ILLINOIS INSTITUTE OF TECHNOLOGY

Framework for Autonomous Delivery Drones

School of Applied Technology

ITM 492 - Embedded Systems & Logic Design

Professors:

Jeremy Hajek

Dr. Dan Tomal

Students:

Raed A. Tawil **Mark E. O. Milhouse**

Abstract

Many types of online purchased merchandise are light in weight and small in size. The distance between the online seller and the buyer, and the absence of delivery methods at some stores like pharmacies show potential for using drones for package delivery.

Shipping trucks can carry a large quantity of small light packages, but can't be used to ship only a single package, and they increase the severity of traffic congestion. Other methods such as car delivery service are only offered within certain industries, but not by pharmacies and governments. Another important factor is the necessity of asphalt roads for previous methods to perform their task, which prevents them from working in areas that lack these types of roads, like some third world countries and post-disaster areas. Quadcopters have been proposed to carry and deliver packages that are light and small. They can be controlled by a remote control, or they can use autopilot. Here we investigate the impact of using autonomous drones to carry packages between several geographical locations, and the reliability of the automation system in terms of wind resistance, GPS accuracy, and flight stability. We use the 3DRobotics Iris quadcopter to test and build the automated system.

Table of Contents

Chapter 1
INTRODUCTION

A rising demand for flexible and efficient forms of delivery that utilize renewable energy resources and minimize on human impact, especially traffic, has recently called attention to the possibility of using autonomous unmanned aerial vehicles (drones) in a widespread commercial application. Many organizations worldwide, from retail outlets to national governments, have expressed interest in utilizing this technology for diversifying their fleet. Additionally there are many opportunities to implement drones to reach areas that were previously too costly or too dangerous to deliver supplies to. In light of these opportunities both commercially and socially, we set out with the goal of developing a vehicle that could meet these demands.

The drone vehicles currently available come in a wide variety of configurations depending on the features, power, battery life, and vary in price and quality. Due to their nature structural integrity and modularity is an essential factor, and the manufacturer we selected, 3D Robotics, is known for meeting these two requirements. The model selected was the Iris quadcopter, which is a sturdy all-inclusive vehicle with autopilot, radio telemetry, manual controller, and Global Positioning System (GPS) included in addition to all of the necessary hardware components, and a payload capacity of 400 grams, which is sufficient to the scope of this project.

These vehicles would have to be able to maneuver themselves between at least two locations carrying a payload as commanded by a remote user (the destination). This

includes autopilot, mission planning, and user-interface software, in addition to the 3D Robotics Iris vehicle, server, and radio telemetry hardware required. Success would entail accomplishing automated navigation to and from a destination without human intervention, and would require the development of server side software for automating the pre-loading of navigation specifications, pre-arm checks, and launch of the vehicle.

The project was divided between hardware and software in an effort to capitalize on strengths and maximize on the short amount of time allotted. The software portion addressed determining flight paths, methods of communication, and automating the flight process utilizing pre-existing programs where possible and developing tie-ins to allow for remote user requests. The hardware portion focused on mastering the actual flight of the drone, calibrating sensors and motors, telemetry, navigation systems, mastering drone parts assembling and disassembling, configuring RC transmitter, and drone repairing. Through a series of agile iterations the research, development, and testing cycle moved from first assembly of the drone and testing initial functions, through thorough flight and process analysis, to ultimately successfully conducting remotely initiated automated flight and return.

This report includes sections detailing our research and findings, and future prospects for further development. Chapter 2 is an overview of the systems involved, and briefly explores the process necessary to a successful trip. Chapter 3 is a list of the hardware components included in the Iris vehicle and their function. Chapter 4 is a discussion of the software used and developed for this project, how it communicates, and its function. Chapter 5 explores the experiments conducted during the duration and their

content. Chapter 6 discusses the results of the project and what was discovered. In Chapter 7 we conclude the project, and finally in Chapter 8 we present future possibilities for further development.

Chapter 2
System Overview

In this chapter we'll explain how the system is set up in order for the drone to deliver the package. The architecture of the system is as shown in Figure 2.1 below.

