Abstract

With the increasing need for product delivery services, the potential for drone delivery in Hong Kong is huge. However, typical drones do not have enough battery life for long-range flights. To save energy, drones should take buses, just like to way we do. The project is called “Land-Air Hybrid Drone Delivery System”. Drones will deliver their packets by riding on buses and flying in the air. Remote control over the Internet and computer vision will be the focuses of this project.

Currently, the remote control mechanism is all right and the drone can now locate landing spots and detect bus numbers using its camera. Also, the drone can fly across Google Maps using GPS signals with obstacle avoidance available. However, the delays in Internet connection will be a significant limitation of the system. It will be improved and compensated in the future.

Acknowledgment

First and foremost, thanks to my supervisor, Dr. Loretta Choi, for her continued support and encouragement. I offer my sincere appreciation for the opportunity to propose and start such a unique project.

Apart from my supervisor, I am very grateful to a PhD student, Mr. Jiaming Xie, for gratuitously lending me an expensive drone. This project could not have been started without his support. My heartfelt thanks.
# Table of Contents

1 Introduction ........................................................................................................................ 1
   1.1 Overview ................................................................................................... 1
   1.2 Motivation ................................................................................................. 1
   1.3 Major Challenge ........................................................................................ 2
   1.4 Proposed Solution ..................................................................................... 2
   1.5 Outline ....................................................................................................... 3

2 Objectives ........................................................................................................................... 4
   2.1 Conceptual Model ..................................................................................... 4
   2.2 Scope ......................................................................................................... 5
   2.3 Problem Definition .................................................................................... 6

3 Methodology ...................................................................................................................... 7
   3.1 Fully Customized Environment for Remote Controls ............................... 7
      3.1.1 Approach ............................................................................................ 7
   3.2 Image Recognition System (Phase 1: Distance) ..................................... 10
      3.2.1 Approach ............................................................................................ 10
   3.3 Image Recognition System (Phase 2: Bus) ............................................. 12
      3.3.1 Approach ............................................................................................ 12
   3.4 Auto Landing and Take-off (Phase 1: Stationary Spot) .......................... 14
      3.4.1 Approach ............................................................................................ 14
   3.5 Auto Landing and Take-off (Phase 2: Moving Spot) .............................. 15
      3.5.1 Design ................................................................................................. 15
      3.5.2 Approach ............................................................................................ 15
   3.6 GPS based Automatic Flight ................................................................... 18
      3.6.1 Design ................................................................................................. 18
      3.6.2 Approach ............................................................................................ 20
   3.7 Obstacle Avoidance Algorithm ................................................................ 21
      3.7.1 Approach ............................................................................................ 21

4 Evaluation ......................................................................................................................... 23
   4.1 Fully Customized Environment for Remote Controls ............................. 23
      4.1.1 Results ............................................................................................... 23
1 Introduction

1.1 Overview

Drone Delivery in Hong Kong

With the advent of drone technology and growing industrial use, drone delivery could be the future of the transport industry. Hong Kong is a geographically small city (approximately 21 km in radius). By taking Lai Chi Kok as the delivery center, there is a huge potential for the implementation of drone delivery.

![Figure 1: The map of Hong Kong with its circumference highlighted](https://www.google.com.hk/maps)

1.2 Motivation

In recent years, online shopping becomes more popular than in-store shopping. The need for product delivery services grows with each passing day, especially after the outbreak of COVID-19. However, the traditional methods of transportation have not improved much in this age of E-Commerce. It relies heavily on manpower. Not only do we need drivers to transfer batches of packets, but also workers to deliver packets client by client.

If drone delivery is employed, a single drone operator can control multiple drones at a time with the assistance of semi-autonomous flight control. No more drivers and workers are required. Therefore, human costs could be greatly reduced.
1.3 Major Challenge

The flight range of typical consumer drones cannot fully cover the area of Hong Kong. Logically, to make a round trip, the distance between the drone and the target location should be shorter than half of its maximum flight range.

<table>
<thead>
<tr>
<th>Drone Model</th>
<th>Max Flight Time</th>
<th>Top Speed</th>
<th>Max Flight Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>DJI Mavic Air 2</td>
<td>34 minutes</td>
<td>19 m/s</td>
<td>18.5 km</td>
</tr>
<tr>
<td>DJI Mavic 2 Pro</td>
<td>31 minutes (at 6.94 m/s)</td>
<td>20 m/s</td>
<td>18 km</td>
</tr>
<tr>
<td>DJI Mavic Mini</td>
<td>30 minutes (at 3.89 m/s)</td>
<td>13 m/s</td>
<td>7 km (C)</td>
</tr>
<tr>
<td>DJI Mavic 2 Zoom</td>
<td>31 minutes (at 6.94 m/s)</td>
<td>20 m/s</td>
<td>18 km</td>
</tr>
<tr>
<td>DJI Phantom 4 Pro</td>
<td>30 minutes</td>
<td>20 m/s</td>
<td>18 km (C)(S)</td>
</tr>
<tr>
<td>Powervision PowerEgg X</td>
<td>30 minutes</td>
<td>17.9 m/s</td>
<td>5.95 km</td>
</tr>
<tr>
<td>Parrot Anafi</td>
<td>25 minutes</td>
<td>15 m/s</td>
<td>11.3 km (C)(S)</td>
</tr>
<tr>
<td>Parrot Bebop 2</td>
<td>25 minutes</td>
<td>18 m/s</td>
<td>13.5 km (C)(S)</td>
</tr>
<tr>
<td>DJI Mavic Pro</td>
<td>27 minutes</td>
<td>17.9 m/s</td>
<td>13 km</td>
</tr>
</tbody>
</table>

(C) for calculated value, Distance = Speed × Time; (S) for unstated speed, Assumed Speed = Max Speed / 2;

Table 1: The most popular consumer drones with their maximum range listed [1,2,3,4]

From Table 1, average maximum flight distance = 13.8 km < 21 km (Hong Kong radius). Even we do not take wind resistance and payload weight into account, a drone clearly cannot fly across Hong Kong within one battery life, not to mention making any return trip.

1.4 Proposed Solution

Land-Air Hybrid Drone Delivery System

Drones should take buses for energy saving. Just like how we take a bus, a drone with packets will fly to and park on a bus that has the correct bus route number such that it can get close to the target spot. After arrival, the drone will take off and fly from the bus stop to the target spot. Then, it will deliver its packets. Finally, the drone will return to the delivery center using the same method.
1.5 Outline

This report contains six main sections: objectives, methodology, evaluation, required hardware and software, and future planning.

First, the objectives of the project will be discussed. The detailed definition of the problem is described in this section.

Second, for the methodology, a fully customized environment for remote controls and an image recognition system are completely developed. Therefore, remote control of the drone over the Internet is made possible. Also, drones can now perform auto takeoff and landing and GPS flight with obstacle avoidance.

Third, for the evaluation, all of the features were tested. The results and limitations were listed in the section.

Forth, the required hardware and software are listed. Due to the limit of budget, the development drone is borrowed from a PhD student.

