Drone Delivery System

Brian Krentz, Mayank Kumar, Eva Liu, Vinh Nguyen, Elvis Tsang, Erin Walk, Billie Wei, Gary Wen

Harvard-HKUST International Design Experience

Harvard School of Engineering and Applied Science and Hong Kong University of Science and Technology

August 7, 2015

Table of Contents

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The goal of the Harvard-HKUST Summer Design Experience 2015 was to develop a novel way of delivering small goods to customers. The project was given direction after research indicated that delivering goods within a 1 mile radius is inefficient in terms of time and cost for small businesses. The system developed in response to this need incorporates an application interface that customers use to order items, the drone itself, as well as a gripper and landing gear which are used for delivery. This document outlines the creative process taken in developing the drone delivery system.

The novel delivery system enables business owners to offer a curated selection of small items for drone delivery within a 1 mile radius to their customers. Customers can place an order on the application interface, and vendors can use the application to view pending customer orders. Vendors and customers can both track the drone on the application screen. In addition, customers receive a notification when their order is approaching so they can pick it up.

The resulting system currently can transport a small object to an address which is input in the application, and has basic object avoidance. The computer application communicates with an Android device using wifi, and then employs the radio telemetry to send messages to the drone. The application also enables business owners to track their products while they are being delivered, as well as customers to track their deliveries while in progress, creating an expectation for greater accountability and expediting delivery on a local level.

1. INTRODUCTION

In recent years, drones have become increasingly affordable for smaller businesses and hobby users rather than solely the government and large companies [1]. The growing prevalence of drones, as well as their rapidly increasing capabilities, make them an ideal system to work with. Despite negative connotations thanks to their military history [2], as the field of drone research expands, they become available for a wide variety of engineering projects, from gaming systems [3] to speakers [4] and delivery systems [5].

The goal of this project was to develop a novel usage for a drone system through identifying a problem which it could be used to solve. Initial ideation produced a wide range of ideas from a lawnmower drone to a drone which shows you to your seats for a performance. After narrowing down these ideas based on interest and feasibility, four main choices stood out from the rest based on usefulness and feasibility. The first was a party drone which played music and produced a light show, the second a drone gaming system which allowed users to play games based on the control of drones, the third a drone which dropped water balloons on unsuspecting victims, and finally the delivery drone which we ultimately chose.

Conducting surveys revealed that there was a need for the drone delivery system, more so than for the other systems. Interest in the project and feasibility also played a role in choosing which project to pursue. In the end, the delivery drone system was chosen for development due to the clear issues with current delivery systems, issues which this product can solve.

1.1 Initial Ideation Results

During the initial ideation process, the pros and cons of pursuing different drone projects were discussed and documented.

Table 1: Initial Ideation Results

Drone Type	Pros	Cons
Party Drone	● novel, fun idea ● clear understanding of the target audience ● easy to think of people who would definitely use our product ● easy access to target audience for testing	● why better than speakers and lights? ● safety issues, especially inside ● rather unfocused - do we do lights, and speakers, and music, or just one?
Games Drone	● definite demand ● clear marketable future ● challenging project complexity	● would need to program two drones to interact ● difficulty in inventing own game ● safety issues with children
Prank Drone	● clear single task to test	● questionable market ● needed more complexity ● potential danger

Delivery Drone	● interesting applications: use of tracking, maps, etc… ● clear market ● understanding of target consumer, target consumer is tech savvy	● will have to differentiate from Amazon

1.2 Problem Statements

Currently, businesses in Cambridge, MA perform between 50% and 100% of their deliveries within a 1 mile radius. However, surveyed customers, as elaborated on in the background research section, are accustomed to waiting an hour for these deliveries, and frequently pay $10 or more based on our preliminary survey on the business. Furthermore, vendors experience a lack of reliability from those delivering items, even with local deliveries. This may be partially due to the fact that many businesses deliver by bike. In response to this need, a more efficient delivery system for local deliveries must be developed.

The project presents two problem statements: one for the vendor who would purchase the product, and another for the customer who would be using the app to order from the vendor.

1. Young professionals need an efficient, affordable delivery system to deliver them goods from local stores because their busy schedule keeps them from going out, and they are willing to pay a little extra for convenience.
2. A business owner wants to target local professionals during the work day with a new delivery system which delivers goods to his customers within a more predictable time range and with lower operation costs

Though other companies such as Amazon are currently developing drone order systems for large scale delivery, this product envisions a system in which drones are affordable for small businesses performing local deliveries and advertising. It provides proof of concept for performing short distance deliveries by drone, using a phone app for ordering.

Surveying local businesses and consumers, and identifying a need in the market for cheaper and more reliable local deliveries, revealed that the intuitive choice for a target market was food delivery. Many of the businesses surveyed were restaurants, since many have begun offering these services to attract workers who do not have time for a long lunch break. However, wait time is usually around an hour and the cost for delivery is around $10. Deliveries cannot be tracked, resulting in a lack of accountability for delivery boys. The small time window and sub-par delivery services made it an ideal area in which to make a technological impact.

1.3 Target Users

The drone delivery system has two sets of target users: those who will purchase the system to use at their business, and those who will order from the business and choose drone delivery. Since the vendor is the one paying for our system, in a conflict of interest their requirements will take precedence. However, the device must also be customer friendly otherwise people will be unwilling to use it and businesses will not purchase it.

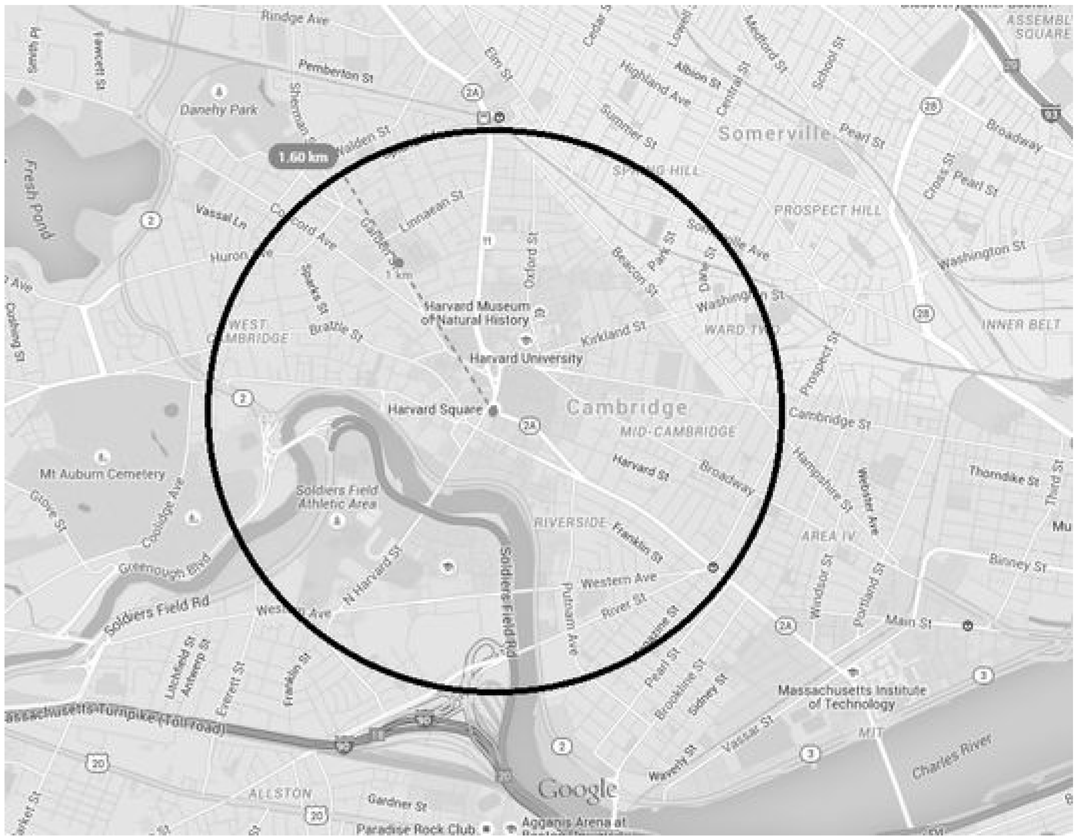


Figure 1: 1.6 Kilometer (~1 Mile) Radius from Harvard Square T Stop

1. The persona for a target vendor is Jackson, a young, local, tech savvy business owner. He can afford ~$1,500 investment into a drone delivery system, and often needs to deliver items of a smaller size. Many of his customers are local and would like to have products delivered to them. Similarly, he works in an urban area with many professionals who appreciate the time they save by using delivery services. He wants both a reliable way to send items to local customers and to create publicity for his company.
2. The target customer is Ella, a 22-40 years old graduate school student or young professional with an income less than or equal to $100,000 a year. She lives near her favorite stores and wants a more affordable delivery service. Her fairly busy schedule prevents her from running multiple errands, and she does not like unpredictable delivery times or long wait times. As a

result, she is willing to pay a little extra for the convenience of delivery. She also enjoys keeping up with technology trends and news.

Utilizing smaller drones to complete quick deliveries (about 1 mile radius) in urban areas is this project's unique response to address these needs. These drones are more affordable for small businesses, and have simpler systems which require less maintenance. The drones will be ordered through an app interface to appeal to young tech savvy customers. Furthermore, the current novelty of these systems will allow deliveries to double as advertising stunts.

2. BACKGROUND RESEARCH

2.1 Technology of the Quadcopter

Quadcopter, by definition, is an aerial vehicle with four rotors, aka propellers, that provide the thrust to lift and move the vehicle. There are three main rotations that the quadcopter can make by changing the spin rate of different motors. These are outlined in Figure 2.

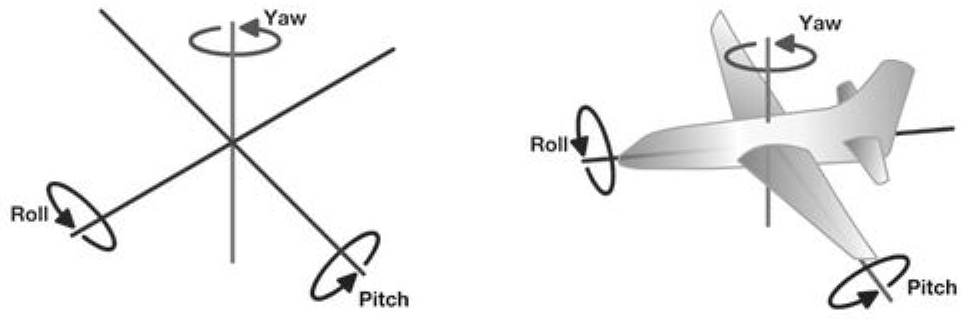


Figure 2: Schematic illustration of roll, yaw and pitch (image courtesy of theboredengineers.com)

There are many variations on the quadcopter design. Copters with 6, 8, or more rotors have been made. More rotors can give a copter greater lift, so it can have a heavier frame or carry more weight. A quadcopter has the simplest flight dynamics because only four motors need to be adjusted in flight. A quadcopter can come in multiple configurations depending on how the 'front' of the copter is oriented. A '+' configuration has the front pointed over one of the arms of the quadcopter, while the 'x' configuration has the front pointed between two arms. The configuration affects which motors need to spin faster to create rotation. There is no strong benefit to either configuration; however, the 'x' configuration is considered the standard for quadcopters, and thus was provided by the two universities for this project.

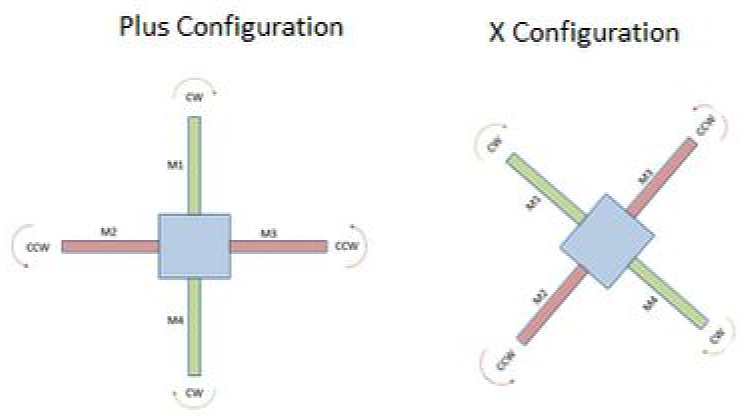


Figure 3: Schematic Illustration of 2 quadcopter configurations (image courtesy of technicaladventure.blogspot.hk)

The ‘brain’ behind the quadcopter will take in sense data from various sensors on the quadcopter to use as inputs in algorithms for calculating the attitude and motion of the vehicle. These algorithms will help the microcontroller determine the rotation speed for each motor. In our design, a PixHawk autopilot system was used as the ‘brain’ of the quadcopter. The PixHawk refers to the entire board that contains the STM32 microcontroller, sensors, ADCs (analog-to-digital converters), outputs to ESCs (Electronic Speed Controller), inputs from various connectors, and more. The PixHawk system includes five sensors: an accelerometer, a gyroscope, a barometer, a magnetometer, and GPS. Combining the sense data, algorithms, and navigation code is the onboard firmware. The PX4 is the firmware code being run on the PixHawk board. It is an open source platform for UAV development and control.

2.2 Firmware

Firmware code available at https://github.com/13rianK/Firmware

The PX4 Firmware is complex, but its layout follows a clear structure to allow for simpler development. The firmware has multiple levels. At the very top is the NuttX operating system which can run applications. A level below that are the modules. These are less abstract than the NuttX and deal with flight operation, eg. navigation, position estimation, attitude estimation, etc. The lowest level is the drivers. These are the files that directly interact with the PX4 board and the sensors on it. Connecting the drivers to the modules is uORB (micro Object Request Broker), an application for using data structures to communicate between different levels of code.

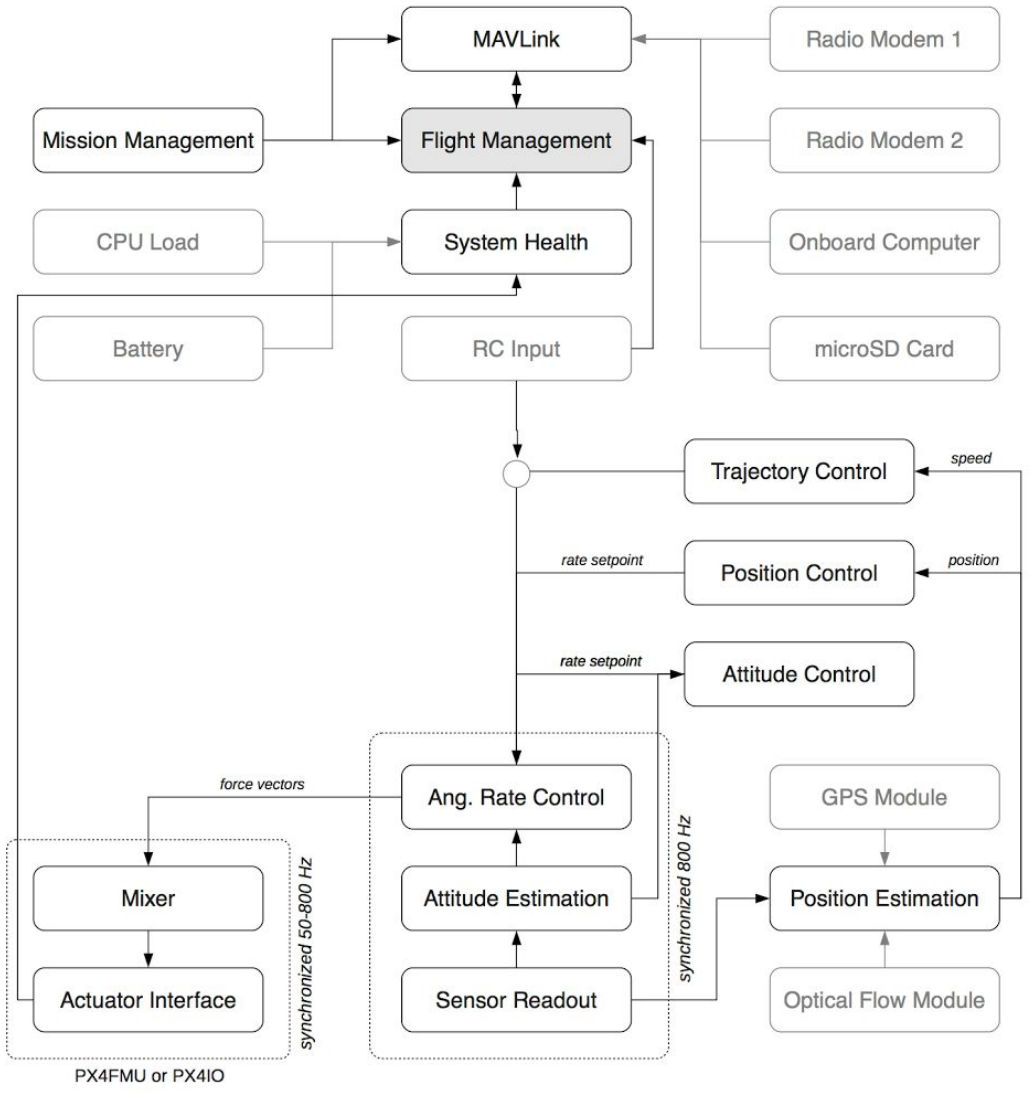


Figure 4: Illustration of Firmware repository structure

The PX4 performs multiple functions for the drone as illustrated in Figure 4. The main function is flight management. The flight is influenced by the mission management onboard, the MAVLink commands being sent to the quadcopter, and the various actuator controls. The sensors are used as inputs to the algorithms which calculate estimations for attitude and position. Those estimations will then feed into controls (where the copter is estimated to be vs. where it should be) which help guide the copter to where it needs to be through PIDs. Mission management is usually done by an offboard program to program missions on to the PixHawk. MAVLink is the communication protocol with the offboard devices.

2.3 Delivery Drones

Current technology in this area falls into two categories: drone delivery and food delivery apps. This project combines these two ideas for a novel product. The London based sushi restaurant Yo! Sushi is using drones to deliver burgers inside the restaurant [6]. These drones remain within the confines of the building, and are used as a flashy replacement for waiters. It is unclear whether or not the restaurant continues to use these drones now that the promotional period is over for the burgers. During initial testing, many expressed concerns over the safety of the drones, as well as criticizing how long it took to order, and several reporters raised questions as to whether the service would run at all [7], though it was eventually used promotionally [6].

The pizza chain Dominos has been testing delivery by remote-controlled helicopter [8]. They have not yet moved past the testing stage, and it is unclear whether they expect to create a functional system to replace delivery by car, or if it is solely a publicity stunt.

Similarly, Amazon has patented a system to deliver products in 30 minutes or less to customers [5]. This system, which uses eight rotor drones, operates in a relatively contained radius of 10 miles around the local delivery center, and it seems that they are hoping to incorporate it into their business model. However, it has not yet received FAA approval. As a result, they are currently only able to run tests and cannot actually offer this service. However, they have chosen to obtain a patent for their system as of April 30th, 2015 [9]. In this patent, they describe their product as “a plurality of unmanned aerial vehicles, each of the plurality of unmanned aerial vehicles configured to aerially transport items.” The system is controlled through some sort of management system. The overall requirements for the system in the patent are rather vague.

The fact that food delivery apps have been increasingly popular indicates that novel and simple food delivery is in high demand [10]. In fact, there are many different ones offering the same services which manage to co-exist in large cities. The delivery system developed through the course of this project combines the novelty and utility of drone delivery with the simplicity and popularity of ordering through an application interface.

2.4 Delivery Customer Surveys

One of the first steps after choosing to design a delivery system was conducting surveys in order to better identify customer requirements. The first survey targeted those who would be using the delivery system to order food and other items. Questions addressed topics such as what customers would want delivered, how much they would be willing to pay and how long they would be willing to wait. Surveying potential customers provided vital information to inform the system’s design requirements, as well as revealed the intent of the project and provided numbers with which to engage local businesses.

After sending out the customer survey, members of the design team went in person to talk to business owners in Cambridge. In total, 10 different businesses around the area were surveyed, and their responses are available in the appendix. These included a local sub shop, a local stationery store, and a regional pizza chain. Through discussing current delivery practices, insight was gleaned on how to improve on the current situation, as well as which values should be emphasized in the new system.

3. DESIGN GOALS

This section is a discussion of the specifications for the drone delivery system. These specifications were determined based upon survey responses, which revealed what vendors and customers expected in a system, as well as standards of safety. In many cases, the numbers for the specifications are less rigorous than what would be expected from a fully operational system. Though this will be discussed in further detail in the future work section, these expectations were lowered as a result of the time constraints on the project, and the necessity for proof of concept rather than a device which is ready for marketing. The main goal of these specifications was to guide the design process through setting expectations for the device which had to be achieved in order to provide proof of concept.

3.1 Customer Requirements

The two surveys shaped and ultimately determined the customer requirements for the system. The first survey was sent over mailing lists which design team members were subscribed to, as well as shared on Facebook and with family and friends. This survey was for those who would be using the application to order food items.

Survey Results

[For charts, please view Appendix]

- 33 people responded to survey version two, which included the demographic questions
- 73% of survey takers were female and 27% male
- 84.9% were aged 16-25
- 81.8% said that they normally ordered from 0-4.8 km (0-3 miles) away
- Meals, snacks, drinks, and small items (phone, keys, id, etc.) were amongst the most desirable items for delivery
- Users said that speed and affordability would be the most important features in a delivery system
- People expected to wait an average of 60 minutes for delivery within 16 km (10 miles)
- People also said they were most available to receive packages from 6:00 pm - 12:00 am and also from 12:00 am - 8:00 am (30.3% of respondents were available 12am - 6am, and 36.4% 6am - 8am)

To determine what the vendors required in a delivery system, the team went door to door at businesses in Harvard Square. A set list of questions guided the discussion, and one person would perform the interview while the other wrote notes. Many businesses were very open to talking about their delivery practices. To ensure that the questions did not bias their responses, the team used a script which was carefully adhered to. After these notes were compiled, they were analyzed for patterns in what the different businesses required.

