Swarm of Drones Choreography

Formation and Shape Control

Project Report

By Oz Arya and Arnold Cheskis

Supervised by Ofer Danino

Project beginning: Spring Semester 2023

Project submission: February 2024

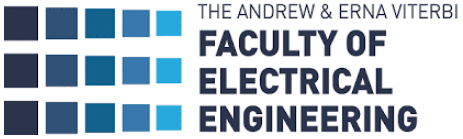

Table of Contents

Abstract

In intricate situations, where human hands cannot reach, or a scan over a large area is required, by connecting multiple drones to the same network we can then harness the capabilities of artificial intelligence to handle complex tasks and fulfill them successfully.

The goal of our project is to synchronize a given number of drones, which communicate through a UDP network, by using artificial intelligence that instructs them move in a given closed path (e.g., a circle, triangle, etc.). This interface receives an image of a closed path as an input, and makes the drones move on it without collision. This can be used to scan an area or in rescue missions.

Introduction

A drone is a flying piece of hardware that can be wirelessly controlled. The drone consists of 4 or more motors, each attached to a propeller. The ability to control the speed of each propeller individually grants the drone the ability to move in 3D with closer to no restrictions. Nowadays, it is common for drones to have a gyroscope, a GPS, a camera, and a chip (such as Bluetooth) that enables it to communicate with a computer or other drones wirelessly.

These features allow the drones to act autonomously, and their usage have been accelerating exponentially since 2015[1].

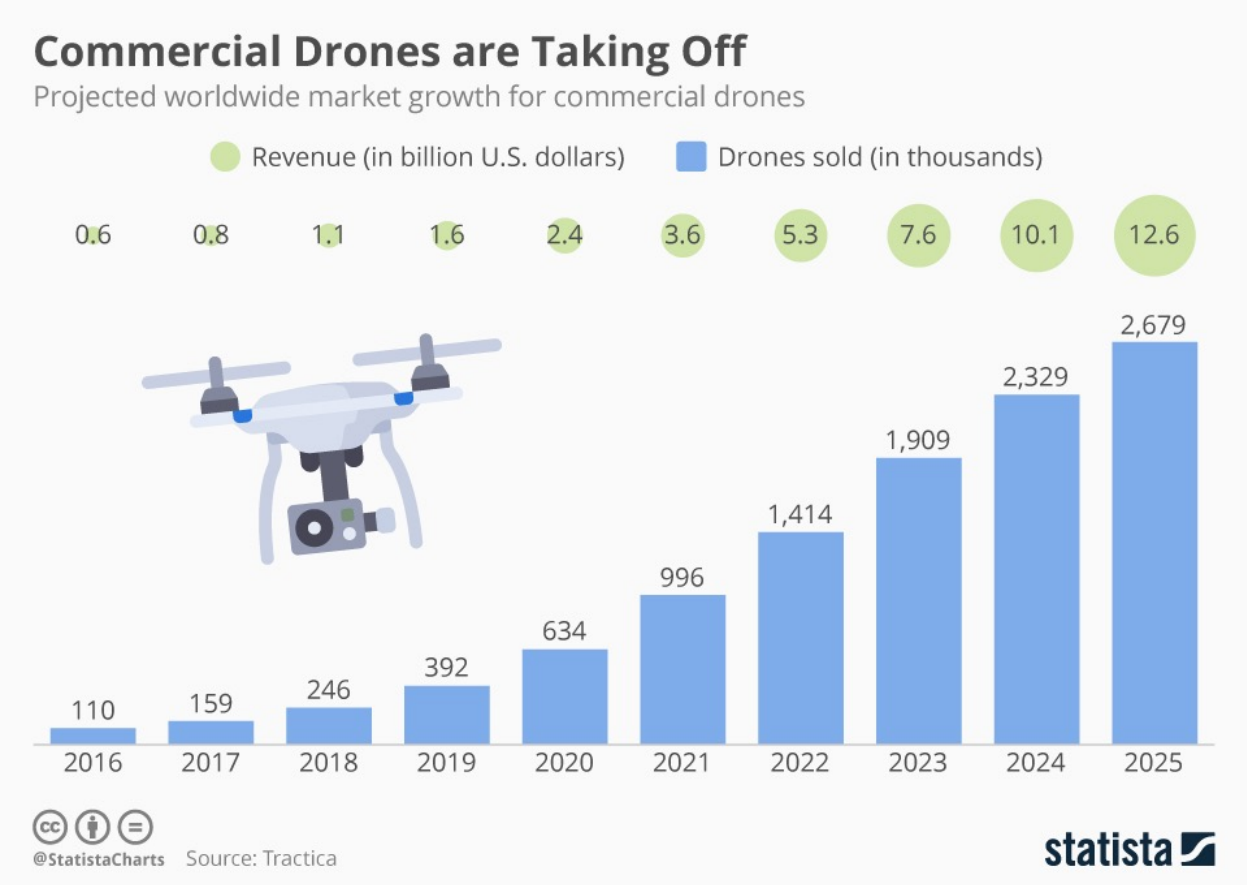

Figure 1: Drones usage growth since 2016. Image from Statista, 2019.

When it comes to drones, the most useful and significant utilization of their capabilities applies when operating a swarm of drones. Swarms are able to cover vast areas and tremendously increase their potential and accumulated power to perform tasks that only a few decades ago would be considered imaginary. Examples of such applications could be seen in the realm of entertainment on October 2022 in Shenzhen, China, during the boat festival[2], or during the 4th of July in 2023, breaking a Guiness record[3]. Furthermore, the autonomous potential swarms of drone have can be seen in security, since the increased number of drones can be used to survey a large area[4]. An even further step can be seen in their utilization in agriculture, where it even can save a country in times of war, as is evident in the Russia-Ukraine war[5] for instance.

Figure 2: examples of drone's swarm usage. The first image taken from the Boat festival at Shenzhen[2], second is an example of a Guinness breaking record[3], below an illustration of their usage in security to survey large areas, and lastly an image of their agriculture usage provided my MIT Technology Review.