Last, some possible future tasks are mention and there is a Gantt chart to visualize the overall progress.
2 Objectives

2.1 Conceptual Model

Sequence of Events during a Drone Delivery

1. A product is ordered by a customer.
2. A packet containing the ordered product is attached to a drone.
3. The flight operator looks for the fastest bus route from the delivery center to the customer’s location according to the given address.
4. The flight operator controls the drone to fly towards and land on the starting bus stop according to the bus route.
5. The drone’s camera points towards the street and looks for a bus with the correct bus route number.
6. Once a correct bus is detected, the drone automatically flies above the bus and search for a landing spot marking on top of the bus using its camera.
7. Once the landing spot marking is detected, the drone performs auto-landing onto it.
8. The drone waits on top of the bus until gets close to the customer’s location.
9. According to GPS signal, the drone automatically flies towards the customer’s location.
10. By using the drone’s camera, the flight operator locates the customer’s flat and drops the packet at the flat’s balcony.
   (If the flat does not have a balcony, drop the packet to self-pickup spot [5] nearby)
11. Use a similar method to make a return trip back to the delivery center.

Figure 2: The schematic diagram of the drone taking a bus
2.2 Scope

**Semi-autonomous Flight Control**

A complete Land-Air Hybrid Drone Delivery System should consist of two major components, namely semi-autonomous flight control and grips. The latter is omitted from the scope of the project. This project aims to utilize computer vision to automate the take-off and landing of drones on moving vehicles and provide assisted control during the flight. The system will be able to detect landing spot and bus route number printed on a bus. The system will also tell a drone to dodge when obstacles are detected in front of it.

The payload for delivery should be small in size and light in weight, which can be loaded on typical consumer drones. For example, toothbrush, computer mouse, cup, and etc.

**Proof of Concept**

This project is a proof of concept. It focuses only on the technological field, legal issues [6] are to be taken care of by the flight operators after implementation. Take-off and landing will be performed on a homemade moving platform that can act as the rooftop of a bus. The payload will be attached and fixed onto the drone to simulate a packet-loaded situation. If the project is brought to reality, the take-off and landing should be done on an actual bus. Partnerships should be established with public transport companies. Moreover, there should also be grips to stabilize drones on buses’ rooftops and allow drones to deliver packets.

Every drone is expensive. Any crashing is insufferable. Therefore, all flight operators should be experienced and well-trained for flying drones.

The homemade moving platform is a cart made by thin wooden plates which have bus number and landing spot printed and stuck onto it.
2.3 Problem Definition

The system can be separated into FIVE major sections:

1. Customized Environment for Remote Controls
   a. Complete remote control through Network
      A mobile device with a cellular network is attached to the drone to facilitate remote control over the Internet.
   b. Java-Python Integration and Asynchronous Processing
      As the drone controller app and the image recognition system are written in Java and Python respectively, a specific way for integration between two languages is required.

2. Image Recognition System
   There are two kinds of information to be extracted from the camera on the drone:
   a. Distance estimation
      When the drone is in the air, it looks for a verified QR code as the landing spot. The distance and bearing between the drone and the spot are essential for auto-landing.
   b. Bus display detection
      When the drone stays at the bus stop, it points its camera to the road and looks for a bus. Auto take-off is performed after the wanted bus route and direction are detected.

3. Auto Landing and Take-off
   With the distance and bearing estimated, a 3D coordinate system that takes the position of the drone as the origin is computed. By tilting the row, pitch, and yaw of the drone, the drone can get directly above the landing spot for landing.

4. GPS based Automatic Flight
   After putting waypoints on the Map, the drone can follow the path and fly automatically.

5. Obstacle Avoidance Algorithm
   During an automatic flight, the flight operator may not be aware of the drone. The drone should be able to keep itself safe without manual control.
3 Methodology

3.1 Fully Customized Environment for Remote Controls

3.1.1 Approach

Complete remote control through Network

The drone manufacturer DJI has published their own SDK (A Ready-To-Run Android Studio Project) [7] including Application Programming Interface (API) to control their drones. A bridge device was built using the resource from their API. It is a medium for the main controller phone (master device) to take complete control of a drone remotely. The bridge device is directly linked to the drone. When the bridge device is connected to the Internet over a cellular network, its corresponding drone can be controlled by a master device.

Concretely speaking, there is a bridge app installed on an Android smartphone (bridge device). It is based on the API from DJI and written in Java language. It converts all received data from its corresponding drone into network packets to the master device and all received network packets into control signals to the drone. After the implementation of the bridge, the master device can remotely control drones even it does not have a direct connection to the drone.

[Diagram of remote control mechanism]

Figure 4: Block diagram of the remote control mechanism
Java-Python Integration

The official DJI API for android is written in JAVA. However, the image processing program is written in Python. Those processes are designed to be Server-based. Two python Flask web servers are responsible to compute the distance and extract the bus number respectively.

Asynchronous Processing and Control Main Loop

As the distance detection is essential to be as fast as possible for automatic flight, the flask server to compute distance is running on the mobile device for smallest delay.

Regrettably, the mobile device does not have enough memory and computing power for real-time OCR. Thus, the bus display recognition server is running on a computer which is connected over the Internet.

The system is powered by a Control Main Loop (the Loop) mechanism, such that a loop is running at 20 times per second to send action command to the drone. An action is executed as long as one of the global parameters is altered to be non-zero. For example, a Yaw Rotation is sent to the drone if the Yaw parameter is not 0 in current iteration. In other words, the Loop
waits for a summary of next action for every 1/20 seconds (or 50 milliseconds). In idle stage, the Loop constantly sends no commands to the drone. Every time the controller device receives sensor data (including distance and bus results), no action commands are directly generated from the data. Instead, the input data is processed and the derived values is added to the parameters of the Loop. As a consequence, multiple action signals from different sensors are summed up and combined together even though they are received and processed from different threads. For example, when the distance results tell the drone to move forward and the height sensor tells the drone to fly up, two information from difference sources (threads) are combined and executed at the same time.

There are three kinds of action commands, their “overriding” priority is arranged in ascending order as following: 1. General Autonomous Actions; 2. Obstacle Avoidance (Autonomous) Actions; 3. Manual Control Actions. For instance, if a Manual Control Action is generated (the analog controllers (joysticks) are dragged by the flight operator), all General Autonomous Actions and Obstacle Avoidance Actions are ignored and bypassed.

This kind of event is called Interruption. The Loop itself acts as the buffer to store up all the actions to be executed. Once an action has a lower priority than the others, it will be ignored and never sent to the drone. This can be put into practice all because of the Control Main Loop mechanism.

Figure 6: Block diagram of the asynchronous processing mechanism with visualized data flow
3.2 Image Recognition System (Phase 1: Distance)

3.2.1 Approach

Computer Vision is the focus of this section. Video will be streamed from the drone’s camera to the mobile device. There are two types of markings to be recognized: landing spot and bus route number. The image recognition system is separated into two phases accordingly.

In the first phase, the system is able to observe and extract positional information from a landing spot marking. A custom QR code pattern was designed to be the marking.

![Custom QR code for landing spot markings](content: Jimmy Lee FYP)

Once the pattern is detected by the drone’s camera, the spherical coordinates of the landing spot, which are specified by three parameters: radial distance (distance to origin), polar angle (pitch), and azimuthal angle (yaw) will be calculated. The detailed process is described below:
0. Input:
Input frame is streamed from the drone at 720p resolution (1280 pixels × 720 pixels)

1. Preprocess:
   Normalize brightness + Grayscale + Bilateral filter + Threshold
   (Using OpenCV image processing library)

2. Find all QR corners:
   Find contours + Find overlapping quadrilaterals

3. Group QR corners (3 per group):
   Erode + Find contours → Contours that overlap with exactly 3 corners = QR codes

4. Estimate 4th corner and center:
   Extend existing corners' edge lines → New coincidence = 4th corner
   Connect diagonals → Coincidence = center

5. Decode and Filter QR code:
   Perspective transform + Decode → verify QR content → filter out incorrect QR code

6. Calculate distances and angles
   (completely original method)
   (formulas shown in Appendix 9.1)

7. Return the result which has the closest distance

Figure 8: Detailed process of distance estimation
3.3 Image Recognition System (Phase 2: Bus)

3.3.1 Approach

In the second phase of the image recognition system, the drone can perform bus display recognition. It is able to identify whether there is a bus route number displayed on an oncoming vehicle. By using Optical Character Recognition (OCR) technology, the numbers displayed on a bus can be read by the program.

