Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

5C21
CPS

CrazyFlie Drone

Prof. Harun Silijak
T.A: Cormac Molloy

Group 1:
Julia Borel
Cornell Castelino
Abhishek Dutta Gupta
Euan David Carroll

Group report for practical component of Cyber Physical System and Control 10 credit module.

2022-2023

Introduction

This report is for the four drone challenges that were performed as part of the Cyber-Physical Systems Lab. These include the handheld controller, the autonomous flight, the advanced controller, and the advanced autonomous flight control. A crazyflie drone is used for all these purposes, which easily connects to the software cfclient or any mobile device. The drone is mapped to the software using an R-F dongle that connects using the drone address. The Crazyflie drone supports various kinds of decks namely the flow deck, the Z-ranger deck, the lighthouse deck, etc... The Lighthouse deck is predominately used here for all the challenges as it supports a co-ordinate based positioning system that enables easy position commands to the drone with high precision compared to other decks.

In the controller part, we explored various control opportunities first through an Arduino Controller with its joystick library and then through a hand tracker using AI libraries Media pipe and open CV. More details about each method are demonstrated below.

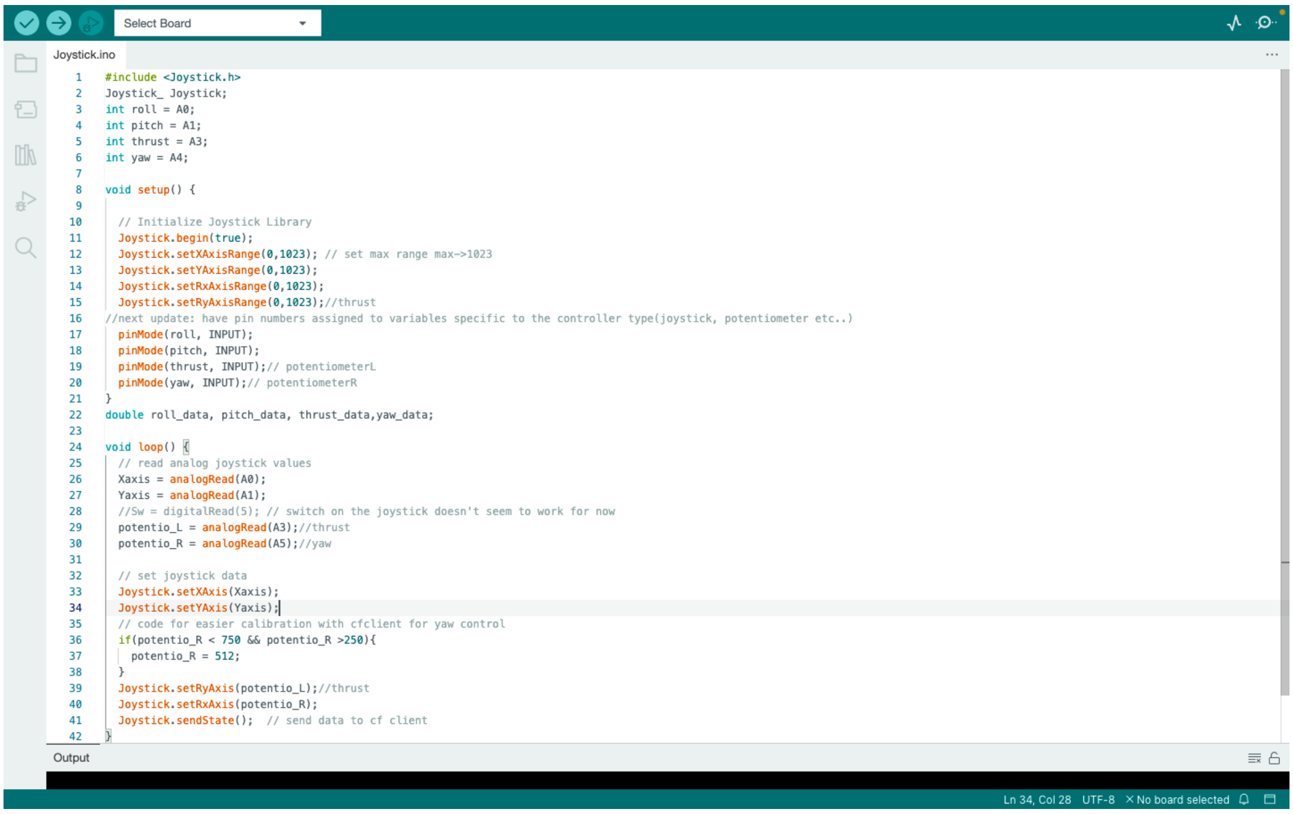
In the autonomous flight control, first, a basic drone forward/backward/left/right movement was performed. In the advanced challenge together with one more drone, a swarm activity of drones was performed with different series. The challenges and details are explored below.

So, let's fly our way through it. But wait.

Handheld the Controller

This was the first challenge in the controller component of this project. It involved the design and verification of a microcontroler-based handheld device that would send data to our drone via the Crazyflie client console. Communicaticlientkey aspect to nail down between the cf dongle and the drone itself. It is done in Compressed Real-time Transport protocol (CRTP). Data is sent in packets to the cfclient using HID descriptors through Arduino.

Using the joystick library in Arduino a simple handheld controller was developed using the Arduino micro board. Our handheld controller can be seen in figure 1. It features two potentiometers and one actual joystick. These components can be seen to offer us a wide range of movement for our drone. We initially used a ultrasonic sensor but for precision purposes we couldn't get a range of freedom that was desired for this application.



```
#include <Joystick.h>
Joystick_ Joystick;
int roll = A0;
int pitch = A1;
int thrust = A3;
int yaw = A4;

void setup() {

  // Initialize Joystick Library
  Joystick.begin(true);
  Joystick.setXAxisRange(0,1023); // set max range max->1023
  Joystick.setYAxisRange(0,1023);
  Joystick.setRxAxisRange(0,1023);
  Joystick.setRyAxisRange(0,1023);//thrust
//next update: have pin numbers assigned to variables specific to the controller type(joystick, potentiometer etc..)
  pinMode(roll, INPUT);
  pinMode(pitch, INPUT);
  pinMode(thrust, INPUT);// potentiometerL
  pinMode(yaw, INPUT);// potentiometerR
}
double roll_data, pitch_data, thrust_data,yaw_data;

void loop() {
  // read analog joystick values
  Xaxis = analogRead(A0);
  Yaxis = analogRead(A1);
  //Sw = digitalRead(5); // switch on the joystick doesn't seem to work for now
  potentio_L = analogRead(A3);//thrust
  potentio_R = analogRead(A5);//yaw

  // set joystick data
  Joystick.setXAxis(Xaxis);
  Joystick.setYAxis(Yaxis);
  // code for easier calibration with cfclient for yaw control
  if(potentio_R < 750 && potentio_R >250){
    potentio_R = 512;
  }
  Joystick.setRyAxis(potentio_L);//thrust
  Joystick.setRxAxis(potentio_R);
  Joystick.sendState();  // send data to cf client
}
```

Figure 1: Arduino Code for Handheld controller

The code was short and simple because the Joystick library did everything for us mostly. We just had to call certain functions and manipulate the data we collected from our controller components to the arduino via wiring as seen in figure 2. The axis range setting functions set the data values between 0 and 1023. As well as the X and Y axis we also have the Rx and the Ry axis which are the rotational axis that will be used to get data from the potentiometer control aspect. The actual set axis function sets the value collected by user toggling of both the joystick itself and the two potentiometers.

