ASPECTS

Christopher Boselli, EE; Alex Breger, EE; Jason Danis, EE; Sandra McQueen, EE

***Abstract*—The increased popularity of unmanned aircraft vehicles (UAVs) has expanded the possibilities for innovation with aerial photography, package delivery services, and other exciting applications. However, many new users are inexperienced with flying and unaware of the guidelines and regulations in place to ensure safety and prevent potentially disastrous collisions. The ASPECTS module – a Raspberry Pi computer connected to a communication device – mounts on the UAV and interfaces with the flight controller to restrict flight in FAA-designated No-Fly zones.**

I. Introduction

RECREATIONAL drone use has increased dramatically in recent years as the hobby becomes more popular. The number of Certificates of Allowance permitting the flight of UAVs in civil airspace, for example, grew from 146 in 2009 to 545 in 2013. [1] These numbers are in addition to the multitude of recreational drones available for public purchase. Furthermore, according to a report by USA Today in October 2015, "An examination of 891 drone sightings reported to the Federal Aviation Administration over a 17-month period found more than half flew too close to an airport."[2] This poses a serious problem: UAV interference with passenger planes could result in property damage and, in extreme cases, fatality. Therefore, with this trend comes a new challenge: defining regulations that will lower the risk of incursions with passenger and commercial planes, and ensuring safe practices among vehicles sharing the airspace. One way this is implemented is by establishing restricted areas where drones cannot legally fly. The Federal Aviation Administration (FAA) currently enforces a five mile No-Fly Zone around all major airports, military bases, national parks, and other sensitive landmarks in the U.S., but very often those perimeters are breached by unmanned aircraft. This poses a threat to safety and security.

This issue has gained the attention of the FAA, the United States Department of Transportation, and several key Senate members, who recently proposed that all consumer drones must feature *geofencing* capabilities in the future.[3] Geofencing is a way of creating virtual geographical boundaries which are defined by a central set of GPS coordinates with a specified radius around that point. The idea of geofencing expands on the basis for other existing technologies such as invisible fences for dogs. This concept can be paired with specified hardware and software components to simulate a physical "fence." This technology requires innovation by manufacturers, and also presents an obstacle for hobbyists who build their own copters. Some drone manufacturing companies, such as DJI, have made firmware updates on their fleets that warn operators of nearby restricted areas via a mobile device and gradually lower the drone as it approaches the No-Fly Zone. This technology, however, only applies to DJI's Phantom 3 copter and newer models. [4] ASPECTS is a portable module designed to be compatible with any existing or future quadcopters that use the Pixhawk [5] flight controller popular among hobbyists, making geofencing technology available to all UAV owners both practically and financially.

In a meeting at Bradley International Airport, officials expressed their concern about the increased presence of UAVs in the airspace. While the threat of collision between drones and other vehicles continues to build, they said, it is not until a disaster occurs that this type of issue receives sufficient attention from the media and policy makers. [6] ASPECTS will not only alleviate apprehension for air traffic controllers monitoring flights into and out of the airport, but will regulate a majority of privately owned drones and ultimately create a safer airspace for travel, business, and recreation.

II. Design

A. Overview

ASPECTS (Airport Safety Perimeter Control System) is a proposed universal geofencing solution which may be incorporated on future drone models or retrofitted on previous designs. The on-board unit will consist of a Raspberry Pi computer, a GPS chip, and a 3G communication module, all of which will interface with the existing flight controller to execute commands. See the block diagram in *Fig. 1*.

By constantly monitoring the location of the drone via satellite, ASPECTS will determine the proximity to local No-Fly Zones through a software algorithm on the Raspberry Pi. Using the 3G communication module and an onboard SIM card, ASPECTS will send a message warning the user when he/she is within a predefined buffer zone of about 500 meters around the No-Fly Zone (indicated by the blue boundary in *Fig. 2*) allowing an opportunity for the user to manually redirect the flight path.