The recognition algorithm is combined with a Rule-based text detection using OpenCV and an OCR model which is built using ResNet and LSTM with CTC. A pre-trained model from Github [8] is utilized for this project.

If a bus route number is recognized, the program will check if the number is the one that the flight operator is looking for. The detailed process is described below:

0. Input:
   Input frame is streamed from the drone at 720p resolution (1280 pixels × 720 pixels)

1. Preprocess:
   Normalize brightness + Grayscale + Bilateral filter + Adaptive Threshold

2. Find all possible bus number display:
   Find contours + Find big quadrilaterals + Merge overlapping contours

Figure 9a: Detailed process of bus display recognition
3. Extract and process rectangle
   (remove dotted words):

   Perspective transform + Normalize brightness +
   Gaussian Blur → base
   translate base by (-1,-1) → trans1
   translate base by (-1,1) → trans2

   merge base, trans1, and trans2 using max function
   → dots connected
   Median blur + Bilateral Filter + Unsharp mask +
   Contrast

4. Extract text:
   Grayscale + Thresh + Find contours + Group all
   white shapes (ignore English text line)

5. Feed to OCR model

6. Match output with the existing data of bus
   number and destination pairs:
   e.g. [('大', 0.995), ('埔', 0.992), ('墟', 0.971)],
   [('A', 0.148), ('4', 0.837), ('7', 0.966), ('X', 0.848)]
   → '大埔墟 A47x', 84.3%

   Figure 9b: Detailed process of bus display recognition

The record that has the greatest similarity will be the final output. If no records have their
similarities greater than 50%, the raw output is then forwarded and marked as unmatched.
3.4 Auto Landing and Take-off (Phase 1: Stationary Spot)

3.4.1 Approach

Using the built-in function from DJI SDK, the drone can seemingly stay still in the air even when it is blown by the wind. Moreover, the drone can perform tiny adjustments of its position in the air. The drone moves according to its pitch, roll, and yaw angles. Pitch and Roll angles control the drone’s position, while Yaw angles control its heading direction.

**Brute Force Algorithm**

In this stage, an intuitive simple algorithm is applied. Once the drone detects a correct bus display, it flies upwards and searches for the QR code. As a 3D coordinate system is computed according to the QR code, 8 directions of flight can be derived: “Front”, “Left”, “Back”, “Right”, “Front and Left”, “Back and Left”, “Front and Right”, and “Back and Right”. The drone flies towards the QR code with small steps repeatedly until the horizontal distance between the drone and the landing spot stays nearly zero. It can be classified as a brute force algorithm. The following is the definition of a “step”:

During a “Front” step, the drone tilts its Pitch angle 5 degrees down for 0.2 seconds.

During a “Back” step, the drone tilts its Pitch angle 5 degrees up for 0.2 seconds.

During a “Left” step, the drone tilts its Roll angle 5 degrees anti-clockwise for 0.2 seconds.

During a “Right” step, the drone tilts its Roll angle 5 degrees clockwise for 0.2 seconds.

With the above settings, the approach for auto takeoff and landing is intuitive.

1. The drone on the ground tilts its camera up to 20° and waits for its target bus.
2. The drone flies 1.5m upwards once it detects its target bus in its camera.
3. The drone tilts its camera down to 40° and looks for the landing spot (QR code).
4. As distance and direction are extracted from the landing spot, the drone moves slowly towards it with continuous tiny adjustments. The camera’s pitch angle is continuously adjusted to keep track of the landing spot, according to the pitch difference between the camera view and the landing spot. If the drone fails to detect the landing spot for 3 seconds, the drone flies backward slowly until the landing spot is detected.
5. Once the camera’s pitch angle keeps being more than 80° down for 1 second, it is steady upon the landing spot. Therefore, landing is performed.
3.5 Auto Landing and Take-off (Phase 2: Moving Spot)

3.5.1 Design

Stepping into the second half stage of the entire project, the auto-landing algorithm must be upgraded so as to support moving landing spots. Previously, the drone is limited to land only on a stationary landing spot due to insufficient speed.

3.5.2 Approach

Mathematical Techniques

Every time a pair of distance and bearing is returned from the distance server, the exact time (unit: millisecond) is recorded with it. After that, with two recorded pairs of distance and time, the velocity of the spot relative to the drone can be derived.

\[
R \overline{R} = \frac{d_0 - d_{-1}}{(t_0 - t_{-1})/1000} \text{ m/s},
\]

where \(d_0\) and \(t_0\) are the current distance and time, \(d_{-1}\) and \(t_{-1}\) are the previous ones.

With the relative velocity calculated, two types of information can further be calculated.

First, the future distance can be predicted. According to the official specification of the current drone model “DJI Phantom 4 Pro”, the delay of sending the camera frame from the drone to the mobile device is 220 milliseconds [9] and there is an additional delay for the distance server to return the distance result. To at least compensate for the delay of sending the camera frame, a prediction of the distance after 0.2 seconds is made.

\[
\overline{d} = (d_0 + \overline{v}_0 \times 0.2) \text{ m}
\]

Second, the actual velocity of the landing spot can be derived. For every 50ms, the actual velocity of the drone which is returned from its sensor is recorded. When the actual velocity of the landing spot is required, the velocity of the drone 0.3 seconds ago is retrieved (assume 0.2 seconds delay of sending camera frame and 0.1 seconds delay of returning distance result). Note that the relative velocity \(v_0\) should be 0.3 seconds ago under this assumption.

\[
\text{Actual Velocity of the Spot, } v_{\text{spot}} = v_{\text{drone}} + v_0
\]
As mentioned previously, a 3D coordinate system is formed using the information of distance and bearing, where the future distance predicted is used as input in this phase. The following is the definition of the 3D coordinate system:

Z: Z-axis points towards the front of the drone. Positive value means the spot is in front of the drone.

X: X-axis points towards the right of the drone. Positive value means the spot is on the right-hand side of the drone.

Y: Y-axis points towards the top of the drone. Positive value means the spot is on top of the drone, which is an impossible result in real life. Negative value means the spot is under the drone.

Figure 10: Custom 3D coordinate system used in this project

Simple Greedy Algorithm: Unsuccessful

In a simple greedy algorithm, the drone flies towards the spot as fast as possible until X and Z become nearly zero. However, it probably does not work in a real-life scenario. A fail case in the auto-landing system is defined as the situation when the drone cannot get stable on top of the spot or the drone completely lose track of the spot.

First, the momentum of the drone is high, it cannot stop effectively when its coordinates reach zero at full speed. It will lead to a pendulum motion where the drone continuously flies across the origin but never stay on top of it.

Second, the effect of delay in returning a distance result may cause the drone to fly too far away from the spot so that the drone is not able to look for the spot again without human help.