Finally, the send state function will send the data collected to the host computer, in our case the cfclient, via HID descriptors. We couldn't get the altitude assist feature to work in the cfclient automatically so we tried to implement it manually by using a button and the button function within the joystick library but sadly failed to do so. This would be something for us to look into when designing improvements for our drones in the future.

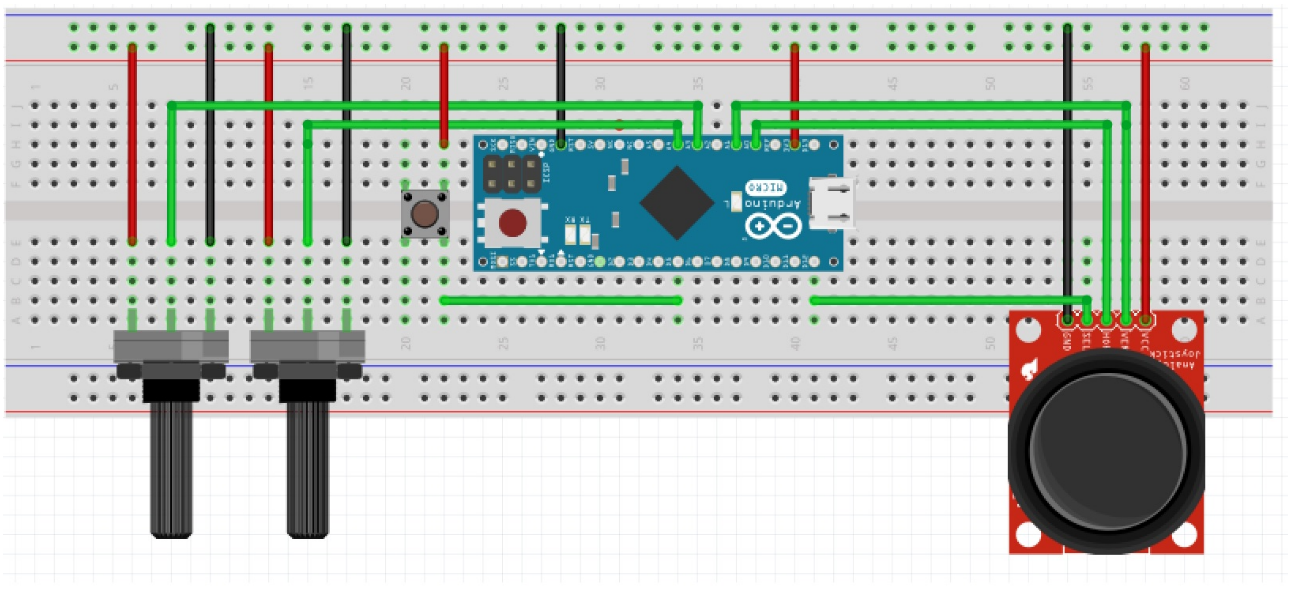


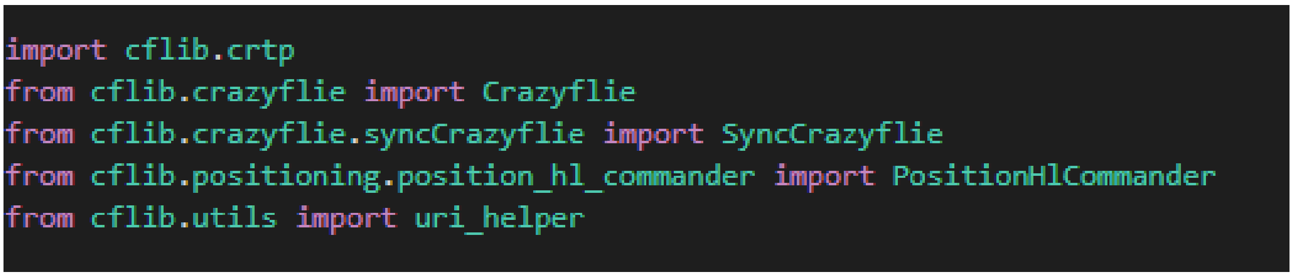
Figure 2: Circuit setup for the controller

Autonomous Flight

The requirement of this flight challenge was to demonstrate basic movements in the drone through autonomous control. The autonomous control was performed using the cfclient control via python libraries. The drone was connected to the cfclient by mapping the address of the drone through an R-F dongle. The lighthouse deck and lighthouse towers were used in this challenge. The calibration of the positioning of the drone in the co-ordinate based system of the lighthouse was performed. After calibration, the drone with the lighthouse deck in it, is kept in the (0,0,0) position. The drone can be controlled using the python libraries of crazyflie (H/L commander and position commander), where the address of the drone is also passed for the drone to be controlled. The drone is now ready to be controlled as per the code.

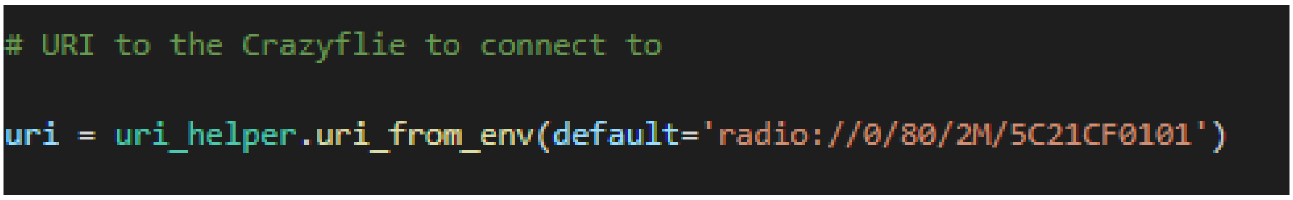
The Code - Autonomous Flight

1. Import of All Libraries
2. Connection of drone to cfclient - The cfclient connects to drone as per the address provided. The address of our drone which is used for the autonomous challenge is //0/80/2M/5C21CG0101.
3. Autonomous Drone movement - With the use of the inbuilt library Position HL commander, the drone is instructed to go forward 1 m in the x-axis, then left 1m, back 1m, and then finally to co-ordinates (0,0,1) and (0,0,0).