Around half of the respondents did not have a delivery system in place because their orders were often too small for it to make financial sense. For the businesses that did do deliveries, almost all of them only delivered locally, with the exception of a large campus bookstore, which used UPS. Besides this bookstore, the businesses that delivered charged flat rates of $10+ on their deliveries.

Design specifications were tailored to correlate with the results of the survey. As a result, the customer side incorporates cheaper delivery costs and more reliability through constant tracking capabilities. The vendor requirements prioritize keeping the components as cheap as possible, and consider maintenance requirements in the costs.

When determining the numbers for the engineering specifications, current technology was considered. For example, the gripper specifications correlate with how frequently UPS loses packages. However, often ideal specifications for deployment were nearly impossible to achieve within the time limit. As a result, important and achievable requirements, such as not dropping packages, were given precedence and these tests were more rigorous. The other tests, such as those for proximity sensors, were kept looser and tests were designed to prove the concept rather than have it ready for immediate deployment at the end of the program.

The next section discusses the Software, Hardware and Mechanical requirements of the system, which were developed through consideration of customer needs and safety standards.

3.2 Software

3.2.1 Tracking

- Client and user can track the drone's path on the app as it is flying
  - Integration with QtLocation, QtPositioning, and OSM or Open Street Map
    - Map type API written in C++ for Qt
    - Mainly used for displaying map, pinpointing location and positioning
  - Place permanent pin on map to indicate business address
  - Geocodes users' addresses from entry form and places a pin on the map to indicate user address
  - Shows the flight tracker for users and vendors
  - Continuously receive updated GPS coordinates from drone (through MAVLink)
- Focus on pickup outdoors where take-off can occur in fields of at least 100 square meters for now for safety reasons, ease of tracking, and indoor flight regulations

3.2.2 Customer Friendliness

- Customer receives in-app notification (dialog window) when the package is within 10 feet of the destination so they can go outside to retrieve it.

- For the tests, the drone will be dropping off the package at the address that the user types into the interface.
    - Collect address information from users and use QtLocation and QtPositioning APIs to geocode longitude and latitude. Store and send this data to the client side.
    - In the future, the users might be asked to enable Location services so that the drone could drop the package as close to the phone as possible.
    - Customers would be asked to go outside or to their door so that the drone does not place the package on their roof.
- No need to ask customer to tip the drone
    - Charge a small delivery fee automatically with payment processing ***OR***
    - No delivery fee because the business owner decides to use the drone primarily as a promotional tool
- Do not want customers to try to touch the moving drone
    - Warnings in the application [in the "Approaching Order" dialog window]
    - Include a button customers can click when they have received their package so that the business can know

3.2.3 Control

- Wireless communication system
    - Has to serve distances of at least ~2.4 km (2 miles), which survey results (see appendix) indicate is the radius in which 50% to 100% of deliveries take place in Cambridge, depending on the business
    - Use the 915MHz (433 MHz in Hong Kong) radio telemetry that the remote control uses to control the drone from a computer. We changed frequencies as a result of what was available and legal in the US [11] and Hong Kong [12]. The range is about 1.6 km (1 mile).
    - Ideally in the future use cellular or an xbee/zigbee, which has a range of about 9.6 km (6 miles) [13], neither of which are subject to as much interference from cordless phones and other devices
- App allows option of manual control (control over throttle, yaw, pitch, & roll) of the UAV in case the user needs finer control of the system within 1.6 km (1 mile), since this functionality is vital feedback for the ultimate automatization
- At 5% battery, about 1 minute left of flight, if the drone has not returned safely, and is more than 0.16 km (0.1 miles) from “home,” it must land, send out its landing location and wait for a speedy manual retrieval
    - Vendor can press button for “beep” function, causing the drone to make a noise for location purposes
- App should warn workers if anything is going wrong with the drone (malfunctioning propeller, rotor, etc.)
    - Automatically return "home" or land if the issue is dire
    - Else give workers the option of continuing delivery
- Only allow the drone to leave the business with a fully charged battery

- Battery must allow for the drone to fly at least 4 km
    - 3.2 km max distance (1.6 km max delivery radius, and return) w/ safety factor
    - PixHawk has built-in failsafe measures for low battery
        - will auto-land on critical battery levels

3.3 Hardware

3.3.1 Flight Accuracy

- Fly to a second location ≤ 1.6 km (1 mile) away with 3 m accuracy (location is the center of a 3 m radius)
    - Test in the field with markers
    - 1.6 km (1 mile) is the limit of communication, and is within the range of the drone
- Placing a point at origin location and another at final destination on the map in application
- Return "home" automatically after the release of package, whether at destination or due to operational problem by programming “home” in UAV GPS
- Return “home” (programmed into GPS) if communication is lost

3.3.2 Flight Mechanics

- Adjust trajectory to not hit trees, buildings and other obstacles
    - If the drone is travelling at 7 m/s on average, and the proximity sensors can sense objects 7.5 m away, reaction time for avoidance must be at most 0.5 s, taking a safety factor to consider the time needed to implement avoidance motions
- Though ideally the drone would have high standards for avoidance, testing will be less rigorous so that it may be completed in the given time
- To test: Aim the drone at its travel speed at five different objects which present their own avoidance issues such as a tree and a building
- If the five objects are successfully avoided, the sensor system is accurate enough for proof of concept
- UAV should stay above 4 m, while the actual height depends on the real situations, in the air when it is not being loaded or dropping off a package in order to avoid cars, people, etc.

3.4 Mechanical

3.4.1 Gripper

- Must be able to hold a small item (< 300 g)
- Has a maximum allowable drop rate of 0.02%
    - Current UPS loss rate: 0.5 - 1.0%
- The center of gravity of the gripper should be at or very close to the center of the drone in order to have a minimal effect on the drone’s balance

- Needs to be able to be controlled electronically
- The gripper should try to minimize any movement from the item, and any other effects on drone flight

3.4.2 Hull

- Weight: should be minimized to maximize the carrying load (≤ 100 g)
- Strength/Toughness: Hull should protect electronic components from any damage in case of failure or accident, should be robust and capable of absorbing energy.

3.4.3 Landing Gear

- Must be of low weight (< 100 g to not encroach on a payload weight of 400g)
- Impact toughness: > 250 J/m
- Able to prevent the body of the drone from hitting the ground when landing
- Protect the frame of the drone
- Provide good stability when landing

3.4.4 Delivery and Landing

- Place object down carefully and release
- Drone should sense when object is released
- No object should ever be dropped from a height greater than 1 foot
- The definition of a successful delivery should include safety of item

3.5 Business

On the business side of the drone system, the goal was to maintain a cost of less than $1,000 USD per drone.

Table 2: Estimated Cost of Drone System in HKD and USD

	USD One UAV	USD Two UAVs	HKD One UAV	HKD Two UAVs
Parts	$600	$1,200	$4,651	$9,302
Labor	$190	$380	$1473	$2946
Total	≤ $1,000	≤ $2000	≤ $7750	≤ $15500

- Battery costs: $25 USD each ($194 HKD)
- Shipping costs: $10 USD in the USA and Canada ($78 HKD)

4. DESIGN APPROACH

The design process began with the materials (a drone) which were provided by the two schools, and the main work was identifying an issue to address. Once the problem was decided upon, brainstorming was done to determine other possible solutions and consider why using a drone was the best solution. To do this, the pros and cons of current delivery systems, namely delivery by bike and delivery by car, were considered. Most of the businesses in Cambridge use delivery by bike, since they solely perform local deliveries. Efficiency of delivery is a qualitative measurement of how rapidly the delivery can be made and how many deliveries can be made in one trip. As a result, this method is the "standard," or datum from which the other possibilities were measured.

4.1 Pros and Cons of Delivery Methods

One of the main differences between common delivery methods and the drone delivery system is that the cost and safety of a person are not at stake in making the delivery. This breaks down typical delivery conventions such as tipping, as well as making it very simple to incorporate a tracking system.

Table 3: Analysis of Delivery Methods

Delivery Method	Pros	Cons
Current Delivery (Bike)	● low maintenance cost	● small loads ● tracking rarely implemented ● inefficient (potentially long time due to lack of traffic predictability, few orders at a time)
Car Delivery	● can carry large loads	● higher maintenance and gas costs (average $9,100 USD per year [14] for the first five years) ● lack of predictability in traffic
Drone Delivery	● easy to track where the delivery is for both customer and business ● no additional salaries or employees ● traffic not an issue	● cannot carry large loads (otherwise larger drone is needed)

4.2 Design Decision Making Table

The Pugh matrix below shows the process of analyzing different delivery method which are currently in use. It shows that the drone delivery system is a preferable method to the current standards of delivery.

Table 4: Pugh Decision Table to Choose most Viable Delivery Solution

Performance/ Requirement	Current Delivery (Bike) - [Datum]	Delivery by Car	Drone Delivery
Reliability (Tracking)	0	0	1
Efficiency	0	0	1
Cost	0	-1	1
Load Weight	0	1	-1
Total	0	0	2

5. DESIGN DETAILS

Each aspect of the project presented knowns and unknowns in terms of what applications and materials should be used. The unknowns were resolved through experimentation, research and testing of the different possibilities.

5.1 Software

- Known:
  - Need a development platform to code the Android app
  - How to prototype the user experience
  - What core functionality the app should have
- Unknown:
  - To use Android Studio or Qt Creator
    - Documentation was examined for both Android Studio and Qt Creator, with some development taking place on both platforms. However, the MAVLink integration was much more intuitive on Qt Creator, so this platform was used since communication between the drone and the program was the critical path for the project.

        - When a more robust MAVLink library becomes available for Java, Android Studio will be considered for this application. Using Android Studio allows developers to utilize all the functionality of the Android devices, such as serial port, with greater ease. Also, there are more unofficial libraries for Android since it is open-source.
    - How to enable the communication function of an Android device
        - Though it is possible to use Android devices as USB hosts, it is difficult to do this in a C++ platform such as Qt Creator, since it requires Java wrappers. Furthermore, the QtSerialPort library only supports computer operating systems and currently is not workable for Android or iOS. As a result, the communication using telemetry was restricted to the computer application. We look forward to a newer version of Qt which officially supports serial port module.
        - WiFi is a good alternative but the limitation is that the drone and the phone have to be under the same local network.
        - Eventually we decide to combine above two communication methods together in our system. We use the computer as data transfer station. The mobile phone uses WiFi to communicate with computer while computer use 915MHz or 433MHz telemetry to talk to the drone.

5.2 Hardware

- Known:
    - Need to traverse to a given address and return
    - Need to activate a servo motor upon reaching location
    - Using a Pixhawk autopilot device to control the drone (one of the given materials)
- Unknown:
    - The Firmware repository
        - To understand the repository, it was first essential to understand how it was structured on a macro scale. From there, a more detailed understanding was required, e.g. understanding the different classes being used, uORB (micro Object Request Broker) topics which control communication inside the firmware, and PWM. The experiments for this were writing and compiling the code.
    - The proximity sensors analog output and the analog-digital-converter (ADC) conversion of this sensor data for the PX4.
        - To understand how the proximity sensors worked they were characterized using an oscilloscope to measure the output voltage while the sensor was moved to known distances from a wall. To understand the ADC the same experiment was performed, but read in the ADC values instead of the raw voltage.

5.3 Mechanical

- Known:
    - To design a gripper which loads the delivery item and drops it at a specific command
    - To design landing gear to absorb shock on landing/crash
    - To design a shell to house the electronic components of the system

- Unknown:
    - The actuator used to activate the gripper system.
        - In order to choose the actuator used in the gripper system, we wanted something that was lightweight and would be easy to control from the drone. Since we wanted the gripper & load to be <300g, we decided to set a limit on the weight of the gripper system at <30g, so that it would only take 10% of the weight. In order to figure out which actuator was easiest to control, it was necessary understand how the drone sent power to its motors.
    - The material used for landing gear and shell.
        - Deciding the material of the landing gear required a consideration of the design requirements. The landing gear needed to do multiple things: raise the overall height of the drone for easier takeoff, have some shock absorption to minimize damage during hard landings, and provide a stable base to make it difficult for the drone to flip over during uneven landings. It also had to be of light material.
    - The design for the 3 mechanical components.

6. DESIGN EVOLUTION

The design has been developing in three different parts: 1) software (app interface for customers and vendors), 2) hardware (trajectory, stability, simulink model and analysis and etc.) and 3) mechanical (gripper, landing gear, hull and propellor shields). Each team faced diverse and distinct challenges and setbacks throughout the summer. The communication between these parts in a system components level is shown in Figure 5.

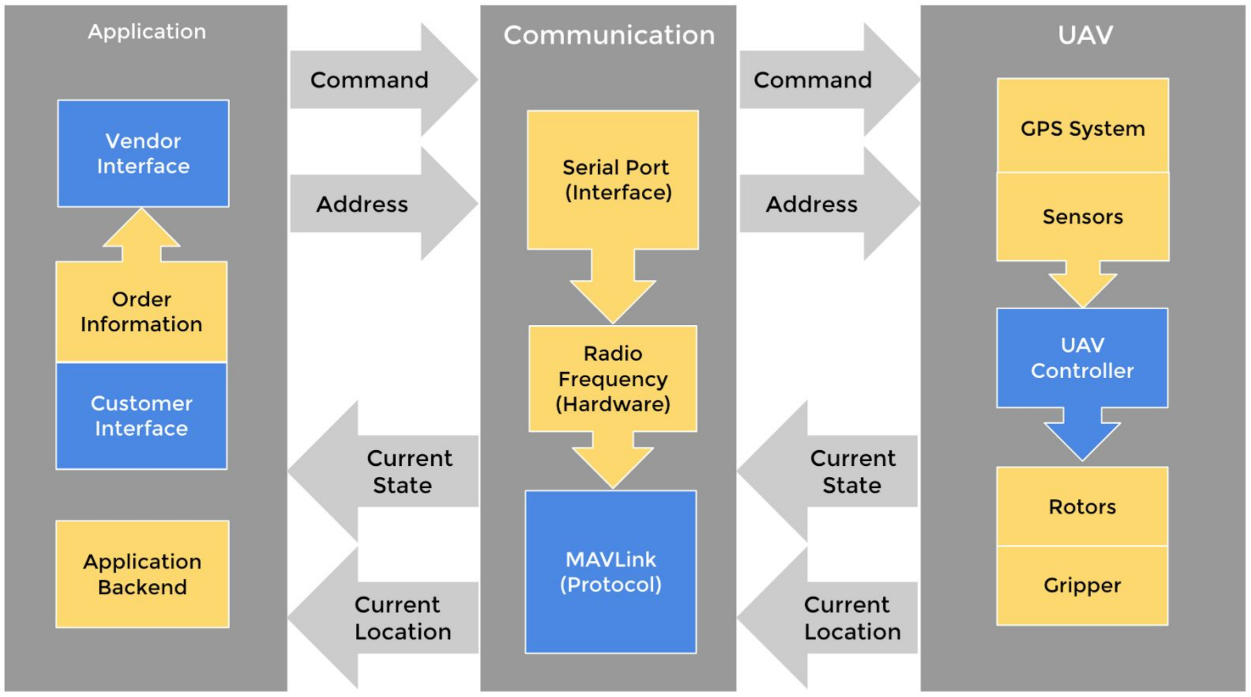


Figure 5: Communication Between System Components (critical components in blue and secondary components in yellow). Grey arrows show direct transfer of commands and status information between system parts. Yellow and blue blocks with arrows show data that is being transferred within each system.

6.1 Software

The software side included application development in two main parts: 1) interface and functional design, and 2) communication between the drone and the computer or mobile device. For the user interface, a paper prototype was developed first to test the usability of the application. Unexpected events and responses to the application were also considered to take care of the real situation. After about three iterations of the initial design, we arrived at the final paper prototype shown in Figure 6.1.

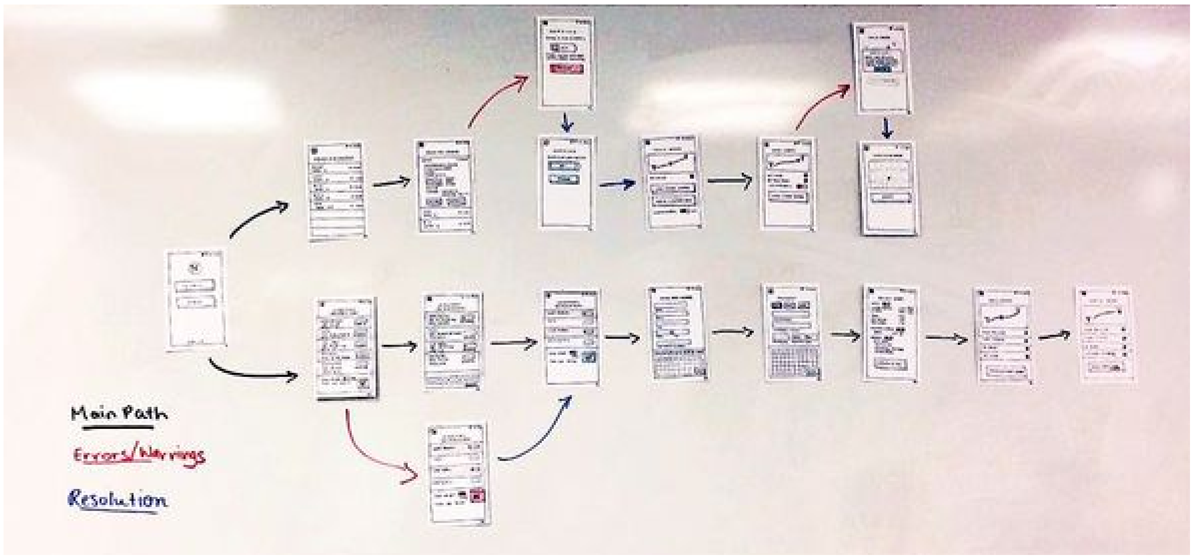


Figure 6.1: Paper prototype of the GUI of the application. The black arrows indicate the normal path, the red arrows indicate errors, and the blue errors indicate the resolution.

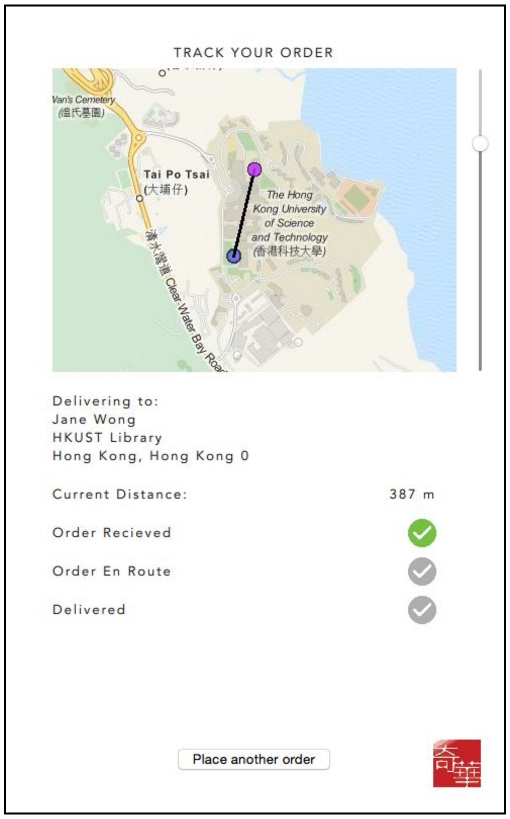
The most suitable options for the mobile application development program were the Android Integrated Development Environment (Studio) and Qt Development Environment. Though Android Studio is the more intuitive platform, with more robust styling capabilities, the libraries for MAVLink, which is the protocol for communication of the UAV over radio frequency, that exist for Java are not as robust as the ones available for C++. As a result, Qt, a development environment that allows coding in C++ and QML (Qt Modelling Language, a JavaScript based user interface markup language for page design), building an Android .apk packet, and then deploying to an Android device, was a better choice for the project. In addition, QML allows developers to build user interfaces in a declarative way. It is relatively simpler to compound QML objects and configure them using property bindings.

The second difficulty was the map and trajectory display. Initially, the Google Maps API and Google GeoCoding API for the trajectory feature were the first choice for the application due to the excellent precision, stability and resources. However, the Qt C++ client that uses Google Maps API is unreliable and has poor documentation. Instead, QtLocation and QtPositioning APIs were used because despite their lower visual quality, they are more reliable and work properly. The QtLocation API enabled accessing and presenting map data, querying for an address, and adding lines and circles. The geocoding is not highly accurate, so extra steps were taken to index into an array of positions and coordinates. The predicted trajectory of the drone was then illustrated by connecting the vendor and the destination points with a straight line, which would be sufficiently similar to the actual trajectory of the drone. In order to create the map type in .qml, we had to implement an Address type, a Map object, map plugin, GeoCodeModel, and MapItemView (code available as VendorTrackPage.qml in the Appendix). The functions of each are described below:

Table 5: Description of Map Parts in QML

Address	Structured address for use in queries and results of geocoding.
Map	A QtQuick item that displays the map on the screen. For unknown reasons, the default height and width is 0,0 and these values are undocumented, leading to unnecessary confusion.
Plugin	A location-based services plugin provides data including map data which is then displayed in a Map object. We used "osm" (Open Street Map), an open-license map.
GeoCodeModel	Queries the Plugin for geocoding translations (address to gps coordinates) and provides access to results via indexes in the model.
MapItemView	Populates the Map with overlay objects based on the data provided by a model, for example, a pin for the vendor / customer's location.
MapCircle	A geographic circle that we use to indicate addresses and the drone location.
MapPolyLine	A polyline that connects the origin and destination points.
Slider	Controls the value field for the zoomLevel of the map

The map, with the home and target locations as well as the predicted trajectory, is shown in Figure 6.2.



Delivering to:
Jane Wong
HKUST Library
Hong Kong, Hong Kong 0

Current Distance:	387 m
Order Recieved	✓
Order En Route	✓
Delivered	✓

Figure 6.2: Trajectory display on the customer side of the application

The third difficulty was related to the code collaboration. GitHub has a steep learning curve, which lead to difficulties since Qt does not easily allow merging files directly inside its text editor. Working on files at the same time was difficult.