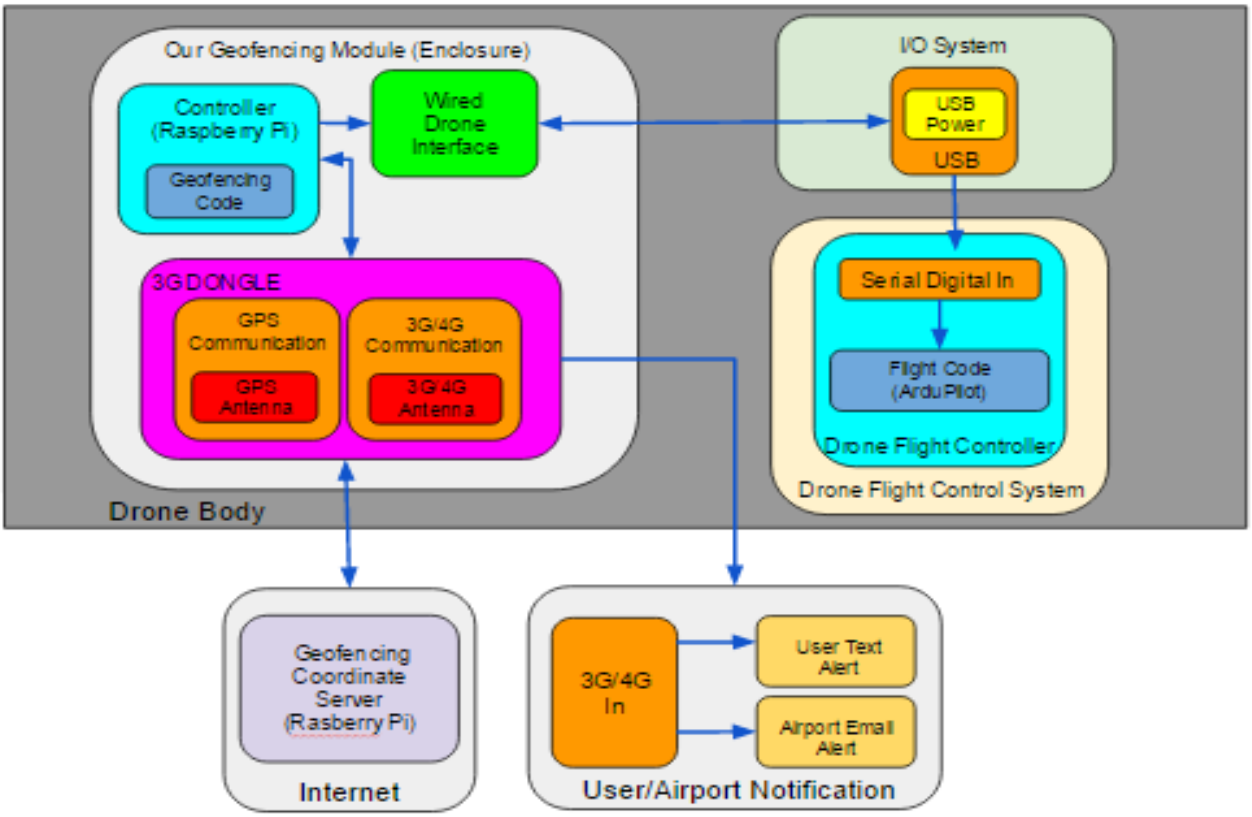

Fig. 1. Detailed block diagram of ASPECTS system.

If the user does not take corrective action before the drone reaches the No-Fly Zone, the on board controller will then assume control of the drone in order to prevent a breach of the critical airspace (indicated by the red boundary in *Fig. 2*). Transferring control away from the user presents a potential unintended consequence: collision with structures such as buildings or power lines which could result in damage to public or private property or to the UAV itself. For this reason we are considering several possible actions, including landing the aircraft, hovering in place, or executing a return-to-launch algorithm. This effectively creates a physical barrier around the perimeter of the airport or other sensitive area and averts a potentially dangerous situation.