We developed an Android mobile app that the customer will use. The app will send the GPS coordinates of the current location of the user to the server. The app and the server will talk to each other using a socket connection. Once the server receives the the coordinates from the app, it will then pass these coordinates to the drone through the MAVproxy server. After the coordinates have been successfully delivered to the drone, then the MAVProxy server, using an already existing set of commands, will take complete control the drone. The instructions that will be sent to the drone will achieve the following process: Taking off, reaching a specific height, start moving towards the destination where the user is, reducing altitude, delivering the package, raising altitude, then flying back to base station. Base station will be referred to later as home station.

In order to perform a successful trip, the following factors must be available and achieved:

1- GPS Lock on the drone: Prior to flight, the drone must have a GPS lock to make sure it knows where home station is so it can come back to it after the package gets delivered. A green light should blink on the drone to indicate that GPS lock is successfully obtained.

2-Preflight check: A usual physical procedure that should happen before every flight. The procedure includes checking all the locking nuts above the propellers to ensure they are tight enough. Setting the antenna in a vertical position to maintain the best connection quality between the drone and the ground station. Making sure the drone is standing over a level ground. Arming and disarming the drone using the RC transmitter. Taking it off to up to a low altitude (2-5 meters) then landing it. Checking that all blades are spinning equally when taking off. And finally, checking that the path is clear for taking off.

3-Connectivity check: After performing the preflight check, then we establish the connection between the ground station and the drone and make sure they're successfully talking to each other.

4-Turning off safety mode: This is the final step. After performing the previous 3 steps, then we'll physically put safety mode off by pressing on the red button on top of the drone until the blinking red light on it turns solid, which indicates that safety mode is off. A confirmation tone will be heard too.

Chapter 3
Hardware Components

We're currently using the 3DRobotics IRIS quadcopter for prototyping. This quadcopter uses a PIXHAWK microcontroller to control the entire system of the drone.

The list below includes all the items of which the drone is built from:

1- A Set of 4 Arms (2 black, 2 blue): The rear arms have a wider angle with the drone body than the front arms.

2- A Set of 4 Propellers: Two of them are built to rotate clockwise, the other two are built to rotate counter clockwise.

3- A Set of 4 Legs: The 4 legs have the same design, but they differ in color for visual purposes. We have a set of 4 short legs and a set of 4 tall legs.

4- Four Motors: Two of them are set to rotate clockwise, the other two rotate counter clockwise. Each two have appropriate propellers connected to them.

4- Drone Body Shell: A plastic case containing the following parts:

- PIXHAWK microcontroller with a MicroSD card

- Power Board that connects to the 4 motors
- Lithium Battery
- GPS Module
- RC Receiver
- Speaker

5- RC Transmitter

6- Ground Station: A 915 MHz connector that can be connected to a laptop or a tablet.

This is device that is used to connect between the home station and drone during the flight.

Fig 3.1, the 3D Robotics Iris Quadcopter with included parts (http://www.adafruit.com)

Chapter 4
Software

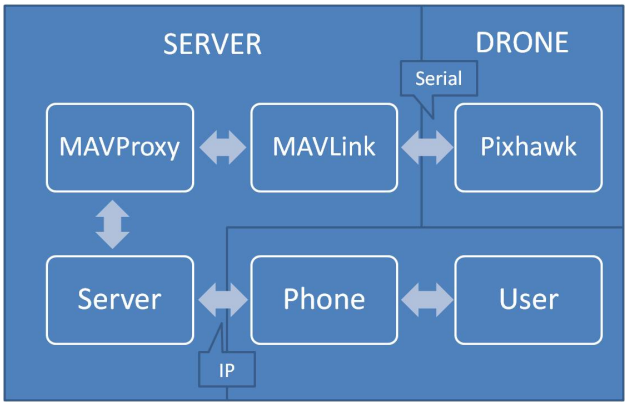
The Iris itself utilizes the 3D Robotics PixHawk autopilot controller and the PX4 autopilot firmware. This communicates with ground station software via the MAVLink communications protocol. In researching a way to communicate with the Iris from a computer we discovered many implementations of software ground control stations, many of which were useful to configuring the drone, but MAVProxy stood out as the most useful to our case. The MAVProxy software is written in Python 2.7 which made it an excellent foundation from which to base our development.

