

DRONE AIR TRAFFIC CONTROL SYSTEM

Technical manual

By

Brendan Mitchell

Title: Drone Air Traffic Control System – Technical Manual

Author: Brendan Mitchell(c00220212)

Supervisor: Dr. Oisín Cawley

Date of Submission: April 3rd, 2020

Organisation: IT Carlow

Contents

Section 1 – Introduction.....	3
Section 2 – System requirements	3
Ubuntu	3
Python	3
To install pip3.....	3
Pyparrot	3
Pygame.....	4
Other modules that need to be installed.....	4
How to start the desktop application	4
Section 3 – Source Code	4
DroneAirControlSystem.py	4
VirtualController.py	12
BebopTestConnect.py.....	18
BebopDroneController.py.....	19
DroneBase.py.....	20
FlightPath.py	22
testMaps.png	24
Plagiarism form.....	26

Section 1 – Introduction

This document will outline how the system should be installed and the relevant modules and libraries that are needed for the user to use the system. This document will also show the user how to use the system.

Section 2 – System requirements

This section informs the user of the requirements that are needed to run the drone air traffic control system desktop application.

Ubuntu

The operating system used during this project was ubuntu version 18.04.3. To check which version of ubuntu you have use the following command:

```
$ lsb_release -a
```

Python

Python 3.6 is required to run the desktop application. Python 3.6.8 was used in the creation of this application. To check which version of Python that is currently install use the command `$ python3 --version`

To install Python3:

```
$ sudo apt-get update
```

```
$ sudo apt-get install python3.6
```

Further help can be obtained from the following link on how to install Python 3:

<https://realpython.com/installing-python/>

To install pip3

```
$ sudo apt install python3-pip
```

Pyparrot

Pyparrot must be installed so its libraries can be used to connect and control the Parrot Bebop drone.

To install Pyparrot:

```
$ pip install pyparrot
```

Further help can be obtained from the following link on how to set up Pyparrot:

<https://pyparrot.readthedocs.io/en/latest/installation.html>

Pygame

Pygame was used to create the graphical user interface for the drone air traffic control system.

To install Pygame:

```
$ sudo apt install python3-pygame
```

Further help can be obtained from the following link on how to set up Pyparrot:

<https://askubuntu.com/questions/399824/how-to-install-pygame>

Other modules that need to be installed

Numpy, zeroconf and shapely also need to be installed.

To install these use the following commands:

```
$ pip3 install numpy
```

```
$ pip3 install shapely
```

```
$ pip3 install zeroconf
```

How to start the desktop application

After installing all the relevant modules as above, the application files are then downloaded and extracted. Once this is done the user should open the terminal in Ubuntu. The user should then navigate to where the folder was extracted. The user should open the folder and type the following command and press enter:

```
$ python3 DroneAirControlSystem.py
```

Section 3 – Source Code

This section outlines the code that was written in this final year's project. In addition to the following code, code written by Dr. Amy McGovern is needed to control the parrot Bebop2 drone. The module that is required to control the bebop2 drone is the Bebop.py. Other modules are also required for connection and sensors. All of these can be obtained from the following link.