Developing the application on Qt Creator resulted in a simpler programming process for the background. Since the translation of C++ and QML code to Java and XML takes place in the background functions of Qt, it was, however, sometimes more difficult and time consuming to debug the application. Qt being mainly used to develop computer applications instead of mobile devices, the support on Android platform is less than that on Windows, iOS and Linux. It took a number of attempts before the application was successfully deployed onto an Android phone.

Upon completion of integration between the QML interface and C++ code, the Android application provides most of the functionality missing only this final communication integration. We solved this issue by adding the Qt Network module and using Transmission Control Protocol (TCP). We run computer ground station as a TCP server connecting Android app and the drone together.

The last but not the least important difficulty was the formatting inconsistency on the Android devices. What looked correct and user-friendly on a computer screen during development looked very different on the Android device. Since running Android simulators on the laptops took too much memory to deploy properly, an actual device was used to see how each formatting change would affect the Android screen, which became a time-consuming process. The functions that retrieve the screen size for the windows did not operate well on the Android device. We solved this by calculating page, button, object dimensions, and spacing in order to make the application responsive to multiple devices. The final iteration is functional on both the computer and Android devices due to use of page width and length percentages in object placement.

Computer Appliction Code available at https://github.com/billiewei/uavattempt

Android Application Code available at https://github.com/billiewei/dronedeliveryapplication

6.2 Hardware/Firmware Design

6.2.1 Overview

The hardware issues to solve broke down into several areas: control, keeping the drone stable during flight; trajectory, guiding the drone along a path to its destination and back; object sensing, creating awareness of the area around the drone; object avoidance, what to do when it has sensed an object; and engaging the gripper from the pixhawk. Creating a detailed model of the quadcopter, which was not necessary for flight but was beneficial for our understanding of the system, was also considered. The design process began with first building the quadcopter and testing the features already on it. Next, the best ways to add sensors on to the PixHawk were researched, the Firmware codebase was explored, and PID values were tested to determine the best solutions. After a thorough amount of background research, writing new code in the firmware began, to allow the UAV to perform the trajectory and object avoidance desired. Lastly, all of our code was tested on the quadcopter and adjustments to the code were made as needed.

Firmware code available at https://github.com/13rianK/Firmware

6.2.2 Control

Control of the copter was obtained through position control and altitude control. The quadcopter has a magnetometer, barometer, accelerometer, gyroscope, and GPS that feed in sense data to the PixHawk, the autopilot platform controlling the drone. The PixHawk already has algorithms that can estimate the state of the quadcopter and then stabilize the drone, but it needs help determining the PID values. The PID values adjust how rapidly the system will respond to inputs. The software uses a Kalman filter and several PID Control Loops to calculate the attitude and adjust. The task was to find the optimal PID values that would allow the drone to respond in the most optimal way. The roll, pitch, and yaw each have their own PID coefficients for the altitude control and position control has control for the X,Y, and Z axis. An experimental approach was taken to determine these coefficients.

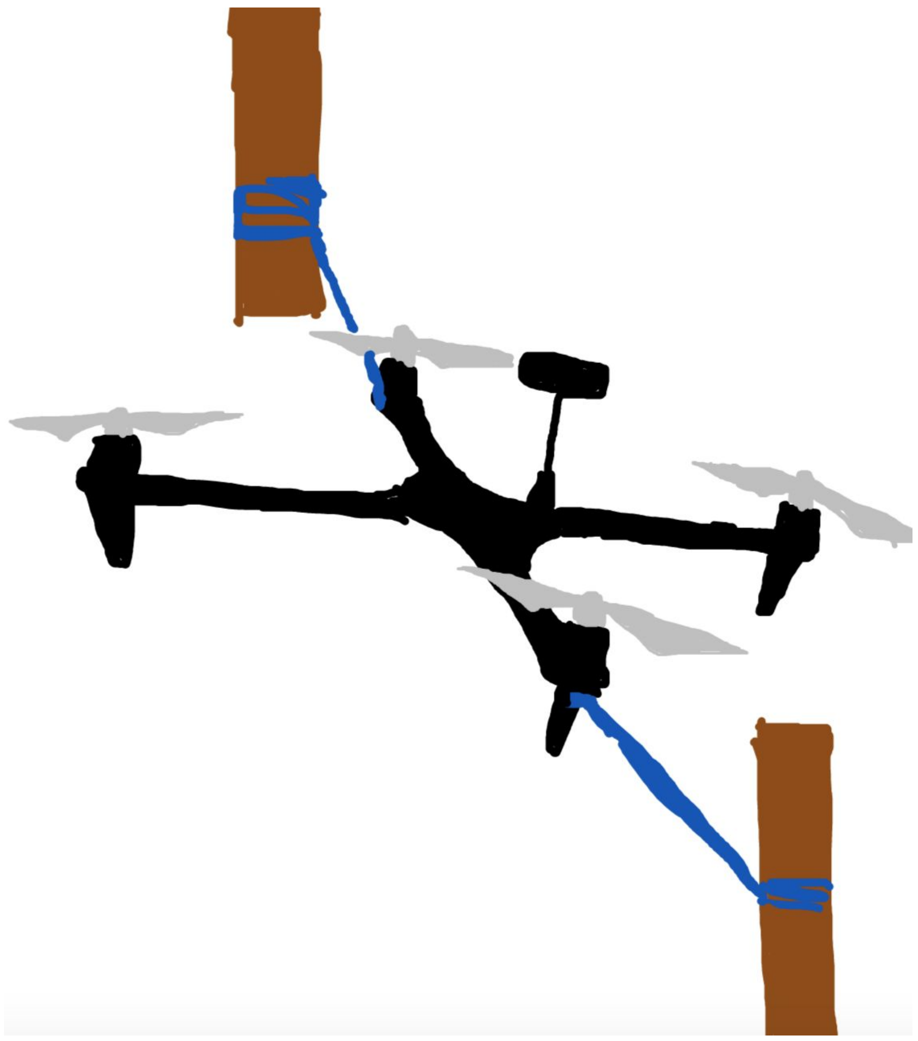


Figure 7: The quadcopter PID setup. It is suspended between 2 poles with a string (blue).

To best determine the optimal value, the quadcopter was isolated so it could only rotate freely around one axis (see Figure 7). With only one degree of freedom, the roll, pitch, and yaw could be explored individually. For each one, the PID values were chosen based on which one gave an optimal response. An optimal response in our case was one that gave the desired motion very quickly, but not so quickly that it would experience overly large overshoot or oscillations. Please look at Appendix item G, PID', for a detailed description of how different PID values affect response times in a quadcopter.

6.2.3 Navigation

There have been many challenges implementing the trajectory of the quadcopter. The most difficult to overcome was the lack of documentation for the PX4 Firmware repository. The ideal trajectory control was to have a single switch, that when turned on, would cause the drone to take-off, fly to a destination, drop off a package, return to its launch site, land, and then disarm itself. The way the firmware is organized has a command code running that adjusts the MAIN_STATES according to RC switches or MavLink commands. These states then determine what navigation files should be implemented. To accomplish the delivery routine a new navigation class was made (in the files delivery.h and delivery.cpp) which could be activated by a new MAIN_STATE. A new priority for the modes has to be created for the delivery state (see figure 8). This state should hold higher priority than all of the other flight modes except for RETURN. The return switch is essential for safely returning the drone to its launch location in the case of emergency and so should always maintain the highest priority, but the delivery routine should take preference just behind it so that the states of other switches cannot affect the flight routine.

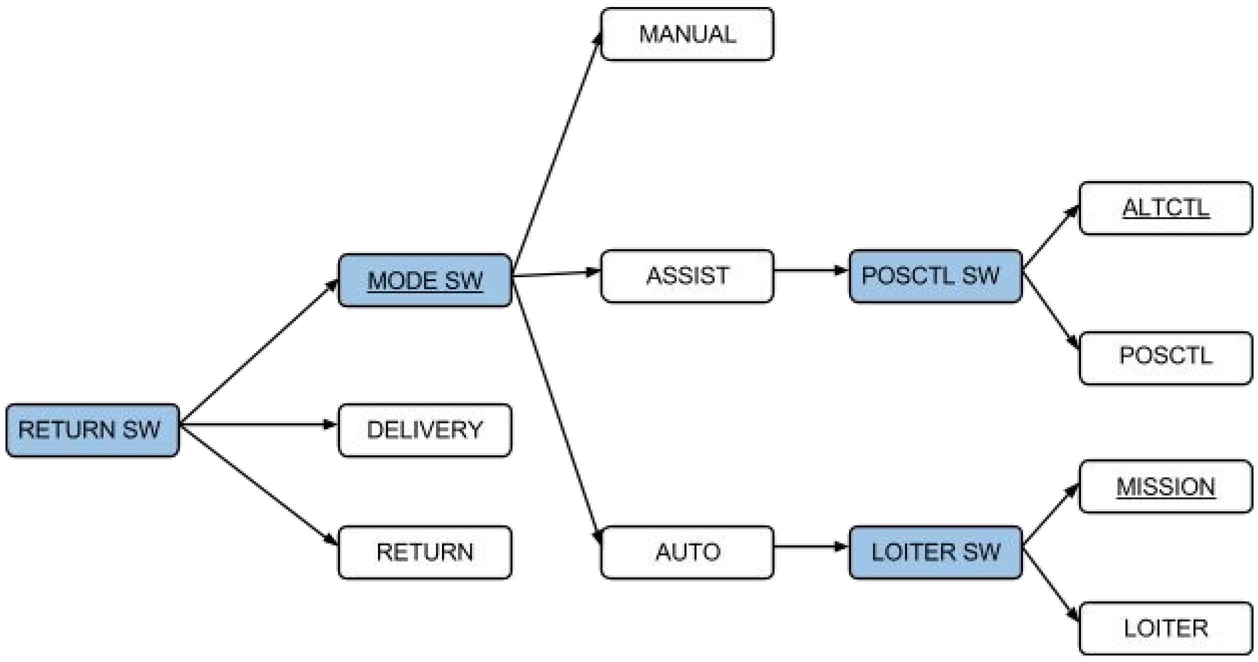


Figure 8.1: Main flight states depending on RC switch positions. Colored boxes indicate separate switches which may be turned off or to any of their corresponding settings, as denoted by the arrows. Underlined states are the default overriding state for the case where the parent switch is in the off position.

Table 6 - Flight States with Definitions	
MANUAL	Allows user control over the throttle, pitch, yaw, and roll, but will maintain a stable attitude
ASSIST	Maintains a stable attitude and uses either POSCTL or ALTCTL
ALTCTL	Maintains a stable altitude using GPS and barometer (will keep stable z, but not x&y)
POSCTL	Maintains a stable GPS position (will keep same x,y,z position)

AUTO	No control from the RC. The quadcopter will either follow waypoints or loiter
MISSION	Follows waypoints stored in memory, if none stored then defaults to loiter
LOITER	Maintains its GPS position at current location
RETURN	Returns back to its home position at a set altitude before landing
DELIVERY	The quadcopter will carry out a delivery routine (described in further detail below)

For this next section, please look at appendix item "delivery.h" for function references (delivery.cpp, the implementation of the header file, can be found at github.com/13rianK/Firmware/tree/master/src/modules/navigation). The delivery routine is a subclass of the MissionBlock class (which is a subclass of the Navigation class). The MissionBlock class was chosen to be the superclass because it has functions that are helpful for using waypoint navigation. PX4 Navigation classes have 3 main functions: *on_active()*, *on_inactive()*, and *on_activation()*. When the main command file is running, it will set each class to be active or inactive based on the RC controls or MavLink commands. The delivery file did not have much to do while inactive. For that function, it would only update the mission items if they were changed. When activated, the delivery file would turn the flight states and variables to starting values. There are six delivery states: DELIV_PREFLIGHT, DELIV_ENROUTE, DELIV_DROPOFF, DELIV_RETURN, DELIV_DISARM, and DELIV_COMPLETE. The *on_active()* function checks the delivery state and implements the appropriate function depending on what is needed. For example, when enroute, the *to_destination()* function is called. This will cause the quadcopter to start reading in the mission items one by one until it is finished. When the delivery state is finished, *advance_delivery()* moves the delivery state to the next one. This continues until the DELIV_COMPLETE state is reached, where the quadcopter will be disarmed and be waiting for the vendor.

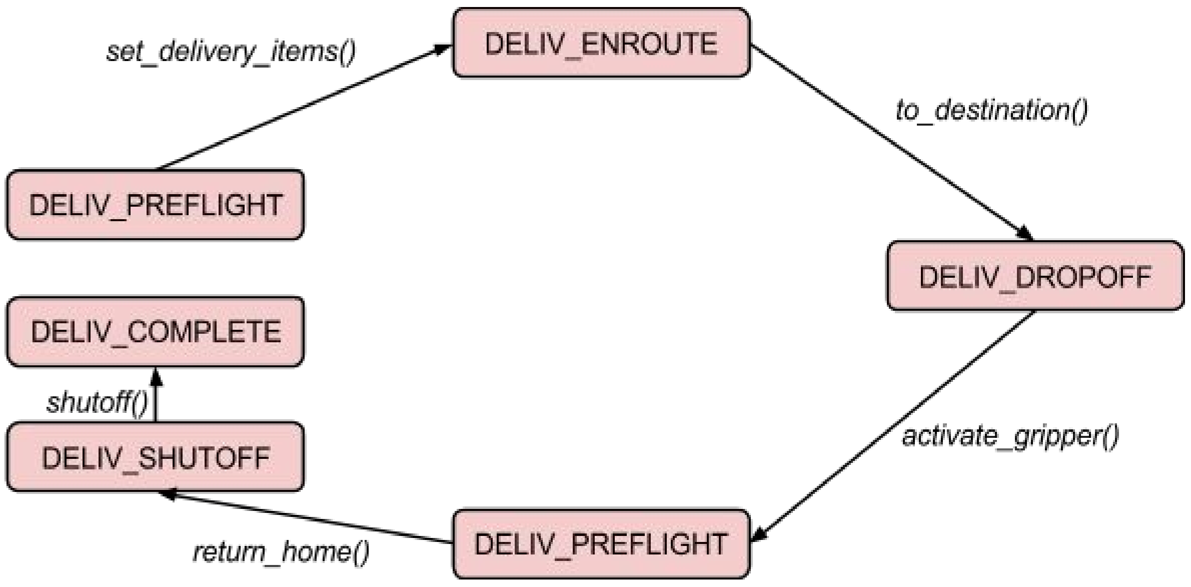


Figure 8.2: Finite State Diagram of on_active() for delivery.h

Implementing a way to activate the delivery file was quite complex. There is no documentation on how to do it and there are ~10 files that need slight changes to implement new modes. Originally, the plan was to use a new switch that would activate the script, but there are limits to the number of RC channels. Instead, the return switch was changed from a 2-position switch to a 3-position switch. A switch holds a decimal value from 0-1 depending on its position. A 2-position switch only will have one threshold value, while a 3-position switch has 2 threshold points that need to be set to differentiate the switch positions. This method still required implementing new navigation and main states, but would not require a new RC channel. Once the new states were created, they needed to be activated at the right time. The switch will cause the quadcopter to go into MAIN_STATE_DELIVERY. When in that main state, the NAVIGATION_STATE_DELIVERY would then be activated. It is this state that sets various requirements for the delivery file (e.g. the GPS has to have signal) and runs the delivery class.

6.2.4 Object Avoidance

For the desired end-goal of having a quadcopter which could navigate around buildings and trees, the drone would need to be able to sense the volume surrounding it. There were many ways that were researched to accomplish this. For example, there are ultrasonic, capacitive, inductive, magnetic, laser, photocell, radar, and a few other ways of sensing distance; however, because the quadcopter needs to sense buildings and trees, none of the sensors that required metallic/magnetic objects could be used. Lasers have a very spatially narrow field of vision, which makes detecting objects that aren't directly in front of the laser more difficult, and receive a lot of interference from the light outside. Ultrasonic is the best type of sensor for a moving vehicle because it has a wide field of vision for detection and will not receive interference from fluctuating amounts of light. The ultrasonic sensors chosen (MB1240) had a field of view that extended out to 7.5 m and 1 m wide. They also included acoustic noise rejection. These sensors could also be chained together somewhat easily to allow multiple proximity sensors on the same ADC input. Chaining together six of the sensors gives us detection in every direction; however, When the sensors are chained together it cannot be determined which sensor triggered the response. To solve this, the future design for the quadcopter would use both ADCs on the quadcopter. The first ADC would chain together four sensors and sense the x-y plane. The second ADC would chain together two sensors and sense the z-axis. In the current state, the quadcopter only has one proximity sensor attached (figure 9) to prove object avoidance can be accomplished. Look at the section 'Future Work' for a more detailed description of how multiple sensors will be used. Placement of the sensors on the copter was also critical for minimizing noise and interference. The greatest amount of noise is created at the very edge of the propellers, while the least amount of noise is found near the center of the body, so the sensors were placed closely to the body of the copter.

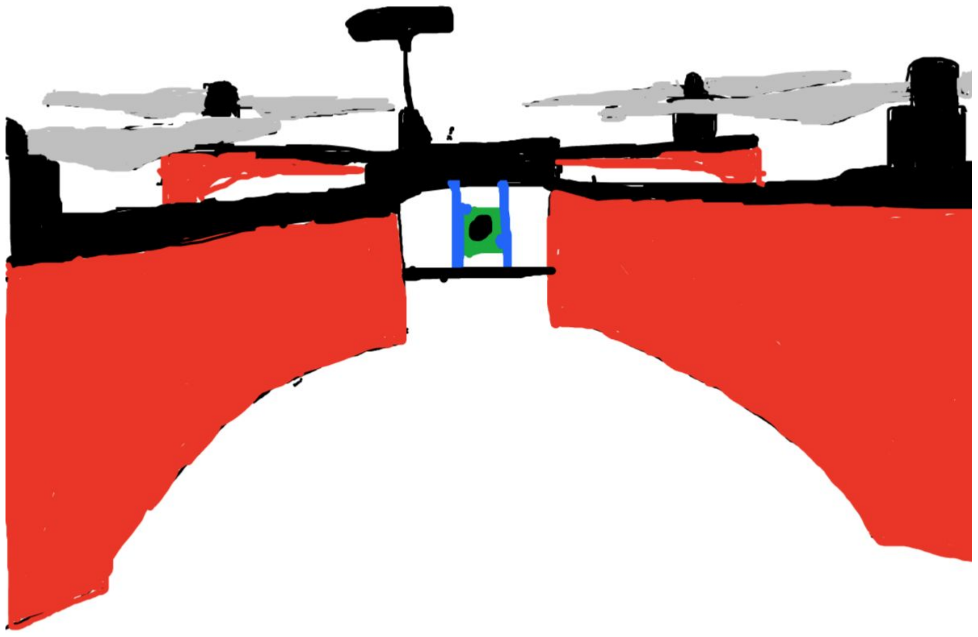


Figure 9: ADC (green) placement on the quadcopter (black frame, red landing gear, grey propellers). It is facing forwards and connected with zip-ties (blue).

Using the input collected by the ADC from the proximity sensors, a proper response to objects had to be determined. Right now, a threshold value has been set for the ADC. When it drops below that threshold, an object has been detected too close to the quadcopter, so all motion will stop and the quadcopter will hover. The quadcopter will then slowly rise upwards to clear its forward trajectory. When the threshold has been cleared, the quadcopter will then resume on its delivery.

6.2.5 Simulation

For modeling the system, initially simulink was used to create a representation of a mathematical model to describe the dynamics of the system. A quadcopter exists in two coordinate frames. The first is the inertia frame. This is a static frame, z is vertical and the x-y plane is parallel to the ground. The angles θ, ψ, and ϕ are the pitch, yaw and roll respectively, and it is the inertia frame that the angles are referenced to (see figure 10). The second coordinate frame is the body frame. This frame keeps the front of the quadcopter facing the positive x-axis, left side facing the y-axis, and the z-axis is going vertically through the quadcopter.

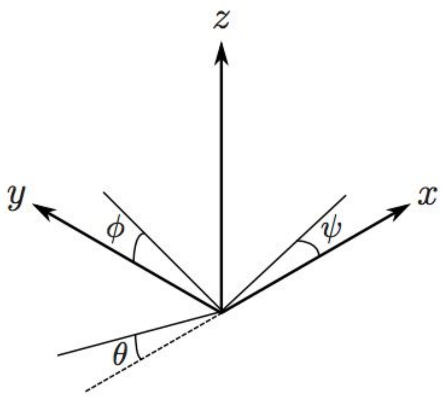


Figure 10: Inertial frame with corresponding angles

The rotation matrix, R (shown below), is the matrix for changing from the body frame to the inertia frame.

$$R = \begin{bmatrix} cos(\phi)cos(\psi) - cos(\theta)sin(\theta)sin(\psi) & -cos(\psi)sin(\phi) - cos(\phi)cos(\theta)sin(\psi) & sin(\theta)sin(\psi) \\ cos(\theta)cos(\psi)sin(\phi) + cos(\theta)sin(\psi) & cos(\theta)cos(\psi)cos(\phi) - sin(\phi)sin(\psi) & -cos(\psi)sin(\theta) \\ sin(\phi)sin(\theta) & cos(\phi)sin(\theta) & cos(\theta) \end{bmatrix}$$

There are two important equations that were used for the model. The first is a sum of forces equation (EQ 1 in figure 10). m is the total mass of the quadcopter (in Kg), R is the rotation matrix, T is the thrust (in Newtons), and K_d is the drag coefficient. This equation relates the acceleration to gravity, thrust, and air drag.

$$m \begin{bmatrix} \ddot{\vec{x}} \\ \ddot{\vec{y}} \\ \ddot{\vec{z}} \end{bmatrix} = -m\vec{g} + R\vec{T} - k_d \begin{bmatrix} \dot{\vec{x}} \\ \dot{\vec{y}} \\ \dot{\vec{z}} \end{bmatrix} \qquad (1)$$

The second is an equation for angular acceleration (EQ 2 in figure 10). It is another form of the equation $Torque = Moment\ of\ Inertia * Angular\ Acceleration$. R is again the rotation matrix, $\acute{\omega}$ is the angular acceleration in the body frame, L is the angular momentum, K is the thrust coefficient, ω is the rotational speed of a motor, I is the inertia, and b is a torque coefficient. All of the dotted parameters imply time derivatives of the original parameters.

$$\begin{bmatrix} \ddot{\vec{\theta}} \\ \ddot{\vec{\phi}} \\ \ddot{\vec{\psi}} \end{bmatrix} = R\dot{\vec{\omega}} = \begin{bmatrix} LK(\omega_1^2 - \omega_3^2)I_x^{-1} \\ LK(\omega_2^2 - \omega_4^2)I_y^{-1} \\ b(\omega_1^2 - \omega_2^2 + \omega_3^2 - \omega_4^2)I_z^{-1} \end{bmatrix} \qquad (2)$$

The simulation consisted of three main subsystems. Four motors were the first subsystem. The input and output are the voltage and the rotational speed and thrust respectively. The sum of forces was the second subsystem. This took in the total thrust and angles θ, Ψ, and Φ to determine the accelerations in the *x*, *y*, and *z* directions using Eq. 1. The third subsystem was the sum of torques. This took in the rotational speed of each propeller and would output the angular acceleration using Eq. 2. The initial equation for the sum of torques was nonlinear and had to be simplified into a linear form. There were numerous parameters for the simulation and they were calculated based on the dimensions of the quadcopter, but many guesses and estimations had to be made (especially for the motors). The simulation ended up being just a tool for further understanding the quadcopter, but did not become part of the final design of the quadcopter. This was because there was some difficulty finding accurate parameter values, and without including PID controllers in the simulation it was impossible to create a stable system. The PID controllers were not included due to time constraints on the critical path for developing a prototype.