Once the MAVProxy module was configured and reverse engineered we began to develop toward the user. This was done by building a server method designed to receive input from external sources and translate them to be sent to the Pixhawk controller utilizing the methods and modules already defined in MAVProxy and further through MAVLink. This involves taking the device's GPS output, relaying it over IP (using Socket) to the remote server which is connected to the drone wirelessly. The server then creates a waypoint file, inserting the coordinates into a series of commands including "takeoff", "waypoint (go to)", "land", and "return to launch". From here the server executes a series of commands on the drone directly to ensure the success of the future mission, a pre-flight check. The drone is then armed, throttled, and sent on its way.

From the point the drone leaves the ground it is now on its own executing the pre-loaded commands autonomously, though the server and any authorized operator is

still able to view flight telemetry data and execute emergency commands. The drone executes the takeoff and flies directly to the destination at the prescribed altitude, executes a landing and deploys its payload, and immediately executes a return to the pre-determined launching point.

Fig. 4.1



Originally the server was intended to run separately from MAVProxy and communications between the two would have been Interprocess, but in order to conduct smoother, more effective communications, the code for the server was injected into the MAVProxy source and called in a new thread. This allowed for the communications to be sent directly to the low-level state handling function inside MAVProxy, and meant the reaction time was reduced to near instantaneous (MAVProxy state refreshes every millisecond).

Each communication from the mobile app to the server is handled as a one-off transaction, allowing the server to field and queue multiple requests per vehicle. This session is terminated once confirmation of the drone's departure is delivered to the user.

Figure 4.1 describes the communication process from the user to the drone and back.

Figure 4.2 is a step-by-step illustration of the processes performed by each piece of software involved.

Figure 4.3 displays a sample waypoint file with known actions and their parameters listed.

Appendix A contains a sample code snippet for translating communications over socket to waypoint instructions to be passed through MAVProxy to the PIXHAWK

