applications of any variety.

Teleoperation is an ideal way to demonstrate the capabilities of a phone and provides a "coolness" factor that can potentially draw users and developers to the WP OS.

Remote monitoring, and monitoring in general, of large parking areas posses a number of challenges as a result of the manpower needed, inefficiency that results from constant patrolling, and the simple fact that the process is very monotonous, yet crucial. The specific problem we are attempting to address is monitoring the parking lots here at Texas A&M University for parking violations and also of providing versatile surveillance that can prevent theft and break-in. An unmanned, aerial vehicle is ideal for ease of use, fast deployment, and efficiency of operation. The aerial craft would ideally navigate between parking lots based on GPS coordinates and traverse parking lots row by row to determine if any vehicles were parked illegally and also to prevent / discourage any criminal activity in parking lots.

The two problems discussed above illustrate an obvious need in two distinct areas but also an interesting possibility of leveraging technology to solve each problem simultaneously. Our group will use our computer engineering backgrounds to combine software and hardware components into a system that effectively solves the parking lot monitoring problem by operating an aerial vehicle controlled via a Windows Phone device.

2.1 Needs statement

Human monitoring and the patrolling of parking lots as a means to prevent break-ins, theft, and parking infractions is very time consuming and inefficient. Surveillance cameras provide a limited view, are non-mobile, and large surveillance infrastructures are costly to implement. A solution to monitor parking lots and other large, open areas is needed to improve current efficiency, increase safety, and enforce regulations.

2.2 Goal and objectives

Goal

Teleoperation of an aerial vehicle will provide a means to remotely patrol and monitor large parking lots in an efficient, cost-effective manner.
The operator will be able to set the destination of the patrol craft using GPS-assisted controls, monitor the area under surveillance via live video stream from the craft to the phone, and record any items of concern or interest using video/photo functionality onboard the craft. The teleoperation component is important because monitoring can be preformed from virtually anywhere with network access and the operator can work from any environment that is comfortable and productive. Often, traffic can prevent patrols from reaching a destination in a timely manner, but an aerial vehicle could circumvent this obstacle with ease. The integrated solution of patrol vehicle and monitoring solution will revolutionize the process of patrolling parking lots and/or other large areas.

Objectives

- Remote operation of aerial vehicle using Windows Phone 7 device
- GPS assisted navigation and flight control
- Automatic flight of vehicle but user determines destination (higher level of teleoperation)
- Live video stream from aerial vehicle to the cellular phone
- Ability to capture images and/or video for documentation purposes
- Automatic navigation of parking rows to monitor for potential parking violations
- Simple aerial vehicle control to deviate from automatic flight path to obtain better views / photos
- Independent camera motion control to ease capture of video / photos and increase versatility
- Aerial vehicle is slated to be a “quad copter” – 4 bladed rotary aerial vehicle
- Network / device connectivity provided by WiFi, 3G, BlueTooth, or similar wireless RF communication methods (phone and aerial craft)
- Automated takeoff and landing functionality for safety
- Bumpers and impact-absorbing materials to prevent damage to the craft and/or vehicles
- Automated shutoff controls for vehicle in the event of loss of control
- Automated landing procedure in the event of loss of connection
- Operation of craft on days of “reasonable” weather conditions (low winds, no precipitation, and good visibility) – not an “all-weather” solution
- Easily controllable and usable by any operator between the age of 16 and 70 with little training
- Minimal system cost of less than $500 (excluding the phone) – goal is to be in the $350 range.

2.3 Design constraints and feasibility

Technical

Phone – the Windows Phone will be running a custom app developed using a combination approach with XNA, Silverlight and C#. Ease of learning, networking requirements and the need for a robust, reliable system lead to this being the best approach. Developing for an unfamiliar platform is difficult but the WP7 SDK and tools offered by Microsoft will help ensure the learning curve is minimized.

The phone will be connecting directly with the aerial vehicle and most of the processing will occur on the phone, therefore, it is imperative that the phone app be designed in a robust fashion. Flight controls, navigation, GPS calculations, camera operation, image/video processing, along with other functions will be implemented through the phone app. The Nokia phone is very new and should provide plenty of processing power to run our application without issues. (1.4GHz processor and 512mb RAM) We anticipate the connection establishment between phone and craft to be one of the main challenges in addition to the programming complexity of automated flight controls.

Aerial Craft – the base requirement of the quad copter is to be able to fly safely and transport the payload that is required – microcontrollers, camera, and wireless communication hardware. The quad copter will need to have automated flight stability functions and wireless communication capability; the craft will need to communicate with the Windows Phone and have the ability to transmit and receive flight commands in real-time. Additionally, real-time video needs to be transmitted from the onboard camera.

Physica*l*

The physical design constraints are primarily associated with the ability of the quad copter to maintain safe aerial flight and the ability to transport a relatively large payload of electronics and camera hardware. Also, the craft must be designed in a way to ensure any damage from a potential crash is minimized not only to the craft, but also to any surrounding items, such as vehicles. Additional weight of any type can throw off the balance of the craft and the lift of the craft becomes much more difficult with more hardware components. Large motors and large diameter propellers are being researched to ensure we have sufficient lift and flight capabilities.

The environment will provide a number of challenges because the wind and other uncontrollable factors play huge roles on the operation of the quad copter, or any type aerial vehicle. The quad copter we are designing is not intended to operate in “non-ideal” conditions due to the challenges of severe weather conditions, but the design can be refined in the future to compensate for wind and other environmental conditions. The copter will be have self-adjusting compensation properties but these will be pushed to the extreme is we subject the copter to winds greater than the usual slight breeze.

Economica*l*

The primary cost associated with the quad copter project is the copter, itself. Durable parts for construction along with the hardware that can support autonomous flight are quite expensive. The goal of the project is to build the entire system with a $500 budget. We believe this is possible, but it will require creativity, selective purchasing, research, and maximizing the efficiency of the parts we are able to afford. The Windows Phone by Nokia was provided for use free of charge, and the development of the app can be accomplished on a free basis since we have access to MSFT tools and we unlocked the phone to allow custom app installation. We are very conscious of our budget and will work to minimize all costs.

Temporal

The project has a large number of components and there are quite a few challenges with the building and assembly of quad copter due to the numerous components required for autonomous flight. Combined with the complexity of the self-assisted flight, establishing a reliable wireless connection between the copter and the phone could prove to be an interesting challenge. Additionally, there is quite a bit of programming complexity required to control and auto-navigate the copter using the phone and also to have the phone application provide the functions that we have outlined in the previous sections.

This project will be a semester-long project but the timetable is fairly aggressive based on the scope of the project and the time each of us has available due to other courses and extracurricular involvement. However, we are confident we will meet the deadlines given as a result of effective planning, time management, teamwork, and the application of parallel task competition.

2.4 Literature and technical survey

- **Parrot AR.Drone** -- (http://ardrone.parrotshopping.com)

$300

The Parrot AR.Drone is the first ever quad copter piloted by a Smartphone. The AR.Drone was built to be controlled by an iPhone, but since its popularity increased the AR.Drone now works on Android devices too. The AR.Drone is marketed as a Video Game where the players or pilots get to control the quad copter through environments indoors and outdoors. Parrot has also created Apps such as AR.Race for multiplayer racing, AR.FlyingAce for multiplayer aerial dogfighting, and AR.Pursuit for hot pursuits like a cops and robbers game. The AR.Drone creates its own Wi-Fi network that can be controlled up to 50 meters. The flight time of the AR.Drone is about 12 minutes on the standard battery supply. The AR.Drone weights about 380g (0.9 lbs.) and is about 20" across.

This was chosen as the platform for out project due to the copter being pre assembled and easily extendable.

- **ELEV-8 Quadcopter** -- (www.parallax.com)

$600

The ELEV-8 Quadcopter is an outdoor flying robotic platform lifted and propelled by four fixed rotors. ELEV-8 uses a HoverFly board with a Propeller multicore microprocessor to electronically control stabilization. The ELEV-8 is relatively light weight (2.5 lbs.) with a size span of about 19” from one motor to another. The lift and propulsion is controlled by 10” propellers, 25A ESCs, and 860kv motors to give a lot of power and stability. The system does not include a camera but there is enough room for a camera mount on the underside of the central frame.

- **Gaui330X-S Quad-Flyer** (http://www.wowhobbies.com/gaui-330x-s-quad-flyer.aspx)

$400+

The 330X-S Quad-Flyer is on the smaller end of quadcopters measuring only 13” across and weighing only 400g so it is quite suitable for indoor flight. It has 8” propellers with 1100kv motors and 10A ESCs. The flight time is only 7-20 minutes depending on battery, payload and flight conditions. The frame uses a collapsible design to absorb minor crashes. The 330X-S does not come with GPS or a camera but does have the payload capacity for these items.

- **Eye Droid 4 Rotor QuadCopter**
  (http://www.robotshop.com/eye-droid-4-rotor-quadcopter-uav-camera-platform.html)

$8500

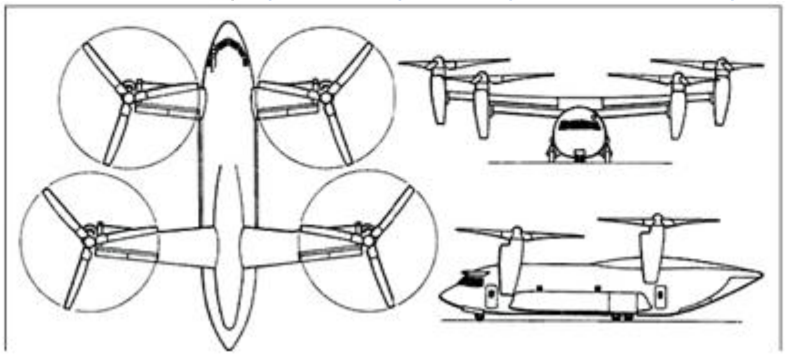
The Eye Droid is unique in that it allows the user/operator to wear FPV (First Person Video) goggles with a live video feed. The Eye Droid uses an 8-channel Radio Controller to operate. The Eye Droid was designed to try to minimize the possibility of breakages with graphite fiber filled composite engine mounts, polycarbonate gear rail brackets, vibration absorption pads, and a polycarbonate dome to protect the electronic systems. The Eye Droid uses a HoverflyPro for an expandable, Plug-n-fly flight control system. The HoverflyPro supports Eye Droid's HoverflyGPS that allows GPS navigation, a GPS position holding function, and a GPS return to home function. The HoverflyGPS system also records its readings allowing you to track your flight and retrace your steps home. The Eye Droid also houses a PhotoHigher Series of Aerial Camera mounts that are securely mounted to the airframe with vibration isolator grommets and are gyro stabilized in pitch and roll to ensure a steady camera feed. The video system is a Fat Shark FPV system that connects to the Fat Shark Goggles for a unique flying experience.

- **Phoenix Family by TiaLinx, Inc. –** (http://www.tialinx.com/phoenixfamily.html)

$8 mil.

The Phoenix Family of UAVs has a multitude of functions to perform. Through integration with other products by TiaLinx, the Phoenix Quadcopters are being tested by the Army in the AEWE (Army Expeditionary Warrior Experiment). The AEWE is using the UAV to land on buildings and detect objects inside. The soldiers found this as a very handy recon tool while hiding in the shadows during military operations in urbanized terrain.

- **Bell Boeing Quad TiltRotor –** (http://en.wikipedia.org/wiki/Bell_Boeing_Quad_TiltRotor)



Unknown $$

The Bell Boeing Quad TiltRotar (QTR) was a proposed four-rotor derivative of the V-22 Osprey tiltrotor. The cargo capacity was expected to be equivalent to the C-130 Hercules, but the QTR would be able to land vertically at unimproved sites like a helicopter. The Bell Boeing team disclosed the QTR design in 1999, but development was not pursued by the US DoD (Department of Defense). Later in 2005, the Joint Heavy Lift program started the hunt for more research and development of the QTR systems. The design of the QTR would join together that of the V-22 Osprey and the CH-47 Chinook. The fuselage would have a 747-inch-long cargo bay with rear loading ramp to carry 110 paratroopers, 150 standard-seating passengers, or 8 of the 463L pallets. Currently, the QTR is still in the design and testing phase for the US Air Force/Army Joint Future Theater Lift program.

2.5 Evaluation of alternative solutions

We had over thirteen different ideas that we came up with. After meeting and consolidating our ideas we ended up categorizing the ideas into five major categories:

· Aerial Vehicle
· Land Vehicle
· Home Management
· Cooperative Game
· Remote Connection

Vehicles are self explanatory. The Home Management was controlling the various aspects of your house with your phone, such as windows, door locks, temperature, lights, and many others. A cooperative game would be some game with RC devices involved. The remote connection was controlling power point presentation or similar application via phone. We decided that in order to be original a vehicle would need to be aerial rather than land based. We eliminated the game as having not enough to do with teleoperation. We then assessed the remaining three over different aspects as in the table on the following page.

Area	Remote Connection	Aerial Vehicle	Home Management
Hardware Complexity	3	1	2
Software Difficulty	1	2	3
Quality of Demo	Easy Demo Hard to show a purpose	Impressive	Demo would only show proof of concept nothing concrete
Overall Difficulty	3	1	2
Probability of Failure	3	1	2
What is Fun	3	1	2
What is best real world system	2	1	2

After going over the list we removed the Remote Connection because it lacked a sufficient hardware component. We decided to go with an aerial vehicle because the home management demo would be little more than a proof of concept.

3 Final design

3.1 System description

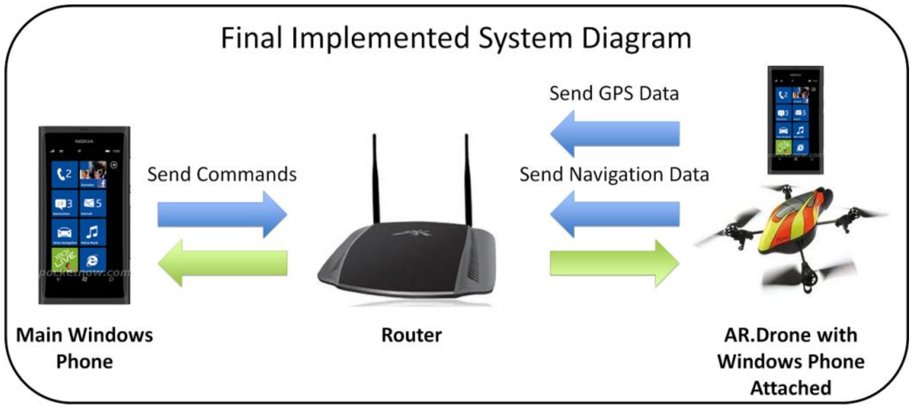


Figure A: High Level System Diagram

Figure A shows an overview of the implemented AR.UAV AutoFlight System. The System uses a Windows Phone that connects via WiFi to the Wireless Router which communicates with the Access Point of the AR.Drone. This is necessary because most Windows Phones do not support the Ad Hoc connection that the AR.Drone supplies. The drone will be controlled by an application running on Silverlight for Windows Phone 7.1. Communication will be through a UDP Protocol on Ports 5554 (Navigation data), 5555 (Video data), and 5556 (AT Commands/Controls). Controls are sent from the phone to the drone while the navigation data, and video stream are sent from the drone to the phone. Control input will be acquired through the phone by using the accelerometer or on screen controls.

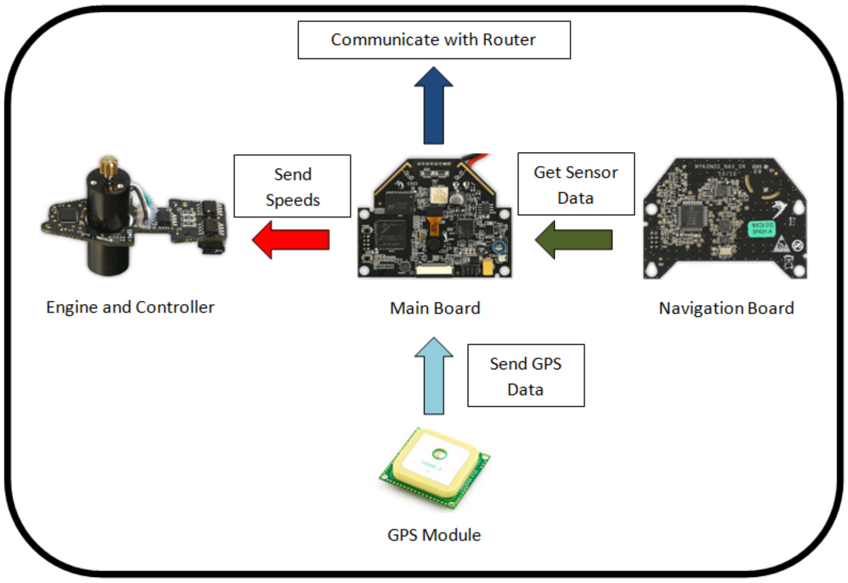
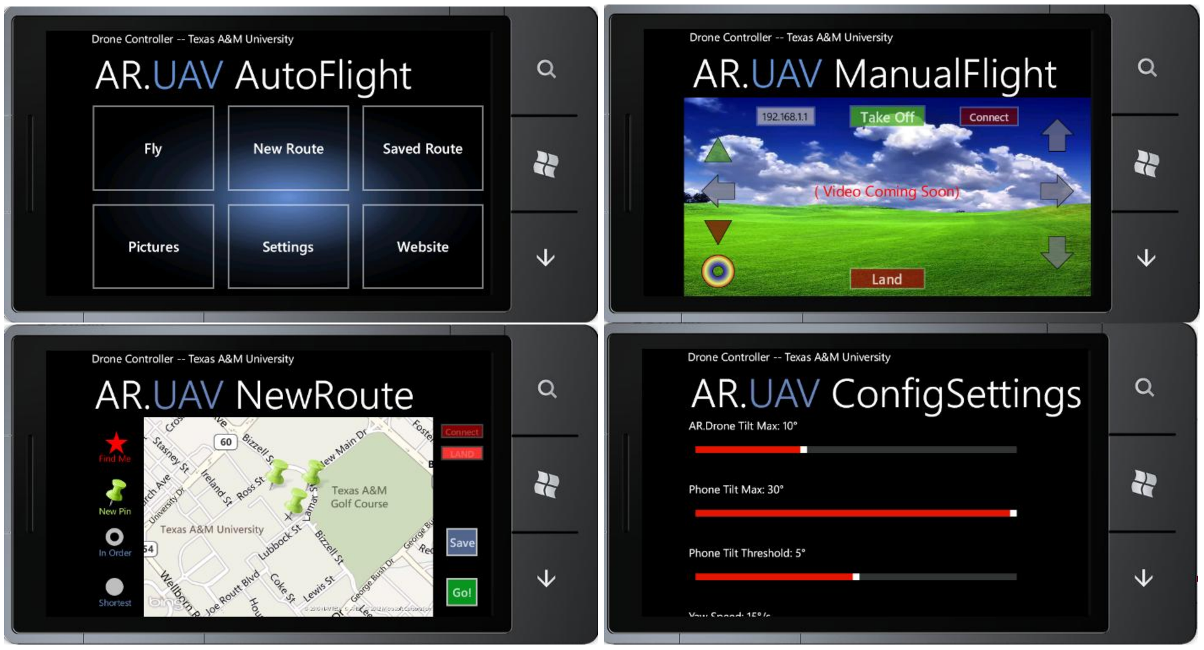


Figure B: AR.Drone Systems Communication

Figure B shows the communication flow of the interior systems of the Drone. The Drone is powered by a Lithium Polymer Battery connected to the Main-board. The Main-board houses an ARM CPU, a video accelerator, and the WiFi chip. The Navigation board constantly sends sensor data from the accelerometer, gyroscopes, and ultrasonic range finders to the Main-board. On the Main-board, the sensor data and control data is integrated to calculate movement commands that are then sent to the Engine Controllers to set the speed of the propellers. The ideal implementation would use a GPS module, but needs extra firmware adjustments. In that case, the GPS module would be attached to the Serial Port on the Main-board to send GPS data through the drone's WiFi connection for automated flight.