Due to concerns about precision, the system only predicts the distance 0.2 seconds later. Even
it can stop earlier when it flies across the spot, it is not significant enough if the drone is flying at full speed due to its high momentum and the delay which is slightly higher than 0.2 seconds.

Enhanced Greedy Algorithm

It is called an “enhanced” one because the velocity of the movement of the drone is highly dynamic and adaptive in this design. Two factors are affecting the velocity:

First, the velocity varies with the distance. It flies at full speed when the drone is far away. The closer the drone flies, the lower the velocity is. Eventually, the velocity becomes extremely low when the distance is less than 0.35m. This change can completely solve all the problems previously as the velocity is low enough for the drone to ignore the effect of momentum and delay.

\[ Velocity = Max(\text{Min}(\frac{\text{distance}}{\text{constant} \times \text{interpolation}}, v_{\text{upperLimit}}), v_{\text{lowerLimit}}) \]

In the current setting, the constant is set to be 250. The interpolation value is 0.65 when distance is higher than or equal to 60, 1 when distance is lower than or equal to 35.

This method can work perfectly if the spot is stationary. Nevertheless, the fact that the drone slows down becomes another problem if the spot is moving.

Second, the velocity of the drone is always greater than or equal to the velocity of the spot. As described previously, the actual velocity of the landing spot can be derived in some way. By adjusting the lower limit of the velocity, the drone can always catch up with the spot.

\[ \text{Final Velocity of the drone} = Max(v_{\text{drone}}, v_{\text{spot}}) \]

Criteria for landing

The criteria for landing are even more strict than that in phase one. For reference, in phase 1, a landing stage is achieved when the camera’s pitch angle keeps being more than 80° down for 1 second. The criteria were loose because the camera’s pitch cannot represent a horizontal position. The drone can land on the ground which is on the left or right away from the spot. Also, the drone is flying 1.5m above the ground and the paper box bus model is around 80cm tall. The camera’s pitch of 80° means 10° (which is 90° - 80°) and a maximum of ±12.15537
cm (which is 70cm×sin10°) tolerance of front and back distance. In other words, the drone has a 2×12 cm tolerance on the front (Z) axis and no limit to the horizontal distance for landing.

In phase 2, the criteria are based on the 3D coordinate system generated by the algorithm. There is a fixed distance threshold of ±5 cm on both front (Z) and right (X) axis. Therefore, it has only a 10 cm by 10 cm tolerance for landing. Once the drone stays in the 10×10 cm2 zone for 1 second long, a landing stage is achieved. And the drone flies downwards.

### 3.6 GPS based Automatic Flight

#### 3.6.1 Design

In the original plan, this function only aimed to correct the direction of flight when the flight operator controls the drone to fly incorrectly according to a planned path. It was only a kind of assisted manual flight.

Nonetheless, during the development, it was found that the system can be more than what was planned. It is upgraded to become a GPS-based automatic flight. It utilizes the interface of Google Map and the extremely precise GPS hardware device built-in the drone. Users can define and record a path before or during a flight. By clicking on the surface of the map, a marker is created as a waypoint of the path.

![Figure 11: Creation of waypoint Marker on Map](image)

18
By holding and dragging an existing marker, the flight operator can edit the path easily.

Additionally, if the marker is clicked, it is deleted. Note that deleting the first marker does not affect other markers’ indices.

After the drone finishes its trip, the flight operator can apply a reverse path onto the map. Therefore, the drone can make a return trip following the reversed path.
3.6.2 Approach

Mathematical Techniques

Similar to auto-landing, two data are essential: distance and bearing. As a marker is created based on the position of the Map, its GPS location (Latitude and Longitude) is defined with it too. By comparing the Latitude and Longitude of the drone and the marker, distance and bearing can be derived.

Consider the distance first:

\[
Radius \ of \ Earth, R = 6378.1370 \ km
\]
\[
latitude \ difference, d_{lat} = \text{lat}_2 - \text{lat}_1
\]
\[
longitude \ difference, d_{lon} = \text{lon}_2 - \text{lon}_1
\]
\[
Haversine, a = \left(\sin \frac{d_{lat}}{2}\right)^2 + \cos \text{lat}_1 \cos \text{lat}_2 + \left(\sin \frac{d_{lon}}{2}\right)^2
\]
\[
Variable \ c = 2 \times \text{atan2}(\sqrt{a}, \sqrt{1 - a})
\]
\[
Distance = R \times c
\]
Consider the bearing:

\[
\text{longitude difference, } d_{lon} = \text{lon}_2 - \text{lon}_1 \\
\]
\[
d_x = \sin d_{lon} \times \cos \text{lat}_2 \\
\]
\[
d_y = \cos \text{lat}_1 \times \sin \text{lat}_2 - \sin \text{lat}_1 \times \cos \text{lat}_2 \times \cos d_{lon} \\
\]

\[
\text{True North Bearing} = \text{atan2}(d_x, d_y) \\
\]
\[
\text{Relative Bearing} = \text{True North Bearing} - \text{Drone Yaw} \\
\]

With the distance and bearing calculated, the drone first turns its Yaw angle until the relative bearing is 0 degree (3 degrees tolerance). During the flight, whenever the relative bearing is not 0 degree, the drone immediately adjusts its Yaw angle. While the drone is not yet inside the green zone of the waypoint marker, the drone flies at high speed. Also, it flies at a low speed towards the center of the green zone while it is inside that zone.

**Marker Management**

To connect all the markers together, a linked list structure is applied. Therefore, deleting markers in-between becomes simpler. By simply changing the pointer of the marker before the target marker to point at the marker after the target marker, the deletion is completed. No swapping of array cells is needed. It is powerful especially when the list is big and many markers are stored in order.

By continuously calling the nextNode() function, all available markers (nodes) can be traversed without handling deleted or obsolete nodes. The simplicity of the traverse method facilitates some important features like updating all markers after a removed marker and drawing lines rapidly connecting all markers.

**3.7 Obstacle Avoidance Algorithm**

**3.7.1 Approach**

There are obstacle sensors on both the front and back side of the drone. While the drone is flying autonomously, its obstacle sensors keep sensing for obstacles. If an obstacle is detected by either side of the sensor, an interruption is called inside the Control Main Loop. In order to undergo the Obstacle Avoidance Algorithm, all action commands except the flight operator’s manual control are blocked.
The algorithm is described below:

First, the drone immediately turns 90° right to sense for spaces to move horizontally. There are in total 4 possible cases to happen.

1. No obstacles were detected in both ways.
2. Obstacle detected in the back side (originally left)
   
   For Case 1 and 2, the drone flies forward (originally right) at 1.5ms⁻¹ for 2 seconds.

3. Obstacle detected in the front side (originally right)
   
   For Case 3, the drone flies backward (originally left) at 1.5ms⁻¹ for 2 seconds.

4. Obstacle detected in both sides (originally left and right)
   
   For Case 4, completely terminate the autonomous flight and prompt the flight operator for manual control.

Then, the drone turns 90° back to the original direction.

Finally, if there are no more obstacles in front of the drone, continue the autonomous flight. Otherwise, repeat the first step until no more obstacles are detected.

Unfortunately, it is impractical to let the drone faces completely off and flies away from the landing spot while hoping it can still keep track on the spot. The drone will simply stay still and wait for the obstacle gone if there is an obstacle during auto-landing. This feature only applies to the GPS-based Automatic Flight. The drone is able to actively dodge obstacles by itself during the flight through the map.
4 Evaluation

4.1 Fully Customized Environment for Remote Controls

4.1.1 Results

**Complete remote control through Network**

All flight and camera control commands were successfully executed. Moreover, sensor readings and video streaming were successfully received.