Below is a table of quantitative specifications which we have already met for MDR, or plan to meet once the system has been integrated.

SYSTEM SPECIFICATIONS

Specification	*Value*
GPS Update Rate	>2Hz
Min. User Response Time	10 seconds
Buffer Zone Radius	>1000 feet
GPS/Cellular Signal Strength	>5dB
Drone Test Flight Time	>15 min

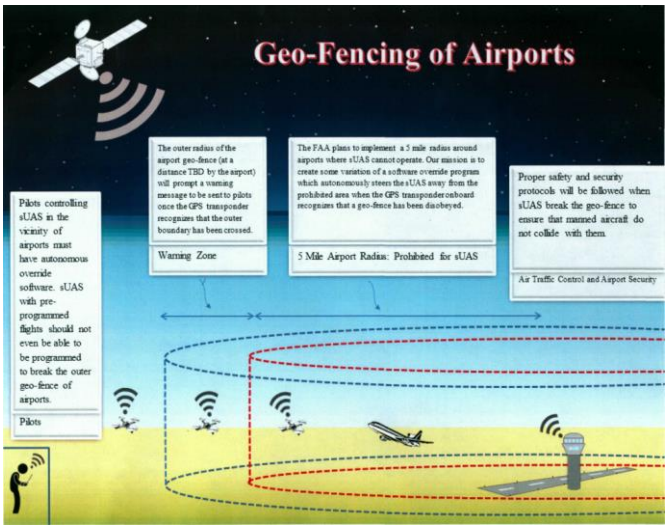

Fig. 2. A visual representation of our *Geofencing* solution. The blue outer perimeter represents the buffer zone surrounding the critical No-Fly zone denoted by the red inner perimeter.

A. Block 1: Drone Body

One of the earliest changes made to our original vision for this project was the decision to build our own drone "from scratch" rather than simply purchasing a fully integrated one. A major factor in this decision was to keep all associated hardware/software components of the project as open source and compatible as possible. In disassembling a 3DR Iris+ quadcopter, we soon realized that we would likely need special

permissions from the manufacturer in order to be able to access and alter the flight software to execute our algorithm once inside a geofenced area. To avoid additional complications, we researched all of the major components required to build a quadcopter of our own.

One of the major aspects that needed to be considered when designing our own drone was the overall weight of the quadcopter and additional on-board hardware which we were going to install as part of our design. This key factor ultimately finalized decisions between alternative hardware possibilities when purchasing the parts for the drone. Ultimately, we stayed within our target limit of 1.5 kg for total weight of the quadcopter parts and additional on-board hardware components. The final parts list with the corresponding weight of each component can be seen in *Table 1*.

Another key aspect of the design process was selecting which battery to purchase that would power all of the on-board components of the drone. We had to take into consideration the power consumption of the drone hardware as well as our additional components, and we had to make some calculations in order to be sure that the battery provided enough power for significant flight time without compromising the balance between increased power and added weight.

B. Block 2: Controller (Raspberry Pi)

In order to interpret GPS data, hold code for overriding the drone's microcontroller, and hold code for landing the drone at a geofence boundary, an on-board controller is needed. We are using a Raspberry Pi Model B [7] to implement this controller unit, which will be housed on-board the drone during flight. This controller is initially responsible for reading in GPS information from a separate hardware component, and interpreting this data as it comes in. The GPS coordinates that are read into the Raspberry Pi are compared with a database of central geofenced coordinates and clearances that are required around each one of them. The GPS data is also changed into NMEA format after it is read in, which makes it easier to interpret. Although our group did not receive a necessary passive antenna before our MDR presentation, we could still demonstrate that the Raspberry Pi could serially read in the formatted data. This geofence information will be compiled into a database, and read in from a separate subsystem upon startup. Coordinates that are read into the controller will be classified into three regions: Clear of all restricted areas, inside of a buffer region near a geofenced area in which the user and owner of the restricted space will be notified of the drone's presence, and inside of a geofenced area. Distance from the central geofence coordinates, illustrated in *Fig. 3*, is determined using the Pythagorean Theorem assuming the difference in latitude is the y-axis distance and the difference in longitude is the x-axis distance.