However, when working with multiple drones, a lot of factors must be considered, such as avoiding collisions, timing the tasks each drone is responsible for, etc. Since the drones often do not have a powerful CPU, they do not possess large computational powers that are required to run AI. Therefore, we used a centralized architecture where a main command unit, such as a server or a computer, is used to send the commands to

the drones. The server runs the algorithm that receives data from the drones, makes the heavy computations, and outputs the movement commands back to the drones. Next, we need to consider the stability. Since drones are inherently not stable even a small gust of wind might make them stray from the path which compounds into large errors. To improve the stability, we need to constantly correct the drones' movements. Essentially, the functionality of the drones in this project can be summarized by receiving commands, broadcasting the image, and flying in any of the 6 perpendicular directions (up, down, forward, back, left, and right). The central command unit in our case will consist of a computer and a server. The computer will be responsible for analyzing the images from the drones, command each drone when to broadcast the image and when to fly, calculate the corrected flight pattern, and communicate with the app on the cloud that is the user interface. The server will host the app, in our case robo-phone.com, which will provide the user with an interface to input the initial pattern in the form of an image, either by upload or by a selection of existing images from the database. Together, the goal of the system is to provide the user with powerful tools to perform complicated operations with drones with the need to write a single line of code. This software can be added as an additional tool as part of a larger toolkit.

The drones we will be using are called Edu-Tello, used mostly for educational purposes, and created by the Tello company. Each drone is equipped with a camera, 4 propellers, and a WIFI chip we will utilize for communication with the server.

System overview

We attempt to create a system with a feedback controller. The system consists of Edu-Tello drones, a router, a server, a green squared paper, and a computer [Figure 4]. The drones are connected to the same Wi-Fi network through a UDP connection. The computer is the "brain" of the system which communicates with the user interface in the cloud, communicates with the drones, and computes the course and corrects the drone's movement when it strays from the path using basic geometrical calculations. The user communicates with the system through the cloud using the app Robo-phone, which utilizes the Firebase interface, where one can either upload the image with the closed path or choose an existing one from the server.

We can divide our system into 3 levels.

Lowest level (Hardware and network connectivity): The drones are connected to the computer through a UDP connection via the router, where each drone operates as a station that receives messages. Through a different port, each station (drone) streams the video to the computer in an H265 format when it gets the appropriate signal.

```
def set_ap(ssid, password, address):
    """
    A Function to set tello in Access Point (AP) mode.

    :param ssid: The SSID of the network (e.g. name of the Wi-Fi).
    :param password: The password of the network.
    :param address: Tello IP.
    :return: None.
    """
    s = get_socket()

    cmd = 'command'
    print(f'sending cmd {cmd}')
    s.sendto(cmd.encode('utf-8'), address)

    response, ip = s.recvfrom(100)
    print(f'from {ip}: {response}')

    cmd = f'ap {ssid} {password}'
    print(f'sending cmd {cmd}')
    s.sendto(cmd.encode('utf-8'), address)

    response, ip = s.recvfrom(100)
    print(f'from {ip}: {response}')
```

Figure 3: The establishment of the UDP connection requires us to change the drone's mode, which

later helps us to connect the drone to the Wi-Fi. To do so we change the drone's mode to "access point" (AP), making the drone act like a station instead of creating its own Wi-Fi. As we can see in this figure we first ping the drone by sending a "command" command, followed by the command "ap <network name> <password>" which is sent to the IP address that is found in the variable "address".

Functional level (path calculation): Using the image received from Firebase we created the Drone class, as shown in figure 4, in order to monitor and update the drone's position and velocity. Whenever we receive from the drone the image stream, at first, we attempted to use AI to calculate its new position and update the coordinates it should go to next, however, the AI was not running quickly enough and therefore we resorted to a simpler algorithm that detects adjacent green pixels with a shape of a square. For convenience, in future references we shall name this algorithm green square detector (GSD). Finally, we send the revised command to the drones, after the algorithm has determined the corrected path, the drones should take. This step closes the circle and creates a feedback controller.