Fig. 4.2

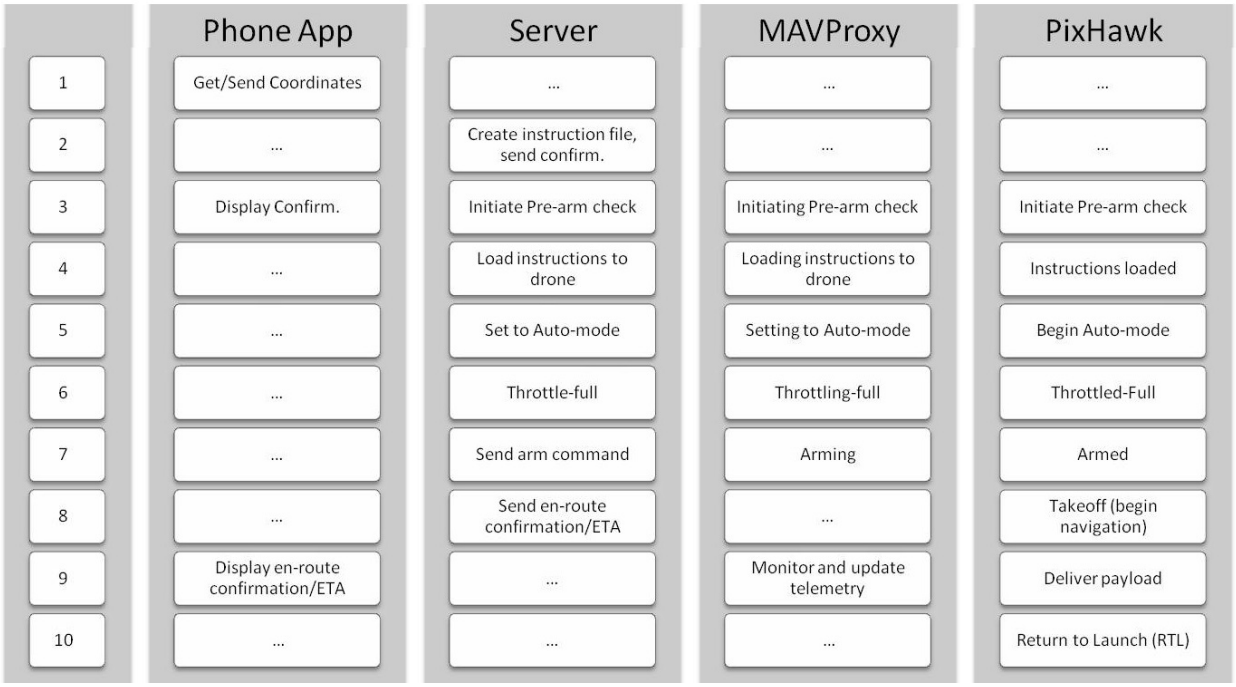

Phone App
Server
MAVProxy
PixHawk
1
Get/Send Coordinates
...
...
...
2
...
Create instruction file, send confirm.
...
...
3
Display Confirm.
Initiate Pre-arm check
Initiating Pre-arm check
Initiate Pre-arm check
4
...
Load instructions to drone
Loading instructions to drone
Instructions loaded
5
...
Set to Auto-mode
Setting to Auto-mode
Begin Auto-mode
6
...
Throttle-full
Throttling-full
Throttled-Full
7
...
Send arm command
Arming
Armed
8
...
Send en-route confirmation/ETA
...
Takeoff (begin navigation)
9
Display en-route confirmation/ETA
...
Monitor and update telemetry
Deliver payload
10
...
...
...
Return to Launch (RTL)

Fig. 4.3

	QGC WPL 110											
	index	current	altmode	cmd	param1	param2	param3	param4	param5	param6	param7	auto
WAYPOINT	1	1	1	16					41.94288	-87.648		1
WAYPOINT	2	1	1	16	DELAY	HIT RAD		YAW ANG	LAT	LONG	ALT	1
LOITER TURNS	3	1	1	18	TURNS		DIR 1=CW		LAT	LONG	ALT	1
LOITER TIME	4	1	1	19	TIME (S)				LAT	LONG	ALT	1
LOITER UNLIMITED	5	1	1	17					LAT	LONG	ALT	1
RTL	6	1	1	20								1
LAND	7	1	1	21								1
TAKEOFF	8	1	1	22							ALT	1
ROI	9	1	1	80					LAT	LONG	ALT	1
DO SET ROI	10	1	1	201					LAT	LONG	ALT	1
COND DELAY	11	1	1	112	TIME (S)							1
COND CHANGE ALT	12	1	1	113	RATE (CM/S)						ALT	1
COND DISTANCE	13	1	1	114	DIST (M)							1
COND YAW	14	1	1	115	DEG	SEC	DIR 1=CW	REL/ABS				1
DO JUMP	15	1	1	177	WP#	REPEAT #						1
DO CHANGE SPEED	16	1	1	178	SPD (M/S)							1
DO SET HOME	17	1	1	179	CURRENT (1)/ SPEC(0)							1
DO SET CAM TRIGG	18	1	1	206	DIST (M)							1
DO SET RELAY	19	1	1	181	OFF(0)/ ON(1)	DELAY(S)						1
DO REPEAT RELAY	20	1	1	182		REPEAT #	DELAY(S)					1
DO SET SERVO	21	1	1	183	SER NO	PWM						1
DO REPEAT SERVO	22	1	1	184	SER NO	PWM	REPEAT #	DELAY (S)				1
DO DIGICAM CONTROL	23	1	1	203								1
DO MOUNT CONTROL	24	1	1	205								1

Chapter 5
Experimental Work

Different experiments were performed to insure success of each part was achieved.

1. Calibration: The IRIS quadcopter need several calibration procedures to be able to fly properly. The importance of this part is extremely high if not the highest. Compass and Sensors calibrations can be done using the MAVLink software or a terminal. The ESC calibration is done using the RC transmitter.

a. *Compass Calibration:* Involves moving the drone around all axiss. This will calibrate the drone to move correctly in the 4 directions.
b. *Sensors Calibration:* Involves positioning the drone on its left side, right side, nose up, nose down, and on its back. This is important for the drone to be stable while flying, and to have a correct reading of its angle with the horizon.
c. *ESC Calibration:* Using the RC transmitter, we follow a procedure of connecting and disconnecting the battery three times to calibrate the 4 motors and make them work with complete synchronization.