6.2.6 Gripper Control

An important part of the delivery drone was enabling a gripper mechanism that would drop off the package successfully at the delivery point. The mechanical design of the gripper will be discussed in more detail further on in the report, but the final gripper design required control over a servo and the ability to turn it slightly over 90 degrees. Servos, specifically the TowerPro SG90 used in this project, turn a certain distance depending on the PWM, or pulse width modulation, they receive from a source. They get this from a signal input, which pulses at 50Hz. The range of the pulse widths that they take is usually between 1-2 ms (Figure 11.1). The SG90 datasheet states that 1 ms turns the servo to -90 degrees from neutral position, and that a 2ms pulse width turns the servo to +90 degrees from neutral position, where neutral position is the point in the middle of the servo's range of motion.

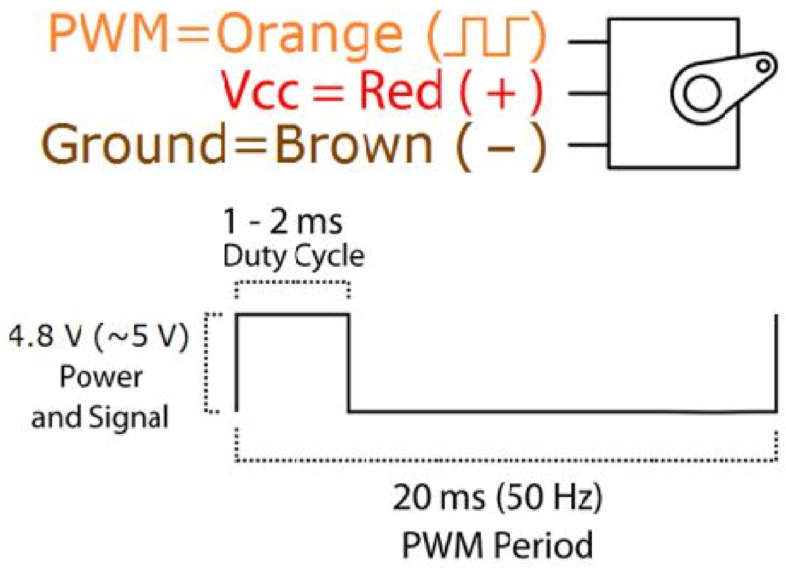


Figure 11.1: PWM specifications for TowerPro SG90. Servo needs a 50Hz square wave to turn the motor. A 1 & 2 ms pulse turns the motor 90° to the left & right, respectively.

Characterizing the particular servo used led to some different results from the datasheet. The SG90, when unpowered, has a range of motion slightly greater than 180°. The servo used turns to its leftmost position at 0.8 ms, and turns slightly to the right of neutral position at 2ms (Figure 11.2). The PixHawk's maximum PWM is set to 2 ms, so the servo couldn't be taken any further.

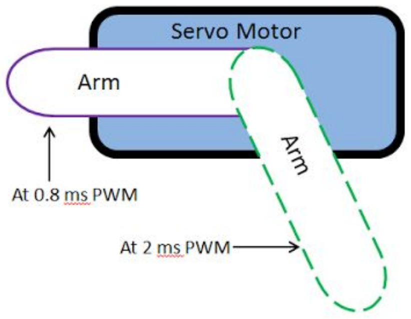


Figure 11.2: Characterization of Servo The positions of the servo with a 0.8 ms pulse width (-90 from vertical) and a 2.0 ms pulse width (slightly right of vertical).

The servo must be connected on the AUX output of the PixHawk device, since the MAIN outputs are reserved for the motors used for flight. However, one of the characteristics of the PixHawk AUX outputs is that the Pixhawk by itself doesn't provide the power necessary to run the AUX outputs; it needs a driver to provide power. This is provided by a LM2596S-ADJ power converter, which took an input from the same battery that powered the drone, and provided a 5V output. This 5V output was connected to the PixHawk, which served as a power driver for the rest of the AUX outputs, and powered the servo motor (Figure 10.3).

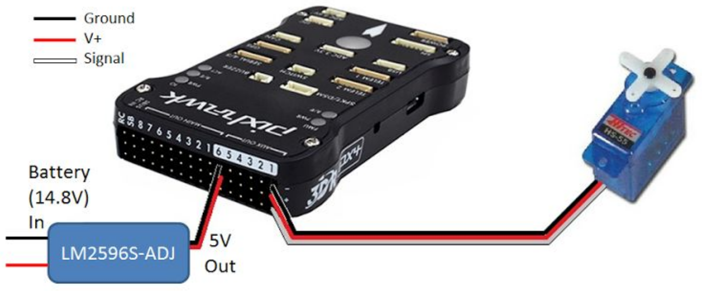


Figure 11.3: Connecting Power and Servo to the Pixhawk. The LM2596S-ADJ power converter takes an input from the battery and outputs 5V into the PixHawk AUX OUT pins to power the servo plugged into AUX OUT. Adapted from copter.ardupilot.com.

The servo is now characterized and fully powered by the battery and PixHawk, so the next step involves sending it commands to turn via the signal output of the PixHawk. In order to do this, a program that sends certain PWM values to the servo based on whether it is told to be open or closed was created., named servo_ctl.c (in the Appendix). It was first made to be run in the command line to test its functionality. This involved creating a function named *servo_ctl_set_pos* which initialized the PWM functionality of the PixHawk outputs, and would set the PWM to the pulse width specified in its arguments. Two functions names *servo_ctl_pos1()* and *servo_ctl_pos2()* called *servo_ctl_set_pos* and set 800 us and 2000 us as input parameters, respectively. After calling these functions in *servo_ctl_main*, along with some logic to handle parameters, the servo would open and close with the commands *servo_ctl pos1()* and *servo_ctl pos2()*, respectively.

In an attempt to make the servo functional within the delivery script, a uORB topic was created. In the delivery script, when the servo position needed to be changed, the new position was published to the topic turn_servo. Within servo_ctl.c, edits were made so that it could begin running a thread on startup. This thread runs in the background, checking for changes in the turn_servo topic. If it detects an update, it will check the topic to see which position it should be in, open or close, and turn to the appropriate position. This was implemented fairly late in the design process, and is fairly buggy. There is an issue in the receiving portion of the uORB requests by servo_ctl.c, so it won't activate during delivery yet. However, one of the switches on the radio control has direct control over the AUX 1 output, where the servo is, so there is currently manual control. This allows for a hands-off delivery, even though it requires one extra flip of the switch.In the future, debugging the uORB thread in servo_ctl.c will enable the servo to automatically open and close within the delivery script for a completely automatic dropoff.

The control of the gripper is mostly functional at this point, with some future improvements to automation possible. The code for servo_ctl.c is included in the Appendix. The hardware components of the drone, especially concerning the gripper, took into account the mechanical design choices that were made during this project, which will be discussed in the next section.

6.3 Mechanical Design

The mechanical components of the drone include a gripper for delivery, landing gears, and a UAV shell (also referred to as the hull). The initial part of the project focused on brainstorming different designs for the components, and evaluating them using decision-matrix tools like a Pugh chart. Initial prototypes of the chosen designs for all the components were made using different cardboards, foams and plastics. The methods of prototyping included quick sketches, cutting foam models, and 3D printing. The gripper mechanism was mainly prototyped using sketches and 3D printing. A 3D print of the shell provided a good qualitative analysis of how strong and tough the final hull would be and was also used for various testing purposes. For the landing gear, styrofoam was used to rapidly prototype the final design. The landing gear prototypes were made by cutting cardboard and linking the mechanisms using wires and threads.

6.3.1 Gripper

A vital feature of the gripper was its ability to hold an object, and release it on command. It did not need to be able to pick up items, because it serves as business delivery drone, and its load can be attached while the user inputs an address to the drone. Initially, the goal was to have a pulley system that could lower the load before dropping it (Figure 12.1). This would have increased general safety, especially because at the beginning the drone was unstable and landing would have posed a danger. The pulley system used a pulley connected to a torsion spring with a solenoid lock. When power was applied to the solenoid, it would retract, and the weight of the object would bring the pulley down. Once the pulley had been fully extended, power would be applied to a second solenoid which would retract. Then a hook hanging on the solenoid would be unsupported, and would fall. This design offered the benefit of straightforward power components (on/off) and the ability to keep the drone flying even when it was dropping off the package.

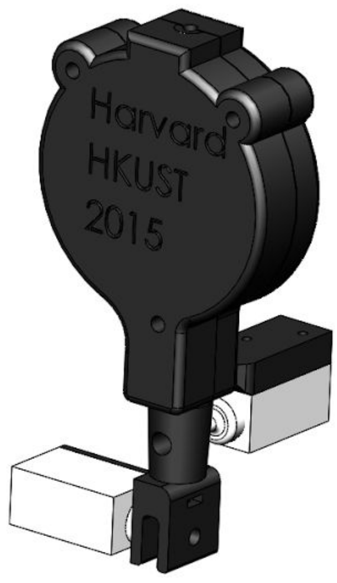


Figure 12.1: Preliminary Gripper Design. Includes a pulley system (top) that is locked by a solenoid (white, right). When released, the gripper (bottom) will descend, where a load will be released when a second solenoid (white, bottom left) is activated and retracts.

The drawbacks of this design proved to be harder to deal with than anticipated. Since the pulley was spring-driven, after the weight was dropped, the force of the bottom of the pulley hitting the top again was significant. Adding friction with the pulley string by adding a wall inside the pulley that would resist the strings movement seemed to be a solution. However, during testing, the force of impact from the pulley going back up actually broke the bottom piece of the gripper system, even without the added weight of the solenoid, as well as the wall added to create friction. Without a better way to control the kickback, other design flaws came to light. As a result of the pulley system, the solenoid at the bottom of the pulley also needed an appropriately long wire connecting it to a power source, which had a high chance of getting tangled. This led to a design change for the gripper.

The new gripper is much simpler, only consisting of a servo and a hook. The hook (Figure 11.2) is of a simple L-shape, with grooves to ensure that any object doesn't slip off during flight. This design is much easier to construct and control, as it only requires two components, the servo and the hook, which are held close to the base of the drone. A servo replaced the solenoid, because the PixHawk runs PWM outputs, which are more suited for controlling servo motors than they are for solenoids. The model shown below is the new design of the gripper part:

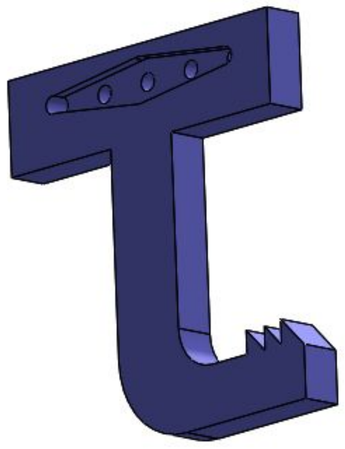


Figure 12.2: Final Gripper Design

6.3.2 Landing Gear

The initial design of the landing gear was made of expanded polyethylene foam (EPE foam), with the two parts of the landing gear attached directly on the arm frame. The design is shown in the following figure.

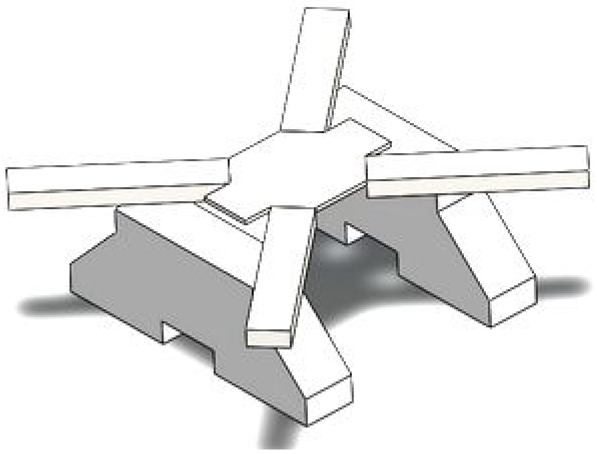


Figure 13.1 - Preliminary design of the landing gear for the UAV

This design was supposed to be able to absorb the shock from landing, or crashing in case of accidents, with sufficient toughness to prevent fracture simultaneously. Some simulations was also conducted to examine the performance, with external force 10 g (gravitational acceleration) to simulate the impulse from crashing or rapid landing, as shown in Figure 13.2 and 13.3. The results showed that there will be significant deformation, but still within the range of elastic deformation.

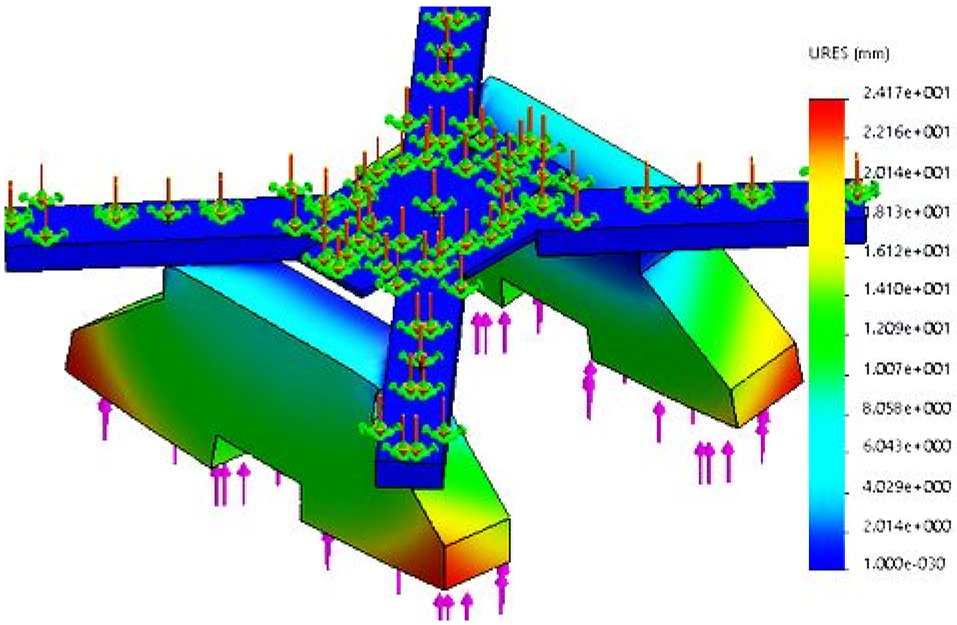


Figure 13.2 - Stress-Strain Analysis of the first landing gear design with scale as (1) displacement

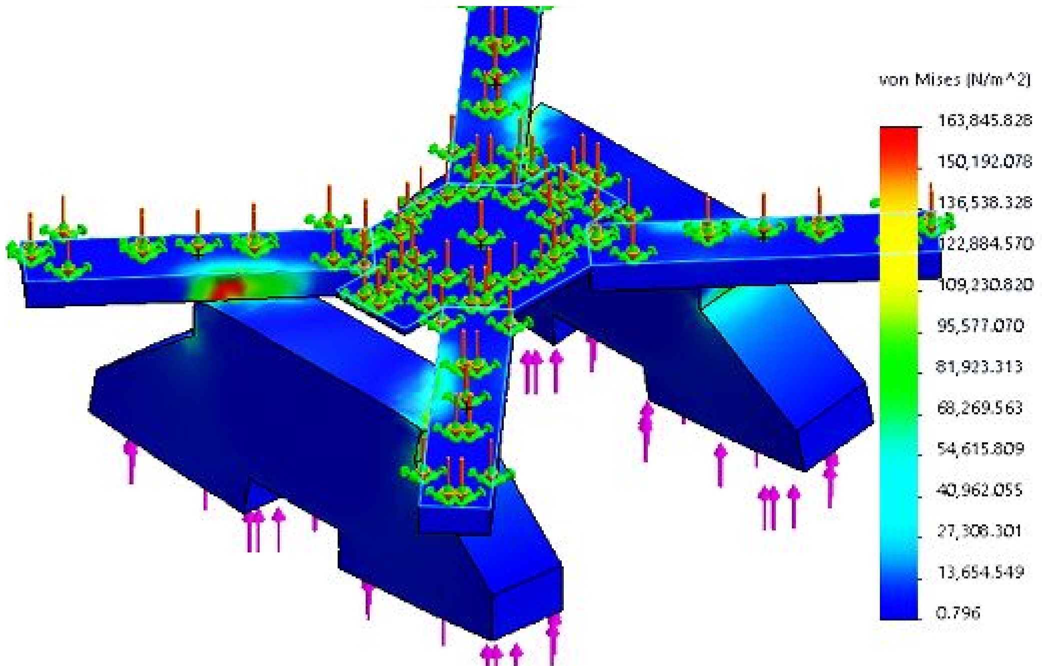


Figure 13.3 - Stress-Strain Analysis of the first landing gear design with scale as (2) Von Mises stress

A prototype was also fabricated to test the actual performance of this landing gear in terms of ability to absorb shocks to protect the frame, and also the toughness of its own. The design was able to achieve all of these goals at first, and also able to balance itself when the drone is not landing directly downward. Yet, the connection between the frame and the landing gear will be loosen to a very large extent after a few testings, which might be due to the connection point and also the design. The next design landing gear was therefore produced as following.

One major problem with the design of the landing gear as shown above was balancing the center of gravity of the drone and ensuring vertical landing and take-off. As the propeller arms were extending out, any angled landing could seriously damage the propellers. Other than that, the chosen material styrofoam offered high enough toughness to absorb any landing shock. The modified design is shown in Figure 13.4 below.

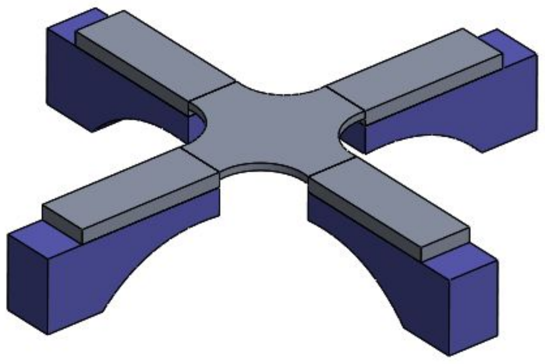


Figure 13.4 - Final design of the landing gear

6.3.3 Shell

The outer shell of the UAV is required to ensure the safety of the components in case of any unprecedented accidents or collision. It also adds some water resistance to the system and prevents damage to electrical components when it rains. In addition to that, it acts as a structural housing for some of the parts like delivery gripper, proximity sensors, and radio telemetry which are to be attached to the body of the shell. The main consideration while designing the shell was to decide which components to house, what materials to use and what shape it should have. One crucial limitation the team had was weight. As the battery consumption increases with increasing weight, a heavy shell would imply less flying time for UAV and hence reduces the area covered for delivery. Based on mechanical analysis of the flight requirements, following numbers were generated:

Mechanical Properties:

Flexural Strength > 90MPa (based on ABS plastic of the thickness 1.25 mm)

Impact Toughness > 100J/m (based on solidworks analysis of the system under constant loading of .4g)

Shore D hardness > 75 (based on ABS plastic of the thickness 1.25 mm)

Restrictions:

Weight < 100g (as the drone's maximum load capacity is 500g, we wish to restrict its own weight to less than 30%)

After the quantitative analysis of the mechanical properties of the structure, the design of the shell was made using solidworks. Various models were made and rapid prototypes were made to evaluate their feasibility. Some of the preliminary designs of the model are shown below:

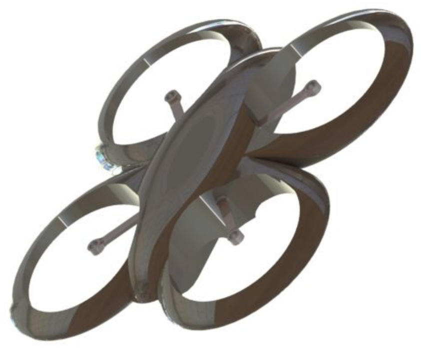


Figure 14.1 - Preliminary design of the shell for the UAV

The figure above shows the preliminary design for the shell of the UAV. Though, adding the propeller shield to the shell increased the safety of the whole system in case of any accident, it also increased the weight of the model drastically. With the same design made of ABS plastic, the weight of the model was around 250 g including the prop shield and only 80g without it. So it was decided that the shell will be 3d printed without the propeller shield. However, efforts were made to use different materials and other manufacturing techniques to make the propeller shield. The details will be talked about later in this subsection.