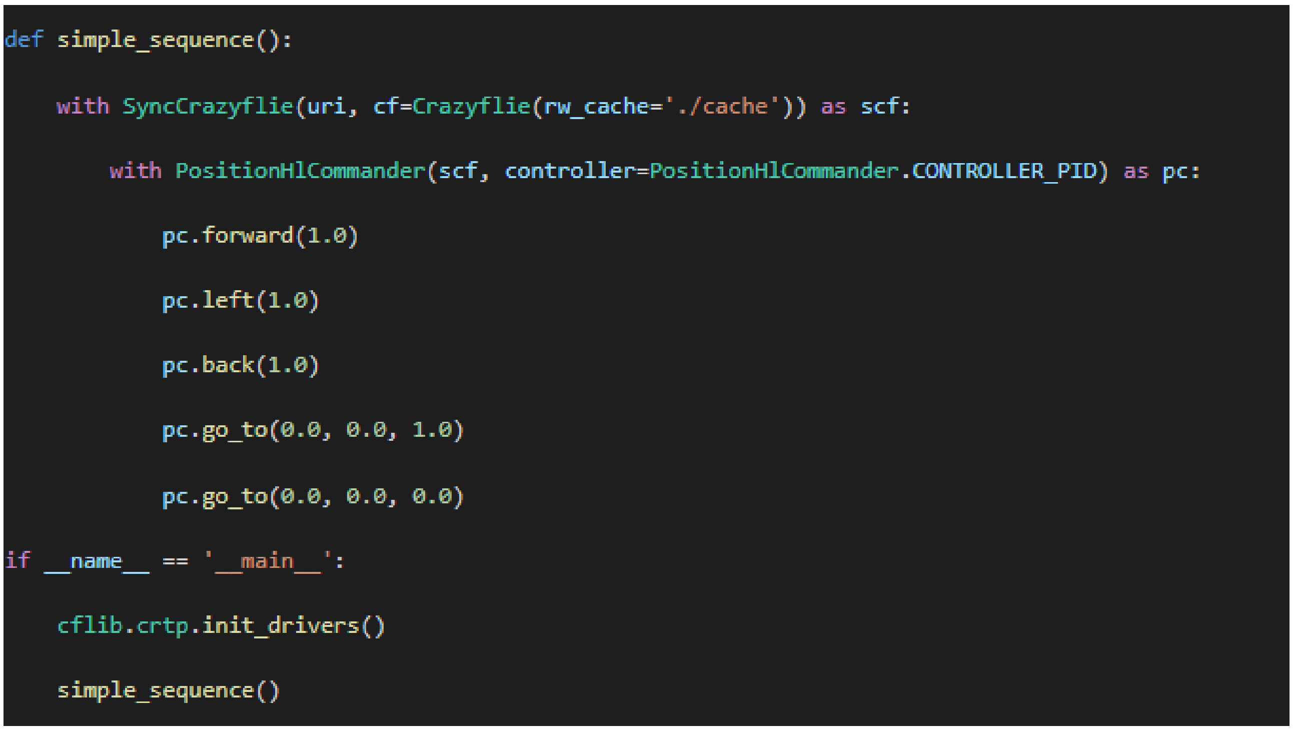
```python
import cflib.crtp
from cflib.crazyflie import Crazyflie
from cflib.crazyflie.syncCrazyflie import SyncCrazyflie
from cflib.positioning.position_hl_commander import PositionHlCommander
from cflib.utils import uri_helper
```

Figure 3: Code snippet1: Imports of all libraries



```python
# URI to the Crazyflie to connect to

uri = uri_helper.uri_from_env(default='radio://0/80/2M/5C21CF0101')
```

Figure 4: Code snippet2: Connection of drone to cfclient



```python
def simple_sequence():

    with SyncCrazyflie(uri, cf=Crazyflie(rw_cache='./cache')) as scf:

        with PositionHlCommander(scf, controller=PositionHlCommander.CONTROLLER_PID) as pc:

            pc.forward(1.0)

            pc.left(1.0)

            pc.back(1.0)

            pc.go_to(0.0, 0.0, 1.0)

            pc.go_to(0.0, 0.0, 0.0)

if __name__ == '__main__':

    cflib.crtp.init_drivers()

    simple_sequence()
```

Figure 5: Code Snippet3: Drone Movement

The autonomous drone movement frame by frame

1. Drone kept at (0,0,0) position. Refer 6
2. Drone goes to default height 0.5m and then forward (1m) in the x-axis as per the code. Refer 7
3. Then it goes left 1m and then back 1m as per the code. Refer 8
4. Finally it goes to (0,0,1) as per the code and then rests at (0,0,0). Refer 9

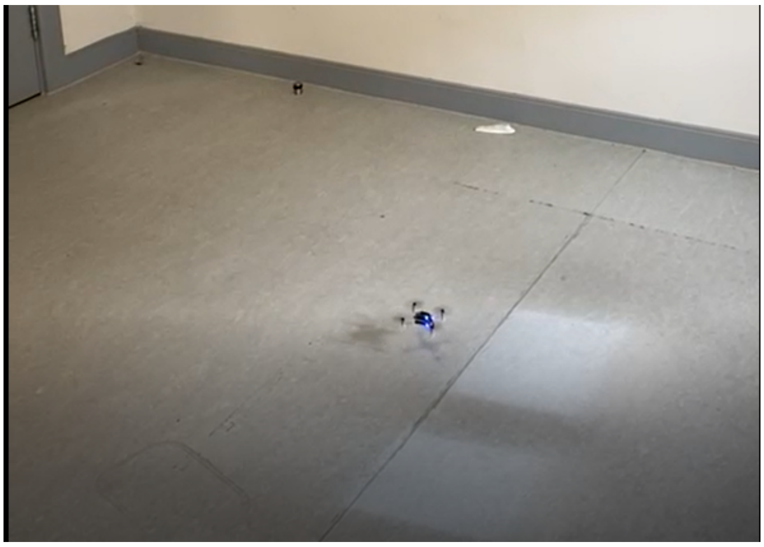


Figure 6: Frame 1: Start (0,0,0)