![Image: The bridge connection between two mobile devices](image)

**Figure 15: The bridge connection between two mobile devices**

**Java-Python Integration**

Before checking the validity of the language integration using Flask Servers. The user interface became a serious problem. Auto Landing and Take-off and GPS-based Automatic Flight are the two major features of this system. There was too much content that needed to be displayed on the user interface. For instance, the value and output frame returned by the Distance server and the Bus server; The Google Maps interface for user input and location visualization; Also, the GPS and Satellite status of the drone. In light of that, the features were separated into two individual interfaces for the ease of use of the flight operator. There were two main user interfaces created: “Auto Landing Takeoff” and “GPS Control”
To switch from one interface to another. The flight operator can simply click on the built-in back button to go back to the list for feature selection and choose the needed one.

![Image of the two separated interfaces for different features](image)

**Figure 16: The two separated interfaces for different features**

Speaking of the language integration which is used in the Auto Landing and Take-off feature, both the distance estimation server and the bus display OCR server were responsive. They were able to return the information processed back to the controller app and also the output frame which contains the visual information extracted from the original camera frame.

**Asynchronous Processing and Control Main Loop**

As mentioned, there are three kinds of actions, namely General Autonomous Actions, Obstacle Avoidance Actions, and Manual Control Actions. When General Autonomous Actions were performed (from both Auto landing and GPS flight), green arrows were shown for visualizing the exact direction of a generated input.
When Obstacle Avoidance Actions were performed, green arrows pointing left and right were shown under both joysticks and on top of the already shown green arrows.

Figure 17: Automatically generated actions shown by green arrows

Figure 18: Obstacle Interruption shown by green arrows
During the auto-landing and GPS flight, the flight operator was always able to perform interruption and override all the actions automatically generated from the algorithms.

There are three cases to show the effectiveness of the Control Main Loop:

1. Joysticks are dragged by the flight controller during the auto-landing.
2. Joysticks are dragged by the flight controller during the GPS flight.
3. An obstacle is detected during the GPS flight (to be tested in the later stage)

The results have shown that the looping mechanism worked very effectively. Complete manual override was performed just by dragging the joysticks. All automatically generated actions were ignored.

Figure 19: The drone is going directly down as the joystick on the left is dragged downwards, in spite of the fact that the auto-landing algorithm has generated a Up and Forward command.
Figure 20: The drone is going directly right as the joystick on the right is dragged to the right, despite the GPS flight algorithm has generated a Forward command.

There is an important reason for creating a fast and direct way to perform interruption. When there is an unexpected error occurs in the autonomous algorithm, the flight operator should be able to perform complete manual control to handle the error so as to prevent accidents. The drone’s propellers are fast rotating objects which are dangerous to anything nearby it. Any accident can be disastrous.

Figure 21: Metal cans that were accidentally cut during testing of the drone
Additional Information

All green arrows were shown if the drone reached its desire locations in different algorithms. For example, the drone was directly above the landing spot during auto-landing; The drone was at the center of the green area under the marker during GPS flight.

4.1.2 Limitations

Communication delay is an obvious limitation of this system. As the control of the drone requires extremely long-range communication, the traditional controller is not suitable. We can only remotely control the drone over the Internet. Delays in both input and output signals of around 0.2 to 0.5 seconds are inevitable.

This situation heavily affects the degree of safety of the whole system. It was already an intense and difficult task to fly a drone normally, let alone flying with control delays.

It is a technological problem that cannot be fixed at least in the near future. However, there is a remedy for it. Additional algorithms are in development to let the drones avoid obstacles by themselves. Thus, the system will be safe and reliable in spite of delays.
4.2 Image Recognition System (Phase 1: Distance)

4.2.1 Results

When the pattern on the landing spot was shown in the drone’s camera image, the program successfully computed the relative spherical coordinates from the drone and the landing spot.

Under a local environment (directly feeding frames from local video files), the program performs extremely well. The average run-time per frame is 57.8ms (counted 100 frames), which is equivalent to 17.3 frames per second. The program is able to perform real-time distance tracking for the project. (The designed tracking rate for distance is 5 frames per second only)

4.2.2 Limitations

1. Detection range is limited to around 2.5 meters by the live stream resolution (720p).
2. Fails to verify QR text content if the QR code is too distorted in the received image.
4.3 Image Recognition System (Phase 2: Bus)

4.3.1 Results

Bus number and destination were detected and extracted within two seconds after the bus display appears inside the frame of the drone video stream.

Bus display that represented a wrong destination was ignored. When the correct destination was detected from the display, the drone immediately started its motors for takeoff.

Figure 25a: The drone stayed still after the wrong bus display was detected

Figure 25b: The drone started takeoff after the correct bus display was detected
4.3.2 Limitations

The rule-based text detection is not adaptive to a real-life scenario.

1. In real life, a bus display may not be exactly a black rectangular-shaped object.
2. Light reflection from the glass in front of the display may block the detection.
3. The glowing effect of the display itself may blur out the text in the display

Figure 26: Real-life bus photo
4.4 Auto Landing and Take-off (Phase 1: Stationary Spot)

4.4.1 Results

Figure 27: Auto landing with brute force algorithm
Currently, the algorithm only works on stationary landing spots. It took an average of 49.625 seconds to perform a successful landing at 80% successful rate.

### 4.4.2 Limitations

1. The algorithm is impractically slow. Especially when the drone lost track of the landing spot and had to fall back to search for the landing spot again.

2. Some area of the drone not landing properly on the spot and the drone fall off from the platform:

   When the drone flies downwards and gets close to the landing spot, the landing spot gets out of the frame of the camera. While the drone keeps flying downwards, the turbulence reflected from the platform makes the position of the drone extremely unstable and makes its final position slightly away from the target landing position.

Possible Fix: Change to use a bigger platform (e.g. 2m x 2m), to reduce the need for accuracy and improve safety. The current homemade bus model is too small (30cm wide) for fast auto-landing. With a bigger platform, greater moving steps can be applied to improve speed.
4.5 Auto Landing and Take-off (Phase 2: Moving Spot)

4.5.1 Results

A new paper box bus model was made for greater room for testing the auto-landing feature. It is 48cm wide, 67cm long, and 67cm tall.

![Figure 28: The new paper box bus model](image1)

A new and bigger bus model is not built for increasing the speed of the auto-landing algorithm. As mentioned, the criteria for achieving the landing stage are even more strict and difficult in phase 2. As long as the landing criteria do not get loose, enlarging the landing platform can never help improving the landing speed.

![Figure 29: Measurement of the paper box bus model](image2)
The true reason for making a larger platform is safety. During the testing in phase 1, when a fail case occurred, the drone did physically fell off from the paper box and does serious damage to the propellers of the drone. Even though enlarging the platform may affect the fairness of testing between two versions of the auto-landing algorithm, the safety of the drone and the environment are the most important. Therefore, enlarging the landing platform is inevitable.

![Drone fallen off](image)

Figure 30: The drone fell off from the paper box and resulted in a fail case

Two kinds of tests were performed on the phase 2 version of the auto-landing algorithm:
1. Landing on stationary platform
2. Landing on moving platform

**Landing on stationary platform**

To compare the performance between the two versions, the drone was tested to land on a stationary platform again even though it could follow a moving platform already.