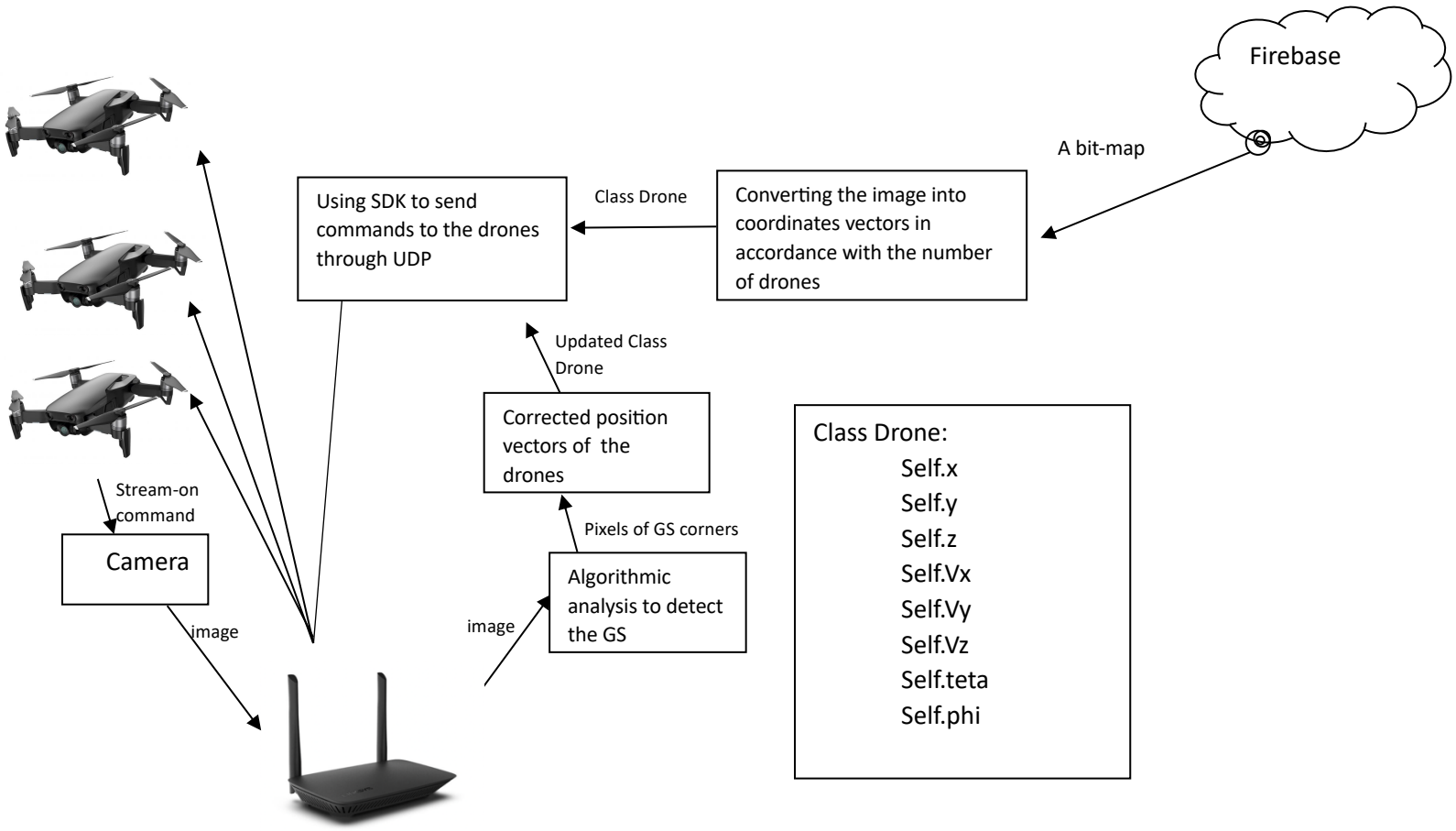

Figure 4: The complete outlines of the system; starting from the user, who provides the input via Firebase, then converting it to coordinate vectors that determine the possible locations of the drones, i.e. the path. Next, we build for each drone a class named Drone which contains the next position the drone should go to, such as velocity and position. Class drone is used to know what values to send via the SDK to the drones, e.g. where to move. Then the drones stream back via the router what they see, the GSD recognizes the GS, and the software updates class drone accordingly.

Highest level (cloud): The cloud which we communicate with through Firebase, holds the possible paths the drone can take, e.g. triangle, circle, etc. The user may choose one of them or upload a new closed path. Then, the cloud sends the start command to the functional level software along with the image.

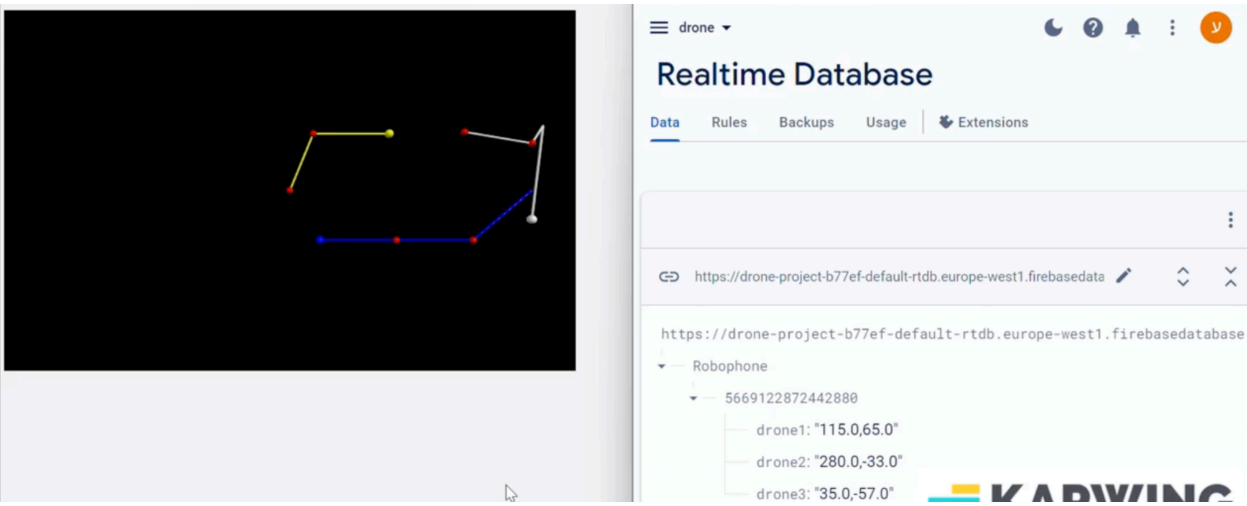

Figure 5: We send the drones' positions to Firebase, i.e. the cloud, as can be seen on the right, and create a visualization of their movement pattern and the correction that are made using Vpython as can be seen on the left.

Tools and environment

Robo-phone

An IoT based application which provides an interface for building and running programs on hardware such as robots, or in our case drones. It enables us to both create an abstraction for building complex code for hardware in a user-friendly environment and establish a more convenient communication channel between different devices and the server. We utilized it as a distributed operating system for our project to connect between the server, aka the cloud, and the device that runs the app to control the drones.

Firebase

Google's product that allows us to store and manage live data on their servers, which we refer to as "the cloud". Using Firebase's API, Robo-phone is able to establish a communication channel between different devices.

Vpython

A python library that provides a graphic visualization of certain processes. In this project it was utilizes to monitor the movement of the drones, provide feedback, and to simulate the movement for code testing purposes.

Packet Sender

An interface for sending packets between different station in the same network. In this project it was utilized to send packets from the PC to the drones to test the initial versions of the code and the drones' responsiveness.