Part	Description	Weight (g)	Qty	Total Weight Contribution (g)
Quadcopter Frame	Tarot 650 Carbon Fiber Frame	485	1	485
Flight Controller	Pixhawk	100	1	100
Battery	nano-tech 4000mA	333	2	333
Propellers	3DR	N/A	3	0
Motors	3DR Black Top Motor	68	4	272
ESCs	3DR 20 A	21	4	84
Our on-board controller	Raspberry Pi Model B	45	1	45
3G/GPS Communication	Adafruit FONA 3G Breakout	9	1	9
TOTAL WEIGHT				**1328 g**

Table 1. On-board hardware and corresponding weight contribution to the design.

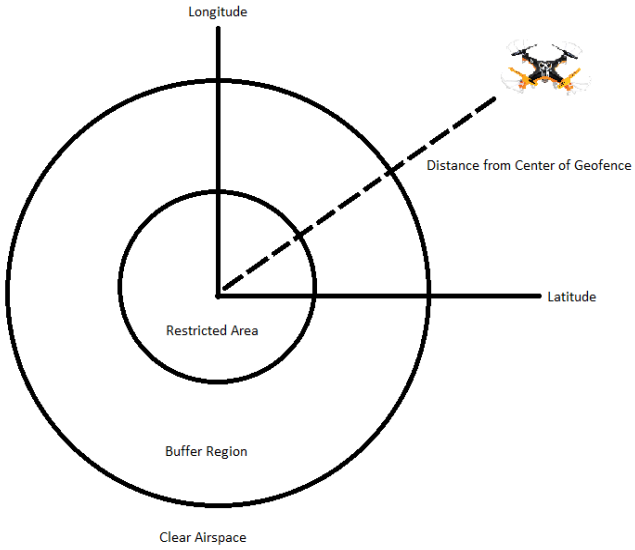

Fig 3. Breakdown of regions surrounding a geofenced area

After it is determined that the drone has breached a restricted area, the Raspberry Pi will communicate with the Pixhawk microcontroller, which is the flight controller installed on the drone, to override its software and safely land the drone. The software to land the drone will be executed after it has been confirmed that the Raspberry Pi has priority over the Pixhawk.

LED indication has been used to test the Raspberry Pi's ability to interpret data that it receives from the separate GPS unit. A yellow LED indication, as well as a text message notification, has been demonstrated when the controller is fed coordinates that are within the buffer region of a preset geofenced location. A red LED was designed to turn on when the controller was given coordinates that were inside of a geofenced area. Lastly, we demonstrated that there is no LED indication when the controller was clear of all restricted areas. Python scripts have been used to implement and test different aspects of the controller unit.

Now that we have verified that the Raspberry Pi can

interpret incoming GPS information, we can send user and airport messages when the drone has entered a buffer region, which will be implemented using a different functional block, and we can execute an algorithm to land the drone when it had entered the restricted airspace of a geofenced location.

C. Block 3: Communication Device

The geofencing module requires two forms of communication: 3G to the UAV user and the airport via the cellular network, and satellite communication to receive GPS coordinates as discussed in the previous section. The Adafruit FONA breakout modem [8] provides both capabilities and was selected for its efficiency and its compact size. Data is sent to and received from the Raspberry Pi through the hard-wired transmit and receive pins. The various 3G and GPS functions of the FONA are accessed using a library of built-in "AT" commands provided in the datasheet. [9]

Then the antenna is connected and *read-in* is enabled, the GPS coordinates are read continuously with an update speed of 2 Hz. This data is sent to the Raspberry Pi where it is parsed and analyzed to determine when the UAV is in danger of breaching a No-Fly zone.