Time taken for landing is derived from the moment the drone leaves the ground to the moment the drone touches the surface of the bus model.

\[
\Delta t_{\text{landing}} = t_{\text{land on bus}} - t_{\text{leave ground}}
\]

Time is recorded based on the timestamp of the video player. In the following example, the time taken for landing is 00:30 – 00:08 = 22 seconds.
Figure 31: The method of calculating landing duration
In a best-case scenario, the drone could even land within 15 seconds.

Figure 32: The best performance of the phase 2 version of auto landing
The results were surprisingly good. It took an average of 20.1 seconds to perform a successful landing at 100% successful rate. For reference, in phase 1, it was 49.625 seconds and 80%. It had more than double the speed than before. The time taken for landing is decreased by a significant 59.5%. However, the successful rate in the phase did not have any have reference value due to the change in the size of the landing platform.

Additionally, note that there was noticeable wind under the bridge where the testing environment locates. The drone had proven its stability against wind.
Landing on moving platform

Unfortunately, the landing on the actual surface of a moving platform was abandoned. It will be further explained in the limitation section.

As an alternative, the ability of the drone to keep flying on top of the moving landing spot is tested.

The estimate the speed of the moving platform, the bricks on the ground were used as the reference objects. As measured, the edge of every brick is 22.67cm long. It is estimated by measuring the square with my own hand and my own hand with a ruler.

\[
\text{Length of Edge of Square Bricks} = \frac{16}{12} \times 17 = 22.666 \ldots \approx 22.67 \text{cm}
\]

Figure 34: The length of an edge of the square bricks

By counting how many bricks have been passed in a certain duration, the speed of the moving platform can be estimated.

\[
\text{Speed of Moving Platform} = \frac{n_{\text{brick}} \times 0.22666 \ldots}{t_{\text{end}} - t_{\text{start}}}
\]
In the best-case scenario, the duration is 00:31 – 00:29 = 2 seconds. It passed 7 bricks in 2 seconds. Therefore the speed is \( \frac{7 \times 0.22666\ldots}{2} = 0.79333\ldots \approx 0.8 \text{ms}^{-1} \)

Figure 35: The method of calculating the speed of the platform
However, when the man pulling the platform (me) suddenly added up his speed to 4 bricks in a second ($\approx 0.9\text{ms}^{-1}$), the drone stayed at its location and stopped moving.

Figure 36: The drone stopped following the platform at to 4 bricks in a second
As the platform was pulled by a human being, its speed could not be adjusted precisely. The test results could only show that the drone can follow a moving platform up to $0.8\text{ms}^{-1}$. The critical speed limit is between $0.8\text{ms}^{-1}$ and $0.9\text{ms}^{-1}$.

If the landing spot moves within the speed limit, the drone can still keep track of the spot even the spot changes its direction of movement.

![Figure 37a: The drone can track on the platform while the platform is changing its direction](image)

Most importantly, the drone could still land on the spot when the platform finally stopped.

![Figure 37b: The drone lands successfully when the platform stops moving](image)

### 4.5.2 Limitations

**Abandonment of landing on the actual surface of a moving platform**

Due to two factors, the drone cannot land reliably within the limited area of the platform.

First, the distance tracking rate and the delay were too low and long respectively. Even though the algorithm was feasible in theory, the drone could never accurately predict the speed of the moving platform. It was a hardware limit and could not be fixed in the current stage.

Second, the goal of landing on this $48\text{cm}\times67\text{cm}$ moving platform was extremely dangerous. There was seemingly no room for failure and it had simply no tolerance of inaccuracy. Consequently, no trials have been done to make the drone land while the platform is moving.
Limited range of distance estimation

As mentioned, the resolution of the live-streamed video frame is limited to 720p which makes the image of the QR code unreadable at around 2.5m. The QR code can be printed larger to compensate for this problem. However, due to the limited area of the landing platform, a larger QR code cannot be bigger than the current size by a lot.

4.6 GPS based Automatic Flight

4.6.1 Results

The drone was able to fly across all markers using a similar amount of time with different weight settings of payload on it (up to 400g). For a fair test, a specific path was recorded inside the controller, so that the drone could fly according to the same path in different weight settings.

Figure 38: “Apply Path” button is pressed to recall the recorded path
Different weight settings

Originally, according to the plan, the goal of the project was to carry up to 300g of payload on the drone while having it flying properly. On top of it, a maximum of 500g of the payload was added to the drone.

As shown in the picture below, one weight block weighs 50g. A pair of it means 100g.

![Figure 39: The weight of the weight block](image)

The following shows the performance of the drone using different weight settings:

0. No weights

![Figure 40: The drone with no payload](image)

Results: Successful in 1 minute and 13 seconds
1. 100g weight

![Figure 41: The drone with 100g payload](image1)

Results: Successful in 1 minute and 20 seconds

2. 200g weight

![Figure 42: The drone with 200g payload](image2)

Results: Successful in 1 minute and 22 seconds
3. 300g weight

![Figure 43: The drone with 300g payload](image1)

Results: Successful in 1 minute and 17 seconds

4. 400g weight (heavier than planned)

![Figure 44: The drone with 400g payload](image2)

Results: Successful in 1 minute and 46 seconds
(Weather get cloudy during the test, GPS connection lost once)
5. 500g weight (heavier than planned)

Results: Aborted

Even though the drone was still able to fly at this weight but it flew in an unstable manner. It kept wobbling seriously especially there was some wind blowing against it. The momentum was so high too and it could not stop fast enough when the joystick was released. For safety reasons, the flight was aborted.
Figure 47: The heavily loaded drone flies in an unstable manner
Screenshots

0. No weights

Figure 48: The zero weights flight
1. 100g weight

Figure 49: The 100g weight flight
2. 200g weight

Figure 50: The 200g weight flight
3. 300g weight

Figure 51: The 300g weight flight
4. 400g weight

Figure 52: The 400g weight flight
4.6.2 Limitations

Weather is an important factor. If it is cloudy, the GPS signal can easily get lost. Like what happened during the 400g test.

![The drone lost GPS signal under a cloudy environment](image)

When the drone flies near a building, signal can get lost easily too.

4.7 Obstacle Avoidance Algorithm

4.7.1 Results

Since it was impossible to put obstacles for testing in the air during a GPS flight, the test was undergone inside the simulator.

![Simulator setting](image)
Simulate/Remove Obstacle buttons were added to block the drone's way during GPS flight.

The drone successfully braked for dodge action after the front obstacle button was pressed.
After braking, the drone turned right to use its front and back sensors to check for right and left obstacles respectively. All 4 possible cases of obstacle detection were simulated:

1. No obstacles were detected in both ways.

![Figure 57a: The drone went right if no obstacles were detected in both ways](image1)

The drone flies forward (originally right) at 1.5ms$^{-1}$ for 2 seconds. After that, it turned 90° back to the original direction and continued the flight.

![Figure 57b: The drone continued flying in its original direction](image2)
2. Obstacle detected in the front side (originally right)

The drone flies backward (originally left) at 1.5ms⁻¹ for 2 seconds. After that, it turned 90° back to the original direction and continued the flight.

Figure 58a: The drone went left if obstacles were detected on the right

Figure 58b: The drone continued flying in its original direction
3. Obstacle detected in the back side (originally left)

The drone flies forward (originally right) at 1.5\(\text{ms}^{-1}\) for 2 seconds. After that, it turned 90° back to the original direction and continued the flight.