AR.Drone Tilt Max: 10°

Phone Tilt Max: 30°

Phone Tilt Threshold: 5°

Figure C: Graphical User Interface for Windows Phone AR.UAV AutoFlight Application

Figure C shows our GUI for the Windows Phone application named AR.UAV AutoFlight. The main page contains buttons to navigate to the manual flight page, new route page, saved route page, camera roll, settings, and our website. Since we did not get video feed capabilities working, the camera roll page was not implemented. Our website page navigates to http://students.cse.tamu.edu/jabam89/wp7drone/.

The GUI for the Manual Flight Page has static button controls as well as accelerometer capability. There is a text box for inputting the IP address of the drone, and a connect button to initialize the connection to the drone. The take off and land buttons do a controlled take off and landing command. The static controls are done by using the grey and colored arrow buttons. The left and right grey buttons control drone yaw (rotation around vertical axis) while the up and down grey buttons control pitch (rotation forward and backward around horizontal axis). The green up and red down arrows control altitude. The accelerometer controls are activated by holding the rainbow colored circle in the lower left hand corner. Unfortunately, there is no video feed currently working with our application, but a placeholder has been put into the background of the manual flight page.

The GUI for the Automated Flight Page (“NewRoute”) has buttons for centering the map (“Find Me”), adding a new way point (“New Pin”), and choosing the traversal method (“In Order” and “Shortest”). Just like the manual flight page, there is a connect button to connect to the drone as well as a land button if you feel the drone is in trouble. The route you create can then be saved by clicking the “Save” button or you can start the automated flight by pressing the “Go!” button.

The GUI for the Settings Page has multiple sliders for adjusting the sensitivity of the controls in manual flight. AR.Drone Tilt Max controls the pitch and roll of the drone which in turn controls the speed the drone will change directions. The Phone Tilt Threshold and Max control the accelerometer sensitivity of the phone. Other sensitivity settings include Yaw Speed, Altitude Speed, and Maximum Altitude Threshold.

3.2 Complete module-wise specifications

Phone Application

Developement and testing done on a Nokia Lumia 800 Phone running windows phone OS 7.1. Capabilities used: accelerometer, wi-fi networking.

The code will be split into three separate components, the UI, GPS movement and path analyzer, and the controller.

The controller UI will have two main pages. The main page will contain connection setup and control configuration to allow users to setup custom control schemes. The control page will contain the either a live video feed from the copter or a Bing map display with waypoints overlaid as well as various on screen buttons for copter control.

The GPS movement and path analyzer will take GPS data from the controller and plot paths from the waypoints. It will then send movement commands back through the controller. The copter will follow paths that are simply connecting the dots of waypoints with straight lines. We can determine the copter heading and position from the GPS readings and then direct it to the waypoints in sequence with appropriate commands.

The controller assembly will contain all classes related to constructing the commands, sending them to the copter, and decoding the response data.

The main DroneController class will contain the interface between the UI pages and the controls. It will contain functions to connect, disconnect, takeoff, land, emergency reset, and control movement. It will contain events to signal when new video frames are available, when new navigation data is available, when GPS data is available, and when new debugging messages are received. This class will also initialize threads to receive the navigation data, video feed, and GPS data. Nested classes will contain the implementation details for constructing commands and communicating with the drone.

The DroneControllerConfiguration class will serve as a helper class to the DroneController by containing information related to the various parameters related to the controller.

The VideoImage class will contain the code for decoding the video stream. New data from the stream will be added via a function and an event will signal a finished video frame.

The NetworkHelper class will be a static class encapsulating the code related to sockets. It will contain functions to initialize new sockets as well as send and receive data on open sockets. It will implement fixed timeouts on requests to prevent deadlocking.

The NavigationDataInfo class will contain all the code to process the navigation data. I will contain nested structures defining the layout of the navigation data in the stream. Data will be placed into the queue for decoding by a function and the decoded drone status will be accessed via a property. This class will also do checksum verification on the received navigation data.

A StopWatch class will be created to encapsulate the common timing functionality needed for the other classes.

All the AT command formatting will be encapsulated into an ATcommands class that will contain format strings for all the different commands, and functions to create strings for each command.

Enumerations will be created for the Connection status, the Led animation types, the drone status flags, the notification source for trace messages, the Notification level for trace messages, and the different video channels.

Hardware Components

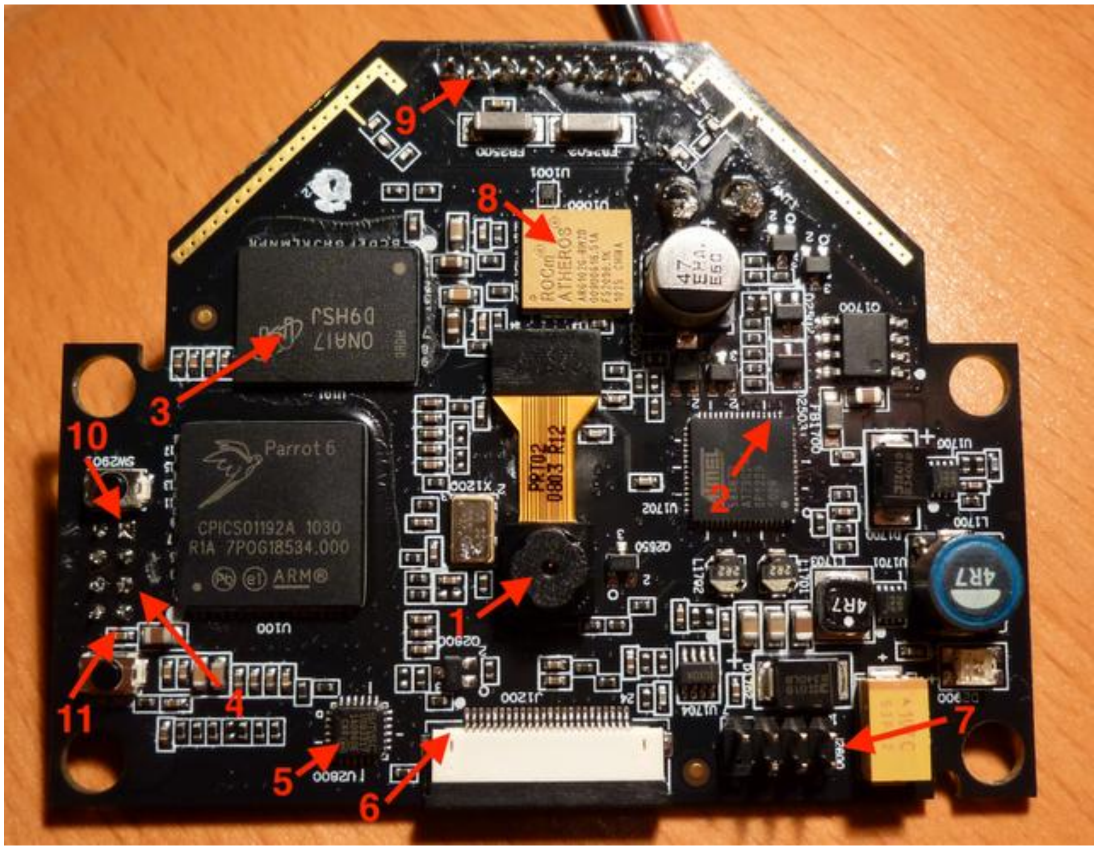
Router: Ubiquiti PowerAP N Router
The Ubiquiti PowerAP N is an indoor ultra-long range 802.11n WiFi access point capable of connection 100 meters away at 300 Mbps speeds. PowerAP N, Features 5 x 10/100 BASE-TX (Cat 5, RJ-45) Ethernet Ports and two external antenna connectors (dual omni-directional antenna include). Power AP N, utilizes version 5 of Ubiquiti's AirOS build upon the market leading intuitive user-interface loaded with advanced wireless configuration and routing functionally.

We are using the router because the Windows Phone can not directly connect to the AR.Drone. The Windows Phone does not support Ad Hoc connections so to get around this we have connected the Windows Phone to the router and forced the router to connect to the Drone. The router also helps to stabilize the signal and to extend the transmission range of the WiFi signal from 50m (Drone WiFi range) to 300m (Router WiFi Range).

Drone: Parrot AR.Drone 1.0
The AR.Drone is built using PA 66 high grade plastic with 4 crossed carbon fiber tubes to make the structure lightweight while not compromising on integrity. The outer hull is made of EPP (Extended Poly Propylen) to keep it lightweight and able to take a few bumps without sustaining major damage. The high-efficiency propellers are designed to optimize power consumption and thrust, and they are attached to the motors with high grade plastic gears and stainless steel shafts. Drone has a 1000 mAh, 3-cell Lithium Polymer (Li-Po) Battery with a Flight time of roughly 12 minutes and a charge time of 90 minutes. The propulsion system contains 4 - 15W Electric Brushless Motors that operate from 10,350 RPMs to 41,400 RPMs which corresponds to about 3,500 RPMs at the propellers.

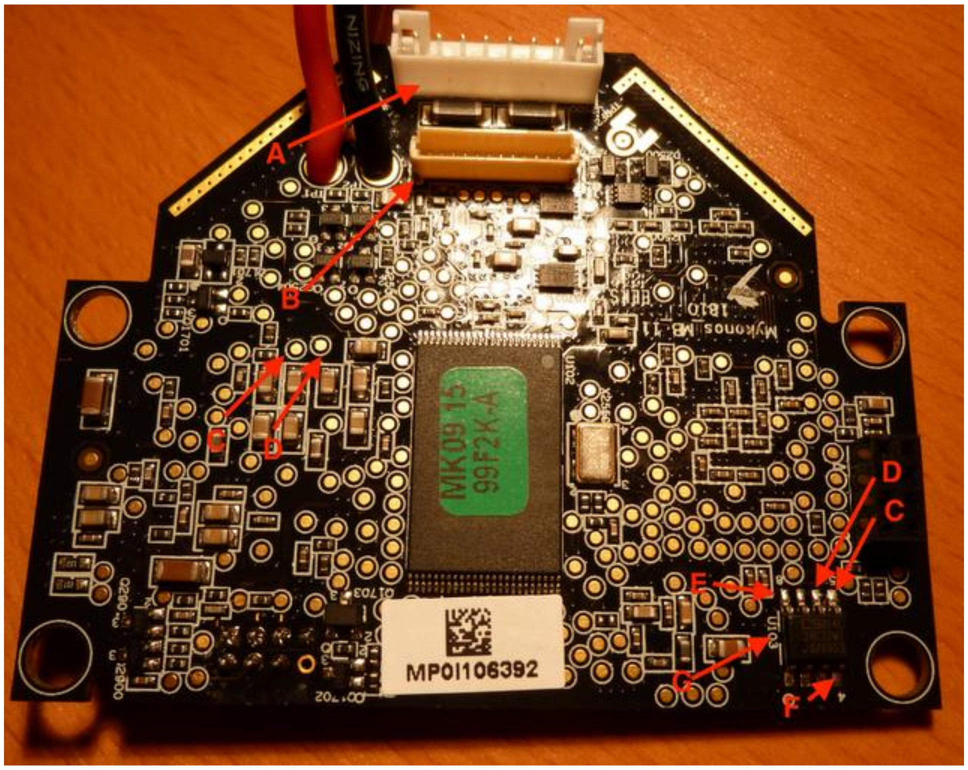
The Drone has many external systems for stabilization and control. The Drone has 2 cameras: Front and bottom. The front camera is a VGA (640 x 480) CMOS camera with a 93-degree wide angle lens able to supply a 15 FPS video feed. The bottom camera is a QCIF (176 x 144) CMOS high speed camera with a 64-degree diagonal lens able to supply a 60 FPS video feed. The bottom camera is used for horizontal stabilization by taking pictures during flight and using those pictures to maintain a stable location while there is no input. The vertical stabilization is done by the altimeter, which is comprised of two ultrasonic range finders that are accurate up to 6m and use an emission frequency of 40 kHz.

The Flight Control System of the Drone uses an internal guidance system and an embedded computer. The Guidance system has a MEMS (Micro Electro Mechanical System), which includes a Triple-Axis Accelerometer, a Bi-Axis Gyroscope, and a Single-Axis Yaw Precision Gyroscope. The embedded computer is an ARM9 RISC 32-bit 468 MHz processor with 128 MB of DDR RAM, WiFi b/g, a Linux OS kernal, and a USB socket.



Mainboard (Top-View)

1: Camera, I2C bus 0 address 5d
2: I2C connectors for I2C Bus 0 arrive here
3: DRAM cache
4: Connector to navigation board, at least one serial port
5: USB driver SMSC USB3317 (datasheet)
6: Connector to front camera, probably not an I2C device
7: External connection port with TTYS0 serial port and USB port (see link)
8: ROCm Atheros AR6102G-BM2D wireless
9: Power supply for the engines
10: Ground
11: VCC +5V



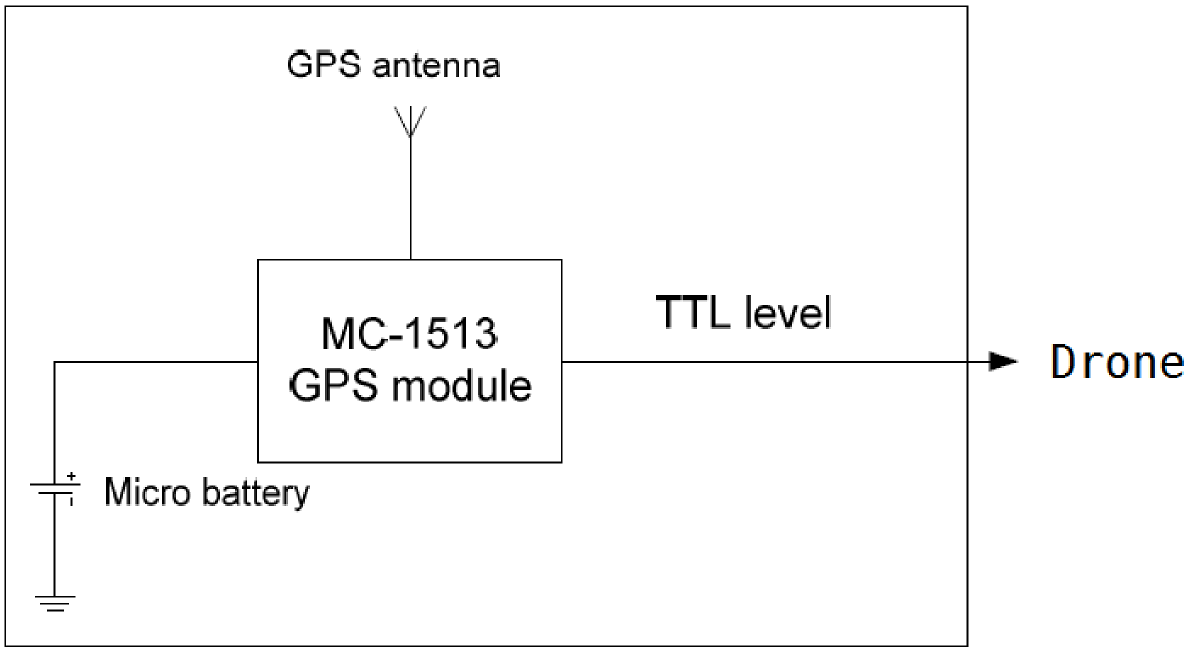
Mainboard (Bottom-View)

A: Power supply for the engines
B: Connector to the engine-boards
C: I2C Bus (data)
D: I2C Bus (clock)
E: 5V
F: Ground
G: I2C EEPROM CSI 24C32WI, i2c bus0, address 50 (datasheet)

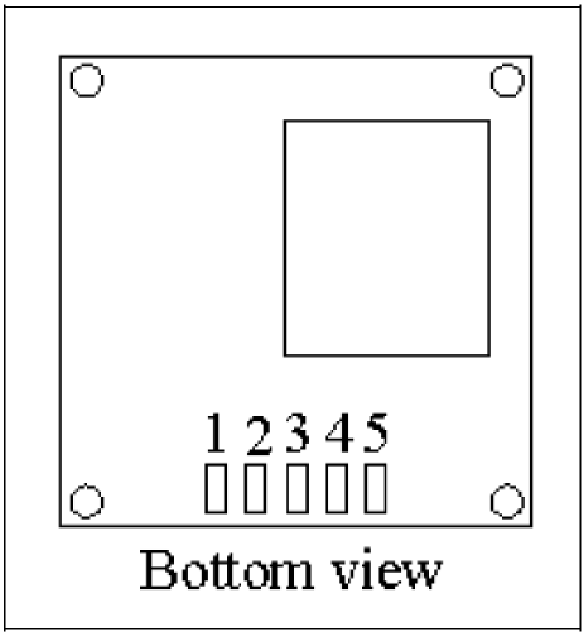
We are going to use (10) and (11) on figure X for our VCC and Ground of the GPS module. The VCC of the Mainboard is 5V and the GPS module takes a VCC input of 3.3V so we will attach a regulator in between the connection. For the RX and TX data of the GPS module, we are going to use pins 4 and 6 on the serial port at (7) on figure X. This GPS data will be in the NMEA standard format and can be viewed from the /dev/ttyPA0 package in the Drone's API.

GPS Module: 66 Channel LS20031 GPS 5Hz Receiver
The LS20031 GPS receiver is a complete GPS smart antenna receiver, that includes an embedded antenna and GPS receiver circuits. This low-cost unit outputs an astounding amount of position information 5 times a second. The receiver is based on the proven technology found in LOCOSYS 66 channel GPS SMD type receivers that use MediaTek chip solution.

The GPS smart antenna will track up to 66 satellites at a time while providing fast time-to-first-fix, one-second navigation update and low power consumption. It can provide you with superior sensitivity and performance even in urban canyon and dense foliage environments. The capabilities meet the sensitivity requirements of car navigation as a well as other location-based applications.



System Diagram for LS20031 GPS Receiver



Pin assignment of LS20031

Pin #	Name	Type	Description
1.	VCC	P	Power input
2.	RX	I	Data input (TTL level)
3.	TX	O	Data output (TTL level)
4.	GND	P	Ground
5.	GND	P	Ground

The VCC has an input range from 3V - 4.2V but the typical input is 3.3V. The RX and TX Data are in the NMEA (National Marine Electronics Association) standard format. The GPS module will be connected to the Mainboard by attaching the VCC and Ground from the Mainboard to the GPS module and the RX (pin 2) and TX (pin 3) from the GPS module to the Mainboard serial ports as explained earlier.

3.3 Approach for design validation

Testing of our system was primarily performed through unit testing, experimentation in the field, and user study tests. The unit testing framework, experimentation methods, and user study framework will be discussed to explain how our intermediate steps and final solution were evaluated. We attempted to create an evaluation framework that covered the main areas of concern and the functionality that is most crucial to the operation and success of the whole system.
The unit tests were designed and created to evaluated and determine the correctness / validity of each software module. Each module that was non-trivial in the implementation of the control system and route finding algorithms was tested to prevent issues that could impact the integrity of the entire system and even the small individual functions / operations that were under test.
Experimentation in the field was conducted to primarily evaluate GPS accuracy and validity of the data that is obtained by the GPS module along with the data received by the phones and additionally how that data compares with that from online web / mapping sources. Hands-on

data measurements were taken and comparisons were made to ensure the accuracy and reliability were within the bounds of our defined thresholds and ideal operational ranges.
User studies were used to evaluate the functionality and ease-of-use of the smartphone application. One of the biggest concerns at the beginning of the project was ensuring the app was ease to learn for a wide user-base and that the flight controls were implemented in a very intuitive manner. For our studies we simply allowed the users to start the app and "explore" without any instruction until they had exhausted their patience and then we would provide small hints to see where the true problematic areas and aspects were located.
The following is a list of the Design Constraints enumerated early in the design phase of our project; each will have a short discussion related to the evaluation of that constraint during testing.