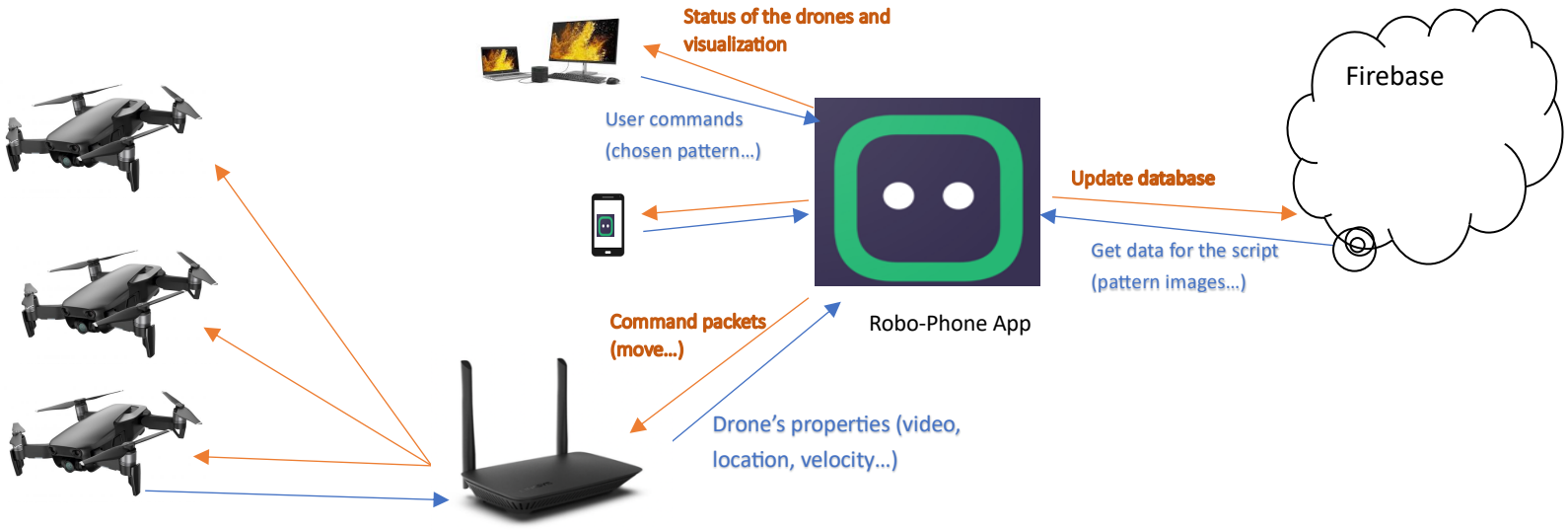

Figure 4.1: An abstraction of the system shown in figure 4, by using the Robo-phone app as an operating system that adds a high-level interface for the users and wraps the script which communicates with the cloud and the drones.

Different solution approaches

Initially we considered using an external camera as part of the feedback controller, instead of using the drones' cameras. However, our goal is to

create an interface that would be easy to reproduce and convenient to the user. Adding another external camera will increase to complexity to set up the system.

Another option we tried was to avoid the usage of the drones' cameras completely and rely solely on the other sensors, such as the gyroscope. The problem with this method appears to be lacking precision when the change in the drone's position is too big. The error we got upon calculating the location of the drones in such case was severe.

We considered connecting each drone to a different computer, and build on top another CRML project we found, however, that would be impractical for people to recreate, and our goal is to build a system that would be as easy to use as possible for the user.

Unveiling the Inner Workings of Our Innovation

Binary image to shape:

When we first receive the bit map from firebase, we need to convert it into the path the drones should follow. While it is not necessary, we would like to check first if the image is a closed path, since we always want to give them the ability to perform numerous iterations of it. Therefore, if the curve is not closed we return false. Ideally, the drones should be traversing a continues path, but in reality, the drones fly in a straight or a curved line from point A to B. Therefore, to give a drone the command to follow a given close path, we must mark N spots on it, and each time a drone reaches one of these spots, we instruct it to go to the next one. On the one hand, we need to consider the maneuvering abilities of the drone and make the distance between every 2 spots big enough; for Edu-Tello we set it to be at least 0.5m for proper functionality. On the other hand, the whole shape needs to be small enough to fit in a confined space. Thus, we allow the user to configure the size of the path, and the number of spots (N > number of drones) is then calculated by our program, as shown in figures 6-7.

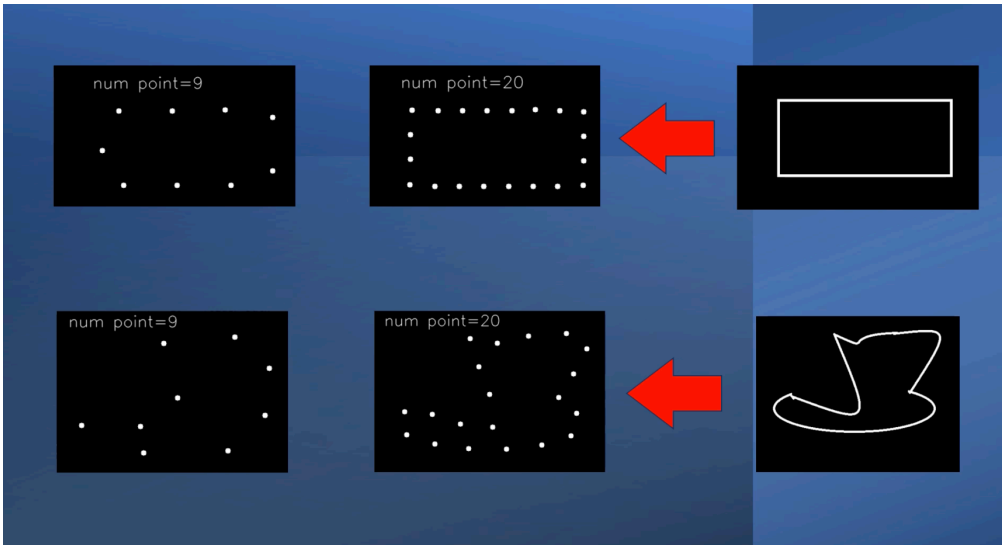

Figure 6: 2 examples of conversion from an image of a continues shape to a discrete structure which can be represented as a finite system of coordinates.