Utilizing existing communication technology, such as AT&T's cell towers, is critical to the ASPECTS design because it minimizes overall cost of the system and facilitates implementation. As the UAV approaches the No-Fly zone it will encounter the geofence coordinates defined in the controller software. The FONA will then receive a command from the Raspberry Pi's transmission (T_X) pin, which contains the cellular telephone number of the user along with a text message warning the user of his or her proximity to the airport and instructing him or her to turn around. In order to allow the user sufficient time to read the message and react, we estimate that the text must be received between 10 and 20 seconds before the drone reaches the critical 5-mile radius. Since this requires fast and reliable communication, we chose to use the 3G FONA in favor of the previous 2G model. The maximum latency observed was 8 seconds from the time the message was sent to when it was received by a mobile device. From this delay, the radius (B) of the buffer zone added on to the initial 5 miles could be calculated based on the maximum speed of the average UAV:

$$B = \text{avg. max speed } (m/s) * [\text{reaction time} + \text{max delay}] (s)$$
$$= 22.3\ (m/s) * [15 + 8] = 512.9\ m$$

In addition to notifying the user when the UAV has breached the buffer zone surrounding the critical airspace, the FONA sends another text message update following the warning. This message will indicate that either the user has successfully cleared the No-Fly zone and the surrounding buffer, or that the flight controls have been taken over and the UAV will land or return to its launch point.

D. Block 4: Geofencing Data Server (2nd Raspberry Pi)

In order to upload the most current geofencing information to our device throughout operation, our team has designed a remote server to host this information over the Internet. The physical server (the host) consists of a second Raspberry Pi device connected to the Internet and configured to function as an FTP server using GNU's vsftpd protocol.[10] Our geofencing controller (the client/first Raspberry Pi) can access the server's file system at custom intervals using a Linux command line script and its 3G data connection. The client need only ping the server's static IP address over the FTP protocol, and it receives immediate read permission to download the hosted files (stored as .txt files). Once downloaded by the client, these text files will be parsed by our GPS program into meaningful navigational data.

Designing the server mainly involved UNIX shell programming (ECE 353 and 558), network configuration (ECE 374), and database programming (ECE 242). On the server side, the main challenge was using the Linux shell editor to alter the server's FTP configuration file in order to enable and customize the server. The host device and local network also had to be configured to forward FTP ports 20-22 and maintain a static IP address. On the client side, we must still design a Linux command line script to be called by our GPS program during operation in order to fetch new geofencing coordinates. For example, the server could contain a database of different coordinate files for various different regions. The server script would be configured to execute upon entering each new region, in addition to executing an automatic check for updates at a custom interval.

There are two experiments we will use to verify the functionality of the data server. The first, which we have already demonstrated for MDR, is issuing a manual server call using a Linux FTP program on the client device. As pictured in Fig. 4 below, when the host's IP address and port information is properly entered into the client, our test file 'MDR_test.txt' can be accessed and downloaded to the client's file system. The program reported a file transfer of 84 Bytes in 1 second, which is a more than optimal latency for this aspect of the project.

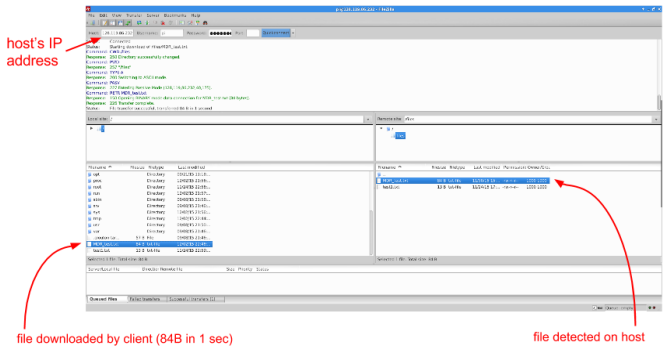

Fig 4. Example of a Manual Server Call using Linux GUI

Once we get our drone in the air and design the data fetch script, our next experiment will consist of conducing a server download over 3G while our entire system is in operation and

subsequently verifying that the new data file was received by the client after landing.