Design Constraints

- *Easily controllable and usable by operator between the age of 16 and 70 with little training*
    - User studies indicate the app and flight controls are intuitive, but a brief demo or even instruction screen is VERY helpful in assisting the users when first attempting to use the app. Suggestions were made regarding the placement of buttons, increasing the size of buttons, and the potential of attaching small labels for easier use.
- *Cost efficient - $400-$600*
    - Cost is projected to be within this budget, but due to technical difficulties, we were required to spend more; however, this was expected due to the nature of prototyping.
- *Operation of craft on days of "reasonable" weather conditions (low winds, no precipitation, and good visibility) – not an "all-weather" solution*
    - Field experimentation emphasizes this is NOT an all-weather solution. Wind affects the drone significantly and the power required to compensate for the wind forces significantly reduces battery life and flight time.
- *Automated landing procedure in the event of loss of connection*
    - Automated landing is not executed simply because that would have required modifications to the drone's firmware, but in the event of connection loss, the drone simply enters into a hover mode and remains in the location where the connection was dropped.
- *Bumpers and impact-absorbing materials to prevent damage to the craft and/or vehicles*
    - Protective, full hull prevent damage to craft or vehicles in the event of crash or malfunction
- *Windows Phone 7 does not allow ad-hoc WiFi (default of the quadcopter)*
    - Force drone to connect to access point using a telnet command and additionally newer drone firmware implements access point wireless abilities
- *WP7 only executes "safe" code – prevents many of the 'quick' tricks that are useful*
    - Cannot bind to specific ports to send and receive data - impacted video streaming
- *Custom encoding of video requires custom decoding*
    - N/A since video could not be received, however, WP does decode video format of the drone
- *GPS data must be streamed through unused UDP port*
    - Due to technical difficulties, an additional WP was used to stream GPS directly to the main Windows Phone executing the AutoFlight app
- *Easy and intuitive flight controls*

    - Defaults to regain control and hover when user is not engaged
    - User studies show controls are intuitive but button placement suggestions were made and updates have been performed

Validation and Testing

- Phone Tests conducted
    - Connection verification and packet handling
    - Correct interpretation of accelerometer movement
    - Encoding of navigational commands used to send to drone
    - Nav data processes and decoding *(N/A since did not receive data from drone)*
    - Video decoding *(N/A since did not receive data from drone)*
    - GPS data reading and interpretation
    - Heading determination from GPS data
    - Accuracy of route finding based on waypoints selected
    - UI functionality
    - Error handling
- Copter Tests conducted
    - Connection tests and “sync” packet handling
    - GPS data read through serial port correctly *(yes - while GPS module functional)*
    - GPS packetized correctly in expected fashion *(N/A - didnt get to packetization)*
    - GPS proxy app sending packets on correct port without modifying packets
    - Transmit and receive acknowledgement handling
    - Error handling and recovery
    - Connection loss / drop handling
    - Emergency signal and shutoff in “desperate” situation

The system / solution that we created has many components and pieces that interact to form the complete solution; however, the primary objectives for prototype testing are not mutually dependent and thus they formed a good basis for evaluation to pinpoint the strengths and weakness of our final solution. The solution was tested thoroughly within our group to ensure functionality and expected behavior, but as listed below, we conducted tests with persons unfamiliar with our solution and technology in general to evaluate the effectiveness of our UI, flow of control, ease of use, and ability to handle corner cases. The objectives listed below are high level and are meant to cover the high level function and operation of our project as a whole rather than individual components / modules.

OBJECTIVE	**Criterion to Test**	**Minimal level of Completion**	**Optimal Level of Completion**
Teleoperation of Copter	1. Takeoff 2. Landing 3. Rotation 4. Directional travel	Control using one method (static or accelerometer based controls)	Control using either/both control methods and the ability to select between the two control systems

GPS AutoFlight	1. Find and select waypoints 2. Choose / save route 3. Copter follows path	Copter navigates from start point to end point	Copter navigates a series of waypoints. Multiple routes saved for the copter to navigate.
Video Stream	1. Video stream from copter to phone is active 2. Video is clear (decoded correctly) 3. Optimal – save data	Video displays on phone screen and image is clear	Video is displayed and there is the ability to save video / images
Easy to Use	1. Phone app easy to use and learn 2. Flight controls intuitive 3. Crash prevention	Successful user tests with users of 3 age levels with no previous knowledge of system – youth, college, and 40-60 age range.	Satisfy minimal level without any instruction and/or influence/help. No training nor help.
Complete Solution	1. Teleop. Satisfied 2. GPS Flight Satisfied 3. Video Stream Satisfied 4. Easy to Use Confirmed	Any combination of completion levels (minimum at least) of the previously enumerated objectives. Completed ON TIME	All objectives satisfied with "Optimal" level of completion. Completion prior to deadline.

NOTE: Please see the **Experimentation Results** section for detailed results

4 Implementation notes

Phone Controls

Communicating with the AR.Drone

The AR.Drone is configured to send/receive data through four separate channels: AT* command channel, Navigation Data, Video stream, and the Configuration Data channel.

The AT* command channel is a purely receiving channel. The drone receives AT* commands from the controller through UDP port 5556. For proper operation the commands must also come from the UDP port 5556 on the controller, using the same port on both sides of the UDP data stream is a requirement enforced by the drone. This restriction makes the drone inherently incompatible with the Windows Phone but this can be circumvented by running a proxy application on the drone beside the standard firmware. This application will be detailed later in this section.

Tests done by parrot have demonstrated that smooth drone movement can be achieved by sending AT* commands every 30 ms. If the drone receives no commands for a period of 2 seconds it will assume that the connection is lost so commands must be sent with an interval no greater than 2 seconds.
The available AT* commands are outlined starting on page 29 of the AR.Drone Developer Guide. The ones applicable to this project are outlined below. The first argument of every command is the sequence number
Any floating point parameters must be inserted into the command strings as the integer value of the binary representation. This can be accomplished in C/C++ using a pointer cast from float* to int*. In C# this is implemented using the BitConverter class.

- AT*REF
  - Syntax: AT*REF=arg1,arg2
  - This command controls the basic behaviour of the drone.
  - The second argument is a bitfield describing the desired behaviour
    - bit 9 is 1 to cause the drone to take off, 0 to land
    - bit 8 is 1 to signal an emergency, 0 otherwise
      - signaling this causes the drone to cut off the engines no matter what state it is in.
    - No other bits are used
    - bits 18,20,22,24, and 28 should all be set to 1, all other bits set to 0
- AT*PCMD
  - Syntax: AT*PCMD=arg1,arg2,arg3,arg4,arg5,arg6
  - This command controls the flight motions of the drone
  - arg2 is a bitfield enabling the use of progressive commands and.or the combined yaw mode
    - if bit 0 is one the drone will ignore all other parameters and hover at the same location set it to 1 to make the drone respond to the other parameters
    - if bit 1 is set then combined yaw mode is enabled
  - The final four arguments are, in order, roll, pitch, vertical speed, and angular speed
    - These are expressed as floating point values in the range [-1,1]
    - They represent percentages of the maximum angles and speeds set in the drone's configuration.
- AT*FTRIM
  - Syntax: AT*FTRM=arg1,
  - This command tells the drone that is is currently level.
  - This sets the drone's reference for the horizontal plane in the internal control system.
- AT*CONFIG
  - Syntax: AT*CONFIG=arg1,arg2,arg3
  - This command sets one of the drone's internal configuration valuse
  - The second argument is the name of the option to set between double quotes
  - The third argument is the value to set between double quotes

The navigation data is sent and received through UDP port 5554. The stream is started by sending a packet with data optional to the drone. The stream is then sent back. Similar to the AT* command channel the ports must be the same on both sides of the link. A more detailed specification of the navigation data can be found starting on page 39 of the AR.Drone Developer

Guide. Navigation data was not used in this project so a description of the format will not be included here.
The video stream is very similar to the navigation data. data is sent and received through UDP port 5555. The stream is initialized by sending an empty packet to the port and then video data is sent out The ports on both sides must be identical just like the AT* commands and the navigation data. The stream is sent as a sequence of compressed images. One image per packet. Each packet is approximately 17 kB. The description of how the images are encoded can be found starting on page 43 of the AR.Drone Developer Guide. They are encoded in a proprietary format based on a simplified version of the H.263 UVLC format. A detailed description of the format will not be provided hereto avoid repetition.
The configuration data stream is sent and received through TCP port 5559. The configuration is sent when the appropriate AT* command is sent on the AT* command channel. More information can be found starting on page 56 of the AR.Drone Developer Guide. Configuration data was not read in this project so the description was omitted.

The Drone Controller Application
The Drone Controller set of classes handles communication of the drone utilizing a separate thread for each communication channel. When a connect is requested a thread is started to handle the AT* commands, the navigation data, the video feed, and the config data.
The navigation data and video stream threads are similar. they both send a initialization packet to their respective ports on the drone and then receive and process the data coming back. The navigation data thread processes the data into simple structures that can be read easily and then signals an event that can be monitored by objects that need to read the data. The video stream thread takes the received data and passes it through a decoding function that creates the bitmap of the image. This image is then sent to all interested in it through an event.
The AT* command thread sends AT* commands at regular intervals by sleeping for fixed intervals. At commands are queued up to be sent by adding them to a synchronized queue of commands and then the worker thread extracts commands from the queue, adds an AT*REF command containing the desired state of the drone as a method to keep the drone from timing out if no movement commands come for an extended period of time, and sends the packet off to the drone
The entire class exposes functions to queue up various AT commands to be sent to the drone and to set or unset to takeoff bit in the AT*REF commands that are sent by default. It also exposes the events for reception of navigation data and new video images, as well as trace messages to be able to monitor what the controller is doing.

The Phone UI
The phone control ui contains the changing image generated from the video feed as well as static buttons for connecting to the drone, taking off and landing. It also has buttons for directional controls, and a button to toggle using the accelerometer for drone control. The accelerometer controls were set up to make the drone roll become the pitch of the phone and the drone pitch to become the phone roll. The phone pitch and roll were also transformed based on configuration settings to allow for an assortment of mappings from phone orientation to drone orientation.

The important configuration options that were offered were: Drone tile max, phone tilt max, phone tile threshold, turn speed, and vertical speed. The phone tilt threshold defined a dead zone such that any angles with magnitude less than the setting will result in no drone movement. This provides a comfort zone so that the operator doesn’t have to hold the phone completely horizontal for the drone to not move. The drone tilt max and phone tilt max settings are used to scale the angles of the phone to drone angles. If the settings are: drone tilt max 10, phone tilt max 25, and phone tilt threshold 5, then phone angle 5 = drone angle 0, phone angle 15 = drone angle 5, and phone angle 25 = drone angle 10.

The Proxy Application

The purpose of the proxy application on the drone Linux kernel is to bridge the gap between the incompatible interfaces on the Windows Phone and the AR.Drone firmware. The application communicates with the firmware in the way it expects and then relays that data to the Phone in the way acceptable for the phone.

The application communicates with the firmware by using NAT routing rules to spoof packets from a different IP but the same port. The NAT rules used for the video feed were as follows.

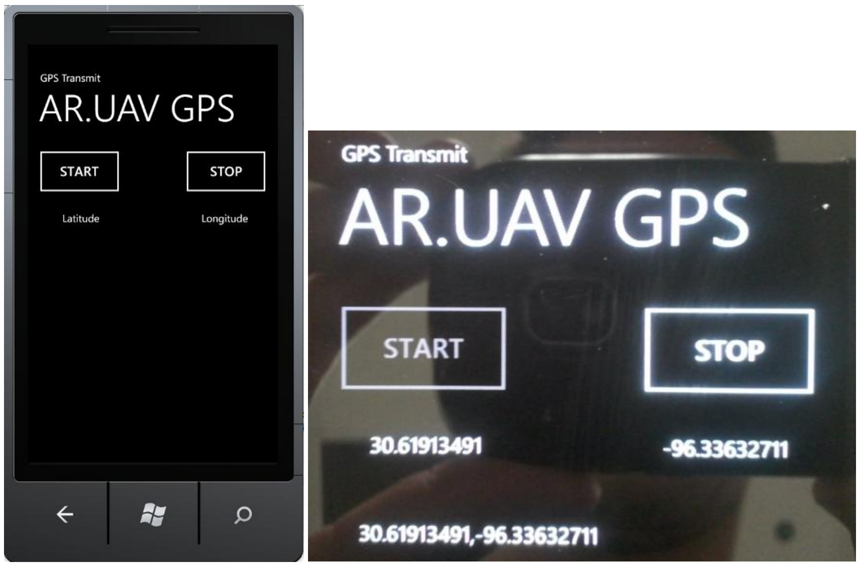
- POSTROUTING
  - if destination port = 5555
  - then source = 192.168.1.2:5555
- PREROUTING
  - if source port = 5555
  - then destination = 192.168.1.1:15555

After inserting these rules the Proxy app can communicate with the firmware from its port 15555 to the firmware port 5555 and the firmware will believe that the packets are coming from the IP 192.168.1.2 port 5555.

Sending the data back to the phone is not complicated. A UDP port was opened to act as the drone video feed, we used port 6666. The application waits for a packet to come in on the port and then records the source IP and port. It then starts the communication with the firmware and relays the data back through port 6666 to the recorded IP and port of the Phone.

GPS Module

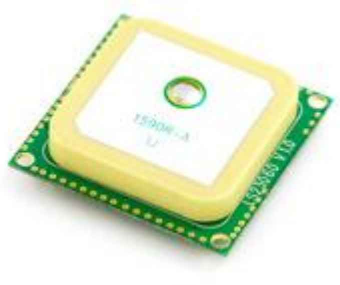
This section is divided into two parts: Our actual implementation with the second Windows Phone providing GPS data, and our proposed implementation with the GPS module attached the to drone.



30.61913491
-96.33632711
30.61913491,-96.33632711

AR.UAV GPS App on Windows Phone: (Left) Emulator, (Right) In Action

The AR.UAV GPS application uses C# code to take GPS (Latitude and Longitude) data and send it over the WiFi connection to the primary Windows Phone running the Flight Application. We were forced to use this method because two of our GPS modules burnt out (most likely due to an accidental short circuit) while mounting them to the drone.

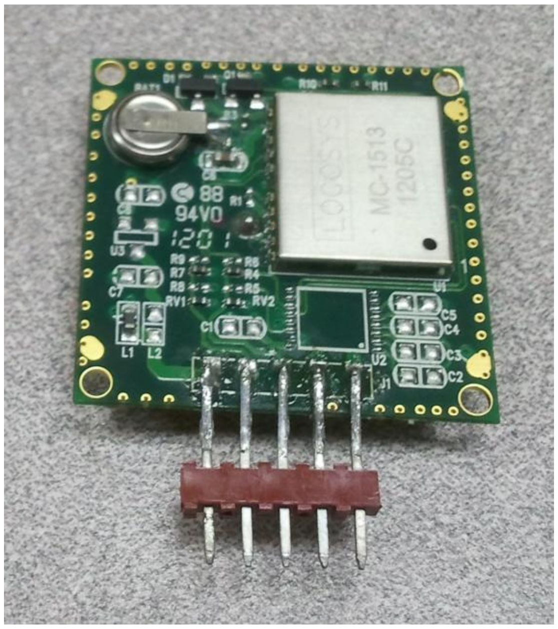


GPS Module

The GPS Module used in our design was the 66 Channel LS20031 GPS 5Hz Receiver purchased from sparkfun.com (http://www.sparkfun.com/products/8975). The process to ensure that you have a stable, working GPS module has multiple stages and checkpoints. The first stage is to create the connections and check that the module is working properly. The second stage is to make sure you can read data through the PC. The third stage is to make your drone fully self-contained with GPS capability.

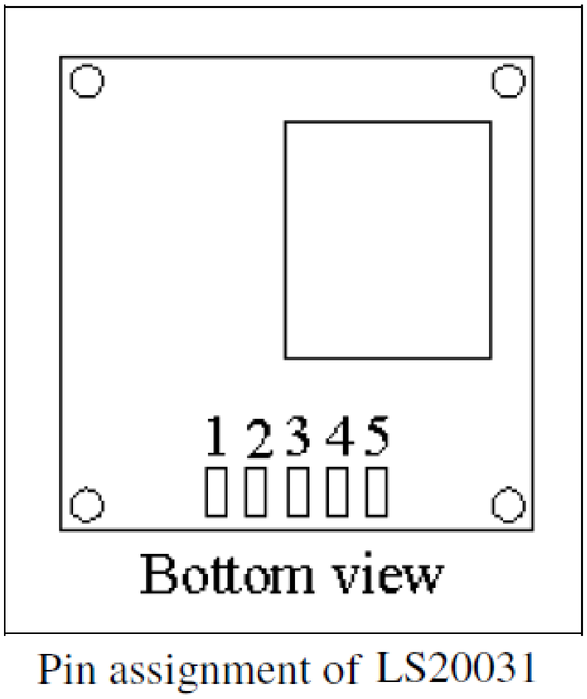
Stage 1 - Setting up the GPS module and testing power.

To start you need the GPS module, a 5-pin header, solder, and a third-hand tool. The connection pads are large enough so that the soldering isn't too difficult, but be sure to make clean connections to minimize the risk of short circuiting the system. The site we bought our GPS from (sparkfun.com) also had a tutorial for soldering the header that might help (http://www.sparkfun.com/tutorials/176).

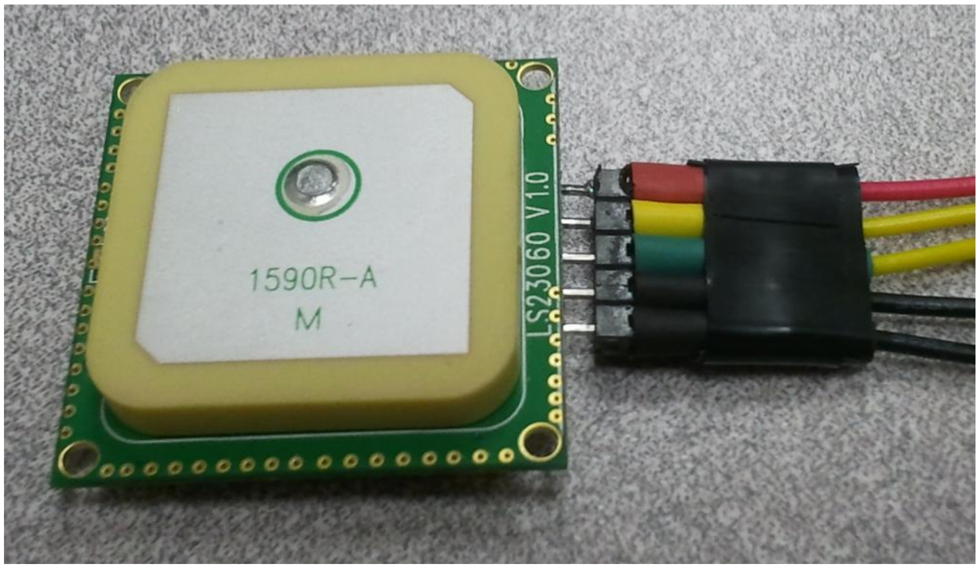


GPS with 5-Pin Header

Next, you need to attach some jumpers to the header so that you can attach your GPS module to the FTDI board and later the drone. Once again you need some more solder and the third-hand tool. Make sure to color code your wires so that you can easily identify what wire belongs where. I use red for Vcc / power, yellow for Rx, green/yellow for Tx, and black for GND / ground. The pictures below show the pin layout for the GPS Module. For more information, the data-sheet can be found below.
(http://www.sparkfun.com/datasheets/GPS/Modules/LS20030~3_datasheet_v1.2.pdf).