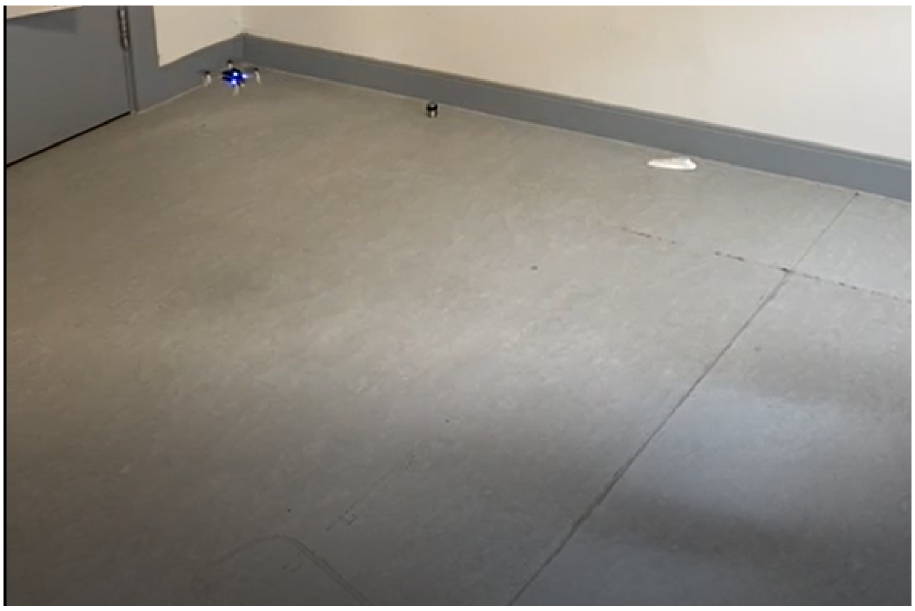
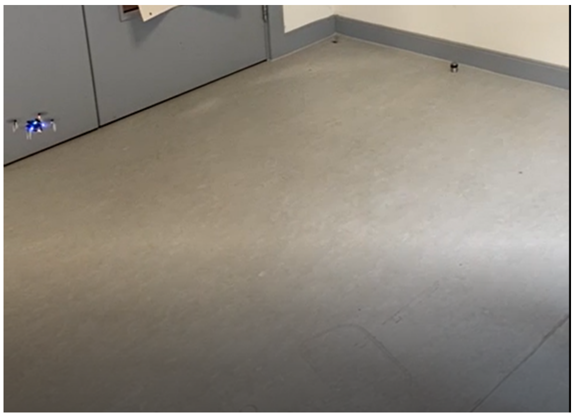


Figure 7: Frame 2: pc.forward(1.0)

Watch the complete autonomous drone challenge : `https://youtube.com/shorts/HKiqkjQ97to?feature=share`


(a) Drone Left 1m

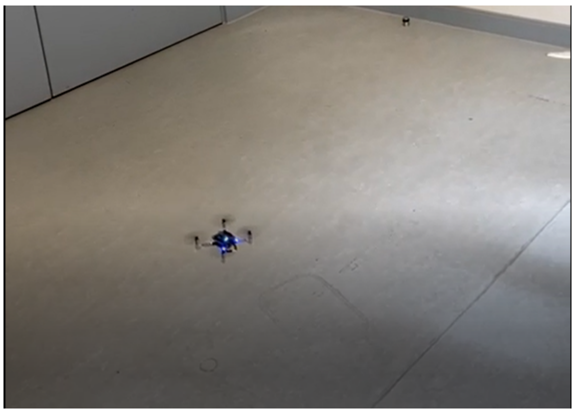
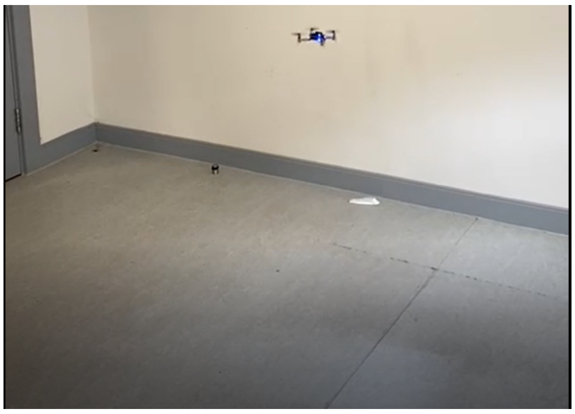

(b) Drone back 1m

Figure 8: Drone Movement left and back as per code


(a) Drone (0,0,1)

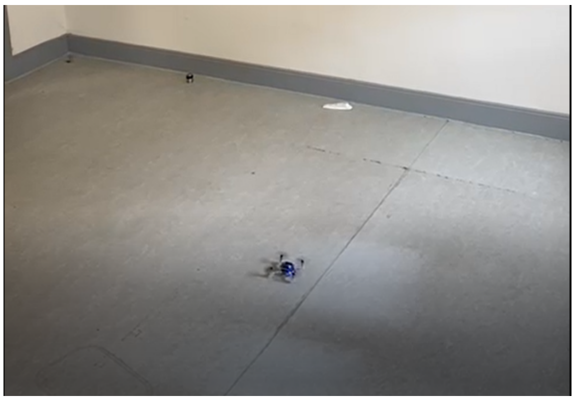

(b) Drone back (0,0,0)