2. Testing the multiple flight modes: Using RC Transmitter, we tested Stabilize mode, Altitude mode, RTL (Return to Location), and observed how the drone reacted to these

modes. We also observed how smooth the transition was between several modes while flying.

3. Testing MAVLink: We used the Mission Planner software to fly the drone, observed how it reacts to the pre-flight plan.

4. Testing the Server: After finishing the server code, we tested it first using the simulator. The simulator basically shows a virtual drone on a map that reacts exactly like the IRIS. The simulator after several experiments performed successful flights. The main goal of using the simulator was testing the quality of flying the drones using the server. The server controls the drones by sending commands to it to tell it what to do. After achieving the desired results virtually, we moved to testing the server on the real drone. Using guidance mode, which feeds the drones GPS location to fly to, the drone received a takeoff command, then successfully reached the destination, after that it received a land command then successfully landed without crashing.

5. Testing the Android App: We've developed several applications to send GPS location to the server, but we encountered some difficulties in generating the correct location. These difficulties were caused because of the high complexity of Android development. We used later simulated commands that simulates the GPS coordinates that were supposed to be sent from the app. Doing that saved us time to focus on the automation

part and the main goals of the project. The app development process will continue later to successfully complete the task.

Chapter 6
Results and Discussion

The ongoing process of the project taught us a lot about autonomous flight control systems, drones aerodynamics properties, battery power, physical threshold of the equipment, and real world applications management.

We'll list the results we've obtained then a discussion will follow.

Results:

1. Wind turbulence is a major obstacle. It can lead to crashing the drone very easily in a blink of an eye. It should be taken very seriously.
2. Accurate Calibration is extremely important. Inaccurate calibration could be a deal breaker.
3. Frequent checking on drone settings and RC transmitter is necessary.
4. We can use PIXHAWK to control the extra sensors and the package delivery mechanism in the future, which is better than using an additional microcontroller such as Arduino or Intel Galileo.
5. In reference to our flight time and past experiments, it's reasonable to state as fact that the current battery power we have, is enough for the drone to safely perform a round trip over a distance that is up to 5 Kilometers with no package mounted, and up to 3 kilometers with a package mounted.

6. Automation algorithm can be generated and fed to the drone once only prior to flight. After that point, the drone doesn't need any further instructions from the server to perform its trip, and it can fly totally on its own.
7. If a trip was planned over spaces, where the drone can fly freely with no possible obstacles in the way, it displays excellent performance without the need for any additional sensors.

Discussion:

1. **Wind Turbulence:** When testing drone resistance against the wind in the windy city, you learn a lot about wind turbulence. Flying the drone in an open space will significantly reduce the chance of crashing in a tree or a building because of the wind. In this scenario, wind turbulence will affect battery power. The more wind fights the drone against the direction it's flying towards, the more battery consumption the motors will need. When flying the drone in a windy weather, the sound of the motors increasing their power to resist the wind was heard loud and clear. This result means that even when a mission is planned over open space, the wind condition will affect the battery life which means the trip should be studied carefully in terms of the distance to wind power ratio to ensure it can be done.

   Now when there are obstacles in the way, especially trees with empty branches, flying on low altitudes is not recommended. The risk of crashing the drone gets very high when flying next to a tree on a low altitude, one wind blow is enough to move the drone towards the tree. If that happens, the possibility of a crash is 100%. Whether the drone was controlled with the RC transmitter or the server (Autopilot Mode), and since the drone is very sensitive to the RC transmitter sticks movement and also very powerful, trying to fly it away from the tree is very difficult and tricky, and in such a short time (seconds) there won't be a high chance of flying it away from the tree or any other obstacle.

2. **Calibration**: When calibrating the sensors of the drone using MAVLink, a very slight tendency could cause an inaccurate calibration. Even if the drone is only 1 degree inaccurate, when it flies and tries to maintain its altitude, it will drift toward one of the 4 directions. It will also lead the drone to flip when taking off causing it to crash instantly afterwards. In the drifting scenario, trying to stabilize the drone using the RC transmitter is challenging, and a manual trimming procedure, using the 4 trimming functions on the transmitter, should be done while the drone is in air. Calibration errors can lead to unexpected flight errors and possibly crashes, therefore the procedure of calibration should be done very carefully.