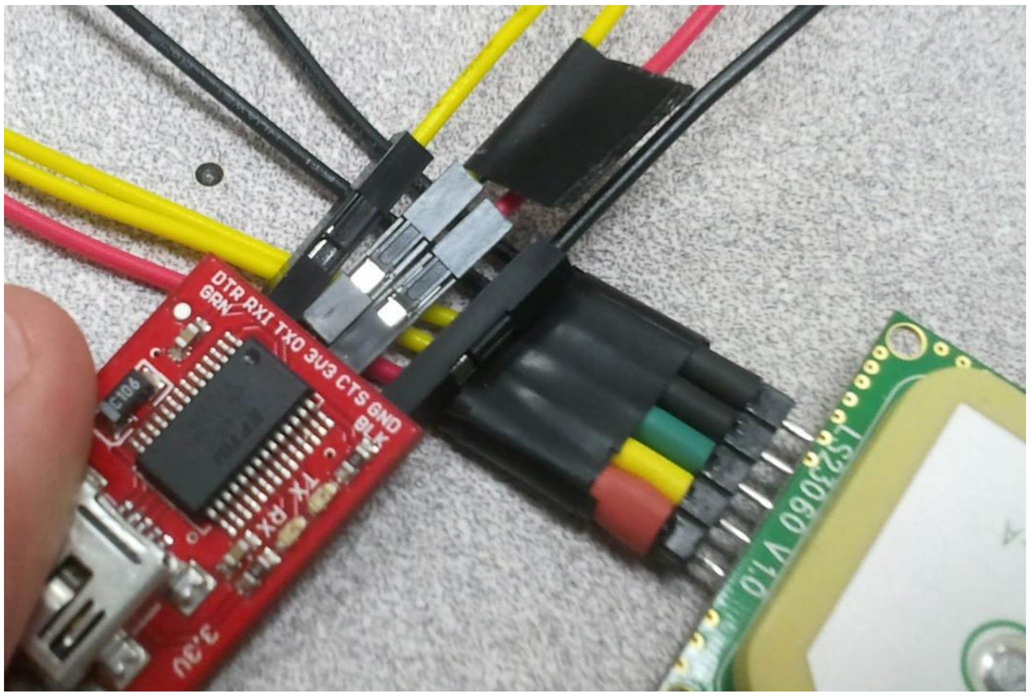
GPS Pin Layout



GPS with Color Coded Jumpers

Pin 1 -- Vcc -- RED
Pin 2 -- Rx (Receive) -- YELLOW
Pin 3 -- Tx (Transmit) -- GREEN
Pin 4 -- GND -- BLACK
Pin 5 -- GND / No Connection – BLACK

Once you have the jumpers connected, you need to pull out a FTDI breakout board rated for 3.3V Vcc. We used obtained ours from sparkfun.com (both of the following work: http://www.sparkfun.com/products/9873 and http://www.sparkfun.com/products/10009).



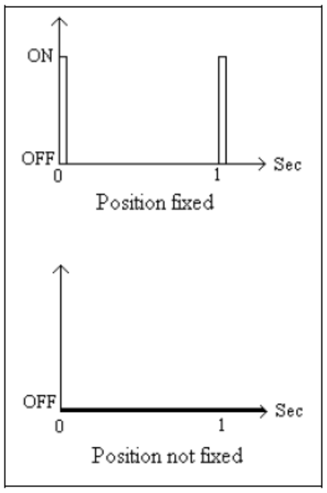
GPS Attached to FTDI Board

Follow the Picture above to connect your GPS to the FTDI breakout board: Vcc (GPS) to 3V3 (FTDI), Rx (GPS) to TXO (FTDI), Tx (GPS) to RXI (FTDI), and Pin 4 GND to GND.
To connect to the board to the computer, this breakout board uses a USB Mini-B cable. (http://www.sparkfun.com/products/598)

For the next step you want to pay attention to the GPS module closely.

There is a Red LED light on the front side of the GPS module (opposite the header). When you power the module, the status light should blink on and then turn off. If you are in range of satellites (outside), the light should blink at a 1 Hz rate. If you are out of range (inside), then the light should blink on then turn off and stay off.

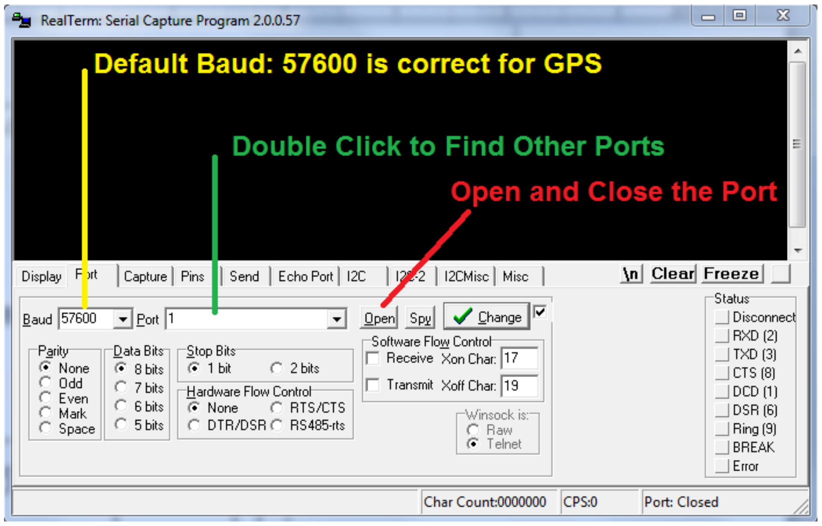
**IMPORTANT: If the light remains on for longer than 2 seconds unplug the module. There could be lose wires, crossed connections, wrong current, wrong voltage, etc. Recheck your connections and try again.



LED indicator of GPS positioning status

Stage 2 - Connecting to the PC and reading data.
Now that you know the GPS module is working correctly, you need a terminal program to read the GPS data. We used RealTerm as our terminal program. (http://realterm.sourceforge.net/)

When you first open RealTerm, go to the Port tab and click the Open button to close the connection. Double click the Port text box to find other ports. Reopen the port once you have selected the correct port number (It won't hurt if you try them all. If you open the wrong port, nothing will show up so just try another port). The change button will allow you to change these settings while a port is currently open so if you want to change the Baud rate or Port number while another port is open, just adjust the information in the text boxes and press "Change".



Baud 57600 Port 1

Parity: [✓ None] Odd Even Mark Space

Data Bits: [✓ 8 bits] 7 bits 6 bits 5 bits

Stop Bits: [✓ 1 bit] 2 bits

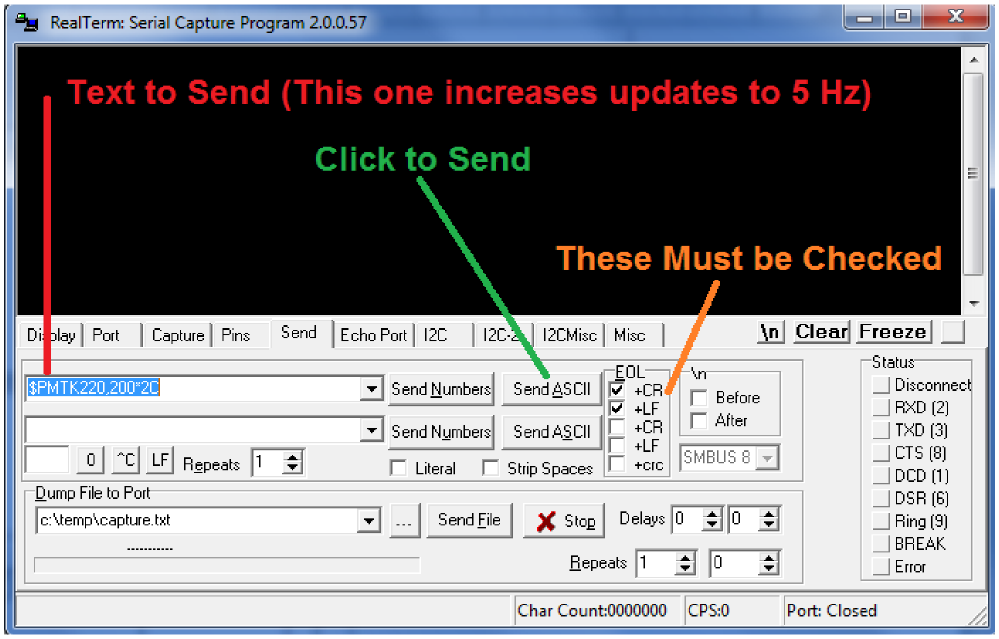
Hardware Flow Control: [✓ None] RTS/CTS DTR/DSR RS485-rts

Software Flow Control: Receive Xon Char: 17 Transmit Xoff Char: 19

Winsock is: Raw [✓ Telnet]

Char Count:0000000 CPS:0 Port: Closed

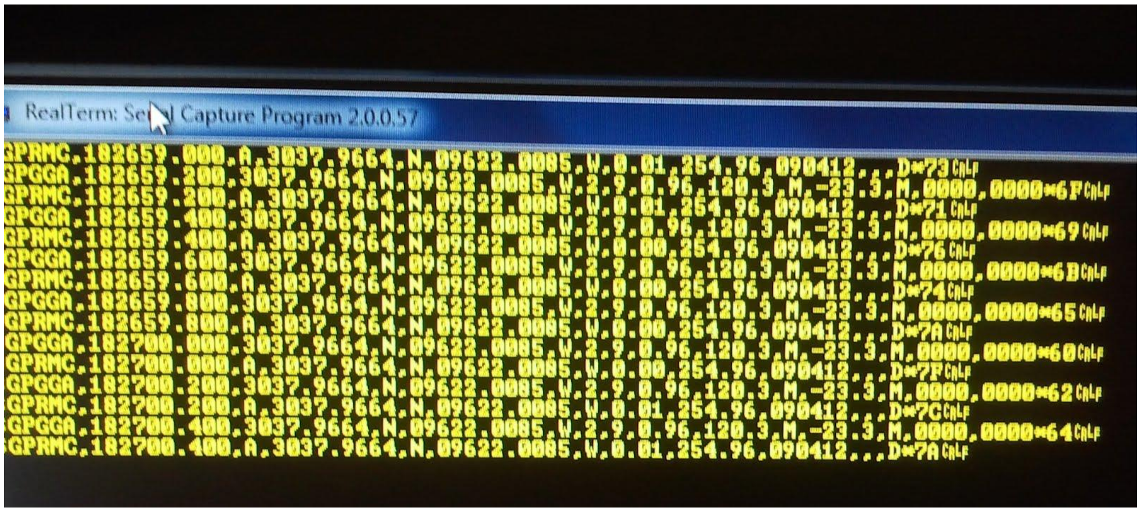
If you want to send commands to the GPS module, you must have the Rx cable of the GPS attached to the Tx of the FTDI breakout board. Then click on the Send tab, type in (or copy paste) the command you want to send. Check the boxes in EOL for +CR and +LF, and press Send ASCII when ready. You should see the TXD (3) Status light on the right side light up when you do.



```
$PMTK220,200*2C
c:\temp\capture.txt
```

Send Tab in RealTerm

If your GPS is working correctly and sending data, you should see a screen similar to this one below.



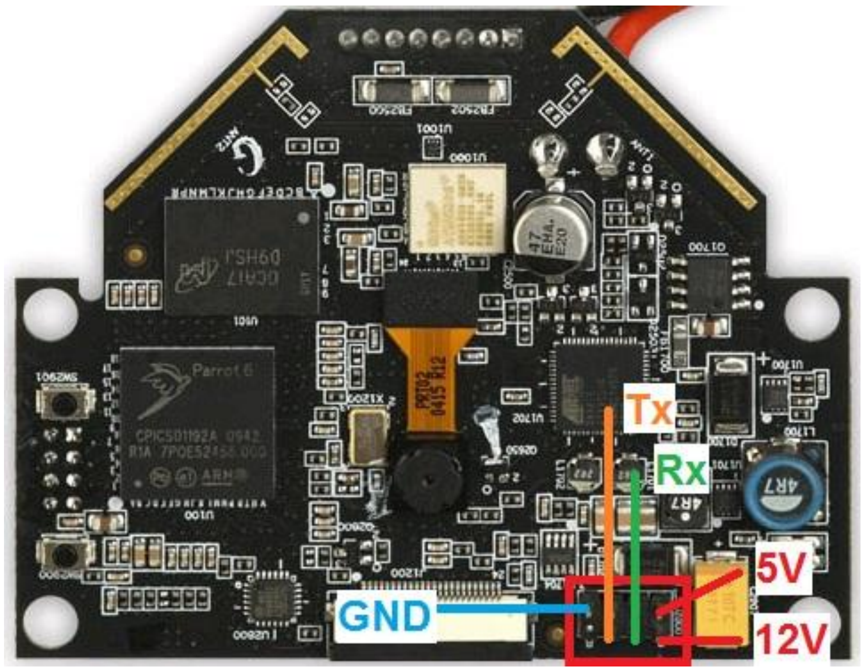
```
GPRMC,182659.000,A,3037.9664,N,09622.0085,W,0.01,254.96,090412,,,D*73CRLF
GPGGA,182659.200,3037.9664,N,09622.0085,W,2,9,0.96,120.3,M,-23.3,M,0000,0000*6FCRLF
GPRMC,182659.200,A,3037.9664,N,09622.0085,W,0.01,254.96,090412,,,D*71CRLF
GPGGA,182659.400,3037.9664,N,09622.0085,W,2,9,0.96,120.3,M,-23.3,M,0000,0000*69CRLF
GPRMC,182659.400,A,3037.9664,N,09622.0085,W,0.00,254.96,090412,,,D*76CRLF
GPGGA,182659.600,3037.9664,N,09622.0085,W,2,9,0.96,120.3,M,-23.3,M,0000,0000*6BCRLF
GPRMC,182659.600,A,3037.9664,N,09622.0085,W,0.00,254.96,090412,,,D*74CRLF
GPGGA,182659.800,3037.9664,N,09622.0085,W,2,9,0.96,120.3,M,-23.3,M,0000,0000*65CRLF
GPRMC,182659.800,A,3037.9664,N,09622.0085,W,0.00,254.96,090412,,,D*7ACRLF
GPGGA,182700.000,3037.9664,N,09622.0085,W,2,9,0.96,120.3,M,-23.3,M,0000,0000*60CRLF
GPRMC,182700.000,A,3037.9664,N,09622.0085,W,0.00,254.96,090412,,,D*7FCRLF
GPGGA,182700.200,3037.9664,N,09622.0085,W,2,9,0.96,120.3,M,-23.3,M,0000,0000*62CRLF
GPRMC,182700.200,A,3037.9664,N,09622.0085,W,0.01,254.96,090412,,,D*7CCRLF
GPGGA,182700.400,3037.9664,N,09622.0085,W,2,9,0.96,120.3,M,-23.3,M,0000,0000*64CRLF
GPRMC,182700.400,A,3037.9664,N,09622.0085,W,0.01,254.96,090412,,,D*7ACRLF
```

NMEA Formatted GPS Data in RealTerm

If you want more help, this tutorial by DIYDrones.com for setting up the GPS has other pointers, but be sure to use the information I provided (some of the numbers in this tutorial are incorrect). (http://diydrones.com/profiles/blogs/using-the-5hz-locosys-gps-with)

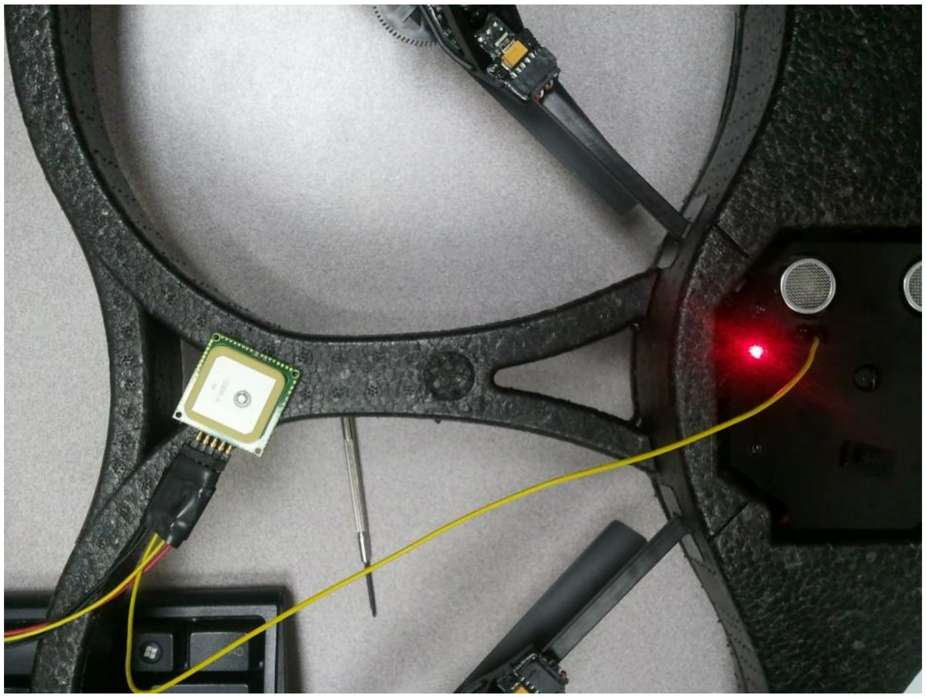
Stage 3 - Creating a Self-Sustained GPS Drone

Now that we have the GPS module working, try taking the Tx pin from the GPS module and attaching it to Pin 4 (Rx) of the drone, and also attach the Pin 5 GND from the GPS to Pin 7 (GND) of the drone. The drone for some reason does not like it when only one wire is attached to the usb connector on the bottom, so the second GND pin fixes this issue.



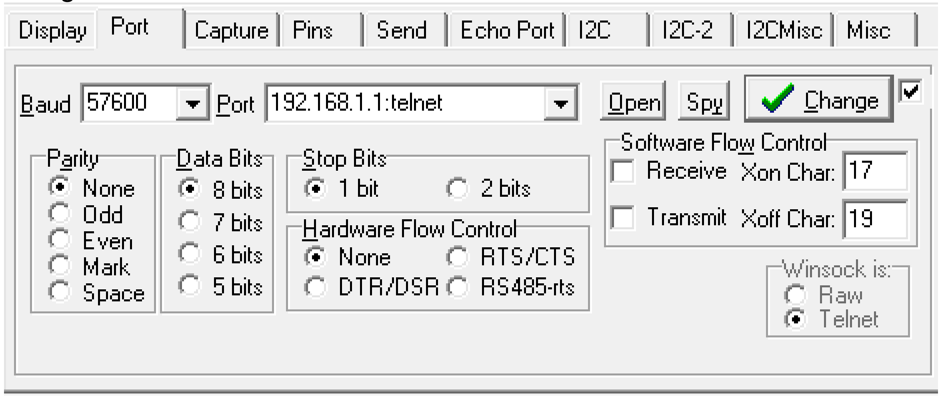
Main-board USB/Serial Pin Out

Pins are numerated from top right to bottom left (with what would be pin 8 filled in).
Pin 1 - 5V
Pin 2 - 12V
Pin 4 - Rx
Pin 6 - Tx
Pin 7 – GND



GPS Attached to Drone Serial Port

Once you have attached the GPS module to the drone, but before you take the GPS off the power supply / FTDI board. Try to see the data through the drone. (Your computer must have WiFi capabilities to do this next step.) Open up your Wireless Networks and connect to the AR.Drone WiFi network. Go back to RealTerm and on the Port tab type in "*192.168.1.1*" to telnet into the drone's Linux Kernel. Then type "*stty -F /dev/ttyPA0 57600*" to change the baud rate of the serial port to 57600. Next type "*cat /dev/ttyPA0*", and you should start seeing the GPS data coming out.