<https://pyparrot.readthedocs.io/en/latest/pyparrot.html#module-pyparrot.Bebop>

DroneAirControlSystem.py

```
1. #Author: Brendan Mitchell
2. #Student Number: c00220212
3. #Description: In this class virtual drone objects are created, telemetry of this dr
   one
4. #is updated as the drone moves along the flight path to its destination and the
```

```

5. #drones are rendered on the map. The destination point are also rendered on the map
   . This class
6. # also contains the menu which the user can use to select an option such as create
   a virtual drone
7. # enter a flight path, assign a flight path, get information about a drone or infor
   mation
8. # contained in a flight path and the user can then display the drones created on th
   e map and
9. # see their journey from the starting coordinates to its destination.
10.
11.
12. #importing all the relevant modules
13. import pygame, sys, math, random
14. from pygame.locals import *
15. import pygame
16. from pygame.locals import *
17. from PIL import Image, ImageDraw
18. #from Virtual_controller import Virtual_controller
19. #from VirtualControllerTwo import VirtualControllerTwo
20. #from VirtualControllerThree import VirtualControllerThree
21. from VirtualController import VirtualController
22. from FlightPath import FlightPath
23. import numpy as np
24. from shapely.geometry import LineString
25. from numpy import interp
26.
27. import pygame
28. from pygame.locals import *
29. from PIL import Image, ImageDraw
30. #from Virtual_controller import Virtual_controller
31.
32. spriteSize =15
33.
34. #User menu
35. def menu():
36.     loop_condition = True
37.     while loop_condition == True:
38.         print("\nWelcome to drone air traffic control system!")
39.         print("\nPlease enter a number for what you want to do.")
40.         print("Enter 1 to add drone.")
41.         print("Enter 2 to create a flight path.")
42.         print("Enter 3 to search drones")
43.         print("Enter 4 to search flight paths")
44.         print("Enter 5 to view flight path information")
45.         print("Enter 6 to assign a flight path")
46.         print("Enter 7 to view drone information")
47.         print("Enter 8 to quit menu and view map .\n")
48.
49.         main_input = int(input("What would you like to do? "))
50.
51.         if main_input == 8:
52.             loop_condition = False
53.             break
54.         elif main_input == 1:
55.             addDrone()
56.         elif main_input == 2:
57.             createFlightPath()
58.         elif main_input == 3:
59.             searchDrones()
60.         elif main_input == 4:
61.             searchFlightPaths()
62.         elif main_input ==5:
63.             viewFlightPathInformation()
64.         elif main_input == 6:
65.             assignFlightPath()
66.         elif main_input == 7:

```

```

67.         viewDroneInformation()
68.
69. #creating a virtual drone
70. def addDrone():
71.     print("enter name for drone")
72.     name = input()
73.     print("enter start latitude from 52.967109 to 52.723674")
74.     lat = float(input())
75.     print("enter start longitude from -7.159379 to -6.700014")
76.     lon = float(input())
77.     print("enter destination latitude from 52.967109 to 52.723674")
78.     destLat = float(input())
79.     print("enter destination longitude from -7.159379 to -6.700014")
80.     destLong = float(input())
81.     #self.name = VirtualControllerThree(52.84083,-6.92611,52.735163726,-
7.01833326,20,5,5,6,0,100)
82.     name = VirtualController(name,lat,lon,destLat,destLong,15,5,False,6,0,15)
83.     Drones.append(name)
84.
85. #creating a flight path
86. def createFlightPath():
87.     print("enter name for flight path")
88.     flightPath = input()
89.     print("enter start latitude from 52.967109 to 52.723674")
90.     lat = float(input())
91.     print("enter start longitude from -7.159379 to -6.700014")
92.     lon = float(input())
93.     print("enter destination latitude from 52.967109 to 52.723674")
94.     destLat = float(input())
95.     print("enter destination longitude from -7.159379 to -6.700014")
96.     destLong = float(input())
97.     #self.name = VirtualControllerThree(52.84083,-6.92611,52.735163726,-
7.01833326,20,5,5,6,0,100)
98.     flightPath = FlightPath(flightPath,lat,lon,destLat,destLong,15,5)
99.     flight_paths .append(flightPath)
100.
101.     #searching flight paths in the system
102.     def searchFlightPaths():
103.         for x in range(len(flight_paths)):
104.             print (flight_paths[x].getFlightPathName())
105.
106.     #searching drones in the system
107.     def searchDrones():
108.         for x in range(len(Drones)):
109.             print (Drones[x].getName())
110.
111.     #Viewing information on the chosen flight path
112.     def viewFlightPathInformation():
113.         print("enter flight path name")
114.         inputFlightPathName = input()
115.         for x in range(len(flight_paths)):
116.             #print(flight_paths[x].getFlightPathName)
117.             if(flight_paths[x].name==inputFlightPathName):
118.                 print("")
119.                 print("flight path name is ", flight_paths[x].getFlightPath()[0]
)
120.                 print("starting latitude is ", flight_paths[x].getFlightPath()[1
])
121.                 print("starting longitude is ", flight_paths[x].getFlightPath()[
2])
122.                 print("destination latitude is ", flight_paths[x].getFlightPath(
)[3])
123.                 print("destination longitude is ",flight_paths[x].getFlightPath(
)[4])
124.                 print("altitude is ", flight_paths[x].getFlightPath()[5])
125.                 print("speed is ", flight_paths[x].getFlightPath()[6])