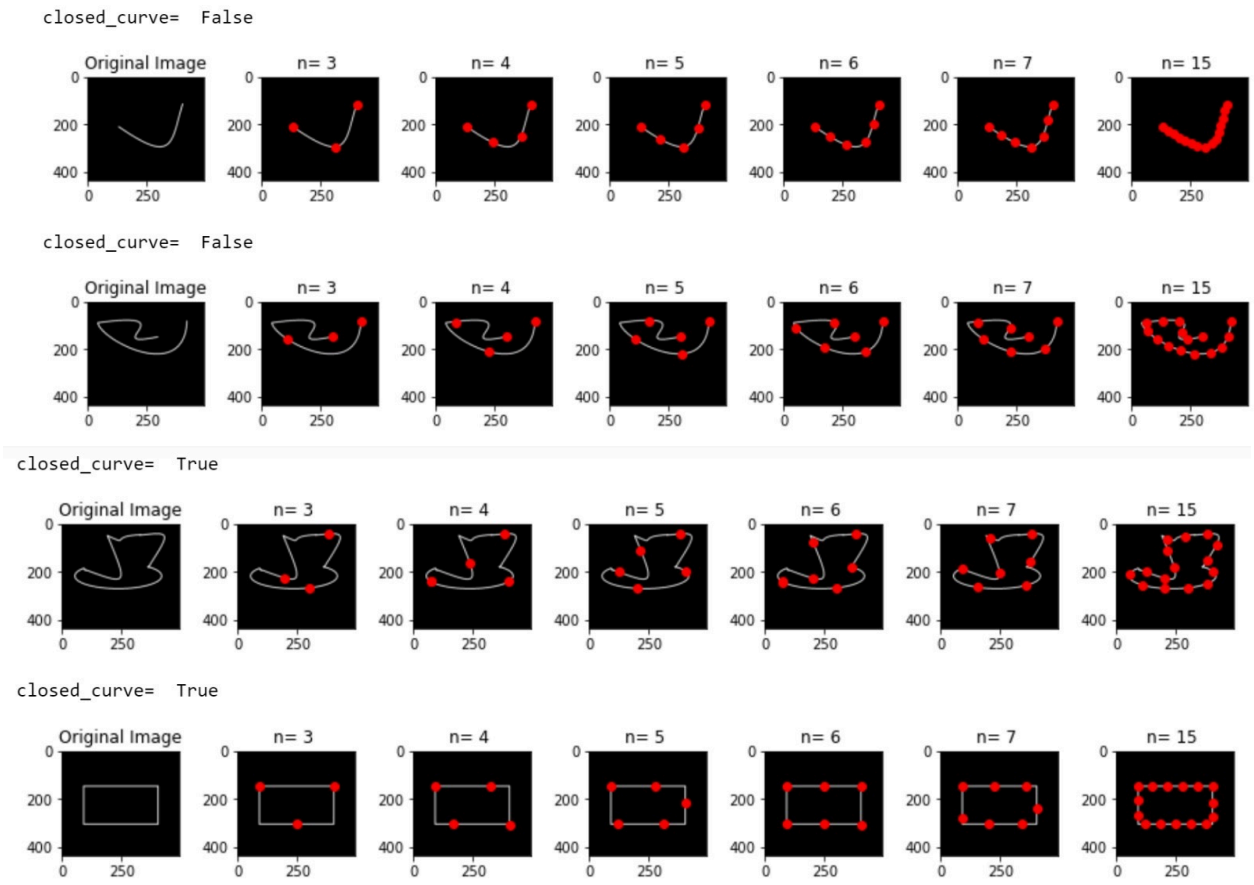

Figure 7: Shows several examples of open and closed curves (we only accepted closed curves in our case), and the number (n) of spots we choose, such that $n = \min\{\max\{num\_drones, \min\_spots\_distance\}, max\_shape\_size\}$

Communication and integration:

At first, we considered using TCP instead of UDP since it is more secure, however, we were restricted by the SDK of the Edu-Tello Drone. In addition, UDP is faster which is crucial since the bandwidth of the router is our bottleneck when we stream the images from multiple drones. By default, each drone has uses direct Wi-Fi (peer-to-peer) to connect to the computer, but to communicate with multiple drones, we need to change the mode of each one to an access point (AP) and connect them to a router, and turn the router into a bridge. This was the biggest challenge of this part, since it was not clear whether the bugs we encountered were

due to a faulty hardware or an unsupported script. After consulting with the Ryzo (that manages the Tello drones) support, a moderator from git that has an SDK repository, and countless forums, we succeeded in changing the router to a bridge and establishing the connection with several drones at once. The next struggle was with the length of the bandwidth, which limited us from streaming videos from several drones at once, as we will see in part c.

Feedback through image recognition

To make the corrections, the drones stream the images they are seeing in front of them when pinged. The main challenge at this stage was to obtain and decode the messages so that the GSD would be able to recognize the green square (GS) from the drone's camera and do it rapidly enough to correct the coordinates before we send the next command to the drone. Since once the drone begins its movement, it cannot change the coordinates mid-flight. The messages was H256-encoded. We tried several methods to decode it, one of each uses several libraries as we described in the guideline's sections. The solution presented in our code uses a class that gets the current frame (stream), using the thread library, and reads frames using PyAV (av library) and cv2 library [7][8].

Next, once we have the image in png format, as previously mentioned we attempted to use a pre-trained AI model to recognize the pixels (position) of the GS, but instead built the GSD algorithm. From it we could infer the position of the drone and correct it as we will see in the next section.

```python
def ordered_rictangle_points(points):
    # output: [top_left, top_right, bottom_right, bottom_left]
    # sort the points by x
    points = points[np.argsort(points[:, 0]), :]
    # sort the points by y
    top_points = points[:2, :]
    bottom_points = points[2:, :]
    top_points = top_points[np.argsort(top_points[:, 1]), :]
    bottom_points = bottom_points[np.argsort(bottom_points[:, 1]), :]
    # top_left, bottom_left,bottom_right, top_right
    return np.array([top_points[0], top_points[1], bottom_points[1], bottom_points[0]], dtype=np.float32)

def find_rectangles(binary_image,ratio):
    # Find contours in the binary image
    contours, hierarchy = cv2.findContours(binary_image, cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMPLE)

    # Filter out contours with internal contours (holes)
    contours_with_holes = []
    for i, contour in enumerate(contours):
        # Check if the contour has a child contour
        if hierarchy[0][i][2] != -1:
            contours_with_holes.append(contour)
    if len(contours_with_holes)==0:
        return None

    for contour in contours_with_holes:
        # Approximate the contour to a polygon
        epsilon = 0.02 * cv2.arcLength(contour, True)
        approx_polygon = cv2.approxPolyDP(contour, epsilon, True)

        # If the contour is a rectangle (4 sides), extract its corner coordinates
        min_score=1000
        if len(approx_polygon) == 4:
            max_x = np.max(approx_polygon[:, 0, 0])
            min_x = np.min(approx_polygon[:, 0, 0])
            max_y = np.max(approx_polygon[:, 0, 1])
            min_y = np.min(approx_polygon[:, 0, 1])
            # find the ratio btween the high and width
            high = max_y-min_y+0.0000001
            width = max_x-min_x+0.0000001

            ratio_poligon = abs(width/high)
            score=abs(ratio-ratio_poligon)
            if score<min_score:
                min_score=score
                min_rectangle=approx_polygon.reshape(-1, 2)
    if min_score>4:
        return None
    return min_rectangle
```