The 3d model was modified according to the new requirements. The propeller shields were designed to be made by EPS foam using the method of hot wire cutting. The Figure 14.2 shows the rendered image of the 2nd version of the preliminary design of the shell:

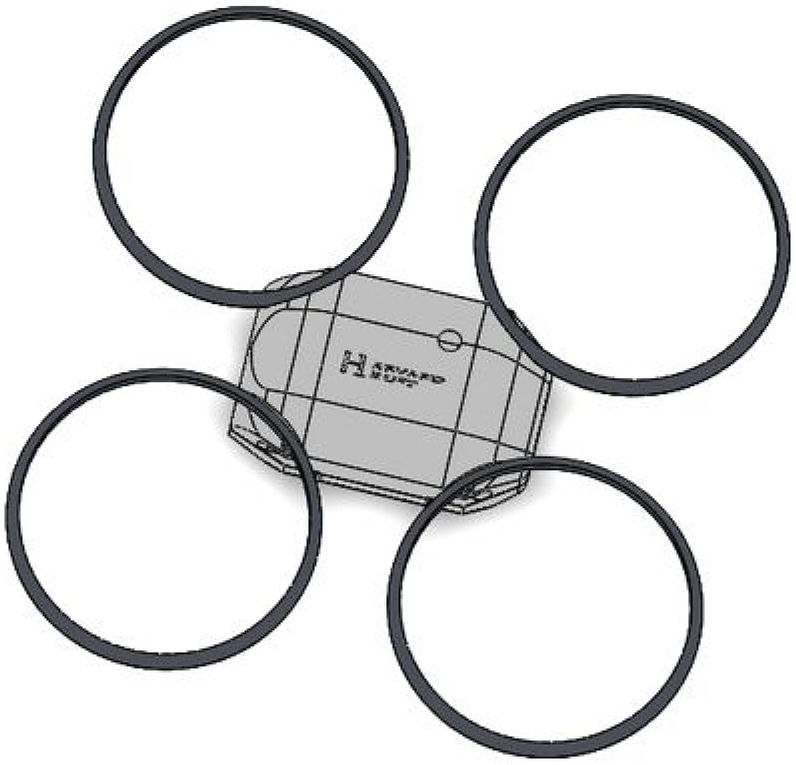


Figure 14.2 - Preliminary design of the shell for the UAV propeller shields

The new mode of the shell had two major separate parts- the upper and the lower housing -and the four propeller shields to absorb any shocks in case of any accident. The housing of the shell, however was too bulky as the weight still exceeded our initial design requirement of having it < 100 g. The lower part of the housing was also redundant as the UAV has the lower metallic frame which gives high structural strength to the system. In addition to that, the propeller shield made up of EPS foam were light enough to meet the design requirements but were not strong enough to provide any additional toughness to the system. So, further modification to the design was made. The new design consisted only of one part which attaches itself directly to the frame of the shell and to the landing gear. This removes the need of having a separate lower cover for the shell and so, reduces the weight by almost half. The previous design as shown above also doesn't cover up the ESCs of the drone which are one of the critical components to be covered. So, the new design also ensures that the covering is extended to support ESCs as well. The following design is the final model chosen to build the shell for the UAV:

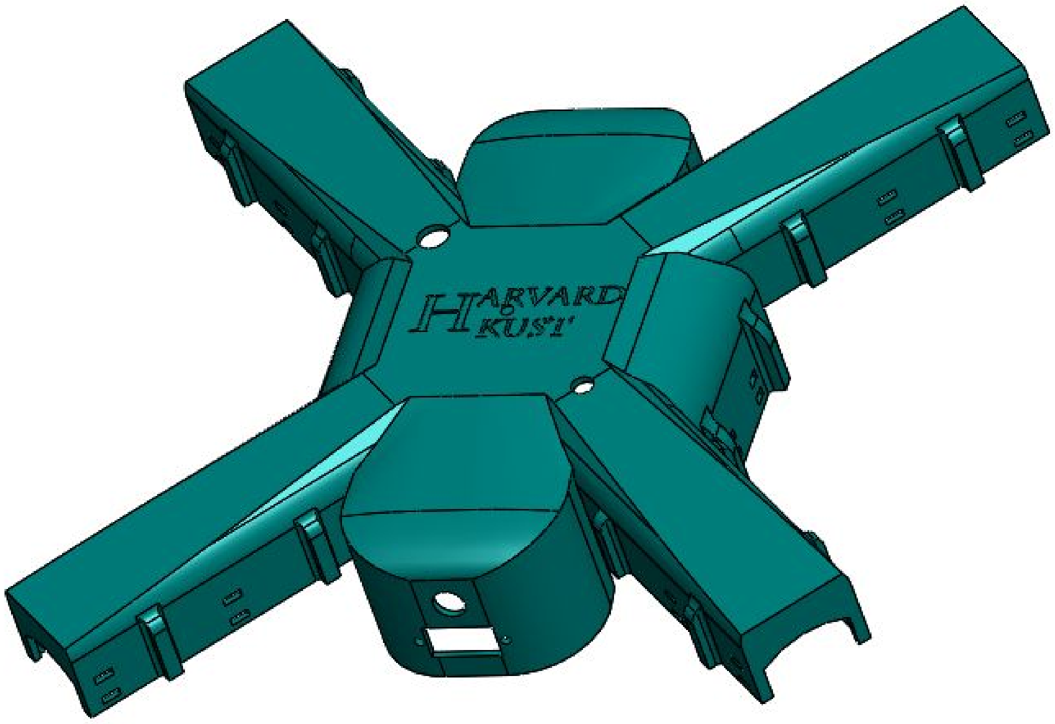


Figure 14.3 - Final Preliminary design of the shell for the UAV propeller shields

The final design of the shell incorporates all the changes from the previous designs and also provides slots for servo-gripper system, radio telemetry,etc. It has arms extended to cover the ESCs and is attached by the main frame using zip ties through the slots, as shown in the Figure 14. The detailed drawing of the solidworks model is shown in Appendix for further reference.

6.4 Other Issues

One challenge which faced the entire team was when, on the first day of successful testing, a lack of control over the throttle resulted in the drone getting stuck in a tree. When it became clear that better equipment was necessary to remove the UAV, it seemed that it would be forced to remain in the tree for the duration of the weekend until a cherry picker could come retrieve it on the following Monday. Since it was supposed to rain over the weekend, planning began to purchase new parts and begin again. Fortunately, the next day some workers were in the area and removed the drone. This experience taught a valuable lesson about ensuring safe testing, purchasing extra parts and what to do when an experiment does not go as planned.

Following the tree incident, a document was written about measures to prevent the same issue in the future. This included programming a maximum altitude, instituting a test which users had to take before testing, and other precautions. Some replacement parts were also ordered so that if a mistake of a similar magnitude occurred in the future it would be less disastrous. Fortunately, this turned out to be a positive learning experience without the harsher consequences which could have occurred had the drone remained outside during the thunderstorms.

Another challenge while testing the drone has been the weather. The drone cannot fly in rain, snow, high winds, or any extreme weather conditions, so testing has been limited to fair-weather days. This became an issue towards the end of the program when more outside testing was necessary, but rain did not allow it. To get around this creative methods were developed to safely test certain features indoors.

7. EVALUATION & VERIFICATION

The Evaluation and Verification section details the testing and subsequent changes made by the design team in order to make the drone delivery system more user friendly and safe. Testing was also done to ensure that engineering standards and design requirements were met.

7.1 Software

7.1.1 User Experience

An important aspect of the Android application is the user-friendliness. Though the application targets a tech savvy user group, it is still necessary to have a simple, easy to use, sufficiently neat and clear interface. Thus, a testing process and survey was designed for potential users, the entirety of which is outlined in the Appendix. All of the interviewees were in no way involved in any portion of the application development process so as to ensure that they are completely new to the application and unbiased. The following is a summary and analysis of the survey results.

The user experience test was divided into two parts, namely the customer side and vendor side. This division reflects the structure of the application. A total of 12 potential users were interviewed during the user experience testing. Table 6 below shows the most important results regarding the user-friendliness for the main features of the application. Apart from the map usefulness, which received a slightly lower score, all of the other parameters show very satisfactory results, ranging from 8.58 to 9.67 out of 10. The lower score for the map feature is a result of some respondents' concern over the distance and estimated time arrival of the delivery drone instead. Overall, the application is clear and easy to use even for beginners, and its many diverse features make it useful for a delivery system.

Table 7 - Overview of the User Experience Survey Results, Survey Size *N = 12*

	Average Score (out of 10)	Standard Deviation

Ease of ordering (Customer)	8.83	1.14
Usefulness of the Map (Customer)	7.75	2.31
Ease of delivering (Vendor)	9.17	0.69
Ease of checking pending orders (Vendor)	9.67	0.62
Ease of checking battery status (Vendor)	8.58	1.38
Overall	**8.80**	/

According to the survey, people generally expect the UAV to deliver smaller objects within a 2 miles radius. These objects include but are not limited to paper & office supplies, gifts, flowers and so on, which are usually sufficiently light in weight to be delivered given the 300g payload limitation of the delivery drone. Furthermore, the respondents expect the package to be delivered the two miles within 36.25 minutes. This is an average speed of about only 1.5 m/s, which is much lower than the typical speed of the drone (estimated ~7 m/s). These results showed that the delivery system designed for this project meets the expectation of the potential users, ensuring sound marketability of the delivery system.

7.1.2 Integration with Drone

In order to provide proof of concept the computer application had to be able to control the UAV using radio telemetry. The application had two aspects: manual control and automatic flight mode. The manual control was necessary to test the connectivity between the drone and computer in a more controlled environment than sending it on an automatic mission. In order to test this functionality, the UAV was taken outside and the throttle was gently raised on the computer. Though initially the drone wouldn't arm, or would fly without any indication on the computer, successive testing and development lead to the point where the computer provided reasonable manual control.

To test the location tracking capabilities of the UAV, the remote control was used to move the drone around a larger area. While this was taking place, the computer application showed a circle which was the drone location. This circle moved as the drone moved, in the directions which it was moving, proving that the GPS location aspect was being correctly transmitted. The process for verifying that battery information was accurate was similar in that the battery levels were taken, then the drone and the computer application were turned on to ensure that the reading on the application was equal to the reading previously obtained.

7.2 Hardware

7.2.1 Stability

To prove the stability of the drone, it would have to be proved that the drone could create a stable attitude and position after experiencing a disturbance. The easiest disturbances to create are the motions created by the RC. The drone can be isolated, with only one axis to rotate around (see Figure 7). If the joystick is tapped to create a sudden rotation, then how quickly the drone creates a level horizon is a good indication of its stability. Multiple tests and videos were taken that show the drone stabilizing after receiving a shock to its level horizon.

7.2.2 Flight Speed

In order to test the average speed of the drone during mission and deliver modes, the drone was sent from one end of the field to the other, a distance of approximately 57 m (derived from Google Maps). Videos of the flight were taken, and the time the drone spent in horizontal flight was recorded from analyzing these videos. The results of this analysis are as follows:

Table 8 - Flight Speeds

Trial	Time in Flight (s)	Mean Speed (m/s)
1	8.50	6.71
2	8.34	6.83
3	7.83	7.28
Overall	**8.22**	**6.94**

7.2.3 Battery Life

In order to test the feasibility of drone deliveries, it is necessary to know approximately how long the batteries used to provide power to the drone can last. The absolute maximum distance the drone can go and still be in range of the radio controller is 1.6km. Therefore the max distance will be around 3.2 km (assume 4.5km for a safety factor of 1.5 and to account for any trajectory change from object avoidance). Assuming a mean speed of 7 m/s, that requires each battery to last around 10.7 minutes in order for this project to work well within a 1.6 km range.

Measuring the life of each battery required a simple test. The drone was powered up with a fully charged battery, and sent 10 meters into the sky, carrying no extra load besides the weight of the dorne and the battery. It was flown back and forth along a length of 50 m every minute until it was low enough on battery that the PixHawk's preprogrammed failsafe measures activated and the drone automatically landed. This test was repeated on all the various batteries of different sizes (2800 mAh, 3300 mAh, & 5000 mAh). The results of these tests are shown in Table 9 (also in Appendix).

Table 9 - Battery Test Results

Battery Type (Battery Weight)	1st Test (Min:Sec)	2nd test (Min:Sec)
2800mAh (220g)	10:05	10:14
3300mAh (320g)	15:20	15:21
5000mAh (541g)	20:43	20:09

From these results, the 10.7 minutes of flight time required for max deliveries is met by the two larger batteries. The battery most suited for delivery is the 3300 mAh one, because the 2800 mAh battery does not meet the specifications and the 5000 mAh is so large that it would severely limit our package load. With the 3300 mAh battery, there is around 1.5 times the recommended time of flight, which also provides safety for any variations in battery time when weight is added. These battery tests demonstrated project feasibility from a power standpoint.

7.2.4 Performing a Delivery

To prove the ability of the drone to complete a delivery routine, the drone underwent several tests of the delivery script. These test consisted of tracking the drone as it went along a delivery route. The actual route of the drone was then overlaid against the desired route as can be seen in Figure 16. The flight path does not match up perfectly; however, this could be due to the accuracy of the GPS (only accurate to ~3 m) or the wind. The important result of the test was that it made it to the destination with the needed accuracy and returned back to the home location.

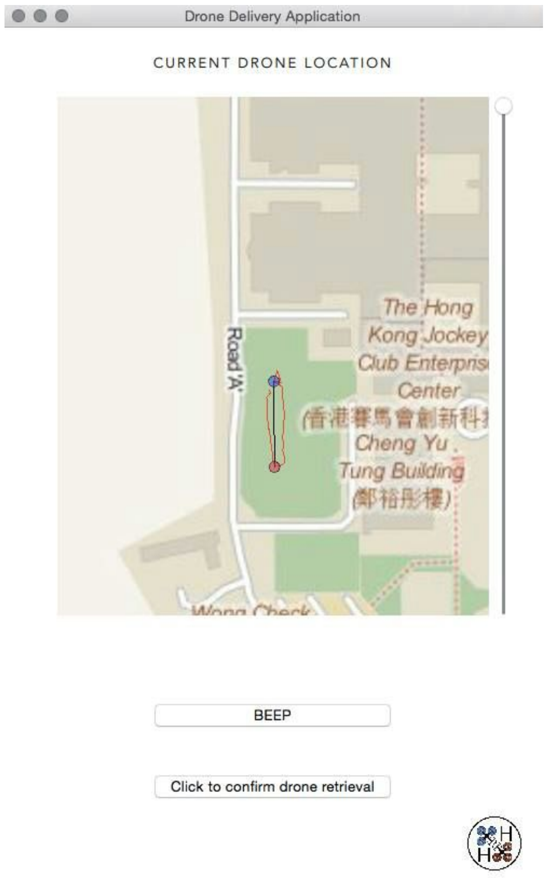

Drone Delivery Application
CURRENT DRONE LOCATION
Road 'A'
The Hong Kong Jockey Club Enterprise Center
Cheng Yu Tung Building
(鄭裕彤樓)
Wong Check
BEEP
Click to confirm drone retrieval

Figure 15: Intended delivery flight path (black line) with actual flight path (red line)

7.2.5 Testing Object Detection

To test the ability of the quadcopter to detect large objects and give an appropriate response, it was flown (slowly) towards a building until the threshold value was reached. When the quadcopter was close enough to trigger the sensor, the response of the quadcopter would be observed. Due to the fragile nature of the quadcopter, it was deemed that flying the quadcopter with a large velocity towards objects would not be an optimal test.

8. BUDGET

The initial spending budget was $2,580.65 USD (20,000 HKD). The exchange rate is 1 USD : 7.75 HKD. About half of the budget was spent on the two drones, the PX4 operating system and the remote control. In addition to this, many other items were purchased as outlined in the appendix, resulting in additional costs of $943.45 USD. This brings the total expenditure to about $2,233.77

USD. When making purchases, which products would perform the actions required while being as cheap as possible was an essential consideration.

To derive the cost-payoff analysis, a time of 6 hours to fully recharge the UAV battery was set. Therefore, if all batteries start fully charged at opening time at 8:00 am, the drone immediately does one delivery, and the entire battery is drained for each delivery, one battery allows for 3 deliveries in one day. This is because the first battery is fully charged again at around 2:00 pm and 8:00 pm. The drone cost is an estimated $825 ($600 for supplies + $200 for assembly + $25 for battery). So, it would take 67 days to pay off the drone cost if there is a $4 delivery fee. Scaling up the deliveries per day, purchasing 6 batteries at $25 each, the cost would be $950 with 1 drone delivery every hour or 16 deliveries per day. In this case it would take 20 days to pay off, charging $4 per delivery. Finally, by purchasing 18 batteries at $25 each (which would allow continuous delivery), the cost is increased to $1250 but it is possible to complete 1 drone delivery every 20 minutes for a total of 48 deliveries per day. In this case, it would only take 8 days to pay off with $4 delivery fee.

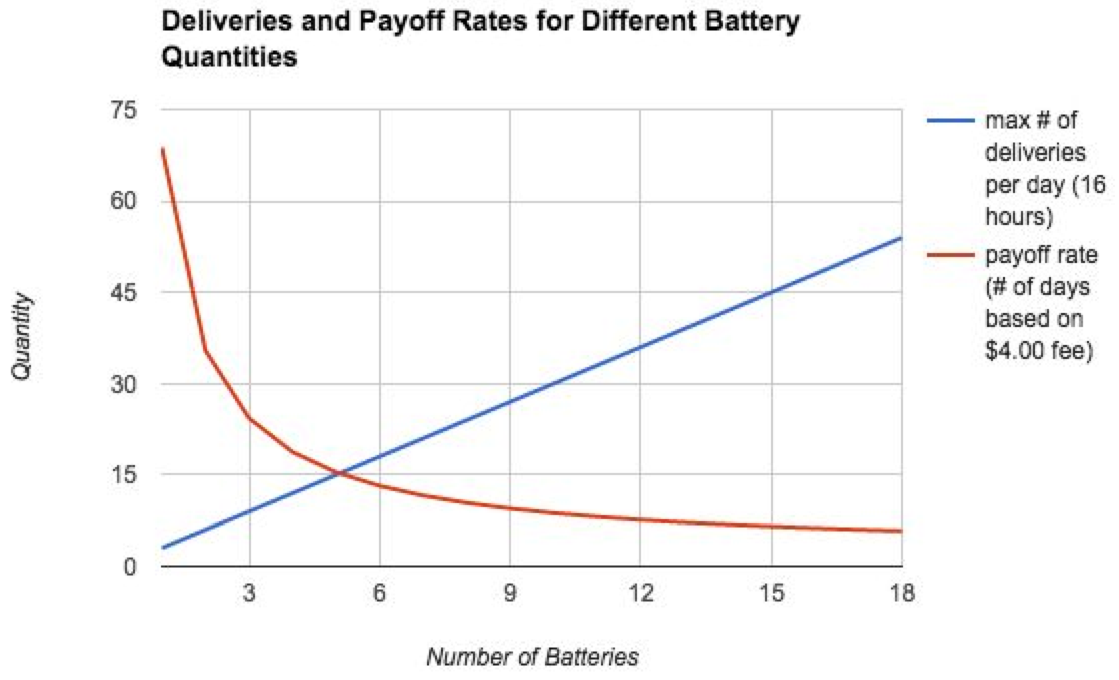


Figure 16: Deliveries and Payoff Rates for Different Battery Quantities

However, drone back-up parts and systems are also necessary considerations. In a scenario where the same amount of money is put into drone maintenance as is initially put into the system, which would be the worst case scenario, the drone with a continuous battery system would still be paid off in 16 days. Since any system which pays for itself in 6 months is viable, this seems like a very reasonable option. Furthermore, when the drones are being mass-produced the parts and manufacturing will cost less. This phenomenon will decrease the price of the system, either directly for the consumer or alternately by creating more profit for the business selling these systems if the single unit price is maintained.

9. CONCLUSION & FUTURE WORK

The drone delivery system satisfies the need for quicker, cheaper and more reliable delivery in urban areas. Through surveying friends and family, especially college students in Cambridge, it was determined that there is a lack of satisfaction with current delivery systems which are expensive, about \$10 per delivery, and also relatively unreliable. In fact, most responders expected a local delivery to take an hour. Taking this information, and talked to local business owners, it was revealed that one issue with current delivery systems is that delivery boys can be unreliable. Furthermore, most deliveries are made within a one mile radius of the business. Some of the businesses we talked to did not have a delivery system because their items were small and low cost to justify sending out a car or bike multiple times an hour to perform the deliveries.

The drone system outlined above performs deliveries in a one mile radius (Cambridge is also less than one mile from Boston) within 20 minutes round trip. According to Google Maps, a one mile round-trip bike ride in Cambridge would take about 10 minutes, which would indicate that both take the same amount of time. However, the drone is unaffected by traffic. Furthermore, reliability is increased through the inclusion of a tracking mechanism on the app which allows both the customer and the vendor to constantly track the drone during the delivery process. As a result, the customer knows exactly when to expect their delivery, as well as if there are any unforeseen issues during delivery. The drone delivery system also saves both the customer and the vendor money. The vendor no longer has to pay delivery people, so the cost of delivery goes entirely towards paying off the delivery system and upkeep of the drone. Since this reduces costs for the vendor, they can charge the customers less for the delivery service. Furthermore, drones do not require any tipping which further reduces costs for the customer. In addition, if the vendor purchases the full set of 18 batteries, enabling continuous delivery, the drones are able to log as many flight miles per day as the average bike courier [15]. Ultimately, the needs of both are met by this system.

In the future, there will likely be many of these systems in effect. As a result, there will be many legal requirements for safety and avoidance in order to keep other drones, packages, pedestrians and buildings safe. In order to achieve this future, the FAA would have to approve the use of drones for commercial delivery purposes. If this system becomes commonplace, one repercussion will be the loss of delivery jobs. This could result in social push-back from these workers and other sources. However, more jobs in maintenance would be created, since an increase in drone use might result in a need for fast, skilled repair.