```

```

126.
127.
128.
129.
130.
131.
132.
133.     #Viewing information on the chosen drone
134.     def viewDroneInformation():
135.         print("enter of drone name")
136.         droneName = input()
137.         for x in range(len(Drones)):
138.             #print(flight_paths[x].getFlightPathName)
139.             if(Drones[x].name==droneName):
140.                 print("")
141.                 print("drone name is ", Drones[x].getDroneInformation()[0])
142.                 print("starting latitude is ", Drones[x].getDroneInformation()[1
143. ])
144.                 print("starting longitude is ", Drones[x].getDroneInformation()[
145. 2])
146.                 print("destination latitude is ", Drones[x].getDroneInformation(
147. ) [3])
148.                 print("destination longitude is ", Drones[x].getDroneInformation
149. ) [4])
150.                 print("altitude is ", Drones[x].getDroneInformation()[5])
151.                 print("speed is ", Drones[x].getDroneInformation()[6])
152.                 print("status is ", Drones[x].getDroneInformation()[7])
153.                 print("model number is ", Drones[x].getDroneInformation()[8])
154.                 print("battery is ", Drones[x].getDroneInformation()[9])
155.                 print("current latitude is ", Drones[x].getDroneInformation()[10
156. ])
157.                 print("current longitude is ", Drones[x].getDroneInformation()[1
158. 1])
159.
160.
161.     #assigning a flight path to a drone
162.     def assignFlightPath():
163.         print("Enter drone name")
164.         droneName = input()
165.         print("Enter flight path name")
166.         flightPathName = input()
167.         for x in range(len(Drones)):
168.             if(Drones[x].name==droneName):
169.                 for y in range(len(flight_paths)):
170.                     if(flight_paths[y].name==flightPathName):
171.                         Drones[x].startLatitude = flight_paths[y].startLatitude
172.
173.                         Drones[x].startLongitude = flight_paths[y].startLongitud
174. e
175.                         Drones[x].destLatitude = flight_paths[y].destLatitude
176.                         Drones[x].destLongitude = flight_paths[y].destLongitude
177.
178.
179.
180.     #setting up a flight path, flight paths are stored in a list
181.     flightPath1 = FlightPath("flightPath1",52.84083,-6.92611,52.735163726,-
182. 7.01833326,15,5)
183.     flight_paths = [flightPath1]
184.     flight_paths_and_Drones = {"drone1":"flightPath1"}
185.
186.
187.
188.     #print out what drone is associated with which flight path
189.     for x, y in flight_paths_and_Drones.items():
190.         print(x, y)