Figure 8: 2 of the functions that locate the GS by analyzing the drone's image.

```python
def calculate_camera_position(intrinsic_matrix, image_points, real_height,real_width):
    top_left, top_right, bottom_right, bottom_left = image_points
    width1 = np.linalg.norm(top_right - top_left)
    width2 = np.linalg.norm(bottom_right - bottom_left)
    img_width = (width1 + width2) / 2
    height1 = np.linalg.norm(top_right - bottom_right)
    height2 = np.linalg.norm(top_left - bottom_left)
    img_height = (height1 + height2) / 2



    ratio_img_world = ((real_width / img_width)+ (real_height / img_height))/2

    fx= intrinsic_matrix[0][0]
    fy= intrinsic_matrix[1][1]
    py=intrinsic_matrix[0][2]
    pz=intrinsic_matrix[1][2]

    dis_x1=fx*real_width/img_width
    dis_x2=fy*real_height/img_height
    dis_x=(dis_x1+dis_x2)/2

    dis_y=(py-top_left[0])*ratio_img_world
    dis_z=(pz-top_left[1])*ratio_img_world

    print(f"dis_x={dis_x}, dis_y={dis_y}, dis_z={dis_z}")

    return np.array([dis_x, dis_y, dis_z], dtype=np.float32)

#return drone's location
def find_location(real_width,real_high,img):

    intrinsic_matrix = np.array([[9.105807265540521e+02, 0, 4.574269390215023e+02],
                                 [0, 9.139776296246570e+02, 3.343168681518988e+02],
                                 [0, 0, 1]], dtype=np.float32)

    ratio=real_width/real_high

    squre_point=find_green_rectangle_points(img,ratio)
    if squre_point is not None:
        top_left, bottom_left,bottom_right, top_right  = ordered_rictangle_points(squre_point)
        img_points = np.array([top_left, top_right, bottom_right, bottom_left], dtype=np.float32)

        camera_position = calculate_camera_position(intrinsic_matrix, img_points, real_high ,real_width)

    else:
        camera_position=None
    return camera_position
```

Figure 9: 2 functions that help us find the location of the drone using intrinsic properties of the cameras and the output from the function from figure 8.

Path correction

All the drones have a joint X, Y and Z axis. Then the drone records the green square (GS) and its position, that we assume to be fixed perpetually in our coordinates system with the center of the GS having the constant coordinates $(0,0,Z_0)$. By using the process we will see in figure 10, by only seeing the image of the GS we can uncover the exact position of the drone.

We assume the GS or part of it is always observable by the drones, and that they are positioned on the same line in the beginning of the movement, adjacent to one another. Initially the computer pings each drone to check connectivity and battery level and then broadcast a takeoff command to all of them.

Then one after the other the computer instructs the drones to move into the initial coordinates, which are the beginning of the path, using the Edu-Tello SDK. Knowing the actual coordinates the drone should be in, using the method described in figure 10, and by figuring out its actual coordinates, we can calculate the next corrected SDK command we need to send to the drone in order for it to return to the correct path.

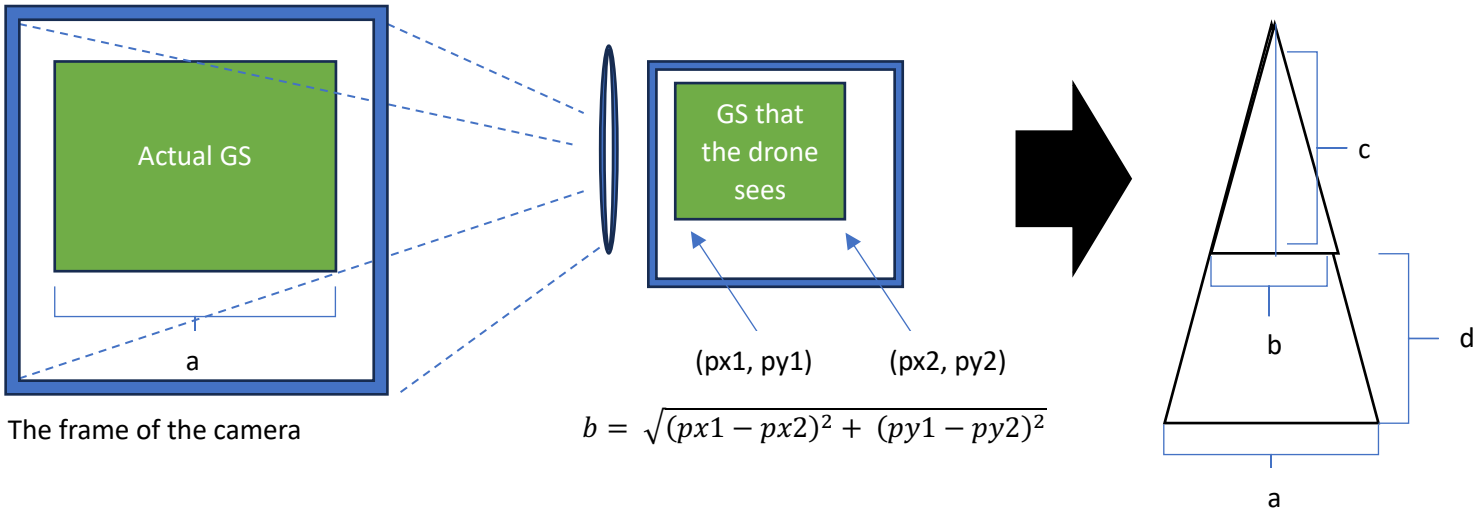

Figure 10: we know in advance, b is the width of the square as seen by the lens. The image recognition from the previous step will give us px1, px2, py1, py2, (or in short "pxy") from which we can establish b. Then c is determined by the internal properties of the cameras and the size of b. Using simple geometry we can find the (x,y,z) position of the drone. We'll define z = d. From simple geometry we know $\frac{c}{c+d} = \frac{b}{a} \rightarrow d = \frac{ac}{b} - c$. *Finally, x and y are inferred from the position of the GS within the frame, i.e. by using pxy we can calculate the* (X_0, Y_0) *of the center of GS. Then the position of the drone is* $(-X_0, -Y_0, d)$.

Figure 11: A demonstration of the drone's usage of the GS. On the left we see their position from an external view and on the right the image as seen from the camera of the middle drone. From this image we can infer the drone's location and correct it if needed.