Future considerations for this project include improving battery duration so that the drone can fly for longer periods of time. Batteries currently do not store a lot of energy, and the need for lengthy recharging makes them non-ideal for long term use in the future. Hopefully technological advances will result in higher capacity and lighter batteries.

Before the system is ready for commercial use, more rigorous testing must be performed to ensure that the device attains higher levels of safety. This especially includes many more tests for the proximity sensors, to ensure that it is safe to use the delivery system in an urban area. Time only allowed for preliminary tests, and none in an actual urban environment. Ideally the drone would scrape an object only 1 in 500 times, the same as the UPS failure to deliver packages rate. The standards for a "fatal" crash which resulted in destruction of the drone would be even more rigorous, ideally only 1 time per year. Estimating about 10 collision opportunities per minute in an urban environment, and 48 twenty minute trips per drone per day, this would mean there would have to only be a collision 1 in 9,600 times.

In addition to this, location accuracy should be improved so that vendors and customers know exactly where the package is being delivered. One idea is that the the drone could deliver the package as close to the phone as possible. One potential flaw with this idea is that packages could end up on the roof or similarly inconvenient places depending on where the delivery recipient is located.

Ideally a method that allows communicating with the drone over greater distances will also be implemented. Currently this is unnecessary, since it must legally always be in the user's line of sight. However, in a future where this is a viable delivery method, the mile radius for delivery may be unnecessarily restrictive. Instead of using the 915 MHz bandwidth (433 MHz in HK) which we were provided with, cellular or an xbee/zigbee would allow the system to serve a larger range of distances.

10. ACKNOWLEDGEMENTS

The Harvard-HKUST Design Team 2015 would like to gratefully acknowledge the financial support of Harvard alumnus Xiang Dong "XD" Yang and Nancy Yang, the Harvard School of Engineering and Applied Sciences, the Hong Kong University of Science and Technology, and the Harvard President's Innovation Fund for International Experiences.

Thank you to those at Harvard School of Engineering and Applied Sciences and Hong Kong University of Science and Technology who instituted the Harvard-HKUST International Design Experience Program 2015, especially Evelyn Hu, Ling Shi and Kei May Lau. Without you, none of us would be here right now enjoying this enriching experience of international collaboration. We greatly appreciated the advice and guidance of our many mentors, including Christopher Lombardo, Daniela Faas, Avinash Uttamchandani, Na Li, Xuan Liang, and Bing Li who skyped in and were available regardless of time zone. We could not have done it without your support. Thank you also to Sarah Lefebvre and Patricia Lai who ensured all of our needs were met in Cambridge and Hong Kong and went above and beyond the call of duty in helping us create Hubway accounts, improving our meal plan, and otherwise making us the happiest and best cared for group on campus.

11. REFERENCES

[1] C. Anderson. (2014). 10 New Breakthrough Technologies 2014: Agricultural Drones [Online]. Available: http://www.technologyreview.com/

[2] M. Zenko. (2012, February 27). 10 Things You Didn’t Know About Drones [Online]. Available: http://foreignpolicy.com/

[3] BYROBOT. (2014). Drone Fighter [Online]. Available: http://byrobot.co.kr/

[4] S. Spilka. (2014, October 13). Spotify ‘PartyDrone’ Specially Built for Dance Festivals [Online]. Available: http://www.psfk.com/

[5] Amazon Technologies, Inc., “Unmanned Aerial Vehicle Delivery System,” U.S. Patent 20150120094, April 30, 2015.

[6] N. Carbone. (2013, June 11). Drone Delivers ‘Flying’ Sushi Burgers at London Restaurant [Online]. Available: http://www.newsfeed.time.com/

[7] H. Khaleeli. (2013, June 10). Yo! Sushi drone delivery – will it take off? [Online]. Available: http://www.theguardian.com/

[8] D. Aamoth. (2013, June 3). Delivering Domino’s Pizza by Unmanned Helicopter: What Could Possibly Go Wrong? [Online]. Available: http://www.techland.time.com/

[9] R. Marsh. (2015, May 12). Amazon drone patent application imagines delivery that comes to you with one click [Online]. Available: http://www.edition.cnn.com/

[10] L. Reich. (2015, March 18). Food Delivery Apps Have Taken Over San Francisco. Here's Why. [Online]. Available: http://www.eater.com/

[11] National Telecommunications and Information Administration. (2011, October). United States Frequency Allocation Chart [Online]. Available: http://www.ntia.doc.gov/

[12] Office of the Communications Authority. (2015, January). Hong Kong Table of Frequency Allocations [Online]. Available: http://www.ofca.gov.hk/

[13] *ConsumerReports.org*. (2012, August). What that car really costs to own [Online]. Available: http://www.consumerreports.org/

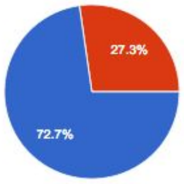
[14] T. Mills. (2011, October 7). What is the maximum range of an Xbee? [Online]. Available: http://diydrones.com/

[15] N. Pesce. (2012, August 21). THE WHEEL WORLD: Real-life ‘Premium Rush’ riders spin NYC bike messenger tales. [Online]. Available: http://www.nydailynews.com/

12. APPENDICES

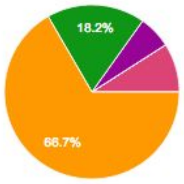
A. Summary of Customer Survey Results

What is your gender?



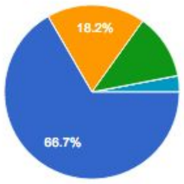
Female	24	72.7%
Male	9	27.3%
Other	0	0%

What is your age range?



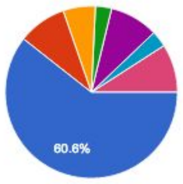
Under 10	0	0%
10 - 15	0	0%
16 - 20	22	66.7%
21 - 25	6	18.2%
26 - 30	2	6.1%
31 - 40	0	0%
41 - 50	3	9.1%
51 - 60	0	0%
60+	0	0%

What is your highest level of education (completed)?



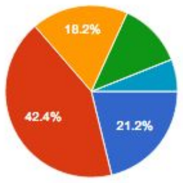
High School	22	66.7%
Associate's Degree	0	0%
Bachelor's Degree	6	18.2%
Master's Degree	4	12.1%
Ph.D. or higher	0	0%
Decline to answer	1	3%

What is your annual income?



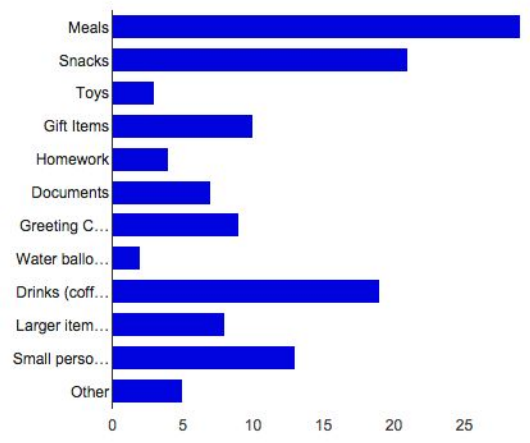
$0 - $20,000	20	60.6%
$20,000 - $40,000	3	9.1%
$40,000 - $60,000	2	6.1%
$60,000 - $100,000	1	3%
$100,000 - $150,000	3	9.1%
$150,000+	1	3%
Decline to answer	3	9.1%

How far are you from the (local) place you order from most?



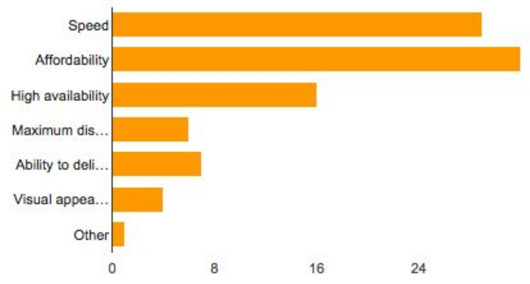
Under 1 mile	7	21.2%
1 - 2 miles	14	42.4%
2 - 3 miles	6	18.2%
3 - 5 miles	4	12.1%
5 - 10 miles	0	0%
10+ miles	2	6.1%

What kinds of items would you want delivered?

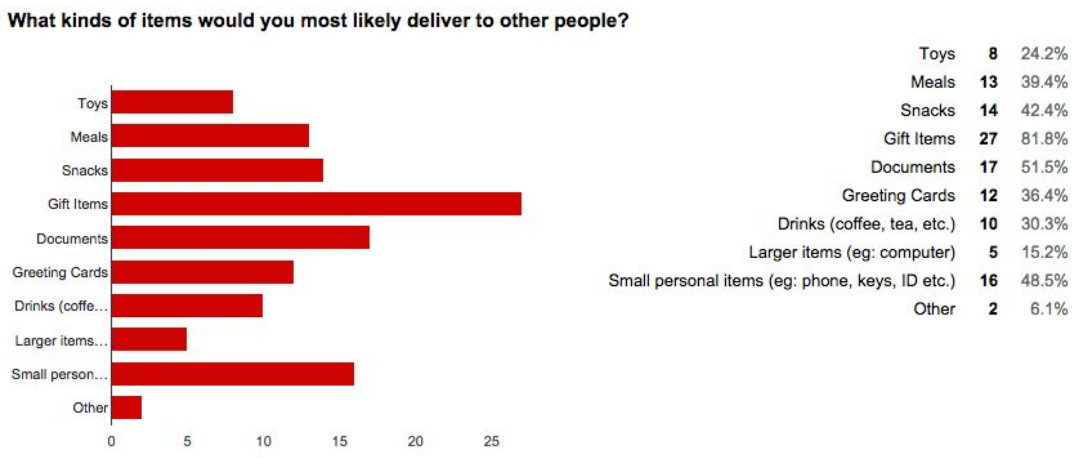


Meals	**29**	87.9%
Snacks	**21**	63.6%
Toys	**3**	9.1%
Gift Items	**10**	30.3%
Homework	**4**	12.1%
Documents	**7**	21.2%
Greeting Cards	**9**	27.3%
Water balloons :D	**2**	6.1%
Drinks (coffee, tea, etc.)	**19**	57.6%
Larger items (eg: computer)	**8**	24.2%
Small personal items (eg: phone, keys, ID etc.)	**13**	39.4%
Other	**5**	15.2%

What features would be most important for you in a new delivery system?

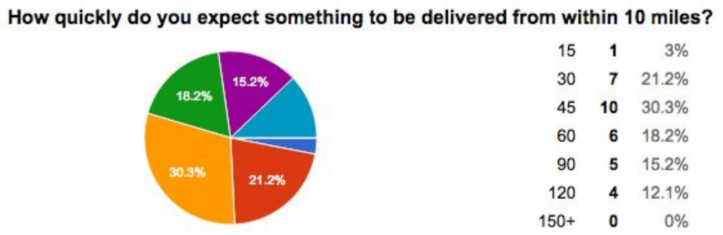


Speed	**29**	87.9%
Affordability	**32**	97%
High availability	**16**	48.5%
Maximum distance of delivery	**6**	18.2%
Ability to deliver a large variety of objects	**7**	21.2%
Visual appearance of system	**4**	12.1%
Other	**1**	3%



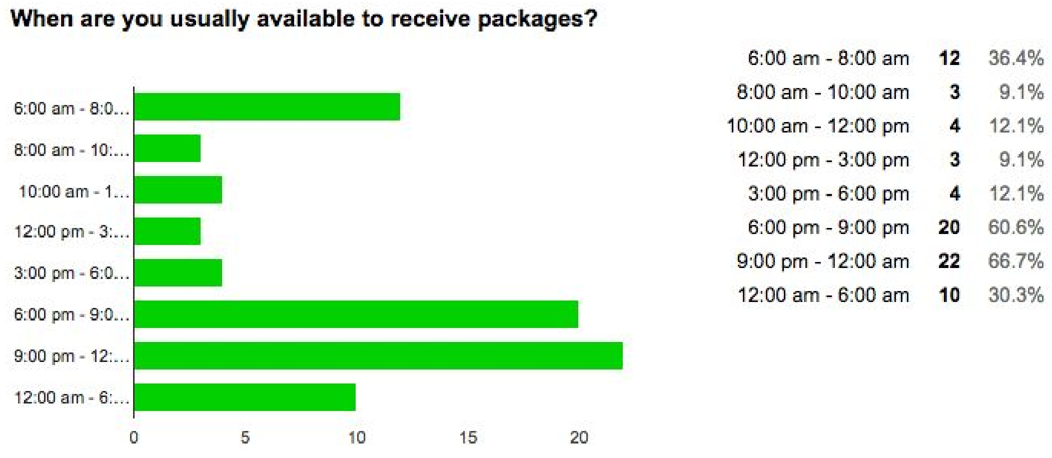
What kinds of items would you most likely deliver to other people?

Toys	8	24.2%
Meals	13	39.4%
Snacks	14	42.4%
Gift Items	27	81.8%
Documents	17	51.5%
Greeting Cards	12	36.4%
Drinks (coffee, tea, etc.)	10	30.3%
Larger items (eg: computer)	5	15.2%
Small personal items (eg: phone, keys, ID etc.)	16	48.5%
Other	2	6.1%



How quickly do you expect something to be delivered from within 10 miles?

15	1	3%
30	7	21.2%
45	10	30.3%
60	6	18.2%
90	5	15.2%
120	4	12.1%
150+	0	0%



When are you usually available to receive packages?

6:00 am - 8:00 am	12	36.4%
8:00 am - 10:00 am	3	9.1%
10:00 am - 12:00 pm	4	12.1%
12:00 pm - 3:00 pm	3	9.1%
3:00 pm - 6:00 pm	4	12.1%
6:00 pm - 9:00 pm	20	60.6%
9:00 pm - 12:00 am	22	66.7%
12:00 am - 6:00 am	10	30.3%

B. Purchases Chart

Item No.	Part-Name	Vendor	Part-ID	Package	Quantity	Cost	Noise	Power
1	2800mAh Copter Battery	Value Hobby	SKU: FLM-	LiPo	2	$34.99		3300mAh

			LP-18 15					
2	3300mAh Copter Battery				1 (two batteries included for each order)	$39.99 / 2		3000m Ah
3	RC Battery				1	$49.99		
4	Microswitch					$-		
5	Proximity sensor	MaxBotix	MB12 40	Custo m	6	$39.95	Air Turbulence Considerations are necessary	3.4mA Avg
6	Cord Reel	McMaster-Carr	1324A 14		1	$15.58		
7	Propellers	DJI	9450		2	$6.00		
8	px4 flight controller kit				1	$334		
9	DJI E310 Accessories pack	DJI	E310		1	$159.00		
10	Pixhawk					522.33		
11	DJI ESCs and motors					189		
12	5000mAh LiPo battery					74.95		

C. Application User Testing Guide

Requirements:

- 1 interviewer
- 1 notetaker
- Android Application
- User Experience Survey
- Assisting images [of drones, etc.]

- Quiet space

Tips

1. User: "Should I press this button?"
    - A: "Go ahead. There is no wrong or right approach. We simply want to see what works and what may have issues."
2. User: "What should I do next?"
    - A: "Proceed as you think you should. It's really important for us to see what might be a roadblock for users."
3. User: "This is really confusing. I have no clue what to do."
    - A: "Thanks for that feedback. We'll be sure that this feature gets improved. Here is the step you want to take [show next step]"
4. User: "This feature sucks/confuses me/doesn't make sense."
    - A: "Okay, can you explain why it is not working for you or why it is confusing?"
5. AVOID saying words that explicitly tell the user what to do:
    - login, place order, button, click, press, fill in that text field,

Guide

1. Introduce user to our project using the following prompt:

    Hi, my name is _____ and this is my teammate _____. Our friends are students working on a summer project involving delivering small objects with drones, also known as UAVs, and we have been asked to help them run some tests.

    If you haven't seen a drone before, here is an image. [show Image 1] This flying vehicle is usually controlled by a remote control like this one [show image 2], but for the project, our friends created a phone application to control the UAV.

    The purpose of the application is to allow users to order something from a store close by. Another side of the application also allows business owners to list items for sale and to use the drone to deliver these items to their customers. Today, we have been asked to have users run through the application, and provide valuable feedback in order to improve the application.

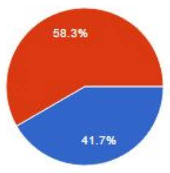
2. Ask user to run through application as a customer.
3. Guide him/her with the following prompts:
    - Your goal today is to order an egg tart successfully from Kee WahWah Bakery. Please proceed to do that on the application.
    - While you are navigating through the application, please say aloud what you are doing, so we can spot points of the application that need improvement.
    - [Address Entry Page] Please enter a real Hong Kong address so we can demonstrate the delivery feature. It doesn't have to be your address.
    - [Payment Page] You may skip this page.

4. Ask user to run through application as a vendor.
5. Guide him/her with the following prompts:
    - Now you are going to pretend you are KeeWahWah Bakery.
    - Please fulfill the egg tart order that has just been placed.
    - Please verify that you have fulfilled the egg tart order.
    - Now please fulfill another order from your customers.
    - Let's say that the drone battery is low. Please cancel this current order.
6. Allow user time and privacy to fill out the user experience survey here.

D. Application User Testing Results

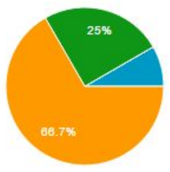
Link to full results:
https://docs.google.com/forms/d/1TSF5atFwKxNoHoT1odLx-qiU-8oH8NAfDsiH7fcAJEU/viewanalytics

What is your gender?



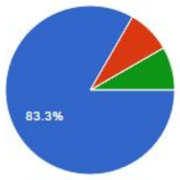
Female	5	41.7%
Male	7	58.3%
Other	0	0%

What is your age range?



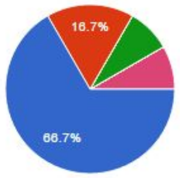
Under 10	0	0%
10 - 15	0	0%
16 - 20	8	66.7%
21 - 25	3	25%
26 - 30	0	0%
31 - 40	1	8.3%
41 - 50	0	0%
51 - 60	0	0%
60+	0	0%

What is your highest level of education (completed)?



High School	10	83.3%
Associate's Degree	1	8.3%
Bachelor's Degree	0	0%
Master's Degree	1	8.3%
Ph.D. or higher	0	0%

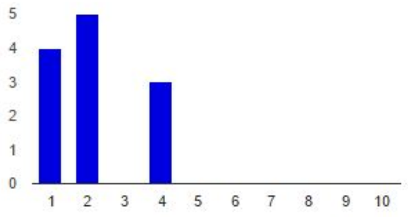
What is your annual income?



$0 - $20,000	8	66.7%
$20,000 - $40,000	2	16.7%
$40,000 - $60,000	0	0%
$60,000 - $100,000	1	8.3%
$100,000 - $150,000	0	0%
$150,000+	0	0%
Decline to answer	1	8.3%

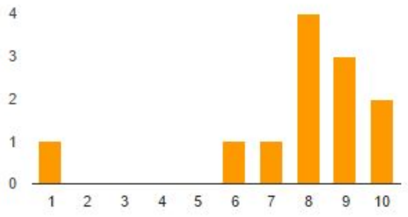
User Experience Survey

How was your experience ordering an egg tart (as a user)?



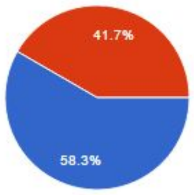
Super Easy:	1	4	33.3%
	2	5	41.7%
	3	0	0%
	4	3	25%
	5	0	0%
	6	0	0%
	7	0	0%
	8	0	0%
	9	0	0%
Impossible:	10	0	0%

How useful for you was the tracking map?



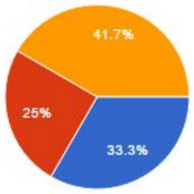
Not helpful at all.:	1	1	8.3%
	2	0	0%
	3	0	0%
	4	0	0%
	5	0	0%
	6	1	8.3%
	7	1	8.3%
	8	4	33.3%
	9	3	25%
Really helpful.:	10	2	16.7%

Did you notice that you had a weight limit for your order?



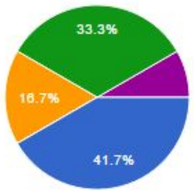
Yes	7	58.3%
No	5	41.7%

What is your opinion of the weight limit?



Didn't care at all. I would use the app even to order 1-2 small items.	4	33.3%
It's okay. The drone experience is cool enough to tolerate it.	3	25%
Eh. I would have liked to be able to order more at once, but the limit is fine.	5	41.7%
I wouldn't order if I couldn't order everything at once.	0	0%
其他	0	0%

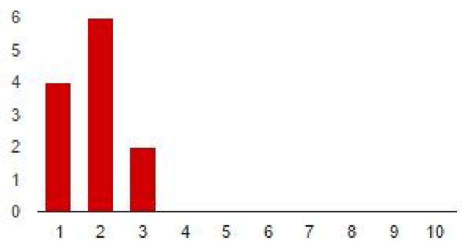
If Kee Wah really offered this delivery service, would you use it?



Yes	5	41.7%
Yes, if shipping was less than 4.00 HKD	0	0%
Only if shipping was free	2	16.7%
Only if it could be here in a certain amount of time.	4	33.3%
Only if I could order more items each time.	1	8.3%
Only if I was far enough from the place.	0	0%
No	0	0%
Maybe	0	0%

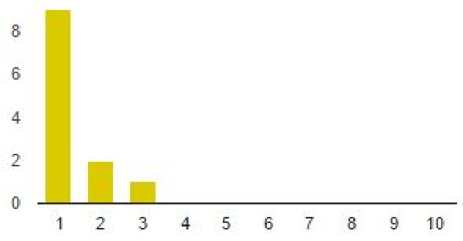
Vendor Experience Survey

How was your experience delivering the egg tart (as Kee WahWah Bakery)?



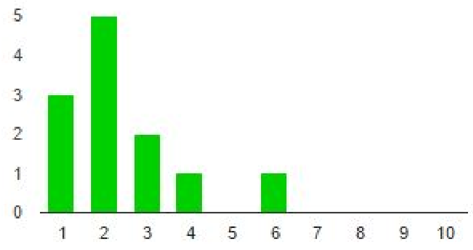
Super Easy: 1	4	33.3%
2	6	50%
3	2	16.7%
4	0	0%
5	0	0%
6	0	0%
7	0	0%
8	0	0%
9	0	0%
Impossible: 10	0	0%

How easy was it to check customer orders?