Figure 9: Drone Movement co-ordinate wise as per code

Advanced Controller

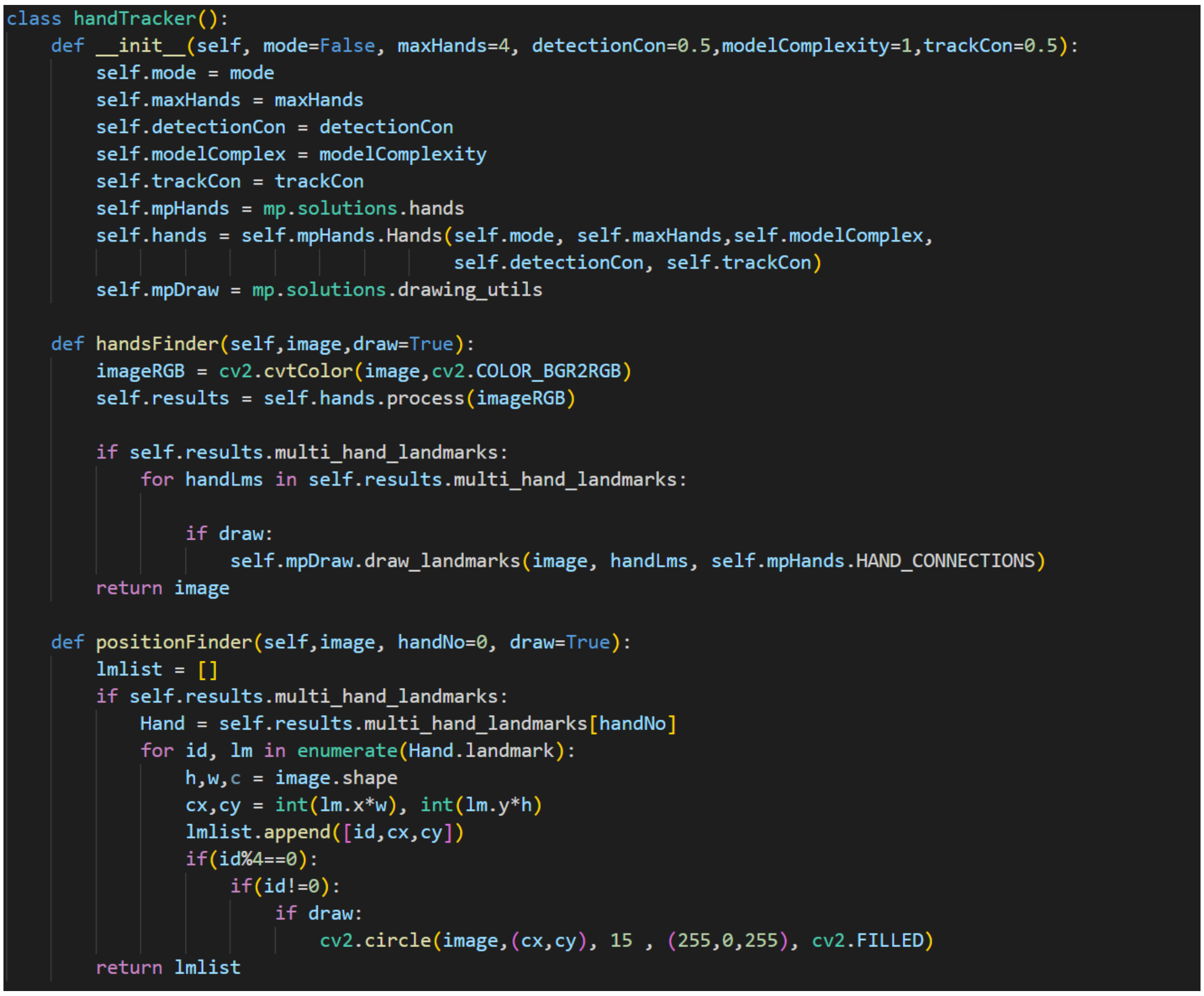
For the advanced controller challenge, we were tasked to take on a novel approach in controlling the drone which improves on the functionality and design of the previous handheld controller challenge. The team decided to use a hand tracker that extracts the raw positions of the index fingers using Media Pipe and openCV libraries to map the position of the index fingers to the height, yaw, pitch, and roll for the drone. The drone also works in combination with a flow deck or a lighthouse deck to achieve altitude hold for a better control experience.

The main idea is to use index fingers to control the drone by extracting the x and y coordinates of the tip of the index finger for the relative control of height, yaw, pitch, and roll. This way, relative to the camera, pointing the index finger up results in a positive pitch or height increase depending on which side of the finger is raised. similarly, pointing down, left, or right results in a height decrease, negative pitch, postive roll, negative roll, positive yaw, or negative yaw.

The algorithm cycle for each frame of the image is as follows:

1. Initialise parameters and camera.
2. Initiate connection to drone using SyncCrazyflie() and PositionHlCommander(). Stop if drone not found.

3. Find hands in the image frame.
4. If hands are found, draw a hand skeleton and update the video stream.
5. Extract the pixel position of the index fingers.
6. Keep a moving average of the position of the index fingers in 3 steps to smoothen the movement for the drone.
7. Map the pixel positions to obtain new height, yaw rate, pitch rate, and roll rate as parameters required for set_hover_point().
8. Use set_hover_point() to send data to the drone.
9. Repeat steps 3 to 8 until video feed or drone is disconnected.



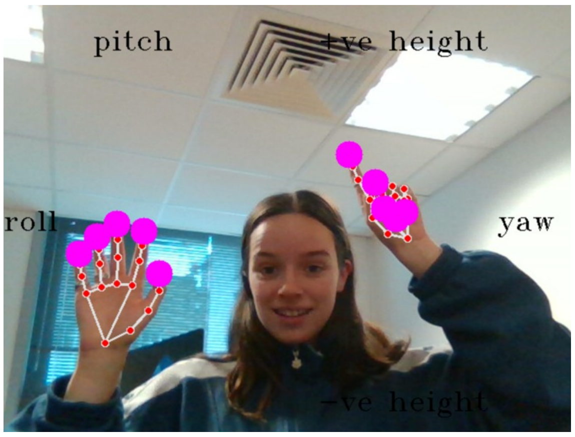
```
class handTracker():
    def __init__(self, mode=False, maxHands=4, detectionCon=0.5,modelComplexity=1,trackCon=0.5):
        self.mode = mode
        self.maxHands = maxHands
        self.detectionCon = detectionCon
        self.modelComplex = modelComplexity
        self.trackCon = trackCon
        self.mpHands = mp.solutions.hands
        self.hands = self.mpHands.Hands(self.mode, self.maxHands,self.modelComplex,
                                        self.detectionCon, self.trackCon)
        self.mpDraw = mp.solutions.drawing_utils

    def handsFinder(self,image,draw=True):
        imageRGB = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
        self.results = self.hands.process(imageRGB)

        if self.results.multi_hand_landmarks:
            for handLms in self.results.multi_hand_landmarks:

                if draw:
                    self.mpDraw.draw_landmarks(image, handLms, self.mpHands.HAND_CONNECTIONS)
        return image

    def positionFinder(self,image, handNo=0, draw=True):
        lmlist = []
        if self.results.multi_hand_landmarks:
            Hand = self.results.multi_hand_landmarks[handNo]
            for id, lm in enumerate(Hand.landmark):
                h,w,c = image.shape
                cx,cy = int(lm.x*w), int(lm.y*h)
                lmlist.append([id,cx,cy])
                if(id%4==0):
                    if(id!=0):
                        if draw:
                            cv2.circle(image,(cx,cy), 15 , (255,0,255), cv2.FILLED)
        return lmlist
```