3. **Frequency of Check Ups:** When more than one person is working on the drone, like in our case, connecting the drone to more than one laptop might easily change some values for important parameters in the drone. That could happen due to different configurations on each laptop, and this will lead to errors in flying it. Therefore, to avoid this scenario from happening, it is recommended to frequently reset the settings on the drone back to the default values, to make sure every value is exactly what it should be to perform a successful flight. A good communication between team members is also recommended to keep all members on the same page regarding the drone settings.

4. **PIXHAWK Abilities:** We noticed that PIXHAWK has many free unused pins on it, which creates the possibility of connecting more sensors in the future to it directly, instead of using an additional device, which would make communication between PIXHAWK and the extra device a complicated and unnecessary task.

5. **Flight Plan Distance:** The Iris quadcopter can reach a flight speed of 30 [Mph]. Although in our tests and experiments we didn't fly the drone outside a circle of 100 meters radius, where we would be standing at the center of that circle, but the speed of the drone was high enough to estimate the distance it can cover with the current battery power.

6. **Automation Algorithm:** Since we only need to feed the drone flight instructions only once prior to flight, this gives the chance to focus on the real time readings from the drone while flying to assure no errors are happening.

7. **Open Distance Advantages:** When the drone receives a mission to fly in an area with obstacles (Buildings, Trees, Street Lights, etc.) in the way, without the ability to sense these obstacles, it's impossible to accomplish the mission without raising the drone to an altitude that's higher than these obstacles. Adding sensors in the future will enable the drone to maneuver between the obstacles. Therefore, in the current

state of the drone, flying among open spaces is achievable without the need of any additional technologies.

Chapter 7
Conclusion

The experiments we conducted on the drone, and the results of these experiments prove that the drone can be efficiently used to carry a light package that weighs no more than 400 grams over a distance of 3 kilometers. The mission time varies between a range of [1-8] minutes depending on distance and wind turbulence, which is a spectacular result. The drone can safely achieve a flight mission with a 3 kilometers radius if mounting package, and 5 kilometers radius without package. This result exceeds the limits of all currently used methods of package delivery in terms of time, and clean energy. The drone is capable of receiving an algorithm of commands that can safely control the flight process during the entire flight, which means automation of flight is reliable and achievable. The drone is capable of tracking its own GPS location constantly during the flight, which enables the calculation of ETA of delivery, and also enables tracking the drone via control station, which leaves no chance of guessing or assuming the location of the drone nor its exact delivery time. The accuracy of the drone in locating the destination point is high enough to depend on it in delivering the package; the drone can land within a circle of 1 meter radius around the destination location. With these results and abilities, future work is very promising. By implementing more sensors and improving battery power among other aspects, the limits of the drone can be much more expanded.

Chapter 8
Future Scope

The use of drones for package delivery is highly promising, and within a couple of years, when they prove to be a reliable technology for package delivery, they will be globally used for that purpose.

Among the many features that could be developed, we list those who have the highest priority for a safe and reliable package delivery.

1. **Sensors:** To make the flight plan 100% automatic, the drone needs to have the ability to sense other objects, and it needs to sense them from a distance far enough to make a change in its flight coarse. We propose two types of sensors that could enable this feature in the drone: Proximity sensors, and Sonar sensors. We need to test these two types of sensors to determine which is more reliable.