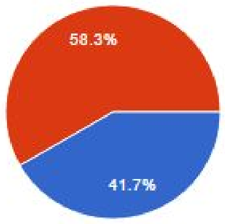
Super Easy: 1	9	75%
2	2	16.7%
3	1	8.3%
4	0	0%
5	0	0%
6	0	0%
7	0	0%
8	0	0%
9	0	0%
Difficult to view: 10	0	0%

How easy was it to check your battery status?



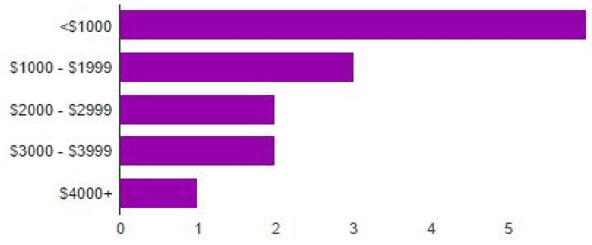
Super Easy:	1	3	25%
	2	5	41.7%
	3	2	16.7%
	4	1	8.3%
	5	0	0%
	6	1	8.3%
	7	0	0%
	8	0	0%
	9	0	0%
Couldn't find it:	10	0	0%

If this was real, would you rather buy or rent the delivery system?



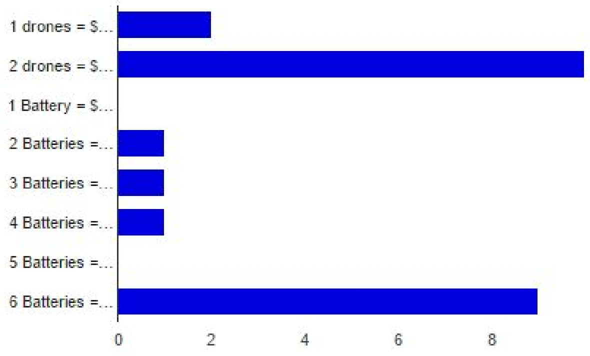
Buy	5	41.7%
Rent	7	58.3%

How much would you be willing to pay for a drone delivery system like this?



<$1000	6	50%
$1000 - $1999	3	25%
$2000 - $2999	2	16.7%
$3000 - $3999	2	16.7%
$4000+	1	8.3%

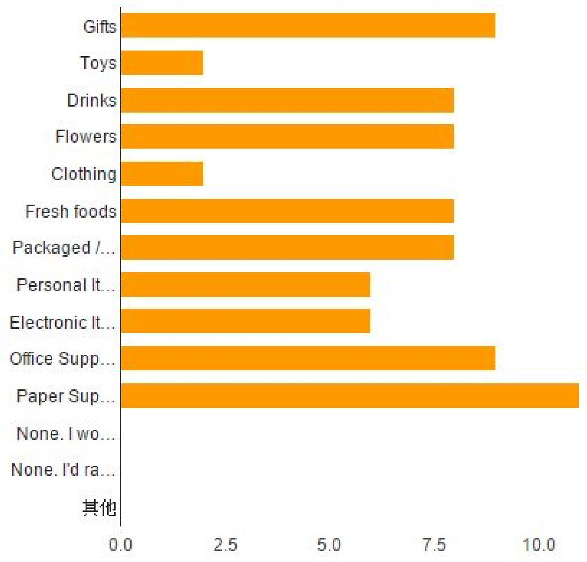
Please indicate the # of drones you would purchase and # of batteries.



1 drones = $1000	2	16.7%
2 drones = $2000	10	83.3%
1 Battery = $25 [1 delivery per 6 hrs]	0	0%
2 Batteries = $50 [2 deliveries per 6 hrs]	1	8.3%
3 Batteries = $75 [3 deliveries per 6 hrs]	1	8.3%
4 Batteries = $100 [4 deliveries per 6 hrs]	1	8.3%
5 Batteries = $125 [5 deliveries per 6 hrs]	0	0%
6 Batteries = $150 [6 deliveries per 6 hrs]	9	75%

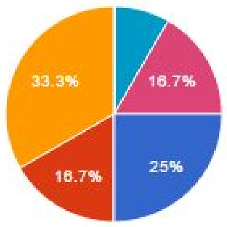
User Experience Survey

What items would you want delivered from businesses within 2 miles from you?



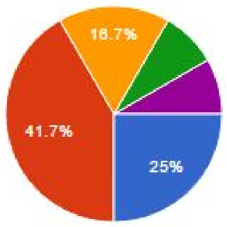
Gifts	9	75%
Toys	2	16.7%
Drinks	8	66.7%
Flowers	8	66.7%
Clothing	2	16.7%
Fresh foods	8	66.7%
Packaged / processed foods	8	66.7%
Personal Items (toothpaste, etc.)	6	50%
Electronic Items	6	50%
Office Supplies	9	75%
Paper Supplies	11	91.7%
None. I would honestly just walk to get it.	0	0%
None. I'd rather not pay.	0	0%
其他	0	0%

How far are you from the (local) place you order from most?



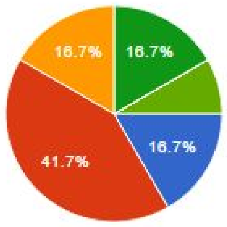
Under 1 mile	3	25%
1 - 2 miles	2	16.7%
2 - 3 miles	4	33.3%
3 - 5 miles	0	0%
5 - 10 miles	0	0%
10+ miles	1	8.3%
Usually don't order / don't know	2	16.7%

How quickly do you usually expect something to be delivered from within 2 miles?



15	3	25%
30	5	41.7%
45	2	16.7%
60	1	8.3%
90	1	8.3%
120	0	0%
150+	0	0%

What do you think is a fair price for a delivery within 2 miles?



5.00 HKD	2	16.7%
10.00 HKD	5	41.7%
15.00 HKD	2	16.7%
20.00 HKD	2	16.7%
25.00 HKD	0	0%
30.00 HKD	0	0%
35.00 HKD	0	0%
40.00 HKD	1	8.3%

E. PID

To illustrate the response times of various PID values, this report will draw on images from a tutorial on Matlab and Simulink built by The University of Michigan. Each of the below figures demonstrates the effect different PID constants can have on a step response, an instantaneous change in input from 0 to 1.

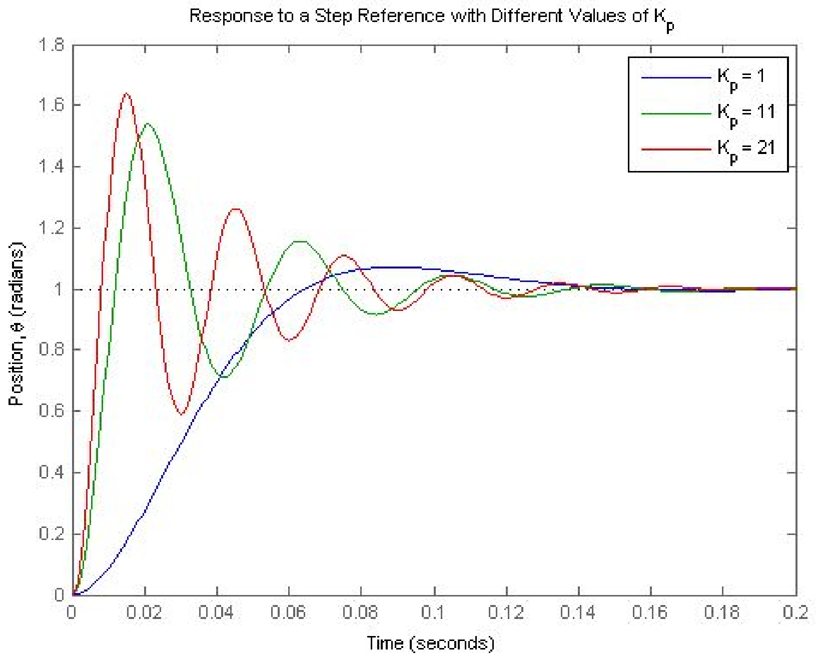


Figure A: Step response for various K_p values

The proportionality constant in a PID controller is the multiplication factor given to the present error in a system. A larger K_p will reach the desired position faster (the red line compared to the blue line in figure A), but will shoot beyond the position. This is not necessarily a bad thing if the overshoot is within an acceptable range. For a quadcopter, some overshoot is acceptable because this creates faster visual response to the control. Flying a quadcopter with very low K_p is like driving a car that takes a couple seconds to respond to a change in pressure on the gas pedal. The larger K_p values can create oscillations around the target position. Oscillations are very bad for systems if they create instability. The red and green lines pictured in figure A demonstrate damped oscillations and reach the desired position quickly, so these would be examples of acceptable oscillations for a quadcopter to experience.

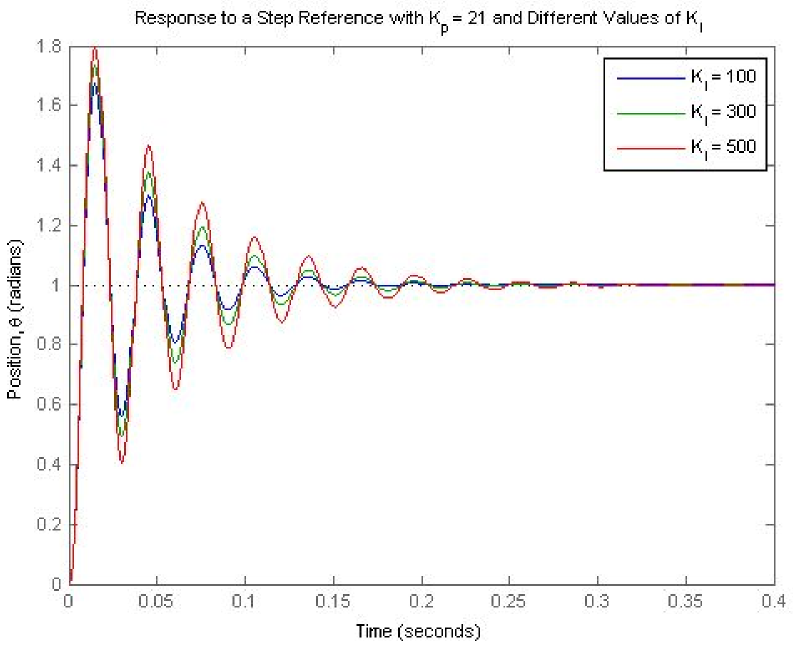


Figure B: Step Response for various K_i values

The integral constant in a PID controller affects the multiplier for the accumulated past error in a system. A larger integral constant will have the PID respond more strongly to the effect of past errors, so if it is expected that the past error has a large effect on the system then a large integral constant should be chosen. For the system being described in figure B, a larger integral constant had a negative effect on the response time. A quadcopter has a similar result. The past error is relatively less important for the response time in a quadcopter because the proportional term is dominant. However, integral control is important for correcting for undesired drifting in the position of a quadcopter.

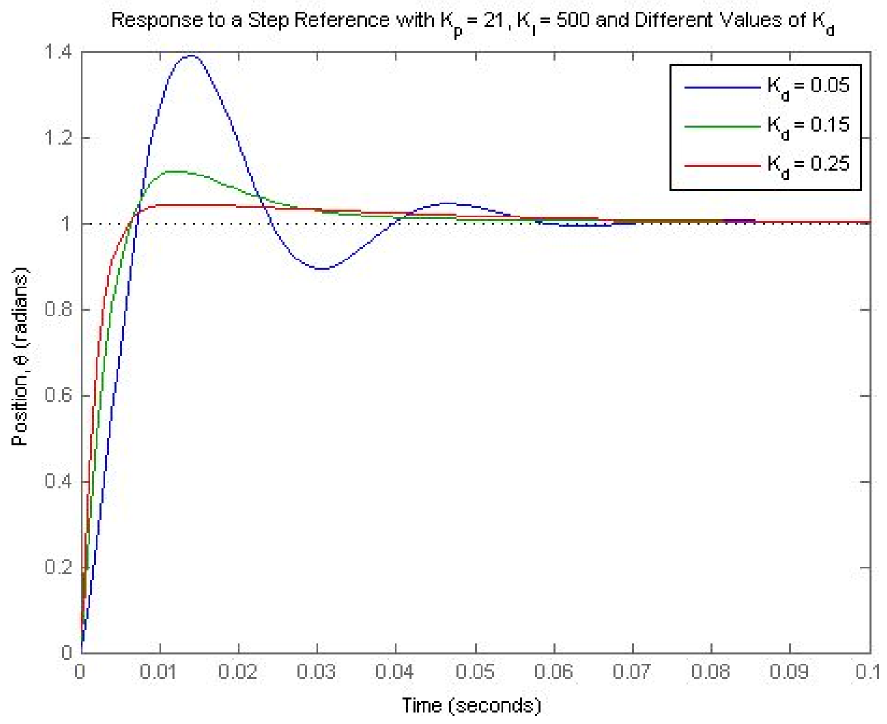


Figure C: Step Response for various K_D values

The derivative constant in a PID controller affects the multiplier for the predicted future error in a system. As the name implies, the future error is estimated by taking a derivative to calculate change. The effect of just a small amount of derivative control can be seen in the change produced in figure C. The response time is hardly different for the lines, but the overshoot sees significant changes from line to line. For a quadcopter, this values tends to be the smallest and is used to fine-tune and limit the overshoot from the proportional constant.

F. Delivery.h Code

```
/****************************************************************************
 *
 *   Copyright (c) 2013-2014 PX4 Development Team. All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 * 3. Neither the name PX4 nor the names of its contributors may be
 *    used to endorse or promote products derived from this software
 *    without specific prior written permission.
```

```
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
* FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
* COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
* INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
* BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
* OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
* AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
* ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
* POSSIBILITY OF SUCH DAMAGE.
*
****************************************************************************/
/**
* @file delivery.h
*
* Navigator mode to perform a delivery route
*
* @author Brian Krentz <brian.krentz@gmail.com>
*/

#ifndef NAVIGATOR_DELIVERY_H
#define NAVIGATOR_DELIVERY_H


#include <drivers/drv_hrt.h>

#include <controllib/blocks.hpp>
#include <controllib/block/BlockParam.hpp>

#include <dataman/dataman.h>

#include <uORB/uORB.h>
#include <uORB/topics/actuator_armed.h>
#include <uORB/topics/vehicle_global_position.h>
#include <uORB/topics/position_setpoint_triplet.h>
#include <uORB/topics/home_position.h>
#include <uORB/topics/safety.h>
#include <uORB/topics/actuator_armed.h>
#include <uORB/topics/vehicle_status.h>
#include <uORB/topics/turn_servo.h>

#include "navigator_mode.h"
#include "mission_block.h"
#include "mission_feasibility_checker.h"

class Delivery : public MissionBlock
{
public:
		Delivery(Navigator *navigator, const char *name);

		virtual ~Delivery();

		virtual void on_inactive();
		virtual void on_activation();

		virtual void on_active();

		/**
		*		Update these at the end of each phase
		*/
		enum DELIVSTATE {
				DELIV_PREFLIGHT,
				DELIV_ENROUTE,
				DELIV_DROPOFF,
				DELIV_RETURN,
				DELIV_DISARM,
				DELIV_COMPLETE
		} delivery_status;

		enum mission_altitude_mode {
				MISSION_ALTMODE_ZOH = 0,
				MISSION_ALTMODE_FOH = 1
		};

		enum mission_yaw_mode {
				MISSION_YAWMODE_NONE = 0,
				MISSION_YAWMODE_FRONT_TO_WAYPOINT = 1,
				MISSION_YAWMODE_FRONT_TO_HOME = 2,
				MISSION_YAWMODE_BACK_TO_HOME = 3
		};

private:

		/**
		 *  Takeoff and go to destination use mission planner with takeoff enabled
		 */
		void to_destination();

		/**
		*		Drop the item
		*/
		void activate_gripper();

		/**
		*		Return Home and land. Activate the RTL with landing enabled.
		*/
		void return_home();

		/**
		*		Shutoff the drone after it touches down.
		*/
		void shutoff();

		/**
		*		Conditions to set upon activation
		*/
		void set_delivery_items();

		/**
		*		Items to set return home feature
```

```
*/
void set_return_home();

/**
*		Advances the delivery_status to the next stage
*/
void advance_delivery();

/**
*		Closes the servo
*/
void load_package();

/**
*		Opens the servo
*/
void unload_package();

/* Mavlink file descriptor */
int _count;
bool _complete;
bool _first_run;
float _drop_alt;
int _armed_sub;
int _servo_sub;
struct actuator_armed_s _actuator_armed;
struct turn_servo_s gripper;
orb_advert_t pub_gripper;

//////////////////////////

/**
 * Update onboard mission topic
 */
void update_onboard_mission();

/**
 * Update offboard mission topic
 */
void update_offboard_mission();

/**
 * Move on to next mission item or switch to loiter
 */
void advance_mission();

/**
 * Check distance to first waypoint (with lat/lon)
 * @return true only if it's not too far from home (< MIS_DIST_1WP)
 */
bool check_dist_1wp();

/**
 * Set new mission items
 */
void set_mission_items();
```

```
/**
 * Updates the heading of the vehicle. Rotary wings only.
 */
void heading_sp_update();

/**
 * Updates the altitude sp to follow a foh
 */
void altitude_sp_foh_update();

/**
 * Resets the altitude sp foh logic
 */
void altitude_sp_foh_reset();

int get_absolute_altitude_for_item(struct mission_item_s &mission_item);

/**
 * Read current or next mission item from the dataman and watch out for DO_JUMPS
 * @return true if successful
 */
bool read_mission_item(bool onboard, bool is_current, struct mission_item_s *mission_item);

/**
 * Save current offboard mission state to dataman
 */
void save_offboard_mission_state();

/**
 * Inform about a changed mission item after a DO_JUMP
 */
void report_do_jump_mission_changed(int index, int do_jumps_remaining);

/**
 * Set a mission item as reached
 */
void set_mission_item_reached();

/**
 * Set the current offboard mission item
 */
void set_current_offboard_mission_item();

/**
 * Set that the mission is finished if one exists or that none exists
 */
void set_mission_finished();

/**
 * Check wether a mission is ready to go
 */
bool check_mission_valid();

control::BlockParamInt _param_onboard_enabled;
control::BlockParamFloat _param_takeoff_alt;
control::BlockParamFloat _param_dist_1wp;
control::BlockParamInt _param_altmode;
```

```
	control::BlockParamInt _param_yawmode;

	struct mission_s _onboard_mission;
	struct mission_s _offboard_mission;

	int _current_onboard_mission_index;
	int _current_offboard_mission_index;
	bool _need_takeoff;						/**< if true, then takeoff must be performed before going
to the first waypoint (if needed) */
	bool _takeoff;								/**< takeoff state flag */

	enum {
		MISSION_TYPE_NONE,
		MISSION_TYPE_ONBOARD,
		MISSION_TYPE_OFFBOARD
	} _mission_type;

	bool _inited;
	bool _home_inited;

	MissionFeasibilityChecker _missionFeasiblityChecker; /**< class that checks if a mission is feasible */

	float _min_current_sp_distance_xy; /**< minimum distance which was achieved to the current waypoint  */
	float _mission_item_previous_alt; /**< holds the altitude of the previous mission item,
					  can be replaced by a full copy of the previous mission item if needed */
	float _on_arrival_yaw; /**< holds the yaw value that should be applied when the current waypoint is reached */
	float _distance_current_previous; /**< distance from previous to current sp in pos_sp_triplet,
					    only use if current and previous are valid */

	//////////////////////////////

	/**
	 * Set the RTL item
	 */
	void set_rtl_item();

	/**
	 * Move to next RTL item
	 */
	void advance_rtl();

	enum RTLState {
		RTL_STATE_NONE = 0,
		RTL_STATE_CLIMB,
		RTL_STATE_RETURN,
		RTL_STATE_DESCEND,
		RTL_STATE_LOITER,
		RTL_STATE_LAND,
		RTL_STATE_LANDED,
	} _rtl_state;

	control::BlockParamFloat _param_return_alt;
	control::BlockParamFloat _param_descend_alt;
	control::BlockParamFloat _param_land_delay;

};
```

```
#endif
```