Figure 10: Class for hand detection

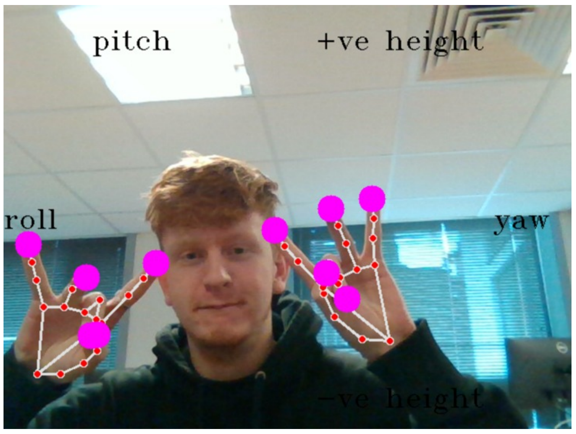
In detail, step 3 uses the Media pipe library which has an in-built hand detection implementation based of AI-Inference[1] to extract positions of each joint in the hand present on the image. Then, step

[1]Media Pipe Hands

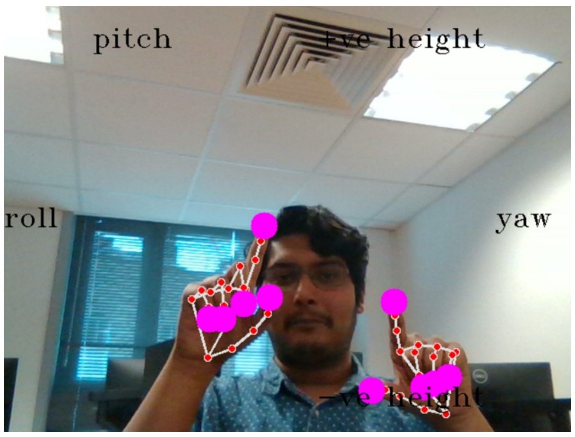
4 uses openCV to draw a hand skeleton on the video feed by using the extracted hand coordinates to observe successful tracking of the hands using the code seen in 10. Step 5 and 6 use the extracted positions of the 2 index fingers to keep a moving average for the smoothening of the movement of fingers to minimize the effect of handshaking for a more pleasant movement of the drone. Additionally, a small area is available at the 2 centers of the finger position to allow for relaxation, only when the finger crosses the invisible boundary, new control data is sent to the drone.



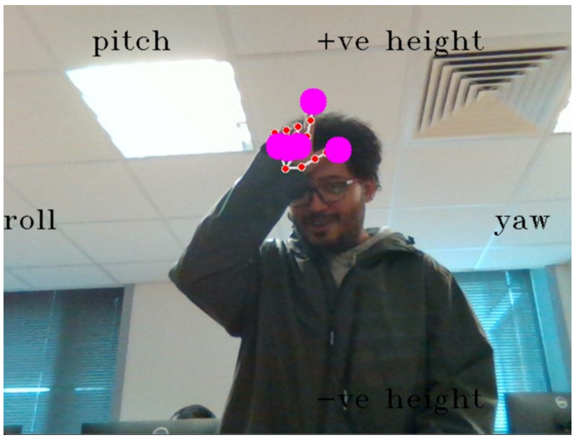
(a) Julia



(b) Euan



(c) Cornell

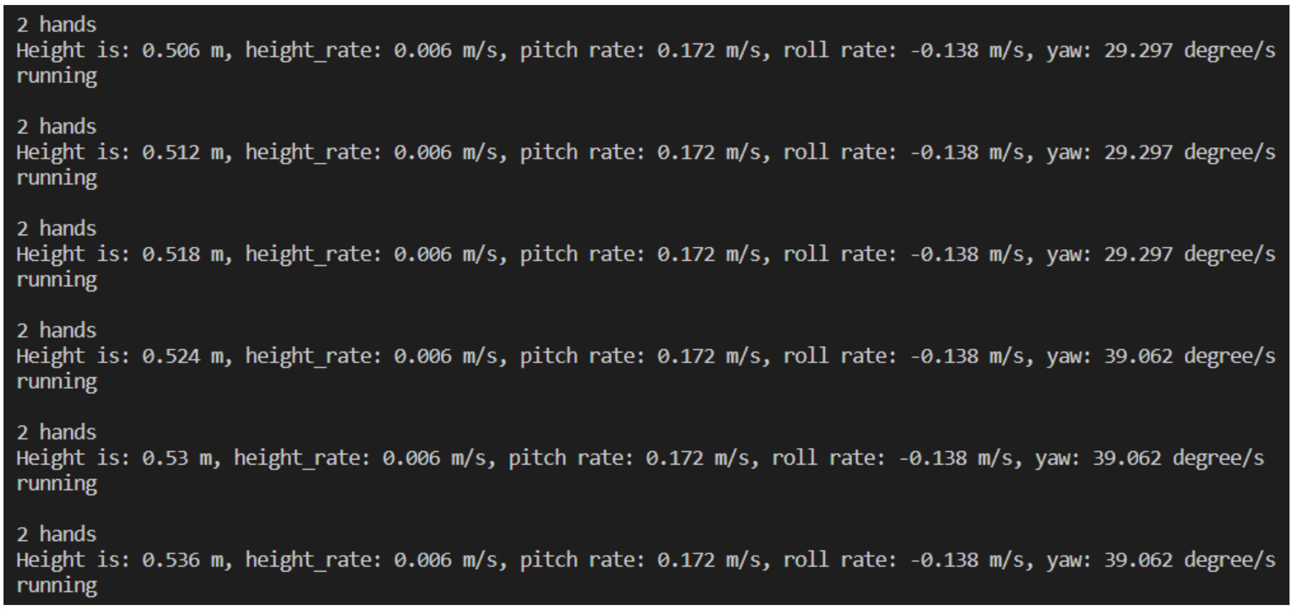


(d) Abhishek

Figure 11: Imposed hand skeleton of each group member from the extraction of hand coordinates from Media Pipe

Finally, steps 7 and 8 map each of the coordinates of the 2 index fingers from the camera feed to the relevant height/yaw rate/pitch rate/roll rate as seen in figure 12 .Figure 11 shows the visual of the camera seen during the control of the drone.