Baud 57600 Port 192.168.1.1:telnet

Parity: [✓ None] Odd Even Mark Space

Data Bits: [✓ 8 bits] 7 bits 6 bits 5 bits

Stop Bits: [✓ 1 bit] 2 bits

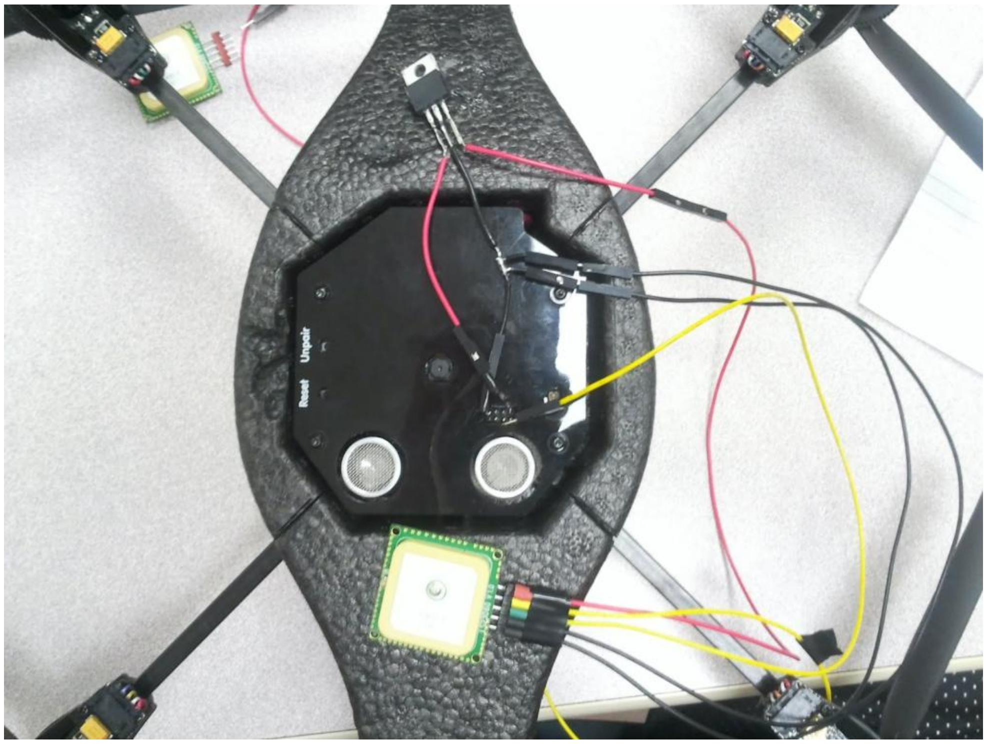
Hardware Flow Control: [✓ None] RTS/CTS DTR/DSR RS485-rts

Software Flow Control: Receive Xon Char: 17 Transmit Xoff Char: 19

Winsock is: Raw [✓ Telnet]

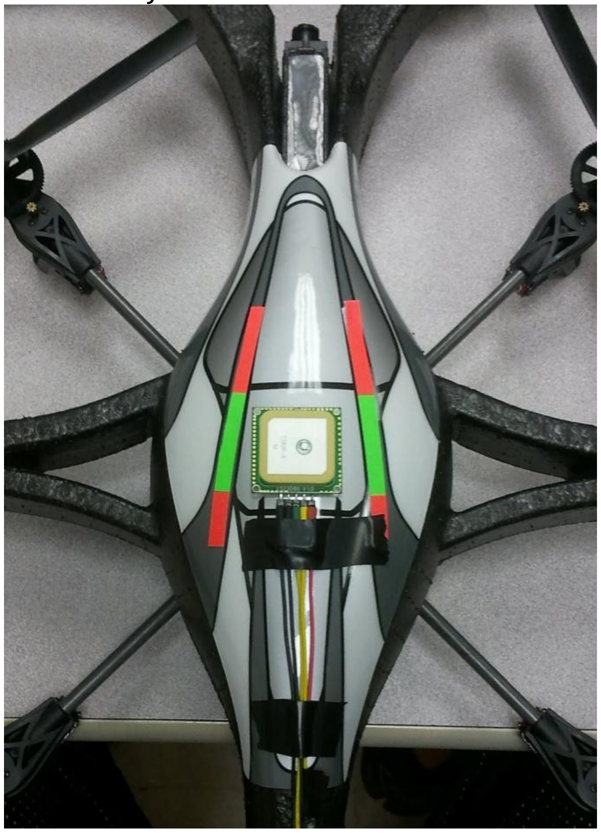
Telnet into Drone in RealTerm

The GPS uses a 3.3V Vcc, but the drone only has 5V and 12V pins available to power the GPS. To efficiently step down the voltage, we used a voltage regulator. The pin that outputs 5V is defaulted to off, but can be turned on by changing some of the firmware. Another option would be to use a 12V to 3.3V voltage regulator or a series of regulators (12V to 5V and then a 5V to 3.3V) since the 12V pin is already turned on (does not require firmware modification).



GPS Attached with 5V to 3.3V Regulator

Once the GPS is all connected, you need to place it on top of the drone so that the GPS receiver will be facing the open sky. One way to do this is to just tape it onto the top as shown below, but another method would be to cut an hole in the hull and fit the GPS inside with the top sticking out so that the wires can stay inside the hull.


GPS Attached to Hull

The proxy app that acquires and sends the GPS data can simply be copied to the drone through FTP and started through telnet using the bash shell. The downside to this is that the app will not start automatically after each power cycle. The app can be made to start after each power cycle by placing commands into the appropriate shell script. The main issue with this process is that actual changes have to be committed to the firmware to start the shell script each time the drone is started; this runs the risk of permanently "bricking" the drone if the configuration and modification is not done correctly. The same process is required if you want to force the drone to connect to a WiFi AP each time rather than sending the telnet command each time the drone is power cycled.

Original source code for the proxy app was found at:
http://code.google.com/p/ardudrone/source/browse/trunk/AtComProxy/atcomproxy.c?spec=svn44&r=44
Early into the project, we became avid AR.Drone fans and used the forum at http://www.ardrone-flyers.com to communicate our ideas and search for help from the AR.Drone developer community.

Route Finding Algorithms and Implementation

***Receiving GPS data from* GPS Module:**
Receiving the GPS data that is being sent using a GPS module is essentially the same process as receiving nav data and video stream data if you choose to implement a proxy app within the drone's firmware. The proxy app is meant to basically forward the data from the serial port of the drone (Rx port: /dev/ttyPA0) to an unused UDP port; our design called for the data to be forwarded to the 6666 port, but implementation of the proxy app was paused due to technical difficulties with the GPS module, in addition to architecture limitations of the Windows Phone operating systems which prevent explicit binding to specific sockets/ports. (see the above section for the proxy app information)
Once the proxy app is implemented and assuming Microsoft updates the policies regarding binding to specific sockets, or you implement multicast transmission of the UDP packets in the proxy app, the first step is to receive the data from the port of your choice or simply listen for data on the multicast channel. The data packet, depending on the specific implementation should contain some type of header or at least a checksum to ensure transmission didn't modify the data, but the actual GPS data that is needed in our implementation of the route finding algorithms are only latitude and longitude. Heading direction/angle information could be useful for future reliability and additional corrective modifications to increase robustness of the system.

The current configuration can be easily adapted if the transmission is in a multicast format; if the value are simply separated by only a comma, the code will automatically split the data, add values to array and then update the current GPS reading with the appropriate latitude and longitude values. The code is configured to basically listen for new data and when new data is available, the code splits, updates, and then repeats.

GPS data from second Windows Phone:
Due to technical difficulties and equipment failure, we coded an application to transmit real-time GPS data from a 2nd Windows Phone attached to the actual drone to the main phone controlling the drone and executing the route finding algorithms. The GPS transmit application is constantly polling for GPS data and transmitting to a multicast group upon the receipt of new data. This is a very resource intensive application - battery life is significantly affected and a large amount of data is transmitted in this configuration. Additionally, since the phone we were using does not have a 3G connection and is not connected to a WiFi AP other than the drone, GPS data is very inaccurate and the error rate should be considered. However, we hope since the value are updated extremely regularly, accurate information will be obtained in most instances.

Like I mentioned, the GPS transmit app broadcasts the GPS data in a comma separated format over a multicast group. The main phone application joins the multicast group once the AutoRoute page of the app is activated - listening for incoming GPS data begins instantly. When data is received, the packet is split on the commas and the values (latitude and longitude) are stored in an array and then they are used to update the current GPS location object within the overarching class. GPS data is updated each time new data is received, however, the reading is only used when after the drone executes an iteration of the path finding algorithm, thus, the inaccuracies are minimized.

Integration with the Map Page within the App
The Windows Phone operating system has a built in Bing Maps library, however, for it to be fully functional, you are required to register and obtain an API key. The key is only used if queries are made regarding locations but it is good to prevent any conflicts (free to students). The maps

application and library are easy to add to the project, simply add a resource and import the map object into the XAML code. The unique functionality is related to the placement of waypoints / destination markers. The default marker is a black rectangle, but this can be easily modified by including your own icon image and again updating the XAML code. The placement of the waypoints has been implement by using a button to drop a waypoint at the current center of the map - the center can be found automatically with the map object. (I also added an 'X' overlay on the map to ease the placement of the waypoints) The waypoint placement button calls the corresponding click function in the C# code - the function finds the middle of the current map, places the waypoint as a child object on the map overlay, and stores the coordinates of the center of the map.

The waypoint object is added to the overlay object associated with the map as a child object when the click function is called. Additionally, the child.Show() function is called to ensure the waypoint icon is displayed on the map. The location of the waypoint object is determined based on the center of the map, however, when the map is adjusted this information/location is not modified because the location is hard linked to the specific latitude and longitude coordinates. Many of the waypoint attributes can be modified through the XAML such as coloring and transparency.

The main functionality of the waypoint click object, however, is to add the coordinate (map center point) to a route queue that is associated with the overarching class. Calling the map.Center() function returns a GeoCoordinate object that contains the exact latitude and longitude information - this object is added to the route queue of GeoCoordinates. The objects in the queue represent the waypoints that the user has selected and will be used in the route finding algorithms.
Different data structures were explored for this functionality, but the queue was used because it directly implements the functionality needed for in-order route traversal.

Once the user selects the waypoints they desire, they can chose in-order or shortest path traversal - queue is ideal for the first option since the queue is build based on the order of the waypoints selected. Additionally, the route finding algorithm dequeues one waypoint each time the previous waypoint is located that way the algorithm can check the length of the queue on each iteration to determine whether the final destination has been found which is implied by a queue length of zero.

An array was originally used for the data structure but due to the frequent additions and removals of objects from the structure in question, the queue simply made the most sense. The queue is implemented over an array and similar functions can be performed and a queue can even be converted into an array, which is actually performed if the shortest path traversal option is selected. Additionally, by using a queue, memory management, allocation, and deallocation is handled automatically without significant overhead and/or modifications - enqueuing and dequeuing can be performed without implementing any other member management logic. The C# queue is a very good, easy solution for our application, but absolute performance could be improved with an array.

Convergence Route Finding Algorithm

The basic principle behind the convergence route finding algorithm that we have implemented is that a stair-step like path is followed from the starting point to the given destination or the waypoint. The angle (slope) between the current position and the destination is calculated and a series of North/South and East/West commands are sent to the drone based on the angle of the path between the positions. This process is repeated multiple times between the start and the destination to ensure accuracy and to decrease distance traveled. The number of steps is set by the user in the number of steps and the thread multiple variables. The main advantage to this method is that the convergence to the destination leads to better accuracy and more fine-tuned control rather than a simple attempt to move straight toward the destination.

The algorithm assumes the drone starts in a north facing position and the calculations are based on latitude and longitude values for the Northern Hemisphere. The accuracy precision of the GPS modules / phone transmitter is not great, so instead of translating latitude and longitude data, a simple comparative based approach has been implemented. The latitude difference between the destination and the current position is calculated - if it is positive, the drone needs to travel north and vice versa; the longitude values in the Northern Hemisphere are negative values in relation to the the degree West, so thus, as you progress West, the values decrease (become more negative). To make the comparisons - the absolute value of the destinations longitude and the absolute value of the current position longitude value are compared and if the difference is positive, the direction of travel is West since the absolute value is of the longitude of a more westward position is larger than that of a more eastern position.

The direction of travel in both directions is saved for use in later calculations.
The travel ratio between the distance latitude (rise) and the longitude (run) is determined for use in making directional movement commands in each direction to the drone. Essentially, if the ratio is 2:1 (slope of 2, rise (latitude difference)) then twice the number of forward commands will be made to the number of side movement commands. This is a bit skewed since the latitude difference is likely to be greater on average than the longitude difference but it provides a good baseline for comparison and navigation. However, there is also an issue if the longitude distance is arbitrarily small - the ratio becomes arbitrarily large and thus the drone receives far too many directional movements in the x-coordinate plane of movement. Our algorithm evaluates the travel ratio and we do not allow more than two times the number of side directional movements to the given number of y-plane directional movements. The number of y-plan directional movements is controlled by the variables mentioned above, and thus the limit is also dependent on those variables.

As an example to make the algorithm a bit more clear: let us suppose we know our current position and the destination - the latitude difference is 0.002 and the longitude distance is 0.001. The direction of Y-travel will be North since latitude difference is greater than zero, and the X-travel direction will be West since the difference between the absolute values of the longitudes is greater than zero. The travel ratio (0.002/0.001) is 2, so the drone will be issued the specified number of forward (north-facing) movement commands, along with ½ that number of east-directional movement commands. Once the movement is executed, the current position is evaluated and the values are recalculated. The process is repeated until the difference between the current position and the destination is less than a user-defined distance (default is 15 meters). The difference in distance is a function implemented by the C# map library that implements the GeoCordinate class.

Angle Finding Method
The angle-based route finding algorithm differs from the convergence method in that the only directional movements of the drone are in the forward direction and the drone is simply rotated to ensure it is pointed in the direction of the destination. The drawback to this method is that the exact heading of the drone is difficult to determine and there are issues issuing repeated yaw (turn) commands to the drone in a repeated fashion. However, the main benefit is that the drone takes a more direct path to the destination than with the convergence method.

The premise behind this algorithm is to think of the unit circle and have the degrees from north be the reference angles from 0* (pi/2 radians) [clockwise increase] to 360* which is also (pi/2 radians). The drone begins in a north facing direction, the angle between the current position and the destination is determined used arctan(latitudeDifference / longitudeDifference). The resulting value of the theta angle is not correlated to the 360* reference angle method we are using, so that angle is converted to the 360* angle based on the direction of travel in the x and y planes; the directions are used to determine the quadrant of the angle and thus determine the appropriate 0-360* angle to assign to the path of travel. Once the angle has been determined, the yaw speed of the drone is set and repeated for a specified amount of time to rotate then needed amount. The heading angle of the drone is assigned the 360* angle measurement and forward movement commands are issue for a certain amount of time based on the distance to the destination. This process is repeated until the distance to the destination is within the specified number of meters.

Determining the angle between the current position and the destination is not difficult simply because it is the arctan(latitudeDifference / longitudeDifference) where longitude difference is found by taking the difference between the absolute value of the destination longitude and the absolute value of the longitude at the current position; however, for the angle determination to be correct in accordance with the angle anticipated, it is necessary to take the negative of the longitude difference because more westward positions have a larger positive absolute value and thus for it to be in the correct quadrant, take the negative of the longitude difference.

Converting the angle to the 360 degree angle in relation to true north is accomplished by considering the direction of travel in both the x and y planes. The arctan of the latitude difference over the longitude difference results in a value between 0-90 in quadrant 1, -90-0 in quadrant 2, 0-90 in quadrant 3, and -90-0 in quadrant 4 (all assuming moving counterclockwise). Considering this factors, if the direction of x-travel is in the East direction, the 360 angle is determined by calculating 90-(theta). In the event that x-travel is West, the angle is calculated with 270-(theta).

The amount of turn required to orient the drone in the correct position is found by finding the difference between the calculated 360 degree angle between the current position and the destination and the current heading value (0-360). If the difference is positive, rotation right (clockwise) is appropriate, and vice versa. However, if the difference is greater than 180, it is more efficient to turn the opposite direction - therefore, the angle change [(360DegAngleToDest - currentHeading)-360] is the angle needed for turning the drone. You could always turn the clockwise direction, but it is much more efficient to turn the minimal amount needed.

Once the change angle is calculated from the previous step, the commands to rotate the drone must be made. The drone's yaw speed is set to a default of 200 units. Therefore, to turn 200 degrees, the drone must rotate (receive commands) for 1 second [200 degree per second rotational speed]. Thus, we will always send yaw speed commands for 1 second to ensure consistency and reduce complexity, but the speed will be determined using the change angle. The speed will be the change angle divided by 200 - thus this the speed needed for the drone to rotate the change angle number of degrees in one second. The thread is simply paused for 1000ms when the yawSpeed is set on the drone, and then the speed is reset to 0 to stop rotation.

Once the angle has been found by the drone, the distance between the current position and the destination is determined using the distance function implemented by the XNA maps library.

The distance is divided by 10 and cast to an integer to obtain a multiple for forward movement. By determining the angle, the movements are much longer when the distance needed to travel is greater and as the destination is approached, the movements become much more fine coarse. The process is repeated multiple times along the path between the current position and the destination to ensure the angle is accurate and a variation or misstep by the drone or GPS calculation doesnt undermine the entire flight path. The degree and number of steps is determined by user variables.
The implementation has been described thoroughly above, however, the algorithm is not currently working. There is an issue related to sending the drone multiple yaw speed commands in a repeated fashion in a very fast pattern. I believe the issue to possibly be the drone receiving one yaw speed set command and setting that speed and turning ever so slightly at that speed but then waiting for an additional rotation command since the speed may be auto reset. The implementation we currently employ is setting the yaw speed once and simply sleeping for 1 second then resetting to zero.
A potential solution is implementing a loop to send the yaw speed set commands multiple times, the number of time needed to achieve the change angle that is needed. This is not confirmed but it is a potential solution. Additionally, wind could possibly affect this implementation significantly because the angle could be changed throughout the flight between each step and thus the drone would converge to the destination but it might be at a very slow rate. However, if the implementation was completed successfully, it is more direct than the convergence method. Also, an area of improvement is that the heading angle should be recalculated based on the change in position between one step position and the next step position, however, the GPS accuracy we are currently dealing with prevents this from being effective and/or useful. The angle finding method is very sleek and efficient, but more calculations and more potential points of failure reduce the reliability of the system. Finally, determining the rotation speed and accuracy of the drone is dependent on many factors such as motor condition, wind, battery charge, ect., thus, it is also quite difficult to ensure the calculations and method will be ideal.

In-Order and Shortest Path

The user has the option to chose between in-order traversal or shortest path traversal between the waypoints they selected on the map within the smartphone app. The in-order traversal algorithm traverses the waypoints in the order they were placed on the map. The shortest path algorithm attempts to navigate and traverse all the given waypoints in the shortest possible path. As we know, this is a variation of the traveling salesman problem - it is not possible to solve in real-time - but we implement a locally optimum algorithm to hopefully select a path that is the local minimum and hopefully the global minimum path.

The in-order traversal algorithm uses a queue as the data store for the waypoints that are attempting to be traversed. When the function is called with the queue, the algorithm determines the current position and dequeues the first waypoint. The follow-route function is called with the current position and the dequeued waypoint as the destination. The follow-route algorithm either executes the convergence based or angle reference method for route finding based on another function argument. Regardless of the selected option, once the destination (distance within given accuracy range), the actual found destination is returned to the in-order traversal algorithm. The returned position is now the current position and another waypoint is dequeued. This process is repeated until there are no elements in the queue implying that the last waypoint has been found, thus the destination has been reached.

The shortest path algorithm is a bit more interesting than the in-order traversal option. The traveling salesman problem can be solved for the number of waypoints we will likely be encountering with our application, however, to ensure real-time performance we implemented a locally optimum selection algorithm. The algorithm first converts the queue of waypoints to an array for easier manipulation and traversal. The algorithm is a while loop that is dependent on a boolean flag and the execution continues until every array object is null, implying each waypoint has been found.