Notice that the inner origin of the coordinates of each drone, i.e. its relative (0,0), changes with each interruption. For instance, if a drone hovers over the same spot, say (0,0), and then an external force moves the drone 1 meter to the right. The drone will still assume it is in (0,0), since it was not given the command to move elsewhere and has no GPS-like system to detect changes. It does have a gyroscope that corrects its movement if a minor external force was applied but given a major interruption it is necessary to correct the position of the next spot, we wish the drone to fly to, as shown in figure 12.

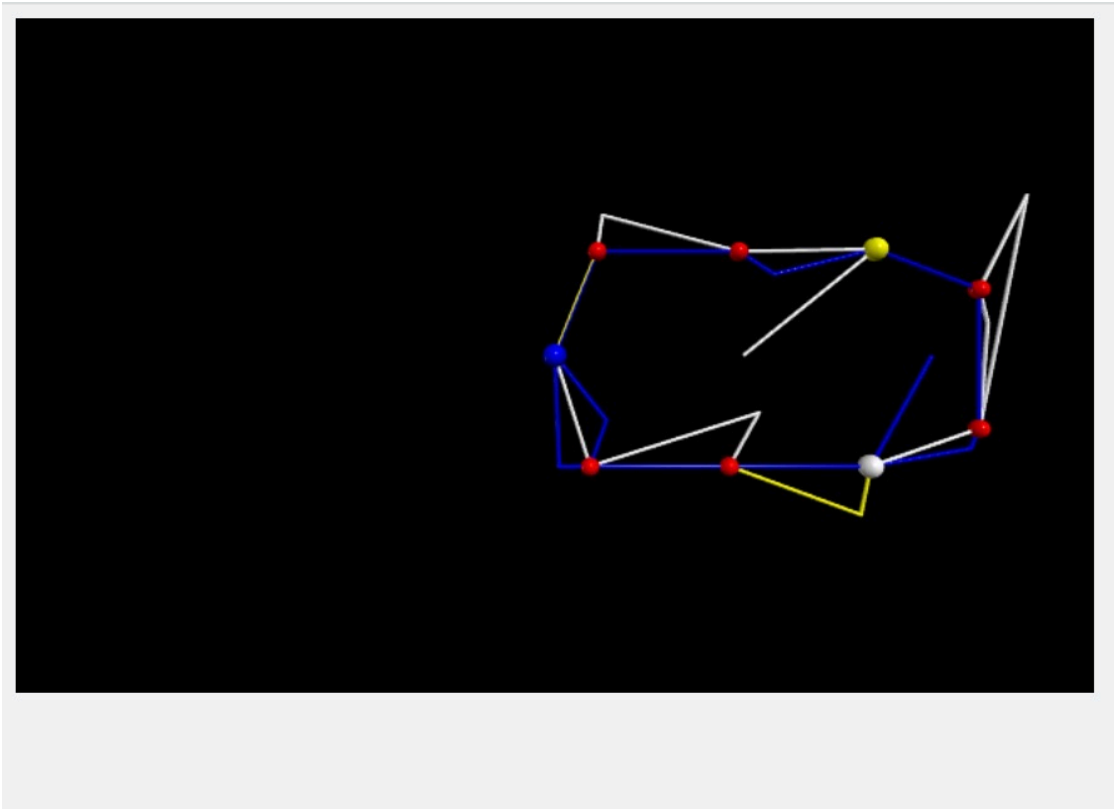

Figure 12: The simulation shows the path 3 drones (blue, white, and yellow) moved on (the red circles) for 2 cycles with interruptions and the corrections that were made. Naturally, we expect to see overlapping routes, which the blue drone came closest to, but due to external interruptions the drones deviate and then correct their paths.

One of the challenges we faced was with the lack of drones, until we purchased additional ones. Thus, until the arrival of the new drones, it was crucial for us to simulate their movement, which we have accomplished by using the vPython library. This proved the usefulness of the VPython simulations.