The flow-deck is the ideal hardware for movement for the drone, however, during the initial setup of the camera the drone acts wildly as if receiving ghost vectors which was never sent but after the camera is initialised and control data is sent, the drone behaves normally. In the first 2 seconds there is good possibility that the drone crashes on the walls or nets due to some unstable feedback causing it to fly crazily. Therefore, while demonstrating the advanced controller challenge, the team opted to use the lighthouse-deck for stability in the first 2 seconds which was successful, However, limiting out flight range between the lighthouse-decks which is not as ideal as the flow-deck.



```
2 hands
Height is: 0.506 m, height_rate: 0.006 m/s, pitch rate: 0.172 m/s, roll rate: -0.138 m/s, yaw: 29.297 degree/s
running

2 hands
Height is: 0.512 m, height_rate: 0.006 m/s, pitch rate: 0.172 m/s, roll rate: -0.138 m/s, yaw: 29.297 degree/s
running

2 hands
Height is: 0.518 m, height_rate: 0.006 m/s, pitch rate: 0.172 m/s, roll rate: -0.138 m/s, yaw: 29.297 degree/s
running

2 hands
Height is: 0.524 m, height_rate: 0.006 m/s, pitch rate: 0.172 m/s, roll rate: -0.138 m/s, yaw: 39.062 degree/s
running

2 hands
Height is: 0.53 m, height_rate: 0.006 m/s, pitch rate: 0.172 m/s, roll rate: -0.138 m/s, yaw: 39.062 degree/s
running

2 hands
Height is: 0.536 m, height_rate: 0.006 m/s, pitch rate: 0.172 m/s, roll rate: -0.138 m/s, yaw: 39.062 degree/s
running
```

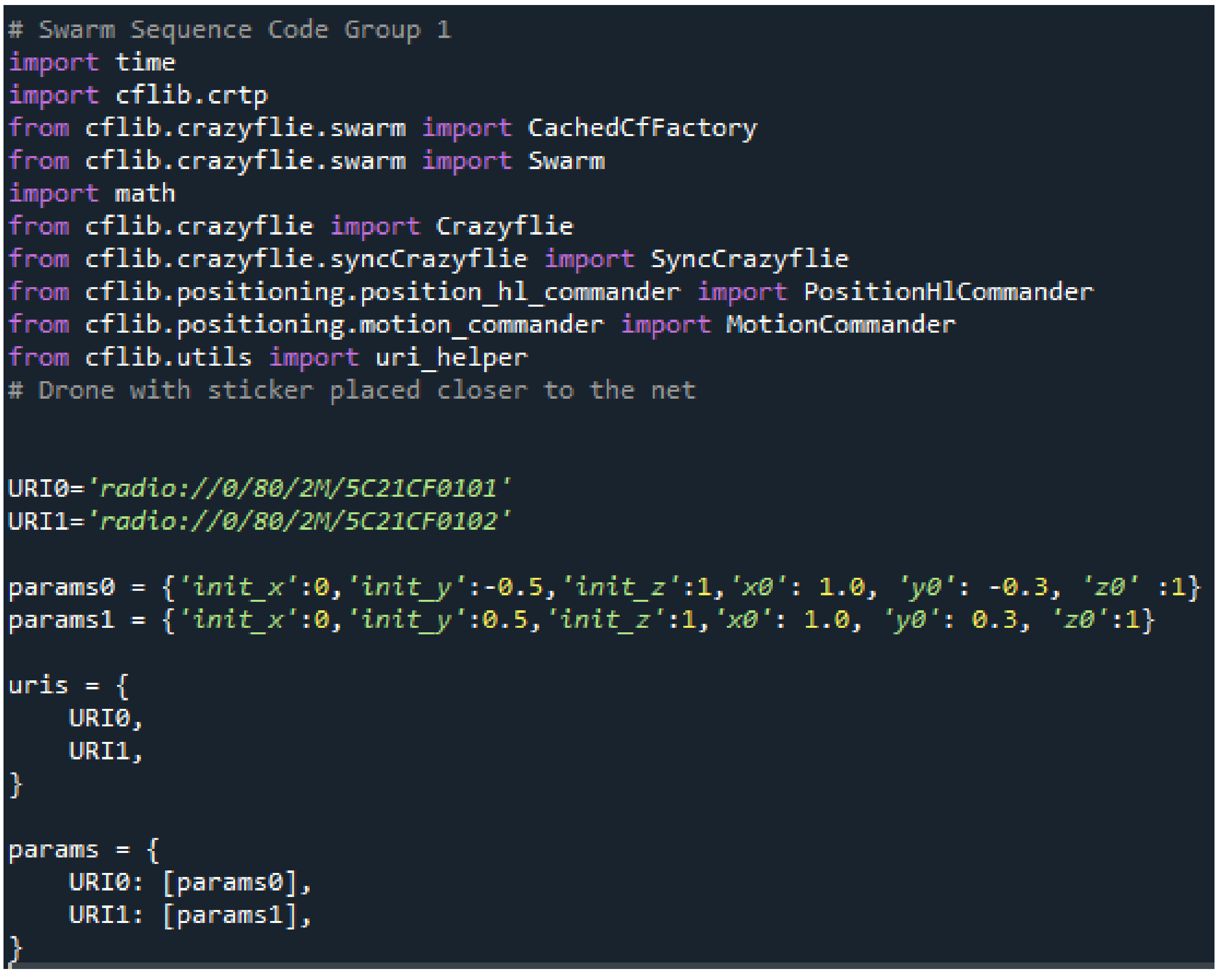
Figure 12: Output values of control for confirmation.

Advanced Autonomous Flight

The requirements of this task were to demonstrate a higher level of understanding and further develop the autonomous capabilities of the drone. The team decided to pursue a combination of a swarm pattern and advanced movement. The approach uses two drones moving in a combined pattern that will be described below. The code for the swarm advanced movement set up is shown in Figure **??**. The libraries used in this code are similar to that of the simple movement but in addition there a libraries to allow for the swarm commands. The addition URI addresses are outlined and can be added to the code if more drones are added to the sequence.

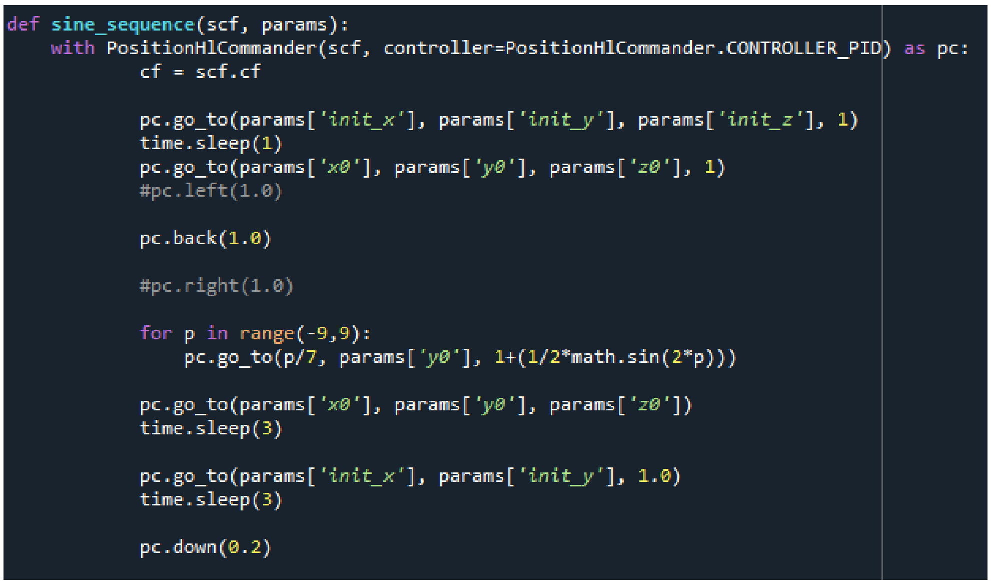
In the first movement, the drones perform a simultaneous movement in a Sine pattern. The The pattern required tuning depending on the dimensions of the available room and to create a visible and recognisable wave. This code is shown in Figure 14.