Each itteration of the while loop in the shortest path algorithm ensures there are non-null objects in the array, and then uses the current position to calculate the distance to the remaining non-null location objects. The temporary distance is found between the current location and the first non-null location in the array and then depending on if a shorter distance is found through the traversal of the array, the index of the shortest path object is recorded and the minimum distance is updated and the comparisons with other locations in the array continues. Each waypoint requires comparisons with (n) elements in the array and thus the runtime of the algorithm is O(n^2), however, this is non deterministic simply because there is significantly more time spent sending movement commands to the drone. Once the comparisons are made with all of the array elements, the index of the array is used to retrieve the ideal waypoint and the route-finding algorithm is called using the current position and this waypoint that was determined to be the optimal selection. Additionally, the index at which the waypoint was stored is set to null to mark the waypoints that have been traversed. When the destination found function returns the actual location, the process is repeated. The algorithm continues until each array position references a null object indicating the entire path has been traversed. There is obviously a possibility that this may not be the optimal path selection, but for the intended use of our system, this is a perfectly acceptable solution.

5 Experimental results

Phone App Ease of Use

For testing the ease of use of our phone app we asked one of our classmates to use it. We powered up the drone, gave him the phone, and provided no other assistance. He started up the app and hit the fly button because it seemed the logical button to press to fly the copter. He pressed several of the buttons but nothing happened. He then saw the connect button and realized that he had to connect to the drone.
After connecting to the drone he quickly figured out what each of the directional buttons did and was able to move the drone about using those buttons. After a short time using the buttons he tried to figure out the accelerometer controls. He quickly found the button for the accelerometer and after several attempts at using it he figured out that he had to hold the button. He flew around a bit using the accelerometer and then landed the copter.
After the test we received feedback on several areas that could be improved in later modifications.

The first was that it was not apparent enough that a connect was required after navigating to the screen for manual flight. The suggestion was to either make the other buttons on the screen unusable until the connect button was pressed, or to make the connect button take up more of the screen until a connection has been established.

The second was that he expected the right and left buttons to strafe right and left when in fact they turn the drone right and left. The suggested change was to make it more apparent that the buttons rotate the drone by using arrows that turn.

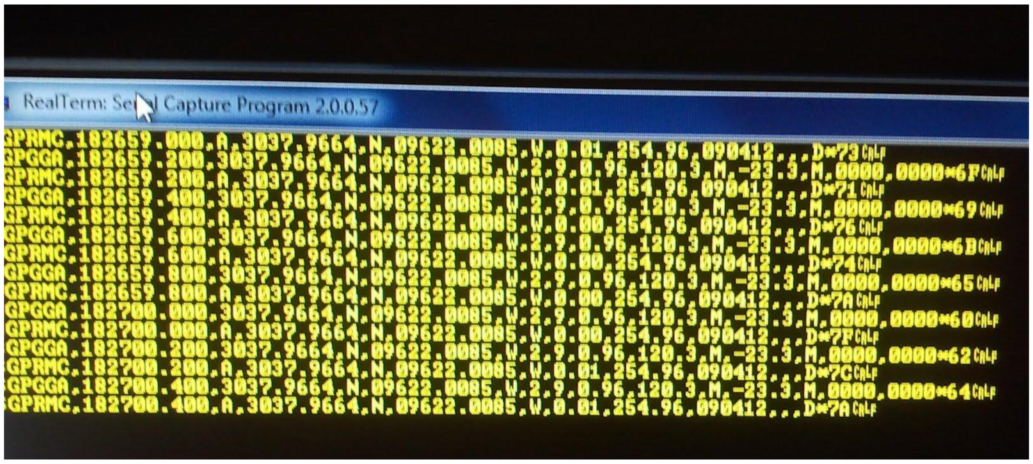
The third was that the button for accelerometer control was not easily noticeable because it was rather small and in the bottom corner of the screen. It was also not apparent that it was the button for the accelerometer control. The suggestion was to make the button bigger and a more noticeable color, and to label it with text like “Accel” or similar to make it apparent that it controls the accelerometer controls.

Due to time constraints we were unable to do testing with other people outside our team but with additional time we would perform similar assessments with people in different age groups and education backgrounds to further prove ease of use.

We did extensive internal testing for not only ease of use of the ui but also for the general responsiveness of the interface. We extensively tested the application user interface to ensure that there was no input that would make the app crash and that each button served its intended function. We also tested the ui extensively for how natural controlling the drone is. We flew the drone around while tuning the flight parameters for what felt most natural, as if the drone was moving exactly according to the movements of the phone, and so that rounding corners and avoiding obstacles was easy.

GPS Accuracy

Because the GPS modules burnt out before we were able to test them thoroughly, the GPS accuracy is not clear. However during the setup phase to get the GPS module working, stationary measurements were taking. The picture below is a screenshot of the GPS module working while attached through the FTDI board to a PC. The difference of the modules coordinates vs. the real position was only 3 meters off.

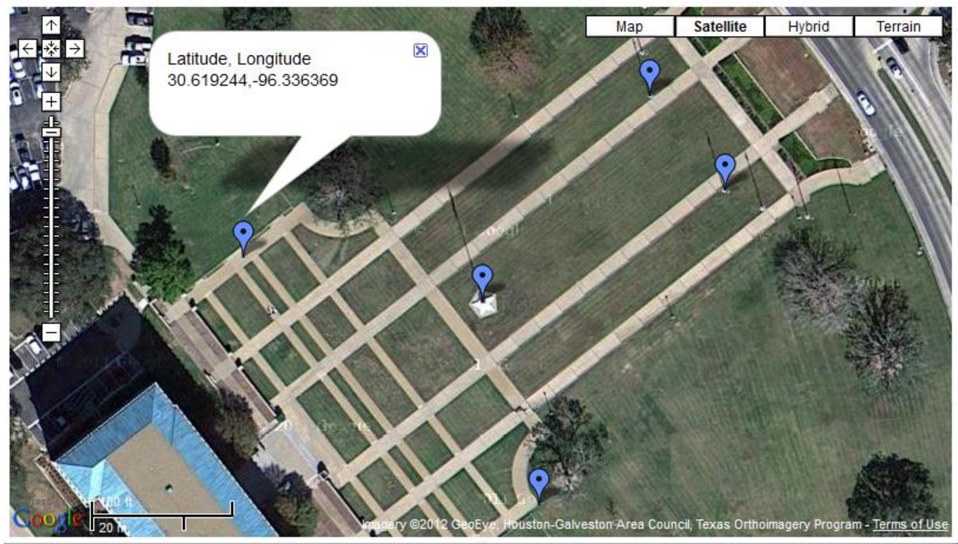


```
GPRMC,182659.000,A,3037.9664,N,09622.0085,W,0.01,254.96,090412,,,D*73CRLF
GPGGA,182659.200,3037.9664,N,09622.0085,W,2,9,0.96,120.3,M,-23.3,M,0000,0000*6FCRLF
GPRMC,182659.200,A,3037.9664,N,09622.0085,W,0.01,254.96,090412,,,D*71CRLF
GPGGA,182659.400,3037.9664,N,09622.0085,W,2,9,0.96,120.3,M,-23.3,M,0000,0000*69CRLF
GPRMC,182659.400,A,3037.9664,N,09622.0085,W,0.00,254.96,090412,,,D*76CRLF
GPGGA,182659.600,3037.9664,N,09622.0085,W,2,9,0.96,120.3,M,-23.3,M,0000,0000*6BCRLF
GPRMC,182659.600,A,3037.9664,N,09622.0085,W,0.00,254.96,090412,,,D*74CRLF
GPGGA,182659.800,3037.9664,N,09622.0085,W,2,9,0.96,120.3,M,-23.3,M,0000,0000*65CRLF
GPRMC,182659.800,A,3037.9664,N,09622.0085,W,0.00,254.96,090412,,,D*7ACRLF
GPGGA,182700.000,3037.9664,N,09622.0085,W,2,9,0.96,120.3,M,-23.3,M,0000,0000*60CRLF
GPRMC,182700.000,A,3037.9664,N,09622.0085,W,0.00,254.96,090412,,,D*7FCRLF
GPGGA,182700.200,3037.9664,N,09622.0085,W,2,9,0.96,120.3,M,-23.3,M,0000,0000*62CRLF
GPRMC,182700.200,A,3037.9664,N,09622.0085,W,0.01,254.96,090412,,,D*7CCRLF
GPGGA,182700.400,3037.9664,N,09622.0085,W,2,9,0.96,120.3,M,-23.3,M,0000,0000*64CRLF
GPRMC,182700.400,A,3037.9664,N,09622.0085,W,0.01,254.96,090412,,,D*7ACRLF
```

GPS Module Data on PC

To test the GPS application’s accuracy, three different phones were taken to 5 different locations and compared to the Map coordinates of those location using http://itouchmap.com/latlong.html. The phones tested were the iPhone 4S, Nokia Lumina 800, and Samsung Focus Flash.

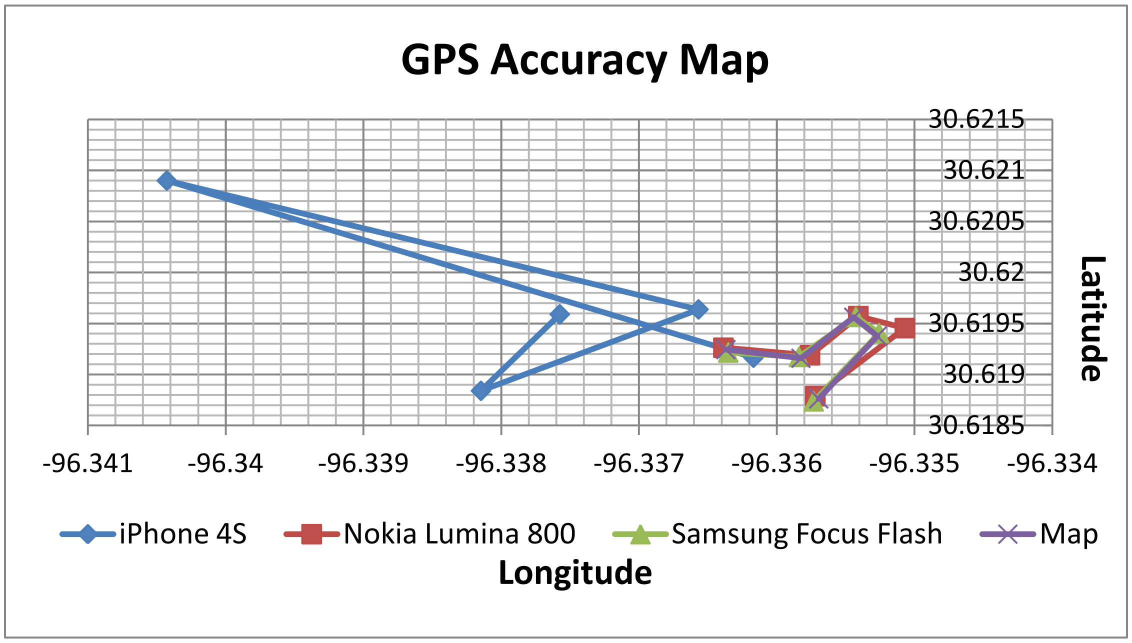
Latitude and Longitude of a Point


Map
Satellite
Hybrid
Terrain
Latitude, Longitude
30.619244,-96.336369
100 ft
20 m
Imagery ©2012 GeoEye, Houston-Galveston Area Council, Texas Orthoimagery Program - Terms of Use

Map of locations used to evaluate GPS data and receive measurements

GPS Accuracy Table

LOCATION		**iPhone 4S**	**Nokia Lumina 800**	**Samsung Focus Flash**	**Map**
1	Latitude	30.61959	30.619262	30.619216	30.619244
	Longitude	-96.337575	-96.336387	-96.336352	-96.336369
2	Latitude	30.61884	30.619189	30.619173	30.619159
	Longitude	-96.338147	-96.335759	-96.335839	-96.335824
3	Latitude	30.619638	30.619572	30.619565	30.619561
	Longitude	-96.336568	-96.335408	-96.335426	-96.335441
4	Latitude	30.6209	30.619455	30.619406	30.619377
	Longitude	-96.340427	-96.33507	-96.335257	-96.335271
5	Latitude	30.619165	30.618788	30.618737	30.618761
	Longitude	-96.336169	-96.33572	-96.335732	-96.335695



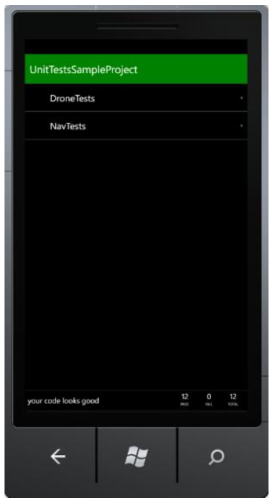
		Difference from Map		
LOCATION		**iPhone 4S**	**Nokia Lumina 800**	**Samsung Focus Flash**
1	Latitude	0.000346	1.8E-05	2.8E-05
	Longitude	0.001206	1.8E-05	1.7E-05
2	Latitude	0.000319	3E-05	1.4E-05
	Longitude	0.002323	6.5E-05	1.5E-05
3	Latitude	7.7E-05	1.1E-05	4E-06
	Longitude	0.001127	3.3E-05	1.5E-05
4	Latitude	0.001523	7.8E-05	2.9E-05
	Longitude	0.005156	0.000201	1.4E-05
5	Latitude	0.000404	2.7E-05	2.4E-05
	Longitude	0.000474	2.5E-05	3.7E-05
	Averages	0.0005338	3.28E-05	1.98E-05
		0.0020572	6.84E-05	1.96E-05
	Medians	0.000346	2.7E-05	2.4E-05
		0.001206	3.3E-05	1.5E-05
	Maximums	0.001523	7.8E-05	2.9E-05
		0.005156	0.000201	3.7E-05

The Samsung Focus Flash turned out to have the least error in GPS accuracy so that Phone was used for the GPS application on the drone. One other factor that we did not account for while flying was the possibility of the magnets in the drone to throw off the GPS data. We did try to use a compass application to get heading, but the magnets affected the performance too much.

Route Finding

Unit testing was used to confirm and validate the expected and intended functionality of the automatic navigation functions. The implemented functions and classes integrate together and to ensure there is not a weak link, unit tests were used to confirm expected behavior in an isolated, controlled simulation. GPS data was generated and updated using a function that simply evaluates the conditions and variables used in the followRoute functions and estimates the change in GPS positioning resulting from each iteration of the algorithm. This is stimulated and is much more ideal than real world situations considering the accuracy of GPS devices, but it proves the algorithm properly executes the algorithm which provides the logic to converge to the intended destination. Essentially, the tests for the automatic route following algorithm was executed on a given start and end point and the destination that was "found" by the algorithm was evaluated to determine if it was truly within the given distance accuracy that is user defined. We defined the accuracy to be approximately 15 meters because defining the radius to be shorter leads to long, delayed "recognition" of the destination since the accuracy of the GPS can easy vary between 5 and 10 meters quite consistently.

- Navigation tests:
    - destinationFound()
    - findLocation()
        - in all directions
    - inOrderTraversal()
    - shortestTraversal()
    - followRoute()
    - angleToDestination()
    - calc360Angle()



Debugging with Step Through for Evaluation

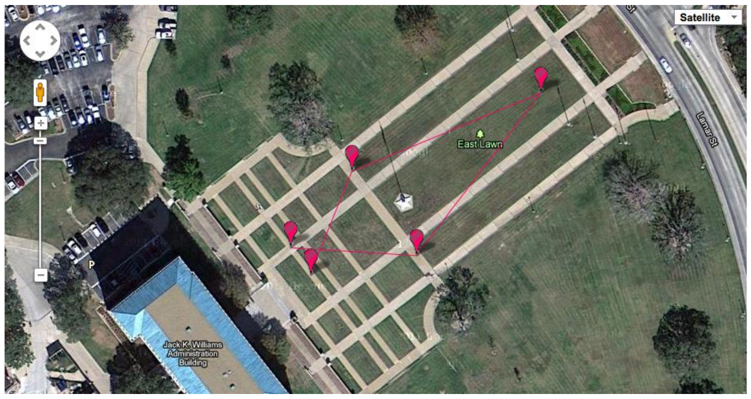
Visual Studio allows debugging with step through to evaluate the correctness of code and to determine if logic errors are present by walking through the entire series of execution. Step through and breakpoint debugging was used to find errors, but it was also used to determine that values were being appropriately configured and updated upon each execution of the algorithm.

As an example, to test the "In Order" and "Shortest Path" traversal algorithms, we saved 3 waypoints and set breakpoints when each intermediate waypoint is "found". At each breakpoint the algorithm should select the next appropriate waypoint as the next destination and this can easily be confirmed since the path is defined and we know how the traversal should work in theory, so we can view the execution to ensure the algorithm is functioning correctly in practice.

Field Testing

We have conducted many field tests to evaluate the autonomous flight capabilities - we were able to confirm waypoints can be found and entire paths can be navigated. The method we used to determine the practical functionality of the route finding algorithms was to define points and “hand-craft” a path. The waypoints are selected by using a Google Maps add-on that gives the exact latitude and longitude values once a pin has been dropped at said point. To ensure this was consistent, we evaluated the points with both Windows Phone devices, an iPhone, and cross referenced another location service online. We would not have a deterministic test if the GPS data was incorrect or was inconsistent with the expected values based on observed data. We conducted in-order path traversal trials and the successful demonstrations were repeated multiple times but unfortunately we did not complete an official demonstration successfully. However, the logic behind the algorithm was clearly displayed as the current distance readout on the phone was clearly converging to the destination before a series of unfortunate incidents occurred. We could have selected a better location for the official demonstration since there were many obstacles to contend with, but it was the easiest and closest location to demonstrate our project.

Wind affected a few of our trials and GPS issues prevented a successful flight on our final attempt. However, we have video documentation confirming the operation of our drone and the autonomous flight capabilities. We successfully navigated a 206 meter (675 foot) path between 3 waypoints and successfully found our defined destination within a distance of 7 meters.



*

Distance from landing point to specified destination: 24 feet (7 meters)

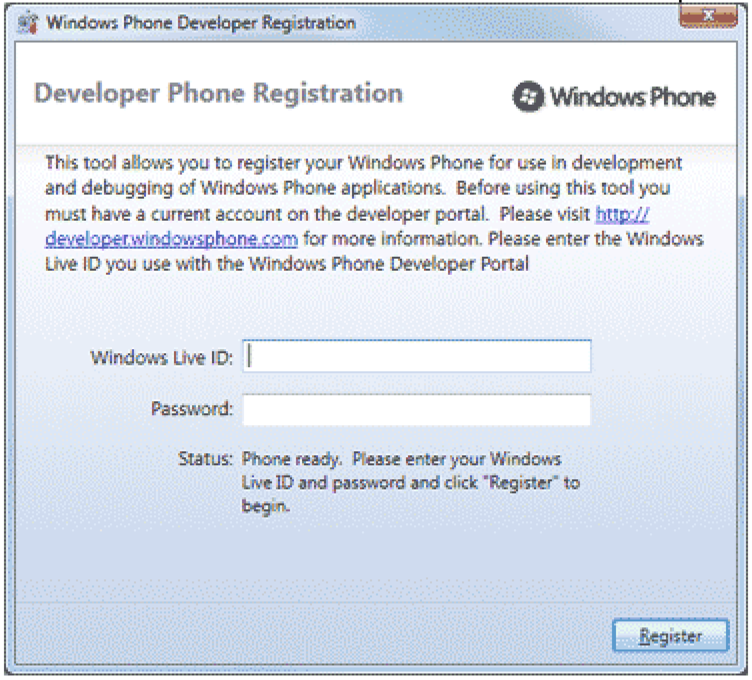
Defined path / intended path for drone to follow - 206 meters (675ft) - 3 waypoints and defined destination

6 User's Manuals

Visual Studio setup including getting code from CodePlex

As this app has not been released on the Windows Phone marketplace you must have the developer tools installed and have a development unlocked phone then download the source code. The following are the steps to do this:

- Install the Developer tools
    - Install the following
        - WP Dev tools
        - WP Dev tools January 2011 update
    - If you do not have a copy of visual studio then the free version will be installed along with the developer tools
- Create an App Hub account
    - Register here
    - Creating an app hub account will allow you to unlock up to three Windows phones for development
- Unlock your phone
    - Start the Zune software on your computer
    - Connect the phone to the computer
    - Launch the Windows Phone Developer Registration Tool



Windows Phone Developer Registration

Developer Phone Registration

This tool allows you to register your Windows Phone for use in development and debugging of Windows Phone applications. Before using this tool you must have a current account on the developer portal. Please visit http://developer.windowsphone.com for more information. Please enter the Windows Live ID you use with the Windows Phone Developer Portal

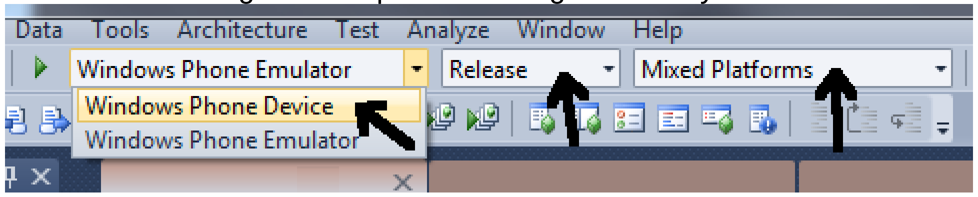
Windows Live ID:

Password:

Status: Phone ready. Please enter your Windows Live ID and password and click "Register" to begin.