G. Servo Control Code

```
/****************************************************************************
*
*   Copyright (c) 2013 PX4 Development Team. All rights reserved.
*   Author: Anton Babushkin <anton.babushkin@me.com>
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
*
* 1. Redistributions of source code must retain the above copyright
*    notice, this list of conditions and the following disclaimer.
* 2. Redistributions in binary form must reproduce the above copyright
*    notice, this list of conditions and the following disclaimer in
*    the documentation and/or other materials provided with the
*    distribution.
* 3. Neither the name PX4 nor the names of its contributors may be
*    used to endorse or promote products derived from this software
*    without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
* FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
* COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
* INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
* BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
* OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
* AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
* ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
* POSSIBILITY OF SUCH DAMAGE.
*
****************************************************************************/

/**
* @file servo_ctl.c
*
* gripper servo via GPIO driver.
*
* @author Vinh Nguyen
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <stdbool.h>
#include <nuttx/wqueue.h>
#include <nuttx/clock.h>
#include <systemlib/systemlib.h>
#include <systemlib/err.h>
#include <uORB/uORB.h>
```

```
#include <uORB/topics/vehicle_status.h>
#include <uORB/topics/turn_servo.h>
#include <poll.h>
#include <drivers/drv_gpio.h>
#include <drivers/drv_pwm_output.h>
#include <modules/px4iofirmware/protocol.h>

struct servo_ctl_s {
        struct work_s work;
        int gpio_fd;
        bool use_io;
        int pin;
};

static struct servo_ctl_s *servo_ctl_data;
static int servo_ctl_pos = 0;
static int sub_gripper;
bool use_io;
bool updated;
int pin;

static bool thread_should_exit = false;         /**< daemon exit flag */
static bool thread_running = false;             /**< daemon status flag */
static int daemon_task;                                 /**< Handle of daemon task / thread */

int                 set_pwm_rate(unsigned rate_map, unsigned default_rate, unsigned alt_rate);
int                 _pwm_alt_rate_channels;
int         _pwm_default_rate;
int         _pwm_alt_rate;

__EXPORT int servo_ctl_main(int argc, char *argv[]);

__EXPORT void servo_ctl_pos1(void);

__EXPORT void servo_ctl_pos2(void);

void servo_ctl_set_pos(FAR void * arg, int rate);

void servo_ctl_stop(FAR void *arg);

void servo_ctl_sub_init(void);

void servo_ctl_sub(void);

int servo_ctl_daemon_thread_main(int argc, char *argv[]);

int servo_ctl_main(int argc, char *argv[])
{
        if (argc < 2) {

#ifdef CONFIG_ARCH_BOARD_PX4FMU_V2
                errx(1, "usage: servo_ctl {pos1|pos2|stop} [-p <n>]\n"
                   "\t-p <n>\tUse specified AUX OUT pin number (default: 1)"
                  );
#endif

        } else {
```

```
		if (!strcmp(argv[1], "pos1")) {
			if (servo_ctl_pos == 1) {
				errx(1, "already at pos1");
			}

			use_io = false;

			/* by default use GPIO_EXT_1 on FMUv1 and servo_ctl_1 on FMUv2 */
			pin = 1;

			/* pin name to display */

#ifdef CONFIG_ARCH_BOARD_PX4FMU_V2
			char pin_name[] = "AUX OUT 1";
#endif

			if (argc > 2) {
				if (!strcmp(argv[2], "-p")) {

#ifdef CONFIG_ARCH_BOARD_PX4FMU_V2
					unsigned int n = strtoul(argv[3], NULL, 10);

					if (n >= 1 && n <= 6) {
						use_io = false;
						pin = 1 << (n - 1);
						snprintf(pin_name, sizeof(pin_name), "AUX OUT %d", n);

					} else {
						errx(1, "unsupported pin: %s", argv[3]);
					}

#endif
				}
			}

			servo_ctl_data = malloc(sizeof(struct servo_ctl_s));
			memset(servo_ctl_data, 0, sizeof(struct servo_ctl_s));
			servo_ctl_data->use_io = use_io;
			servo_ctl_data->pin = pin;
			servo_ctl_pos1();
			servo_ctl_pos = 1;
			warnx("pos1, using pin: %s", pin_name);

			thread_should_exit = false;
			daemon_task = task_spawn_cmd("servo_daemon",
						SCHED_DEFAULT,
						SCHED_PRIORITY_DEFAULT,
						2000,
						servo_ctl_daemon_thread_main,
						(argv) ? (char * const *)&argv[2] : (char * const *)NULL);
			exit(0);

		} else if (!strcmp(argv[1], "pos2")) {
			if (servo_ctl_pos == 2) {
				errx(1, "already at pos2");
			}
```

```
				use_io = false;
				pin = 1;
				servo_ctl_pos2();
				servo_ctl_pos = 2;
				warnx("pos2, using pin: %d", pin);
				thread_should_exit = false;
				daemon_task = task_spawn_cmd("servo_daemon",
							SCHED_DEFAULT,
							SCHED_PRIORITY_DEFAULT,
							2000,
							servo_ctl_daemon_thread_main,
							(argv) ? (char * const *)&argv[2] : (char * const *)NULL);
				exit(0);

			} else if (!strcmp(argv[1], "stop")) {
				if (servo_ctl_pos != 0) {
					servo_ctl_pos = 0;
					warnx("stop");

					use_io = false;
					pin = 1;
					servo_ctl_data = malloc(sizeof(struct servo_ctl_s));
					memset(servo_ctl_data, 0, sizeof(struct servo_ctl_s));
					servo_ctl_data->use_io = use_io;
					servo_ctl_data->pin = pin;
					servo_ctl_stop(servo_ctl_data);

					thread_should_exit = true;

					exit(0);

				} else {
					errx(1, "not running");
				}

			} else {
				errx(1, "unrecognized command '%s', only supporting 'pos1', 'pos2', or 'stop'", argv[1]);
			}
		}
}

int servo_ctl_daemon_thread_main(int argc, char *argv[])
{
		thread_running = true;
		servo_ctl_sub_init();

		while(!thread_should_exit){
			servo_ctl_sub();
			sleep(10);
		}

		thread_running = false;
		return 0;
}


void servo_ctl_pos1()
```

```
{
		servo_ctl_data = malloc(sizeof(struct servo_ctl_s));
		memset(servo_ctl_data, 0, sizeof(struct servo_ctl_s));
		servo_ctl_data->use_io = use_io;
		servo_ctl_data->pin = pin;

		servo_ctl_set_pos(servo_ctl_data, 2000);
}

void servo_ctl_pos2()
{
		servo_ctl_data = malloc(sizeof(struct servo_ctl_s));
		memset(servo_ctl_data, 0, sizeof(struct servo_ctl_s));
		servo_ctl_data->use_io = use_io;
		servo_ctl_data->pin = pin;

		servo_ctl_set_pos(servo_ctl_data, 800);
}

void servo_ctl_set_pos(FAR void *arg, int rate)
{
		FAR struct servo_ctl_s *priv = (FAR struct servo_ctl_s *)arg;

		char *gpio_dev;

		// sets all actions to the FMU/AUX pins
		gpio_dev = PX4FMU_DEVICE_PATH;

		// open GPIO device
		priv->gpio_fd = open(gpio_dev, 0);

		if (priv->gpio_fd < 0) {
				// TODO find way to print errors
				printf("servo_ctl: GPIO device \"%s\" open fail\n", gpio_dev);
				servo_ctl_pos = 0;
				return;
		}

		// sets PWM values

		// initializes all FMU ports as servos, w/ magic numbers (mask value), taken from fmu.cpp
		up_pwm_servo_init(0xff);

		int ret = ioctl(priv->gpio_fd, PWM_SERVO_SET_ARM_OK, 0);

		if (ret != OK) {
				err(1, "PWM_SERVO_SET_ARM_OK");
		}
		else{
				warnx("servo set arm is a go");
		}

		// tell IO that the system is armed (it will output values if safety is off)
		ret = ioctl(priv->gpio_fd, PWM_SERVO_ARM, 0);

		if (ret != OK) {
				err(1, "PWM_SERVO_ARM");
		}
		else{
			warnx("servo legged");
		}

		// sets PWM values and activates servo
		ret = ioctl(priv->gpio_fd, PWM_SERVO_SET((priv->pin)-1), rate);

		if (ret != OK) {
			err(1, "PWM_SERVO_SET(%d)", priv->pin);
		}
		else{
			warnx("the wheel keeps on turning");
		}
}

void servo_ctl_stop(FAR void *arg)
{
		FAR struct servo_ctl_s *priv = (FAR struct servo_ctl_s *)arg;

		char *gpio_dev;

		// sets all actions to the FMU/AUX pins
		gpio_dev = PX4FMU_DEVICE_PATH;

		// open GPIO device
		priv->gpio_fd = open(gpio_dev, 0);

		if (priv->gpio_fd < 0) {
			// TODO find way to print errors
			printf("servo_ctl: GPIO device \"%s\" open fail\n", gpio_dev);
			servo_ctl_pos = 0;
			return;
		}

		int ret = ioctl(priv->gpio_fd, PWM_SERVO_DISARM, 0);

		if (ret != OK) {
			err(1, "PWM_SERVO_DISARM");
		}
		else {
			warnx("go to sleep");
		}
}

void servo_ctl_sub_init()
{
		sub_gripper = orb_subscribe(ORB_ID(turn_servo));
}

void servo_ctl_sub()
{
		struct turn_servo_s turn;
		orb_check(sub_gripper, &updated);

		if (updated){
			orb_copy(ORB_ID(turn_servo), sub_gripper, &turn);
```

```
                    if (turn.stopped){
                            servo_ctl_data = malloc(sizeof(struct servo_ctl_s));
                            memset(servo_ctl_data, 0, sizeof(struct servo_ctl_s));
                            servo_ctl_data->use_io = use_io;
                            servo_ctl_data->pin = pin;
                            servo_ctl_stop(servo_ctl_data);
                    }
                    else if (turn.open){
                            servo_ctl_pos1();
                    }
                    else{
                            servo_ctl_pos2();
                    }
            }
}
```

H. Barrier Avoidance Code

```
/*      commander.cpp     */

static struct manual_control_setpoint_s mcs;
static int timecnt=0;
__EXPORT bool isInAdcMode = false;
#define PROX_THRESHOLD    250

.  .  .  .  .  .  .  .
.  .  .  .  .  .  .  .

orb_check(adc_prox_sub, &updated);    // Check if the adc value has been updated

if(updated){

        orb_copy(ORB_ID(adc_prox), adc_prox_sub, &adc_prox);   // copy the value to a struct

}

int data=adc_prox.data;

if(data < PROX_THRESHOLD && data>19){                  //if proximity sensor value ranges within
                                                       // threshold value for twice,  enter adc mode,
                                                       //  change state to POSCTL
        timecnt++;

}   else   {

        timecnt=0;
        isInAdcMode = false;

}
if(timecnt>1){
        int shuai = main_state_transition(&status,vehicle_status_s::MAIN_STATE_POSCTL);
        if(shuai==1){
                main_state_changed = true;
        }
        isInAdcMode = true;
}
. . . . . . . . . .
. . . . . . . . . .

// ascend when proximity sensor ranges within the threshold

if(isInAdcMode){                                    // if we are in adc mode, publish a thrust in
                                                    // the manual_control_setpoint topic which is
                                                    // subscribed by mc_pos_control.cpp
        mcs.x = 0;
        mcs.y = 0;
        mcs.z = 0.8;        //  set throttle to 0.8
        mcs.r = 0;

        mcs.timestamp = now;

        // Lazy publish
        if(mcs_pub < 0)
                mcs_pub = orb_advertise(ORB_ID(manual_control_setpoint), &mcs);
        else
                orb_publish(ORB_ID(manual_control_setpoint), mcs_pub, &mcs);
}
```

I. VendorTrackPage.qml Code

```
import QtQuick 2.0
import QtQuick.Controls 1.3
import QtQuick.Controls.Styles 1.3
import QtPositioning 5.2
import QtLocation 5.3

Rectangle {
   width: parent? parent.width : 500
   height: parent? parent.height : 800
   visible: true
   Image {
       visible: true
       anchors.right: parent.right
       anchors.rightMargin: page.width*0.05
       anchors.bottom: parent.bottom
       anchors.bottomMargin: page.width*0.05
       width: page.width*.10
       height: page.width*.10
       source: "qrc:/logo.png"
   }
   Text {
       anchors.horizontalCenter: parent.horizontalCenter
       anchors.topMargin: page.height * 0.03
       anchors.top: parent.top
       text: "TRACK ORDER"
       font.family: "Avenir"
       font.letterSpacing: 2
```

```
}
Plugin {
    id: osmplugin1
    name:"osm"
}
Item {
    id: tracking_drone
    anchors.horizontalCenter: parent.horizontalCenter
    y: page.height * 0.08
    width: page.width * 0.8
    height: page.width * 0.6

    Text {
        id: targetlatitude
        visible: false
    }
    Text {
        id: targetlongitude
        visible: false
    }

    Address {
        id: fromAddress
        street: if (vendor_handler.delivery == 1) {vendor_handler.street1}
                else if (vendor_handler.delivery == 2) {vendor_handler.street2}
                else if (vendor_handler.delivery == 3) {vendor_handler.street3}
        city: if (vendor_handler.delivery == 1) {vendor_handler.city1}
              else if (vendor_handler.delivery == 2) {vendor_handler.city2}
              else if (vendor_handler.delivery == 3) {vendor_handler.city3}
        country: if (vendor_handler.delivery == 1) {vendor_handler.region1}
                 else if (vendor_handler.delivery == 2) {vendor_handler.region2}
                 else if (vendor_handler.delivery == 3) {vendor_handler.region3}
        state : if (vendor_handler.delivery == 1) {vendor_handler.state1}
                else if (vendor_handler.delivery == 2) {vendor_handler.state2}
                else if (vendor_handler.delivery == 3) {vendor_handler.state3}
        postalCode: if (vendor_handler.delivery == 1) {vendor_handler.zip1}
                    else if (vendor_handler.delivery == 2) {vendor_handler.zip2}
                    else if (vendor_handler.delivery == 3) {vendor_handler.zip3}
    }
    Map {
        id: map2
        plugin: osmplugin1
        width: tracking_drone.width
        height: tracking_drone.height
        zoomLevel: mapslider1.value
        center {
            latitude: vendor_handler.latitude
            longitude: vendor_handler.longitude
        }
        gesture.flickDeceleration: 3000
        gesture.enabled: true


        MapCircle {
            id: point
            radius: if (mapslider1.value < 13) {200}
                    else {30}
            color: "#3938FF"
            border.color: "#000000"
            border.width: 2
            opacity: 0.6
            center {
                latitude: vendor_handler.latitude
                longitude: vendor_handler.longitude
            }
        }

        MapCircle {
            radius:
                if (mapslider1.value > 13) {5}
                else if (mapslider1.value = 13) {20}
                else if (mapslider1.value > 12 & mapslider1.value < 13) {200}
                else {1000}
            color: "#FF0F47"
            opacity: 0.6
            border.width: 2
            border.color: "#000000"
            center {
                latitude: manual_control_handler.latitude
                longitude: manual_control_handler.longitude
            }
        }

        MapPolyline {
            id: polyline2
            line.width: 3
            line.color: "#000000"
            path: [
                { latitude: vendor_handler.latitude, longitude: vendor_handler.longitude },
                { latitude: targetlatitude.text, longitude: targetlongitude.text }
            ]
        }

        GeocodeModel {
            id: geocodeModel1
            plugin: osmplugin1
            autoUpdate: true
            query: fromAddress
            onLocationsChanged: {
                map2.center.latitude = get(0).coordinate.latitude
                map2.center.longitude = get(0).coordinate.longitude
                targetlatitude.text = get(0).coordinate.latitude
                targetlongitude.text = get(0).coordinate.longitude
            }
        }
        Component {
            id: pointDelegate1

            MapCircle {
                id: point2
                radius: if (mapslider1.value < 13) {200}
                        else {30}
                color: "#C638FF"
                border.color: "#000000"
```

```
                border.width: 2
                opacity: 0.6
                center {
                    latitude: targetlatitude.text
                    longitude: targetlongitude.text
                }
            }
        }
        MapItemView {
            model: geocodeModel1
            delegate: pointDelegate1
        }
    }

}
Slider {
    id: mapslider1
    anchors.right: parent.right
    anchors.rightMargin: 15
    anchors.top: parent.top
    anchors.topMargin: tracking_drone.y
    height: map2.height
    value: 15
    maximumValue: 19
    minimumValue: 2
    tickmarksEnabled: false
    updateValueWhileDragging: true
    visible: true
    orientation: Qt.Vertical

}
Text {
    id: currentdronecoordinates
    y: tracking_drone.y + tracking_drone.height + 20
    anchors.left: parent.left
    anchors.leftMargin: 50
    text: "Current Drone Latitude: " + '<br>' + "Current Drone Longitude: "
    font.family: "Avenir"
    font.letterSpacing: 2
}
Text {
    y: tracking_drone.y + tracking_drone.height + 20
    anchors.right: parent.right
    anchors.rightMargin: 50
    text: manual_control_handler.latitude.toFixed(8) + '<br>' + manual_control_handler.longitude.toFixed(8)
    font.family: "Avenir"
    font.letterSpacing: 2
    z: currentdronecoordinates.z + 3
}
Text {
    id: currentdistance_label
    y: currentdronecoordinates.y + currentdronecoordinates.height + 20
    anchors.left: parent.left
    anchors.leftMargin: 50
    text: "Current Distance:"
    font.family: "Avenir"
    font.letterSpacing: 2
    }
    Text {
        id: currentdistance
        anchors.right: parent.right
        anchors.rightMargin: 50
        y: currentdronecoordinates.y + currentdronecoordinates.height + 20
        text: Math.floor(6371000*Math.acos(Math.sin(vendor_handler.latitude*0.0174532925) *
Math.sin(targetlatitude.text*0.0174532925) + Math.cos(vendor_handler.latitude*0.0174532925) *
Math.cos(targetlatitude.text*0.0174532925) * Math.cos (targetlongitude.text*0.0174532925 -
vendor_handler.longitude*0.0174532925))) + " m"
        font.family: "Avenir"
        font.letterSpacing: 2
    }
    Text {
        id: deliverystatus
        y: currentdistance.y + currentdistance.height + 20
        anchors.left: parent.left
        anchors.leftMargin: 50
        text: "Current Status:"
        font.family: "Avenir"
        font.letterSpacing: 2
    }
    Text {
        id: display_deliverystatus
        anchors.right: parent.right
        anchors.rightMargin: 50
        y: currentdistance.y + currentdistance.height + 20
        text:
            if (delivered_validationsign.source == "checkgrey.png" & returned_validationsign.source ==
"checkgrey.png") {"Delivering"}
            else if (delivered_validationsign.source == "checkgreen.png" & returned_validationsign.source ==
"checkgrey.png") {"Returning"}
            else if (returned_validationsign.source == "checkgreen.png") {"Returned"}
        font.family: "Avenir"
        font.letterSpacing: 2
    }
    Text {
        id: delivered
        y: deliverystatus.y + deliverystatus.height + 20
        anchors.left: parent.left
        anchors.leftMargin: 50
        text: "Delivered"
        font.family: "Avenir"
        font.letterSpacing: 2
    }
    Image {
        id: delivered_validationsign
        y: delivered.y - 5
        height: delivered.height + 10
        width: delivered.height + 10
        anchors.right: parent.right
        anchors.rightMargin: 50
        source:
            if (Math.abs(targetlatitude.text - manual_control_handler.latitude) < 0.0002 &
Math.abs(targetlongitude.text - manual_control_handler.longitude) < 0.0002) {"checkgreen.png"}
        else {"checkgrey.png"}
    }
    Text {
        id: returned
        y: delivered.y + delivered.height + 20
        anchors.left: parent.left
        anchors.leftMargin: 50
        text: qsTr("Returned")
        font.family: "Avenir"
        font.letterSpacing: 2
    }
    Image {
        id: returned_validationsign
        y: returned.y - 5
        height: returned.height + 10
        width: returned.height + 10
        anchors.right: parent.right
        anchors.rightMargin: 50
        source:
            if (Math.abs(vendor_handler.latitude - manual_control_handler.latitude) < 0.0002 &
Math.abs(vendor_handler.longitude - manual_control_handler.longitude) < 0.0002) {"checkgreen.png"}
        else {"checkgrey.png"}

    }
    Text {
        id: currentbattery
        anchors.left: parent.left
        anchors.leftMargin: 50
        y: returned.y + returned.height + 20
        text: if (((manual_control_handler.voltage - 13500) / 31).toFixed(1) > 0) {
                 "Current Battery: "+ ((manual_control_handler.voltage - 13500) / 31).toFixed(1) + "%"}
              else {"Current Battery: 0%"}
        font.family: "Avenir"
        font.letterSpacing: 2
    }
    Rectangle {
        id: battery1
        anchors.right: parent.right
        anchors.rightMargin: 50
        y: currentbattery.y
        width: 75
        height: currentbattery.height
        color: "#00000000"
        border.width: 2
    }
    Rectangle {
        id: batteryfill
        x: battery1.x + 5
        y: battery1.y + 5
        width: if (((manual_control_handler.voltage - 13500) / 31) > 0.0) {
                    (battery1.width - 10.0)*(((manual_control_handler.voltage - 13500) / 3100))}
               else {0}
        color: if (((manual_control_handler.voltage - 13500) / 31) >= 85.00) {"#65E01F"}
               else if (((manual_control_handler.voltage - 13500) / 31) >= 60.00) {"#FF790A"}
               else if (((manual_control_handler.voltage - 13500) / 31) < 60.00) {"#D60000"}
        height: battery1.height - 10
        visible: true
    }
    Button {
        id: return_to_manual
        anchors.horizontalCenter: parent.horizontalCenter
        anchors.bottom: parent.bottom
        anchors.bottomMargin: page.height * 0.14
        visible: true
        text: "Manual control"
        width: cancelreturn.width
        onClicked: {
            vendor_track_page.visible = false
            manual_control_page.visible = true
        }
    }
    Button {
        id: view_other_orders
        anchors.horizontalCenter: parent.horizontalCenter
        anchors.bottom: parent.bottom
        anchors.bottomMargin: page.height * 0.09
        visible: true
        text: "View other orders"
        width: cancelreturn.width
        onClicked: {
            vendor_track_page.visible = false
            pending_order_page.visible = true
            if (delivered_validationsign.source == "checkgreen.png" | returned_validationsign.source ==
"checkgreen.png") {
                if (vendor_handler.delivery == 1) {
                    vendor_handler.reset1()
                    vendor_handler.delivery = 0
                    if (vendor_handler.valid2) {
                        vendor_handler.pass2to1()
                        vendor_handler.reset2()
                    }
                    if (vendor_handler.valid3) {
                        vendor_handler.pass3to2()
                        vendor_handler.reset3()
                    }

                }
                else if (vendor_handler.delivery == 2) {
                    vendor_handler.reset2()
                    vendor_handler.delivery = 0
                    if (vendor_handler.valid3) {
                        vendor_handler.pass3to2()
                        vendor_handler.reset3()}
                }
                else if (vendor_handler.delivery == 3) {
                    vendor_handler.reset3()
                    vendor_handler.delivery = 0
                }
            }
        }
    }
    Button {
        id: cancelreturn
        anchors.horizontalCenter: parent.horizontalCenter
        anchors.bottom: parent.bottom
        anchors.bottomMargin: page.height * 0.04
        visible: if (returned_validation.text == "Y") {false} else {true}
        text: "Cancel Order and Return"
        onClicked: {

            manual_control_handler.setFlightMode(0)
            display_deliverystatus.text = "Returning"
            delivered_validationsign.source == "checkgrey.png"
        }
    }
}
```