2. **Package Delivery Mechanism:** The main goal of this project was to build the framework of the autonomous drone delivery. We have achieved the goal of flying the drone from point A to point B automatically and safely. What comes next is the development of package delivery mechanism. There are several solutions proposed for this feature. This feature can be broken down to two factors:
    a. Holding the Package: The way the drone will carry the object is yet to be determined. Some methods ensure the safety of the package so that it won't fall from the drone, such as using a container box and putting the package inside of it. This method will also protect the package from being scratches if it was held directly by a claw, and also the damage caused by weather. The disadvantage of this method is it will increase wind resistance which will lead to more battery power consumption hence less distance to cover.
    b. Identifying the Correct Customer: The drone should have the ability of delivering the package personally to the person who placed the order. This feature is very important if the package was a legal document or medication. In these cases, having the drone to just land in the garden or the backyard then dropping the package is not an option. We propose two methods at the moment to apply this feature. Facial recognition, and lock code. Facial recognition is helpful in identifying the right person, and decreases the chances of

delivering the package to the wrong person. A lock code on the package could be an extra security checkpoint, where a lock code could be given to the customer after the order is placed, so only him/her can open the box and take the package out. Electronic locks are the best in achieving this, since they can be used quickly.

3. **Battery Power:** Improving the battery power has two obvious advantages: Extending flight time and raising the weight limit of the package.
4. **Operation Algorithm Development:** All the previous features need addition of code on the software end. The sensors must have a driver that can process the readings of the sensors as an input, and changes the flight plan accordingly, which means sending flight commands as an output. The package delivery mechanism should also have a driver that could tell the drone whether the person standing in front of it is the right person or not, then either release the package or raise altitude to prevent the person from stealing the package.

References

Android GPS
http://www.vogella.com/tutorials/AndroidLocationAPI/article.html#locationapi

Socket in Java
http://cs.lmu.edu/~ray/notes/javanetexamples/

Troubleshooting the arducopter:
http://code.google.com/p/arducopter/wiki/AC2_Troubleshooting

Understanding the PIXHAWK:
http://copter.ardupilot.com/wiki/common-pixhawk-overview/

Raw Code for Arducopter
http://dev.ardupilot.com/

MAVProxy
http://tridge.github.io/MAVProxy/

Configuring Python
http://www.anthonydebarros.com/2011/10/15/setting-up-python-in-windows-7/

Arducopter Simulator
http://dev.ardupilot.com/wiki/setting-up-sitl-on-linux/

API for Google's Directions within Google Maps
https://developers.google.com/maps/documentation/directions/

Configuring the APM level pre-flight.
https://code.google.com/p/arducopter/wiki/AC2_Flying

Socket connection
https://docs.python.org/release/2.5.2/lib/socket-example.html

Iris and parts found at:
http://www.adafruit.com
http://www.3drobotics.com

Appendix A

```
from time import sleep
from thread import start_new_thread

def listen_phone():
        global xdata, conn
        conn, addr = xs.accept()
        print 'connected by', addr
        xdata = conn.recv(1024)
        if len(xdata)>0:
        xdata1, xdata2 = xdata.split(';')
        conn.send("confirm")
        print("confirm")
        with open("/home/jsmith/Desktop/waypoints.txt", "w") as wpfile:
        wpfile.write("QGC WPL
110\n0\t1\t0\t16\t0\t20\t0\t0\t0\t0\t0\t1\n1\t0\t3\t22\t0\t0\t0\t0\t0\t0\t5\t1\n2\
t0\t3\t16\t0\t20\t0\t0\t" + xdata1 + "\t" + xdata2 +
"\t5\t1\n3\t0\t3\t21\t0\t0\t0\t0\t0\t0\t0\t1\n4\t0\t3\t112\t30\t0\t0\t0\t0\t0\t0\t1
\n5\t0\t3\t22\t0\t0\t0\t0\t0\t0\t5\t1\n6\t0\t3\t20\t0\t0\t0\t0\t0\t0\t0\t1\n")
        print("xAuto Mode")
        mpstate.rl.line = "auto\n"
        sleep(2)
        print("xArming Throttle")
        mpstate.rl.line = "arm throttle\n"
        sleep(2)
        mpstate.rl.line = "wp clear\n"
        sleep(5)
        print("xSaving Waypoints...")
        mpstate.rl.line = "wp load /home/jsmith/Desktop/waypoints.txt\n"
        sleep(2)
        print("xThrottling Up Up Up and Away")
        mpstate.rl.line = "rc 3 1500\n"
        conn.send("enroute")
        print("xEnroute")
        else:
        conn.send("error")
        print("xError")
```

```
conn.close()

start_new_thread(listen_phone, ())
```