Figure 59a: The drone went right if obstacles were detected on the left

Figure 59b: The drone continued flying in its original direction
4. Obstacle detected in both sides (originally left and right)

There was no room to dodge. The autonomous flight was completely terminated and the flight operator was prompted “The drone is stuck. Manual control is required” for manual control.

Figure 60: Warning when no room to dodge
5 Required Hardware and Software

5.1 Hardware

Master Devices: Samsung S20 Ultra (Android)
Bridge Device: LG G6 (Android)
Bus Display OCR Server: PC with Windows 10 installed
Drone: DJI Phantom 4 Pro
Homemade Moving Platform: A paper cardboard car with bus route number and landing spot printed on it

5.2 Software

Programming languages: JAVA (DJI Bridge, DJI API)
python (Computer Vision, OCR)

Remarks:

Due to budget issues, borrowing a drone from others became the only choice for this project. DJI Phantom 4 Pro was borrowed and chosen as the development drone eventually.

JAVA is the main programming language for this project as Android apps should be written in JAVA and the DJI API should be called using JAVA.

Python is used to perform Computer Vision and OCR as it is well-known for having a lot of well-developed libraries to facilitate a stable, simple, and efficient development on such topics [10].
6 Schedule

Below is the Gantt chart for the overall progress. (Green labels represent completed progress)

<table>
<thead>
<tr>
<th>Task</th>
<th>Sep-Oct</th>
<th>Oct-Nov</th>
<th>Nov-Dec</th>
<th>Dec-Jan</th>
<th>Jan-Feb</th>
<th>Feb-Mar</th>
<th>Mar-Apr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Develop environment for remote controls</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Develop image recognition system (recognize landing spot and compute coordinate)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Develop image recognition system (recognize bus route number)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Develop auto landing and take-off</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Develop obstacle avoidance algorithm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Develop path auto-correction program</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Develop GUI for flight operator</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test the homemade API</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test the image recognition system (phase 1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test the image recognition system (phase 2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test the auto landing and take-off</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test the obstacle avoidance algorithm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test the auto-correction program</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bug fixing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Final evaluation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Overall, there were no significant delay in progress. The schedule was successfully followed throughout the whole project.
7 Future Improvement

Automatic Bus Finder

In the current stage, the target bus to be detected by the auto take-off is manually input by the user. It can be improved by adding an automatic best bus route finder.

Positioning Hardware under low light

As the operating environment of this drone model can only detect obstacles with a surface with a clear pattern and adequate lighting (lux>15) [9]. Better hardware that can detect an object’s distance at low light can be developed

Grips on top of the Landing Spot

If the project is put into real life. Grips must be added to fix the drone onto the bus so as to prevent it from falling off during the ride.

8 Conclusion

All compulsory part of the project is completed. For example, network-based control, image processing system, python integration, auto takeoff, and auto landing, GPS flight, and obstacle avoidance.

However, the system still has room for optimization. Hope future development can help to improve the project.
9 Appendix

9.1 Mathematical Proof for Distance Estimation

Before calculating the distance from an image, the focal length of the camera should be measured.

<table>
<thead>
<tr>
<th>l</th>
<th>length&lt;sub&gt;physical&lt;/sub&gt; (l)</th>
<th>Pixels&lt;sub&gt;under720p&lt;/sub&gt; (p)</th>
<th>Distance&lt;sub&gt;physical&lt;/sub&gt; (d)</th>
<th>l/pd&lt;sub&gt;under720p&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data 1</td>
<td>11.5 cm</td>
<td>193.77 px</td>
<td>50 cm</td>
<td>0.001187 px&lt;sup&gt;-1&lt;/sup&gt;</td>
</tr>
<tr>
<td>Data 2</td>
<td>5 cm</td>
<td>84.98 px</td>
<td>50 cm</td>
<td>0.001177 px&lt;sup&gt;-1&lt;/sup&gt;</td>
</tr>
<tr>
<td>Data 3</td>
<td>16.3 cm</td>
<td>270.46 px</td>
<td>50 cm</td>
<td>0.001205 px&lt;sup&gt;-1&lt;/sup&gt;</td>
</tr>
<tr>
<td>Data 4</td>
<td>7.75 cm</td>
<td>135.38 px</td>
<td>50 cm</td>
<td>0.001145 px&lt;sup&gt;-1&lt;/sup&gt;</td>
</tr>
<tr>
<td>Data 5</td>
<td>12.4 cm</td>
<td>223.31 px</td>
<td>50 cm</td>
<td>0.001106 px&lt;sup&gt;-1&lt;/sup&gt;</td>
</tr>
<tr>
<td>Data 6</td>
<td>5.5 cm</td>
<td>105.33 px</td>
<td>50 cm</td>
<td>0.001044 px&lt;sup&gt;-1&lt;/sup&gt;</td>
</tr>
<tr>
<td>Data 7</td>
<td>3.8 cm</td>
<td>62.51 px</td>
<td>50 cm</td>
<td>0.001216 px&lt;sup&gt;-1&lt;/sup&gt;</td>
</tr>
<tr>
<td>Data 8</td>
<td>9.95 cm</td>
<td>173.88 px</td>
<td>50 cm</td>
<td>0.001144 px&lt;sup&gt;-1&lt;/sup&gt;</td>
</tr>
<tr>
<td>Average</td>
<td>-</td>
<td>-</td>
<td>50 cm</td>
<td>0.001153 px&lt;sup&gt;-1&lt;/sup&gt;</td>
</tr>
</tbody>
</table>

Table 4: Camera Measurement data of DJI Phantom 4 Pro

Equivalent Focal Length (35 mm film) = \( \frac{\sqrt{(36)^2+(24)^2}}{2 \times \tan 80.56^\circ} \approx 25.6mm \approx 24mm \) (official data)

Also, 6 lengths are known: AB = AC = BD = CD = 11.5 cm (measured in real life); BC = AD = 11.5√2 cm (Pyth. theorem)

Point P(x, y, z): ((x of P<sub>2D</sub> - x of image center), (y of P<sub>2D</sub>-y of image center), unknown)

\[
\text{Angle } P = \text{Angle from z-axis} = \tan^{-1}(0.001153 \times \sqrt{x^2 + y^2})
\]

\[
\text{Roll } P = \tan^{-1} \frac{x}{y}
\]

\[
\text{Angle } PVQ = \text{Angle between Point P and Q} = \cos^{-1} \left( \frac{\cos Q}{2 \cos P} + \frac{\cos P}{2 \cos Q} - \frac{\sin P \cos Q \tan P}{Z} - \frac{\sin Q \cos P \tan Q}{Z} + (\sin P \sin Q \cos(\text{roll } Q - \text{roll } P)) \right)
\]

\[
\text{pxPQ} = \text{pixel distance of PQ} = \sqrt{(x \text{ of } P_{2D} - x \text{ of } Q_{2D})^2 + (y \text{ of } P_{2D} - y \text{ of } Q_{2D})^2}
\]

Figure 61: Coordinate system that the calculations use
Figure 62: 3D world to 2D image conversion
After estimating the fourth corner and center (Step 4), 6 points can be extracted from the image: \(V_{2D}\), the center of the view and the location of the viewpoint; \(A_{2D}, B_{2D}, C_{2D},\) and \(D_{2D}\): the location of all 4 corners in the image; \(O_{2D}\): the location of the QR code center in the image.