III. Project Management

MDR GOALS	
Specification	*Completion*
GPS Tracking/Data Logging	100%
GPS Data Interpretation	100%
User Notification System	100%
FTP Server	100%
Copter Assembly	50%

Table 2: Proposed deliverables for Midway Design Review.

As indicated in Table 2 above, the major goals set by the team at the Preliminary Design Review were achieved by MDR. The GPS data can be read in from the FONA to the Raspberry Pi in a specified format and interpreted to distinguish latitude, longitude, and altitude values for later processing. From those coordinates the relative position of the UAV to the center point of the airport is calculated in a python script. Depending on that distance, a text notification may be sent to the user accordingly. Once these three individual subsystems were tested separately, we then integrated them so that the process executes when the Raspberry Pi boots up, and will run continuously thereafter.

Our team meets weekly to review our progress with our advisors Professor Douglas Looze (ECE) and Professor Daiheng Ni (CEE). We also meet separately as a team to talk about our individual contributions and how each piece will fit into the overall system design. Alex Breger is primarily responsible for the server; Christopher Boselli, for developing flight control software and assembling drone hardware; Jason Danis, for GPS processing; and Sandra McQueen, for communication with the UAV, owner, and local airport. When obstacles arose due to delayed hardware shipment, we adapted our deliverables by focusing on other subsystems so that all contributed equally to the final MDR design. While each had his or her particular subsystem to develop, all team members assumed responsibility for project completion.

Earlier this semester we had the opportunity to present our project at Bradley International Airport, where we received valuable feedback from airport officials regarding their experiences with UAV sightings and their concerns for the future of drone use. In addition to providing further motivation to implement our solution, the conversation at Bradley emphasized the importance of notifying the airport as well as the user. This provides them with the data necessary for observing trends and expressing the magnitude of the problem through statistics.

Figure 5 shows a Gantt chart specifying our team's projected timeline for the fall and spring semesters leading up to the Final Design Review, as well as the name of the team member assigned to each task.