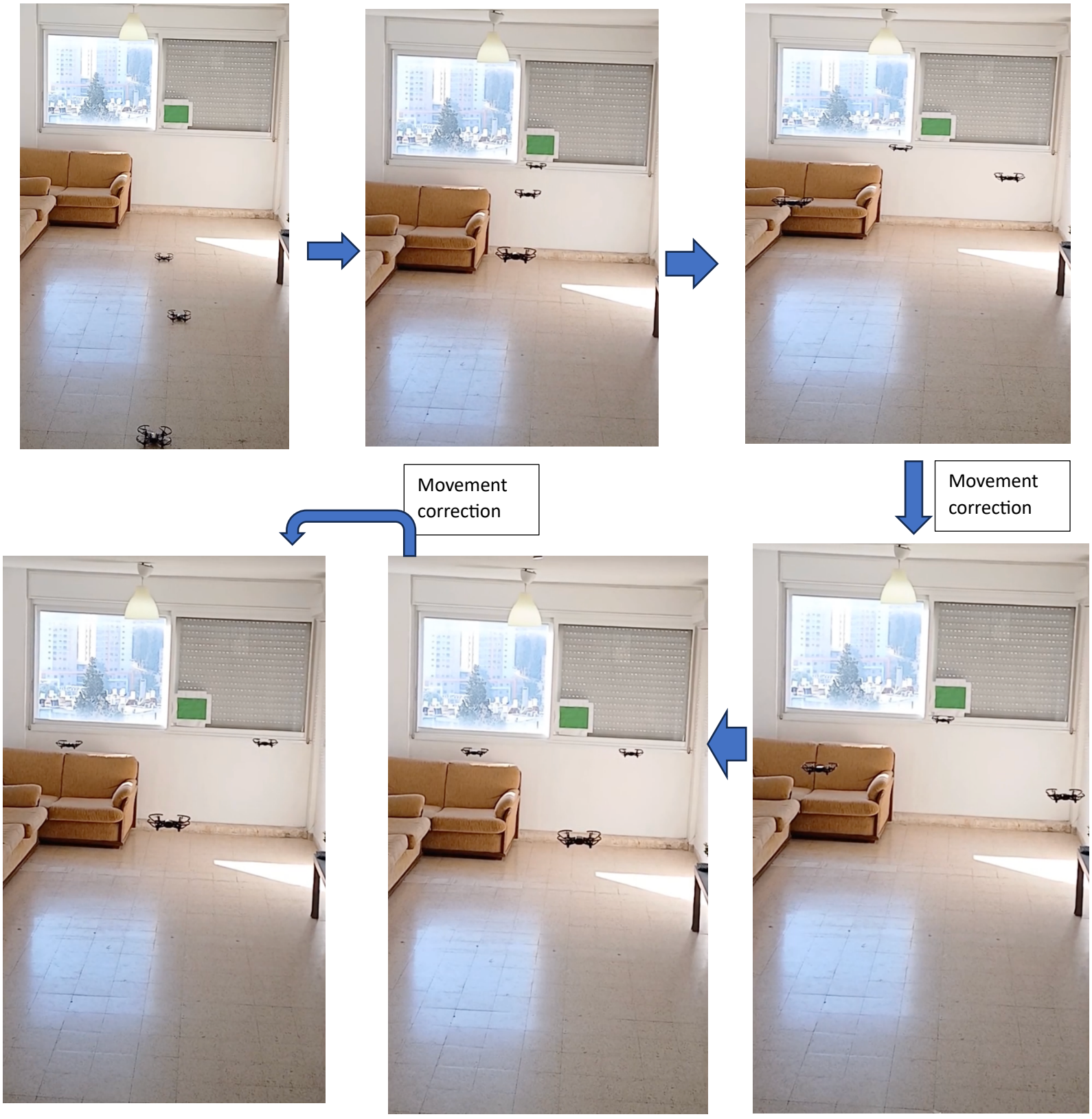

Figure 13: Example of the drone's movement and the correction that occurs in real-time when the drones stray from the path.

Summary

We managed to create a system that communicates with the cloud on its highest (software) level and with a feedback controller on its lowest (hardware) level, making it possible to operate even in "noisy" environments, e.g. outside when it's windy, generate live simulations from the drones' movement, and control them from afar. Unfortunately, the Edu-Tello drones were designed for educational purposes and not to be utilized for real-world applications. Therefore, we don't see any potential use with the current setup. However, given the appropriate hardware it will not require extensive work to repurpose the software to support real-time applications with a compatible drone, since there is a rigorous separation of the section responsible for the SDK in our code.

Key milestones

Date.	achievement
10.4	Converted a bit-map to a route for the drones
1.6	Created a simulation for their flight path (for better visualization)
24.6	Connect several drones to the router
15.7	Integrated the 2 libraries we developed to make the drones receive commands from the simulation pattern
7.8	Decoded the H264 stream to a png image
8.8	Major integrations and upgrades to finalize the system, e.g. created a live log for feedback and a battery check, support for Firebase commands, etc.

Table 1: significant milestones of the project.

What comes next?

The use of a green square simplifies the system but makes it less practical to recreate and scale. Therefore, the next step to improve the system would be to remove the GS altogether and get the location solely from data gathered by the drones' cameras. This might be achieved by algorithms such as Kalman Filter but will require heavier calculations. Another aspect of the project that could be improved is the system's connectivity to the Robo-phone interface. A full integration of the Edu-Tello's drone SDK with the Robo-phone would make it far more friendly to the users. In addition, instead of receiving a binary image, we could request any image that contains a path, e.g. a clear road in a forest, and then use a neural network, similar to the one we used, to extract the path.

Reflections

Throughout the project we noticed the difficulties of handling hardware and real-life problems in comparison to software. Such problems were evident in bug detections; when part of the system did not work, e.g. the drones did not receive the commands, it was by far harder to get to the source of it in comparison to a software bug.

In addition, we had to handle bottle necks such as the bandwidth of the router which limited the amount of imaged that could be broadcast, or the connectivity to Firebase, which prevented us from running the heavy computations on the cloud.

On the subject of heavy computations, our go-to move to detect the GS was to use a pre-trained AI model, which indeed successfully accomplished the job, but took longer than we could afford to waste. By noticing we could achieve the same result using simple geometric

computations, the algorithm ran much faster. This came to show that while AI is convenient to use as a black box, it is often redundant.

It was important to understand the limitations and capabilities of the hardware tools in our possession as well as the existing libraries. This helped us expect and sometimes avoid potential issues and work more efficiently.

While we could have achieved better results by building on top of previous projects and as a result create tools that could prove their usefulness to many other engineers, we are contempt with the progress we made and the potential our project has in serving future Robo-phone users.

Guidelines for future users

- When turning a drone to a station (ap mode) and connecting it to the router, the Wi-Fi name (SSID) must only have letters. Otherwise, you'll get an 'ep not recognized' error.
- We borrowed 1 drone from Intel - wasn't Tello Edu and thus couldn't connect to the router. Make sure you use the black Edu drones, version 3 and above.
- Caveat- in CRML lab there are 2 Edu Tello drones that don't function properly. 1 didn't take off, the other one doesn't recognize any command besides "takeoff". The 3ed one responds well and is stable.
- You need to establish a UDP connection and find out to which port it transmits.
- Decoding the messages we receive from the Tellos Camera.
- Rearranging them in the right order
- Guide for H264 decoder installation and usage-

  - Go to https://github.com/DaWelter/h264decoder/tree/master

  - Install vcpkg; guidelines in this link.

  - Go back to stage 1 (h264 git) and follow the instructions to build/install

  - Give up if these stages don't work.

- To install ffmpeg follow link.
- Drones' details:
  - IP Address: 192.168.10.1
  - MAC: 60-60-1f-dc-61-X; where X is the id of the drone (name of its Wi-Fi)

Bibliography

[1] New technology making drones easier, more affordable, Taryn Luna, Bostonglobe, 2015

[2] The 1,500 drone dragon that danced in the Shenzhen sky, AV Magazine, 2022; link

[3] Patriotic drone show lights up the sky with record-breaking aerial message, Guinness World Record, 2023; link

[4] Echodyne and Univ. of Washington assist DARPA with aerial drone surveillance test, GeekWIRE, 2019; link

[5] Ukraine Will Get Agricultural Drone Silver Lining After War, The Medium, 2023; link

[6] Manual for the Ryzo drone (SDK) in this link.

[7] Drone swarm libraries in this link, and this link.

[8] H264 decoder (for the streaming in UDP channel) in Dewalter's git repository; link.

[9] To start with Firebase we used the tutorial in this link.