To create a more rapid movement that took advantage of the MotionCommander, another sequence was introduced into the code. The code will move both drones simultaneously in circle movements, instructed by the circle left or circle right movements with the MotionCommander. This code is shown in Figure 15.



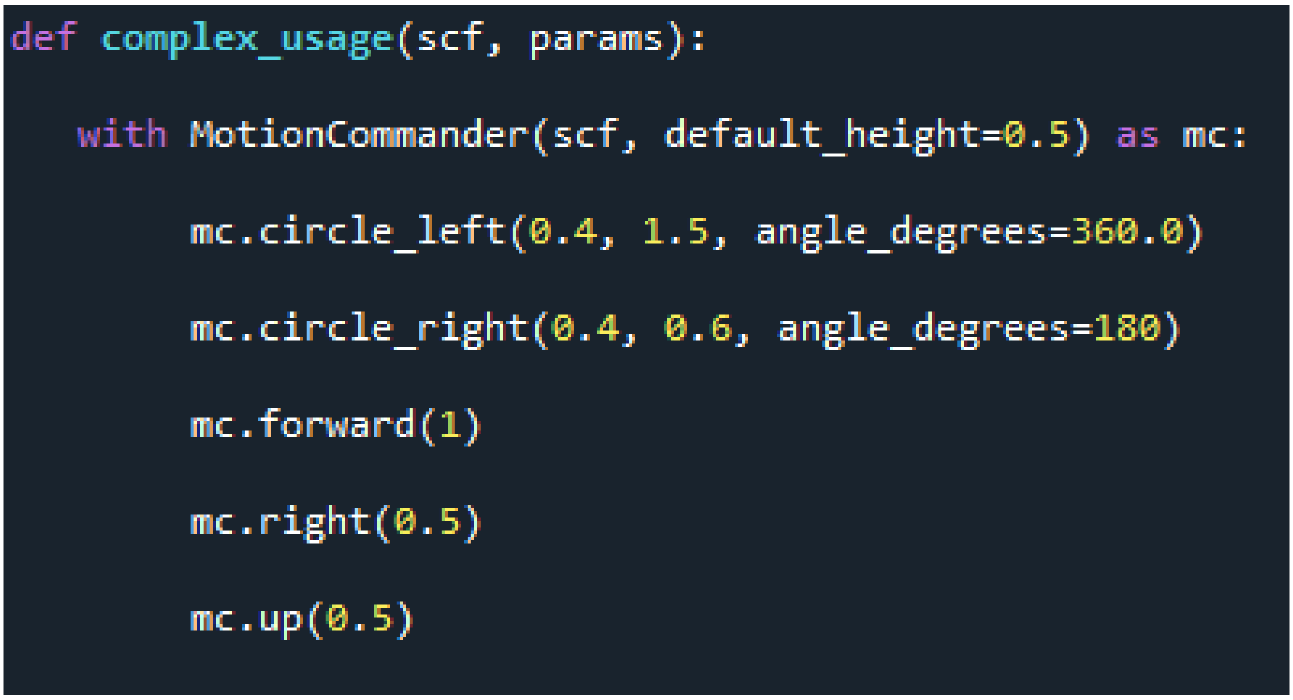
```python
# Swarm Sequence Code Group 1
import time
import cflib.crtp
from cflib.crazyflie.swarm import CachedCfFactory
from cflib.crazyflie.swarm import Swarm
import math
from cflib.crazyflie import Crazyflie
from cflib.crazyflie.syncCrazyflie import SyncCrazyflie
from cflib.positioning.position_hl_commander import PositionHlCommander
from cflib.positioning.motion_commander import MotionCommander
from cflib.utils import uri_helper
# Drone with sticker placed closer to the net


URI0='radio://0/80/2M/5C21CF0101'
URI1='radio://0/80/2M/5C21CF0102'

params0 = {'init_x':0,'init_y':-0.5,'init_z':1,'x0': 1.0, 'y0': -0.3, 'z0' :1}
params1 = {'init_x':0,'init_y':0.5,'init_z':1,'x0': 1.0, 'y0': 0.3, 'z0':1}

uris = {
    URI0,
    URI1,
}

params = {
    URI0: [params0],
    URI1: [params1],
}
```

Figure 13: Code for Autonomous Swarm Flight Set Up



```python
def sine_sequence(scf, params):
    with PositionHlCommander(scf, controller=PositionHlCommander.CONTROLLER_PID) as pc:
        cf = scf.cf

        pc.go_to(params['init_x'], params['init_y'], params['init_z'], 1)
        time.sleep(1)
        pc.go_to(params['x0'], params['y0'], params['z0'], 1)
        #pc.left(1.0)

        pc.back(1.0)

        #pc.right(1.0)

        for p in range(-9,9):
            pc.go_to(p/7, params['y0'], 1+(1/2*math.sin(2*p)))

        pc.go_to(params['x0'], params['y0'], params['z0'])
        time.sleep(3)

        pc.go_to(params['init_x'], params['init_y'], 1.0)
        time.sleep(3)

        pc.down(0.2)
```

Figure 14: Code for Autonomous Swarm Flight Sine Sequence



```python
def complex_usage(scf, params):

   with MotionCommander(scf, default_height=0.5) as mc:

        mc.circle_left(0.4, 1.5, angle_degrees=360.0)

        mc.circle_right(0.4, 0.6, angle_degrees=180)

        mc.forward(1)

        mc.right(0.5)

        mc.up(0.5)
```

Figure 15: Code for Autonomous Swarm Flight Circling Sequence

Watch the complete Advanced autonomous drone swarm challenge: `https://youtu.be/rTdUSdARNCM`

Conclusion including ways to improve the drone

Start writing here