    - Log in with the Windows Live ID associated with your App Hub account
    - Finish the wizard
- Download the source code
    - The source is located here
    - The most recent stable release can be found through the download button on the right.
    - The latest developer snapshot can be found under Source Code
    - Unzip the code to a safe place on your hard drive
- Deploy the code to your phone
    - Open up the Zune software and connect your phone

- Open up the Solution downloaded in the previous step
- Once Visual Studio Loads the solution Find the drop box in the toolbars to the left of the build configuration options. Change it to say Windows Phone Device



- Make sure the solution configuration is set to Release/Mixed Platforms
- Select Deploy Solution from the build menu and wait for it to finish
- Your phone must have its screen unlocked during the deploy process

Drone assembly and parts repair

The AR.Drone comes as a pre-assembled ready to fly kit with an indoor hull, outdoor hull, flight platform, and a battery. All repairs or replacements should be done with parts made by Parrot for the AR.Drone. Parrot has actually made a series of YouTube videos to show users how to repair and replace parts on the drone. (http://ardrone.parrot.com/parrot-ar-drone/usa/repair-your-ar.drone)

GPS module installation (2 pages)
Look at pages 24 - 34.
Attaching headers to the pads of the GPS make it much easier to connect the GPS module to other devices. This process can be tricky, but luckily sparkfun.com provided an easy to use little manual for connecting a 5-pin header (http://www.sparkfun.com/tutorials/176).

Installation and operation of phone apps

Install - AutoFlight, Drone Controller App

A similar process can be followed to install the main AutoFlight application on a Windows Phone. In the package explorer, find the DroneControllerApp, expand and ensure all dependencies and resources are present. Build the project and correct any errors by adding resource dependencies to the Resource folder. Deploy to the emulator to ensure proper compilation and explore the application before installing on your phone. In theory, it is possible to use the emulator to control the drone in the event the computer with the emulator is connected to the drones wireless network; however, this is not suggested as it has not been tested for proper functionality. Additionally, you can debug and step-through the application using the emulator by right clicking on the project, and choosing "Debug, Step into new Instance". The debug mode allows you to see the actual execution steps and procedures of the application and allows easier debugging in the event there is a miscalculation or logic error in the program. To make any updates, modify the C# code implemented for each of the respective pages; this can be found under the *PageName.xaml* files. Each page is independent, but inheritance can be performed from page to page if needed.

Deploy to the phone when you are ready by simply changing the target device within Visual Studio to Windows Phone Device after connecting phone and launching Zune. The app should install and it should be available for you to use and fly the drone. Debugging can also be performed by keeping the phone connected, ensuring the target device is the phone and then stepping into a new instance similar to the method described above for the emulator. The app can be pinned to the main screen as a Tile by simply holding down the main application icon and selecting "Pin to Start" - this is also how applications are uninstalled if needed.

Install - GPS Tracker

The entire solution should be loaded into your Visual Studio package explorer, if not, see the above steps. Expand the GPS tracker app and ensure all resources have been correctly configured in the Resources folder. In the event something is not present, right click on the folder and select add resources. Navigate to either projects or system libraries depending on what might be needed. The main logic / code for the application is located in the MainPage.xaml.cs file under MainPage.xaml; any updates and changes to the C# file will be reflected once the project is rebuilt.

Attempt to build the solution, right click on the top level GPS Tracker folder, and select "build". The application should compile and any warning / error messages will be displayed in the console. If the package builds without errors, attempt to deploy to the Windows Phone emulator (check the box near the top middle of the Visual Studio window and ensure emulator is selected - see above picture for reference). Attempt to open the application by tapping the icon. When the application starts, click start and ensure the default GPS data appears to confirm functionality.

Upon the completion of the previous steps, connect the actual Windows Phone to the computer via USB. Start Zune on the computer and ensure the phone is recognized. Once Zune loads, change the target deployment device drop down box from Emulator to Windows Phone Device. Repeat the steps to deploy to the emulator mentioned above, but since the drop down box was changed, the application will be deployed to your Windows Phone. Ensure you are connected to either a 3G connection or a WiFi connection, and then attempt to launch and start the GPS tracker. If the GPS does not load immediately, press the Stop button and click Start; this should resolve the issues.

Install - Unit Test app

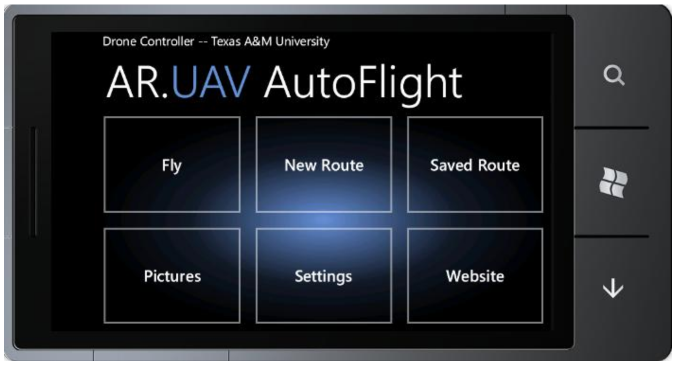
The unit testing framework cannot be executed within Visual Studio due to limitations with Windows Phone development environment. Additionally, unit testing is not implemented strictly for Windows Phone. External libraries must be added if you plan to start a new unit testing project or if there is an issue with the configuration. Essentially, the unit tests for Windows Phone are required to appear as Silverlight unit tests, but they have to be embedded within a Windows Phone app so there is a way to execute them against WP C# code. The process is very convoluted and difficult, but the configuration should be complete if the source code for the entire project is obtained from our CodePlex account. However, if the libraries for this setup are needed, or if you would like more information, please see this link: Unit Tests with WP7.

Additionally, the link above describes the basic UnitTestApp that we have implemented to use as the framework for the unit tests that have been written. The app executes the unit tests inside of the app monitors the results thus providing a real-time graphical display of the testing results. Installation of the app follows the same procedures as mentioned about for the previous two apps.

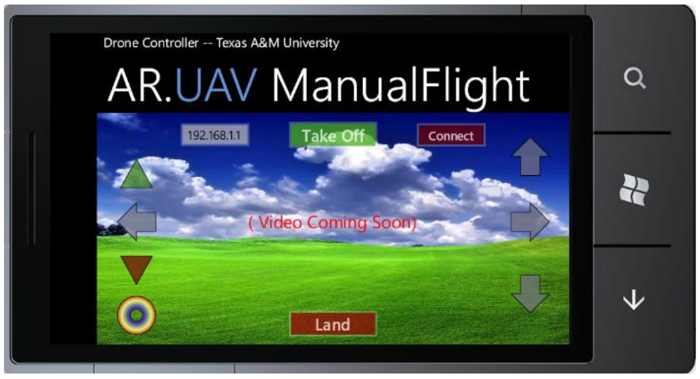
The unit tests are located in C# files within the project and testing classes are constructed to encapsulate the tests into fundamental categories. For example, DroneController tests and NavigationalTests. The test files can be expanded and new files can be added; the tests should be automatically included in the next execution of the application if it is rebuilt. Also, the unit test app can be run in either the emulator or on the physical Windows Phone device.

Operation - AutoFlight, Drone Controller app

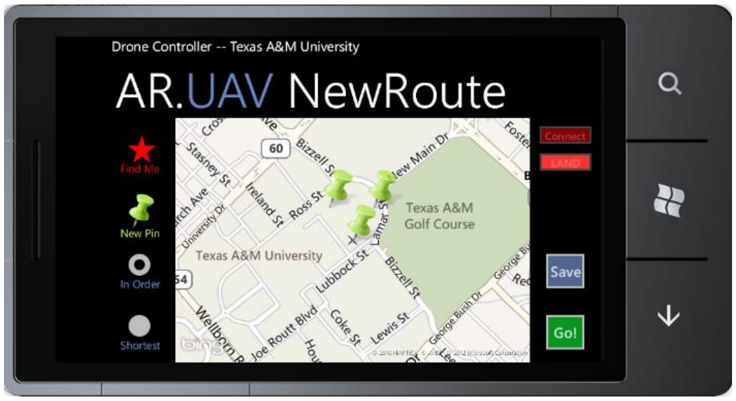
The welcome screen allows the user to select, “Fly”, “New Route”, “Saved Route”, “Pictures”, “Settings”, and “Website”. The application launches in landscape mode and the layout is a bit distorted when placed in portrait orientation - there is a limitation with the Windows Phone in the ability to force one orientation by default. However, the app is meant to be easy, intuitive, and visual appealing. The structure of the application mimics the official application released by Parrot Inc. (manufacturer of AR.Drone) for the iPhone to ensure the experience is reasonably consistent between platforms.



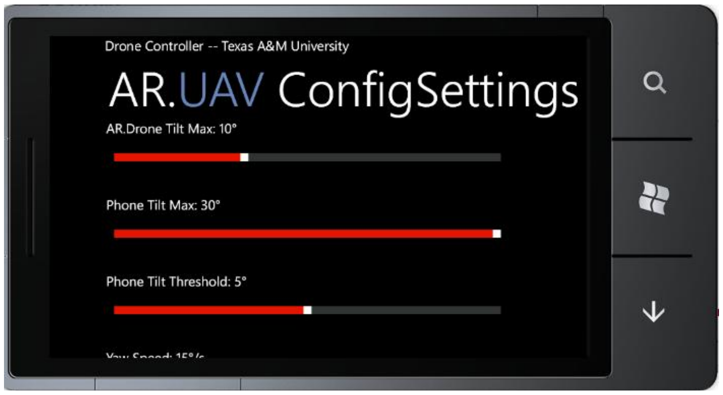
The “Fly” button leads the user to the main flight screen. This page has many buttons and options, but user tests have shown the arrangement is reasonably intuitive. One of the main issues, however, was the requirement to click “Connect” before performing any drone operations. In the future, this could be automated upon launch of the application, but issuing explicit connect commands ensures the commands are handled correctly between the phone and the drone. The main movement commands are the grey arrows on the edges of the screen - the up and down arrows moves the drone forward and backward respectively and the right and left buttons execute drone strafe movements. The green and red triangles on the left side adjust the flight height of the drone. Take off and Landing are placed in the middle of the screen at the top and bottom of the screen respectively. Finally, the rainbow/multi-colored circle allows the drone to be controlled using accelerometer controls. The button must be held in the down position for the accelerometer controls to be active to ensure there is no mistaken flight commands sent leading to unforeseen / unplanned drone action. The intention is to have video streaming from the drone active as the background video source, but due to technical difficulties, this was not implemented.



The “New Route” button leads to the AutoNavigation screen. A map is displayed in the center of the screen which has its position centered on the current GPS reading from the Windows Phone device. All zooming, panning, and movement commands needed for the path can be conducted using basic multi-touch gestures. The “Find Me” button functionality is implemented when the phone app launches and is not needed - planned to remove button, but time was a limited resource. The place waypoint button indicated with a green push pin, allows the user to mark a waypoint and/or destination on the map to use in the automatic navigation of a specified route. The waypoints are placed in the center of the map and the map has an “X” placed in the middle to ensure accuracy when placing the push pins. The pin should be placed on the map when the button is pressed. Below the push pin, the user can select in-order or shortest path traversal between the points that are specified with the waypoints. A connect button and a land button are placed in the top right corner of the screen but are not usually needed as the app auto connects and the drone will automatically land when the destination is located. A text box on below the land button will actively display the incoming GPS readings and allows the user to confirm the data is being received correctly. The “Save” and “Go” buttons are below the text box - the “Save” button is intended to save the queue of waypoints to local phone storage and allow the user to use this route in the future, however, it is not directly implemented at this point. However, the “GO!” button does execute the route finding algorithms to traverse the given path of waypoints. When a user presses the “Go!” button, the drone takes-off, pauses to calculate initial values and then begins executing the algorithms based on the selections of the user. Additionally, a transparent green box is displayed over the map displaying the current distance from the drone to the next waypoint - this information is useful and the progress of the drone can be tracked to ensure it is converging to the specified waypoint. Once all waypoints have been traversed, “Destination Found” is displayed on the green box, the box is closed, and the drone lands.



The “Saved Routes” button is not functional at this point, but the future implementation is straight forward in that it simply has to read the local storage on the physical device and select the stored waypoint queues associated with saved routes. A simple list can be displayed and upon selection, the route could be loaded into the AutoNavigation page and the user could simply perform the same functionality as if they had defined a new path.
“Pictures” has not been implemented either because of the difficulties related to processing the streaming video from the drone. Ideally, the “Fly” screen should have the option to capture video and images during flight once the video has been integrated so the flight can be documented; this would serve as evidence of parking infractions and other important information that would be needed when the drone is applied to the intended use case of monitoring parking areas. The “Pictures” button will simply allow the user to navigate to the local camera photo/video storage to find the videos / pictures that are needed/in question.
The “Settings” button allows the user to customize the settings for the drone when flying in the manual mode using either static or accelerometer based movement controls. The settings modify max tilt, sensitivity, response accuracy, timing, max height, along with a few others. The settings allow more advanced / experienced users to customize the flight experience and fine tune the drone’s performance.


Drone Controller -- Texas A&M University
AR.UAV ConfigSettings
AR.Drone Tilt Max: 10°
Phone Tilt Max: 30°
Phone Tilt Threshold: 5°

Finally, the “Website” button opens a web browser to our project home page. The home page has information about the project, embedded project videos, documentation, and group contact information. This simply allows the user to gain quick instruction or find information directly from the app if it is needed.

Operation - GPS Tracker

The GPS tracker requires a network connection, even if there is not internet access, so ensure you are connected to either 3G or a WiFi network before launching the app. The app is very simple in that it launches and operates with only 2 buttons. The Start button automatically joins a multicast group to transmit data, starts location services, and listens for GPS data. When a new value is received / processes, it is automatically decoded, parsed, displayed on the screen in real-time, and immediately sent to to the mutlicast group. The latitude and longitude are processed from the GeoCordinate information and the directional heading of the phone is determined using the readings of the internal accelerometers, magnetometer, and other sensors. The latitude, longitude, and directional heading values are combined, separated with a single comma, and placed inside a UDP packet for transmission. The processing and transmission sequence happens each time the GPS information is updated / received - the heading is updated much more frequently, but only processed when GPS data is ready. The user can press the Stop button at any time to turn off location updates and turn off the senors used for heading determination. On a side note, the GPS data update fairly frequently and it is crucial to ensure a few updates are automatically processed before connecting with the main AutoNavigation app simply to ensure the app has completed all of the initialization sequences. Also, the heading information could be very useful, however, the magnets and battery on the drone completely alter the directional reading in a HUGE way - to the point that it is completely unusable the error is so high. Also, the app is configured to ignore the phone sleeping but this cannot be confirmed as to whether all Windows Phones support this feature and it is simply easier to prevent phone lock for 5 minutes in the settings of the actual phone. Finally, be aware of the time spent using the app because it is very resource intensive in regards to battery usage and data transmission.

Operation - Unit Test app

Upon launch of the application, a start screen will be displayed and it will automatically navigate to the unit test start screen - the user has the ability to select tagged tests or run everything and, by default, the “run everything” option is chosen automatically after 5 seconds if there is not user interaction. The tests will execute and real-time information will be given as to the passed and failed tests. Once the tests have completed, the test suite is divided by test class at the top level and the user can drill down to the individual tests to gather specific information if needed. In the event a test fails, the data related to that failure will be logged into that actual test case within the application and the user can attempt to solve the issue based on the stack trace that is displayed. If the tests pass, there is no significant information stored, except execution time which can be useful for determining the speed of the app functions, however, it is running inside of a test suite / framework within an actual app, so the speed is likely to not be accurate, however, it is a good method for time comparisons between functions. When tests are added in the C# test files, the tests will automatically be executed if the project is rebuilt before it is redeployed to the emulator or the actual phone. Also, the emulator struggles with testing motion API functions, but it can be simulated with additional configurations within the emulator itself. Also, the same is true if GPS data is needed for simulations - data can be generated / programmed for the individual tests, however, this is a challenge and I simply implemented an ideal GPS modification function for use when testing to simulate the movement of the drone when the movement commands of the algorithm are executed. The unit test app provides a good base for unit testing since it is typically such a pain to test windows phone applications and it is relatively easy to add new test cases. However, the time spent testing is significantly increased since the tests are executed in the actual emulator and not simply run within the Visual Studio environment.

7 Course debriefing

Team Management

Tasks

At the beginning of the semester, each of us identified our strong skill-sets / our preferred work areas and tasks were assigned based on that to ensure we were putting the right people on the most appropriate tasks. David chose hardware over software so he took the lead in drone assembly, GPS module installation and data transmission, along with finances, public relations, and purchasing. He also looked into firmware specifications to get an idea for the proxy app that would have been needed for the GPS and video stream. Alex wanted a software focus, so he took the lead on the phone controls (static and accelerometer) and communication between the drone and the phone. Also, Alex was assigned the task of decoding and displaying the live video feed from the drone. John had experience with unit testing so he took charge of the testing framework and the website; throughout the project, he picked up assignments that were either in danger of not being completed or unforeseen. In addition to being the team lead, John create the application UI, coded the route finding algorithms, and he wrote the GPS transmit app for the second Windows Phone.

The division of tasks worked relatively well since the tasks were aligned with our team's strengths and this would definitely be repeated. At the beginning John was not assigned specific tasks outside of unit testing and the website, which was good since he was able to pick up the dropped tasks, but if this had been more planned, more time would have been available for the entire team for their specific responsibilities.
Consistent completion of the tasks proved to be a challenge for each team member and we were very deadline driven which lead to issues related to the quality of the finished product. Internalized motivation to complete tasks on time and to the degree in which was discussed would have helped but also if there was a better way to hold each other mutually accountable, we might have been able to stay on track to a better degree.

Schedule

John set the task schedule and did the project planning throughout since he was the team leader. The schedule projected at the beginning of the semester was very aggressive and looking back there was an overestimation of available time in addition to and underestimation of the project complexity. However, the schedule was tweaked on a weekly basis to account for unforeseen circumstances and the tasks for the week for each team member were included on the weekly agendas as planned to complete items. However, there was an issue with all team members being aware of the tasks that were assigned and adherence to the schedule that John created. Prioritization of pending tasks was rarely in alignment with the schedule and timetable described in each agenda. As a remedy, John attempted to be more direct the second half of the semester with planned tasks and the tasks from the past week that were not completed, however, this proved to be little help. Communication from all parties could have been improved.