From the figure, there is a pyramid \(VABCD\). It can be separated into two triangular pyramids \(VABC\) and \(VBCD\). Let \(V'\) = \(V'BC'\) = \(V'D'\), as shown in the figure, so that triangle \(V'BC'\), \(V'CD'\), \(V'CD'\) and \(V'B'D'\) are all isosceles triangles.

Consider triangular pyramid \(VABC\) first:
Let \(VA = VA'\), \(VB = bVB'\) and \(VC = cVC'\),
where \(AB = AC = BC / \sqrt{2} = r_{AB}A'B' = r_{AC}A'C' = r_{BC}B'C' / \sqrt{2}\)  (\(ABC\) is a right-angled triangle)
So that the pyramid formed by lines that have length \(VA', bVB', cVC', r_{AB}A'B', r_{AC}A'C'\) and \(r_{BC}B'C'\) is exactly equal to the triangular pyramid \(VABC\).

Applying Cosine Law twice:
\[ A'C'^2 = VA'^2 + VC'^2 - 2(VA')(VC')(\cos AVC) \]
\[ \frac{VA'^2}{A'C'^2} = \frac{1}{2(1-\cos AVC)} \]
\[ r_{AC}^2 = VA'^2 + c^2 VC'^2 - 2(VA')(cVC') \cos AVC \]
\[ r_{AC}^2 = \frac{VA'^2}{A'C'^2}(1 + c^2 - 2c \cos AVC) \]
\[ r_{AC} = \sqrt{\frac{1 + c^2 - 2c \cos AVC}{2(1-\cos AVC)}} \]

When \( r_{AB}A'B' = r_{AC}A'C' \), \( r_{AB} = \frac{pxAC}{pxAB} r_{AC} \sqrt{\frac{1 + b^2 - 2b \cos AVB}{2(1-\cos AVB)}} = \frac{pxAC}{pxAB} \sqrt{\frac{1 + c^2 - 2c \cos AVC}{2(1-\cos AVC)}} \)

For simplicity, all constant terms are combined,
Let \( C1 = \frac{(pxAC)^2(1-\cos AVB)}{(pxAB)^2(1-\cos AVC)} \); \( C2 = \frac{2(pxAC)^2(1-\cos BV C)}{(pxBC)^2(1-\cos AVC)} \); \( C3 = \frac{2(pxAB)^2(1-\cos BV C)}{(pxBC)^2(1-\cos AVC)} \);
\[ b^2 - 2 \cos AVB \ b + 1 = C1(c^2 - 2c \cos AVC + 1) \]
\[ b^2 - 2 \cos AVB \ b + (1 - C1 c^2 + 2 C1 c \cos AVC - C1) = 0 \]
\[ b = \cos AVB \pm \sqrt{C1(c^2 - 2c \cos AVC + 1) - sin^2 AVB} \]
Applying a similar method on line B'C' and A'C',

When \( r_{BC}B'C' = \sqrt{2}r_{AC}A'C' \), \( \frac{b^2 + c^2 - 2bc \cos BVC}{2(1 - \cos BVC)} = \frac{\sqrt{2}pxAC}{pxBC} \sqrt{\frac{1 + c^2 - 2c \cos AVC}{2(1 - \cos AVC)}} \)

\((b^2 + c^2 - 2bc \cos BVC) - C2(1 + c^2 - 2c \cos AVC) = 0\)

(Note that b has two possible expressions)

Clearly, no analytical solutions can be derived from the above equation. However, roots can be found by substituting the value of c from 0.001 to 2.000. When the value of the left-hand side of the equation touches/passes 0 (x-axis), that pair of b and c is one of the roots. And we should always have multiple roots.

To figure the correct pair of b and c, another triangular pyramid VBCD should be considered: Let \( VD = dVD' \); \( VO = oVA' \); (O is the center of the QR code; VO is the estimated distance)

By using \( r_{CB} = r_{BC} = \sqrt{c^2 + b^2 - 2cb \cos BVC} \),

\( \cos VCB = \frac{r_{CB}^2 + c^2 - b^2}{2cr_{CB}} \)

\( o^2 = c^2 + \left(\frac{r_{CB}}{2}\right)^2 - 2c \left(\frac{r_{CB}}{2}\right) \cos VCB = c^2 + \left(\frac{r_{CB}}{2}\right)^2 - cr_{CB} \frac{r_{CB}^2 + c^2 - b^2}{2cr_{CB}} \)

\( o = \sqrt{c^2 + \frac{r_{CB}^2 - r_{CB}^2 + c^2 - b^2}{4}} \)

\( o = \sqrt{c^2 + \frac{c^2 + b^2 - 2cb \cos BVC - 2c^2 - 2cb \cos BVC}{4}} \) //IMPORTANT

By using \( r_{AD} = \sqrt{1 + d^2 - 2d \cos AVD} \),

\( d^2 = r_{AD}^2 + 1 - 2r_{AD} \cos VAD \); \( o^2 = \frac{r_{AD}^2}{4} + 1 - r_{AD} \cos VAD \)

\( \cos VAD = \frac{r_{AD}^2 + 1 - d^2}{2r_{AD}} = \frac{r_{AD}^2 + 1 - o^2}{r_{AD}} \)

\( d^2 = \frac{r_{AD}^2}{2} - 1 + 2o^2 \)

\( d^2 = \frac{1 + d^2 - 2d \cos AVD}{2} - 1 + 2o^2 \)

\( d^2 + (2 \cos AVD)d + (1 - 4o^2) = 0 \)

\( d = \frac{-2\cos AVD \pm \sqrt{-4(\cos AVD)^2 - 4 + 16o^2}}{2} \)

\( d = -\cos AVD \pm \sqrt{4o^2 - (\sin AVD)^2} \), if \( \Delta \geq 0 \)
For each pair of b and c, we calculate its corresponding ds. So we can generate a list of proposed sets of b, c, and d.

\[ r_{BD} = \sqrt{\frac{b^2 + d^2 - 2bd \cos BVD}{2(1 - \cos BVD)}}; \quad r_{CD} = \sqrt{\frac{c^2 + d^2 - 2cd \cos CVD}{2(1 - \cos CVD)}}; \quad r_{AD} = \sqrt{\frac{1 + d^2 - 2d \cos AVD}{2(1 - \cos AVD)}}; \]

By calculating BD = r_{BD}B’D’, CD = r_{CD}C’D’, AD = r_{AD}A’D’ and etc, each set of b, c, and d can form a quadrilateral ABCD (where ABCD is formed by two triangles ABC and BCD). Therefore, we have the estimated real-world lengths of all edges connected by ABCD.

There are two major factors that show whether ABCD is similar to a square. First, the percentage difference between sides AB, BD, CD, AC. Second, the percentage difference between diagonals AD and BC. The overall percentage difference is the average between these two percentage differences. The lower the percentage difference is, the higher the likeness between ABCD and a square is.

If the quadrilateral ABCD formed by a set of b, c, and d is most similar to a square (has lowest percentage difference), that pair of b and c is considered as the correct root.

\[
VA = \frac{11.5}{\sqrt{1 + b^2 - 2b \cos AVB}} = \frac{11.5}{\sqrt{1 + c^2 - 2c \cos AVC}} = \frac{11.5\sqrt{2}}{\sqrt{b^2 + c^2 - 2bc \cos BVC}}
\]

\[Estimated \ Distance = VO = a \times \text{average} \ VA\]
10 Reference