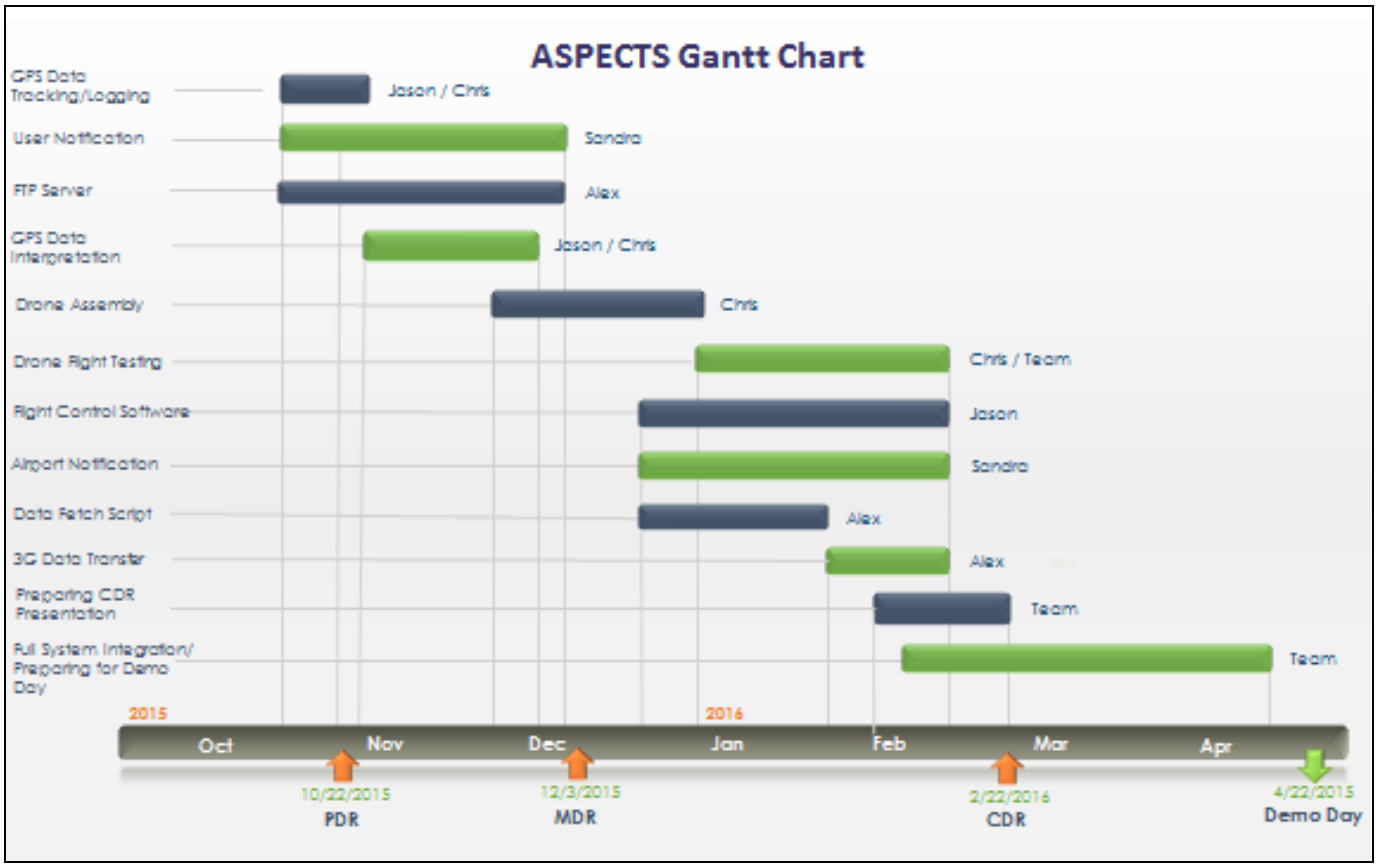

Fig. 5: Time Management Plan and Primary Responsibilities

I. Conclusion

Our team accomplished our goals for MDR even without the use of a passive antenna for GPS data collection. Without this component, we still demonstrated that the data could be collected, formatted and logged, and that the resulting data could be incorporated into an algorithm that would interpret it. We were also able to successfully notify the user via individual SMS messages specific to the location of the drone (in No-Fly zone or simply in buffer zone) sent by our on-board controller upon determining if action was necessary or not. Additionally communication was established between our on-board controller and off-board file server which will eventually house a database of geo fenced coordinates.

Our CDR goals include establishing reliable communication between our on-board controller and the drone's flight controller, creating an algorithm to override the flight controller software at the boundary of a No-Fly zone, creating an enclosure to house our additional on-board hardware attached to the drone, developing a set of regional databases containing geofenced coordinated, and creating an algorithm to provide additional notification to airports via associated official email addresses. For our CDR presentation, we are also striving to have all of these aforementioned subsystems fully integrated and functioning correctly.

The remaining work to be done has been split into phases which we believe is the most practical way to ensure the success of the project. First, we will have the drone fully assembled and tested, and the enclosure to house the on-board electronics will be built. The enclosure can be designed using applications such as AutoCAD, and physically built using a 3D-printer. For testing and demonstration purposes, all members of the team will become proficient in piloting the drone. Next, we will create the algorithm to safely land the drone. We are planning on using MavLINK to accomplish this phase, which is a protocol for communicating with small unmanned vehicles such as drones. During this phase we will simultaneously be designing the email notification system to alert the airports, which will be implemented with the FONA hardware component. After we demonstrate that we can successfully land the drone, the last phase before subsystem integration will be demonstrating that we can automatically synchronize the geofencing database information onto the Raspberry Pi using the FTP host server.