Issues with adherence to the schedule resulted from the inability to hold each other accountable - a huge issue on a team such as this. Some of the delays were due to technical difficulties, but a number of them were a result of poor prioritization and minimal dedication to the project. The group was able to pull together to push out updates and show progress but it was not accomplished in a very effective manner.

Collaboration

Brainstorming sessions were effective at the beginning of the semester to determine the project of choice and to solve initial problems, but group brainstorming tapered during the duration of the semester. Communication was difficult due to sporadic attendance to lab meetings and the deviation from the schedule that was set at the beginning of the semester. A poor start to the semester in regards to meeting attendance set the precedent for the remainder of the semester. Online communication was minimal but using Google Groups enabled our team to share links, ideas, and messages very quickly.

Document sharing was not utilized for the first report and huge issues with completion and integration pushed us to pursue online collaboration using Google Docs. Real-time document sharing has been a valuable choice for the team as the composition of documents has become much easier and it is easier to see who is working and completing sections and who is struggling.

Source control was handled by CodePlex and a team foundation server. The integration with Visual Studio is very good and the online repositories can be shared with the public which is a nice feature of our project. We have received many inquiries related to the project and have had double digit downloads of our application source code.

Safety and Ethical Concerns

Environmental, safety, political, and ethical issues and how they were addressed
Environmental

- *Power usage and parts (electronic/battery) disposal*
    - Maximize battery lifespan by using the entire battery cycle and fully recharging on a balance LiPo certified charger
    - Ensure parts were treated with care - testing with foam case when possible
    - Preventive maintenance on plastic rotor gears to maximize lifespan
    - In the future could research the most environmentally friendly batteries
- *Noise pollution*
    - Conducted experiments outside away from offices
    - Performed indoor studies outside of business hours
    - In the future could exchange the propellers with props having a higher bade number, which would lead to lower RPMs and lower noise levels

Health and Safety

- *Collision damage to people or property*
    - Tested drone in open areas
    - Used full, foam hull casing when possible
    - Flew drone in open fields away from trees and buildings
    - When testing navigation features, team members positioned to retrieve
    - In future, take laptop to control drone and test/tweak in field
- *Weather reliance*
    - Weather is outside our direct control, but increased weight to minimize effects of wind, however, didnt work well
    - In the future, make waterproofing improvements on drone

Political

- *Replace jobs of transportation officers*
    - Not at a stage where this is a direct concern
    - In the future - train the officers to use and control drone so their job description and responsibilities simply shift

Ethical

- *Video privacy and surveillance*
    - Did not capture and record video
    - In the future - post signs notifying people that they may be and their vehicles may be subjected to video surveillance so they can make the decision of whether they feel comfortable or not

Additional steps if project repeated or for future design consideration
One of the biggest concerns is safety, so designing better propeller protection would help very much; in the event with collisions, the protection device(s) would prevent damage to all parties, building, objects, walls, trees, or whatever the case may the receiver of the collision in addition to preventing damage to the drone and the internal components. Also, the video privacy issue will need to be addressed in the near future and steps should be taken to ensure the public is notified of the change and the potential of video surveillance.

Testing
As mentioned the above sections, our manual flight controls, accelerometer based flight controls, and automatic route finding algorithms are functioning correctly. Unfortunately, we were not able to demonstrate the navigational functionality during the project demonstrations for a variety of technical and weather related issue, but the algorithms have proved to be repeatable and accurate on multiple, repeated test situations in the field. Videos have been taken of a few of the trials are included in the files being submitted. The actual flight controls work very well and the dependability of the application we have created is quite good.

Unit testing
Unit testing has confirmed the expected and intended functionality of the non-trivial functions and classes we have implemented in our project - simulated data was used in the test and the algorithms result in the expected outputs and desired control sequences.

- Drone controller tests confirmed:
    - connectToDrone(), disconnectFromDrone(), takeOff(), land(), setPitch(), setYaw(), setRoll(), setGaz(), and setMoving()
- Navigation tests confirmed:
    - locationConstructor(), destinationFound(), findLocation() [in all directions NW, NE, SW, SE], inOrderTraversal(), shortestTraversal(), followRoute(), angleToDestination(), and calc360Angle()

Experimentation and field testing
We have conducted numerous field tests and, as mentioned above, the flight controls work very reliably and the application is quite intuitive. The autonomous flight capabilities has been confirmed with a saved path using in-order traversal. The trials were repeated multiple times and completed successfully. However, the official project demonstrations did not go well for our group. Wind affected a few of our trials and GPS issues prevented a successful flight on our final attempt. Also, we could have selected a location better - there were many obstacles to contend with, but it was the easiest and closest location to demonstrate our project.

- Manual controls
    - Very reliable, tested and confirmed. Demonstrated successfully.
- Accelerometer controls
    - Very reliable, tested and confirmed. Demonstrated successfully.
- Navigation
    - Dependability is an issue and there was not a successful official demo - but we have recorded multiple successful demonstrations
- Poor weather

    - Wind affects the drone significantly and greatly impacts the autonomous flight abilities of the drone. Significant challenge for manual flights as well - huge drain on the battery

User trials

One user trial was conducted with a fellow classmate that was unfamiliar with the product and we received very good feedback. The controls were intuitive and the app was easy to use, but button placement and description could have been improved. The subject of this user trial was to evaluate all flight control methods and functions.

- Ease of use
    - Straight-forward, easily learned the control systems within 3 minutes of experimentation without any instruction
    - Suggested small descriptions to minimize learning curve
- Intuitive flight controls
    - Suggested strafe action with manual side arrow, no rotation - change made
    - Possible popup hints menu could be nice on the first launch

Situations not tested

Poor weather

- Precipitation
    - This situation hasn't been tested, but there are not plans since the drone is not waterproof and water is likely to cause permanent damage to the electronic components

User trials

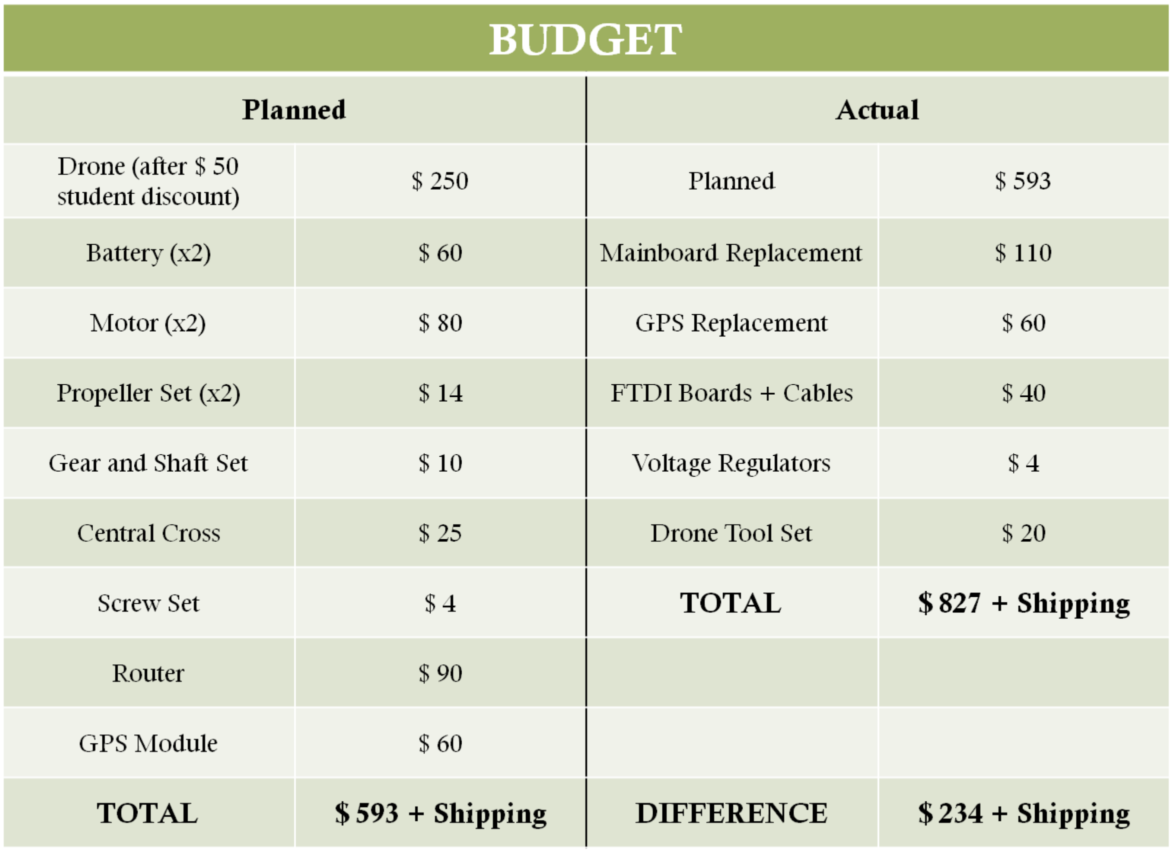
- Different demographics - have only conducted user trial with classmate
    - Young potential users
        - expect to pick up app quickly, but anticipate many crashes due to reckless type of attitude when controlling a "toy"
    - Older potential users
        - expect older users to experience more of a learning curve and may struggle significantly since not very willing to "experiment" with controls and available functions
    - Non-native US users
        - all wording is in English but intuitive flight controls hopefully speak for themselves - expect issues simply from confusion with words
- Autonomous flight
    - User trial
        - No tests conducted with users evaluating the ease of use related to the route finding capabilities of our system
        - Need to test the route definition, and selection of options for path traversal

Additional verification and testing

- All user trials mentioned above
- Field and user trials of autonomous flight
    - Route definition
        - saved path functionality has been tested in the field but user definition of routes has only been tested in simulations
    - Path traversal
        - only in-order traversal has been tested in the field - shortest path traversal has only been confirmed with simulations and unit tests

- Additional unit testing
    - Simulate GPS data using emulator - instead of ideal function
    - Give random data
        - evaluate how systems handles inaccurate data (very real problem)
- Field testing
    - Log GPS data throughout route
        - plot route and determine if miscalculations are occurring
        - determine if there is a way to optimize algorithms
        - plot the path and find patterns leading to problematic performance

8 Budgets



BUDGET			
Planned		**Actual**	
Drone (after $ 50 student discount)	$ 250	Planned	$ 593
Battery (x2)	$ 60	Mainboard Replacement	$ 110
Motor (x2)	$ 80	GPS Replacement	$ 60
Propeller Set (x2)	$ 14	FTDI Boards + Cables	$ 40
Gear and Shaft Set	$ 10	Voltage Regulators	$ 4
Central Cross	$ 25	Drone Tool Set	$ 20
Screw Set	$ 4	**TOTAL**	**$ 827 + Shipping**
Router	$ 90		
GPS Module	$ 60		
TOTAL	**$ 593 + Shipping**	**DIFFERENCE**	**$ 234 + Shipping**

The costs for the prototype can be broken down into 5 main categories: Drone, GPS module, router, replacement parts, and support tools.

- The AR.Drone retails for $300, but after speaking with a sales representative they offered to give us a developer/education discount of $50.
- The GPS module (LOCOSYS 66 Channel LS20031 GPS 5Hz Receiver) cost $60.
- The WiFi router (Ubiquiti PowerAP N Router) cost $90.
- The two extra batteries were a must for testing because of the short flight time (~12 minutes) and long charge time (~90 minutes) of each battery pack.

Some of the drone replacement parts were purchased immediately because we thought they were more fragile, and after our rigorous testing would need to be replaced; however, the only part that we didn't order an extra of (the main-board) was the part that broke. Our first GPS module also burnt out while trying to connect it to the drone, so we had to replace that as well. We believe short circuit caused the first main-board and GPS module to break since they both died within two days of each other.

- The Drone Tool Set is a must for replacing parts on the drone.
- The FTDI board and USB cables were used to attached the GPS module to a PC to check accuracy and configure setting.

Even though our prototype had a proposed budget of ~$600 and turned out to be ~$825, we are strongly convinced that:

Mass produced version could be achieved at ~$300 especially if the project was undertaken by Parrot Inc. The mass produced version would only require the AR.Drone and the GPS module with very minimal hardware modifications.

Parrot Inc. actually announced the AR.Drone 2.0 this year (January 2012) which has a USB 2.0 port ready for hardware extensions without any firmware modifications. The AR.Drone 2.0 is supposed to be available for purchase in the summer (June 2012).

9 Appendices

File Structure Layout of Final Submission:

```
CSCE 483 tWP7:
         AR.drone API.url
         Web_home (website)
         AR.Drone Files:
                  atcomproxy.c
                  bin:
                           atcomproxy_arm
                  Makefile.txt
         droneCommands:
                  ARDrone.java
                  DroneReset.java
                  exampleWorkingCmds.txt
         DroneImages:
                  Background.png
                  cI6EGVSLLjjDFATI.medium.jpg
                  drone.jpg
                  grass.jpg
                  maroonDrone.png
                  maroonDroneSmall.png
         Phone App:
                  DroneControllerApp.7z
                  New Text Document.txt
                  v4l2_test.c
                  Wilke.Interactive.suo
         Reports:
                  Critical Design Review:
                           CDR_WP7_Final.doc
                           CDR_WP7_Final.pdf
                           CDR_WP7.doc
                           timeline_CDR_JBrock.docx
                           WPC_CDR_JBrock_APerovich_DDornier.pptx
                  Final Report:
                           FinalPresentation.pdf
                           AR.UAV_FinalReport_tWP7
                           FinalVideoDemo
                           Pictures:
                                    AR.Drone Deconstucted.jpg
                                    AutoRoute.PNG
                                    Drone no GPS.jpg
                                    Drone Systems Communication.png
                                    Drone with GPS.jpg
                                    Fly.PNG
                                    GPS app.jpg
                                    GPS attached with Regulator.jpg
                                    GPS Data thru RealTerm.jpg
                                    GPS on Hull.jpg
                                    GPS pin layout.png
                                    GPS Status LED Diagram.PNG
                                    GPS Status LED.jpg
                                    GPS to FTDI.jpg
                                    GPS unit.jpg
                                    GPS with 5-pin Header.jpg
                                    GPS with Jumpers.jpg
                                    GPSData_PhoneAccuracy.xlsx
```

```
                                        Main.PNG
                                        Mainboard USB Pin Out.jpg
                                        PointsTestGPS.PNG
                                        RealTerm (Port Tab).PNG
                                        RealTerm (Send Tab).PNG
                                        RealTerm (Telnet to drone).PNG
                                        RouteARUAV.PNG
                                        Settings.PNG
                                        System Diagram.png
                                tWP7 final report presentation.pptx
                        IAP Poster:
                                IAP Poster (tWP7).pdf
                                IAP Poster (tWP7).pptx
                                Phone UI diagram.jpg
                                PhoneMockUp.pptx
                        Proposal Files:
                                Bio&CV_ddornier.docx
                                Phone UI diagram.jpg
                                timeline_JBrock.docx
                                WP7_Proposal_alex.doc
                                WP7_Proposal_david.doc
                                WP7_Proposal_FINAL.doc
                                WP7_Proposal_FINAL.pdf
                                WP7_Proposal_jbrock.doc
                                WPC_JBrock_APerovich_DDornier.pdf
                                WPC_JBrock_APerovich_DDornier.pptx
                Silverlight for Windows Phone Toolkit - Aug 2011.msi
                tWP7 drone controller:
                        DroneControllerApp:
                                Controller:
                                        CommonHelper.cs
                                        Constants.cs
                                        Controller.csproj
                                        Controller.csproj.vspscc
                                        DroneController.cs
                                        DroneControllerConfiguration.cs
                                        DroneProxy.cs
                                        Enumerations.cs
                                        EventArgs.cs
                                        Exceptions.cs
                                        GPSNotificationEventArgs.cs
                                        MutlicastHelper.cs
                                        NavigationDataInfo.cs
                                        NetworkHelper.cs
                                        Properties:
                                                AssemblyInfo.cs
                                        ReceivedEventArgs.cs
                                        StopWatch.cs
                                        VideoFileWriter.cs
                                        VideoImage.cs
                                DroneControllerApp:
                                        App.xaml
                                        App.xaml.cs
                                        AppConfiguration.cs
                                        ApplicationIcon.png
                                        AutoNavPage.xaml
                                        AutoNavPage.xaml.cs
                                        Background.png
                                        ConfigPage.xaml
                                        ConfigPage.xaml.cs
                                        ControlPage.xaml
                                        ControlPage.xaml.cs
                                        cross.png
                                        DroneControllerApp.csproj
                                        DroneControllerApp.csproj.vspscc
                                        grass.jpg
                                        Properties:
                                                AppManifest.xml
                                                AssemblyInfo.cs
                                                WMAppManifest.xml
                                        pushPin.png
```

```
                    pushPinSmall.png
                    SplashScreenImage.jpg
                    StartPage.xaml
                    StartPage.xaml.cs
          DroneControllerApp.sln
          DroneControllerApp.vssscc
          GPSTracker:
                    App.xaml
                    App.xaml.cs
                    ApplicationIcon.png
                    Background.png
                    GPSTracker.csproj
                    GPSTracker.csproj.vspscc
                    MainPage.xaml
                    MainPage.xaml.cs
                    Properties:
                              AppManifest.xml
                              AssemblyInfo.cs
                              WMAppManifest.xml
                    SocketClient.cs
                    SplashScreenImage.jpg
          GPSTransmitApp:
                    GPSTransmitApp.csproj
                    GPSTransmitApp.csproj.vspscc
                    Properties:
                              AssemblyInfo.cs
                    ScheduledAgent.cs
          PortRedirectionApp:
                    app.config
                    PortRedirectionApp.csproj
                    PortRedirectionApp.csproj.vspscc
                    Program.cs
                    Properties:
                              AssemblyInfo.cs
          UDP echo app:
                    Program.cs
                    Properties:
                              AssemblyInfo.cs
                    UDP echo app.csproj
                    UDP echo app.csproj.vspscc
          UnitTestsSampleProject:
                    App.xaml
                    App.xaml.cs
                    ApplicationIcon.png
                    Background.png
                    dll:
                              Microsoft.Silverlight.Testing.dll

Microsoft.VisualStudio.QualityTools.UnitTesting.Silverlight.dll
                              Moq.Silverlight.dll
                              WatermarkedTextBoxControl.dll
                    DroneTests .cs
                    MainPage.xaml
                    MainPage.xaml.cs
                    NavTest.cs
                    Properties:
                              AppManifest.xml
                              AssemblyInfo.cs
                              WMAppManifest.xml
                    SplashScreenImage.jpg
                    UnitTestsSampleProject.csproj
                    UnitTestsSampleProject.csproj.user
                    UnitTestsSampleProject.csproj.vspscc
                    UnitTestsSampleProject.sln
                    update71dll:
                              EULA.rtf
                              Microsoft.Silverlight.Testing.xml

Microsoft.VisualStudio.QualityTools.UnitTesting.Silverlight.xml
          Video decoder:
                    app.config
```

```
                              Form1.cs
                              Form1.Designer.cs
                              Form1.resx
                              Program.cs
                              Properties:
                                      AssemblyInfo.cs
                                      Resources.Designer.cs
                                      Resources.resx
                                      Settings.Designer.cs
                                      Settings.settings
                              Video decoder.csproj
                              Video decoder.csproj.vspscc
                              VideoImage.cs
Videos:
        CDR video demonstration.mp4
        IMG_0047.MOV
        IMG_0048.MOV
        IMG_0049.MOV
        IMG_0051.MOV
        Title Screen:
                Application Walkthrough.TIF
                Autonomous Flight.TIF
                Manual Flight.TIF
                Title Screen.tif
        Title Screen.pptx
        tWP7.wlmp
Weekly Agendas:
 -- All available
Windows Phone SDK nvm_web2.exe
wp7ardrone-16774.zip
```

.