One of the major difficulties we can expect for the remainder of this project is the learning curve associated with learning scripting languages such as Python as well as the correct way to implement the MavLINK protocol. This should not pose too much of a problem, as each member of the team is becoming proficient at scripting thanks to the numerous tutorials and examples that are available online. There is also an abundance of documentation available for implementing the MavLINK protocol. The other difficulty of this project will be learning how to pilot the drone, which each member of the team should be comfortable with before we test our electronic systems on-board the drone. Once all of the individual subsystems have been designed, tested, and integrated onto the drone, we will be ready to give a demonstration of the system's functionality. This demonstration will begin by initiating startup software as the drone is powered on, and will conclude after the drone has been successfully landed, powered down, and the user and fictitious airport have been notified of its presence.

Acknowledgment

All of us on the ASPECTS team would like to thank our ECE faculty advisor, Professor Douglas Looze, for his valuable insight and contribution to team ideas. We would also like to extend a thank you to our Transportation Engineering advisors, Professor Daiheng Ni and Professor John Collura, for their help and advice along the way. We would also like to thank Bradley International Airport's Air Traffic Manager, Ayaz G. Kagzi, for his suggestion and for providing us with the perspective and insight of an airport staff member dealing with these issues every day. His critical advice helped us understand the solution our project is attempting to solve from outside the engineering point of view, and made us realize the severity of the issue at hand.

References

[1] Federal Aviation Administration, Fact Sheet – *Unmanned Aircraft Systems (UAS)*. Washington, DC: US Department of Transportation, January 2014.

[2] B. Jansen, '*Drone sightings spur legislation to fence them in away from planes*', USA TODAY, 2015. [Online]. 7 December 2015. <http://www.usatoday.com/story/news/nation/2015/10/21/drone-rulebreaking-geofencing-dianne-feinstein-charles-schumer/74292314/>

[3] "Schumer Bill Would Require Safety Features, Like Geo-Fencing And Sense & Avoid Technology On All Drones; Would Improve The Ability Of Law Enforcement To Take Action Against Reckless Users By Making Drones Detectable And Identifiable To Pilots & Air Traffic Control," *Charles E. Schumer United States Senator for New York*, Oct. 14, 2015. <http://www.schumer.senate.gov/newsroom/press-releases>.

[4] "DJI Introduces new Geofencing System for its Drones," *DJI*, 17 November 2015. [Online]. Available: <http://www.dji.com/newsroom/news/dji-fly-safe-system> [Accessed: January 22, 2016].

[5] 3DR. "*Pixhawk Overview*" [Online]. Available: <http://copter.ardupilot.com/wiki/common-pixhawk-overview/>

[6] A. G. Kagzi, Air traffic Manager, Bradley International Airport, personal communication, Nov. 2015.

[7] Adafruit. "*Raspberry Pi Model B*" [Online]. Available: <https://www.adafruit.com/pdfs/raspberrypi2modelb.pdf> [Accessed: January 22, 2016].

[8] SIMCom. (2012, Aug. 21). "SIM5320_Hardware Design_V1.07" [Online]. Available: <https://umanitoba.ca/faculties/engineering/departments/mechanical/pdf/NEW-Citing-IEEE-2013.pdf> [Accessed: January 22, 2016].

[9] SIM Tech, "AT Command Set," SIMCOM5320 command set, July 2014.

[10] Christopher Evans, "vsftp – Secure, fast FTP server for Unix-like systems" (2015) <https://security.appspot.com/vsftpd.html>.