```

```

182.
183.
184.     #creating a virtual drone and entering flight information
185.     drone1 = VirtualController("drone1",flightPath1.getFlightPath()[1],flightPat
186.     h1.getFlightPath()[2],
187.     flightPath1.getFlightPath()[3],flightPath1.getFlightPath()[4],flightPath1.ge
188.     tFlightPath()[5],
189.     flightPath1.getFlightPath()[6],False,6,0,15)
190.     #self.drone1 = VirtualControllerThree(52.84083,-6.92611,52.735163726,-
191.     7.01833326,20,5,5,6,0,100)
192.     drone2 = VirtualController("drone2",52.87332984, -7.00666664,52.886996452, -
193.     7.1000000,15,5,False,6,0,15)
194.     drone3 = VirtualController("drone3",52.9366929199, -
195.     7.03684318596,52.797141, - 6.895631,15,5,False,6,0,15)
196.
197.     # #carlow gps 52.827766, 6.935186
198.     # self.drone4 = VirtualControllerTwo( 52.827766,-6.935186,52.735163726,-
199.     7.01833326,20,5,5,6,0)
200.
201.     # #array of drones
202.     Drones = [drone1,drone2,drone3]
203.
204.     class App:
205.         def __init__(self):
206.             self._running = True
207.             self._display_surf = None
208.             self._image_surf = None
209.             self._map = None
210.             # Mouse position for testing.
211.             self._mouseX = 40
212.             self._mouseY = 50
213.             self.screen = pygame.display.set_mode((640, 840))
214.             self.clock = pygame.time.Clock()
215.
216.
217.
218.         def on_init(self):
219.             pygame.init()
220.             pygame.font.init()
221.             self._clock = pygame.time.Clock()
222.             self._running = True
223.             self._running = True
224.             self.image_surf = pygame.image.load('testMaps.png').convert()
225.             #setting the font for the buttons
226.             self.text = pygame.font.Font("freesansbold.ttf",20)
227.
228.             #code for the buttons
229.             def text_objects(self,text,font):
230.                 textSurface = font.render(text,True,(0,0,0))
231.                 return textSurface, textSurface.get_rect()
232.
233.
234.             #This method checks if an event has been triggered.
235.             # eg A mouse click on the map
236.             def on_event(self, event):
237.                 if event.type == QUIT:
238.                     self._running = False
239.                 if event.type == pygame.MOUSEBUTTONDOWN:
240.                     self._mouseX, self._mouseY = pygame.mouse.get_pos()
241.                     #self._drone.processEvent(event)

```



```

242.
243.         #This loop moves the drone along its flight path to its destination. it
        checks if it has
244.         # landed or not. if it is not landed, then the method checks to see if t
        he drone is likely to
245.         # be involved in a collision. if the drone is on a course for a collision
        n, then preventative
246.         # measures are taken to try avoid this collision.
247.         def on_loop(self):
248.             i = 0
249.             while i<len(Drones):
250.                 Drones[i].getTelemetry()
251.                 #checking to see if the drone has reached its destination yet
252.                 if(Drones[i].getDistance() <1):
253.                     print("drone less than 1 distance is ", Drones[i].getDistanc
        e())
254.                     if(Drones[i].status==True):
255.                         Drones[i].setCurrentLatitude(Drones[i].getDestLatitude()
        )
256.                         Drones[i].setCurrentLongitude(Drones[i].getDestLongitude
        ())
257.                         print("drone is landed")
258.                         i+=1
259.                         #check status(landed or not) if not landed then land els
        e pass
260.                     else:
261.                         print("drone is about to land")
262.                         print("drone distance about to land", Drones[i].getDista
        nce())
263.                         Drones[i].hover()
264.                         Drones[i].land()
265.                         Drones[i].setCurrentLatitude(Drones[i].getDestLatitude()
        )
266.                         Drones[i].setCurrentLongitude(Drones[i].getDestLongitude
        ())
267.                         Drones[i].status==True
268.                         i+=1
269.
270.
271.
272.
273.                 else:
274.                     if(Drones[i].status==False):
275.                         #pass in the drone i.e Drones[i] to the check collision
        function
276.                         self.checkCollision(Drones[i])
277.                         i+=1
278.                     else:
279.                         i+=1
280.
281.
282.
283.
284.
285.
286.
287.
288.
289.
290.         def checkCollision(self,drone):
291.             m=0
292.
293.             #getting telemetry
294.             currentLat = drone.getCurrentLatitude()
295.             currentLon = drone.getCurrentLongitude()
296.             destLat = drone.getDestLatitude()

```

```

297.         destLong = drone.getDestLongitude()
298.
299.         while m<len(Drones):
300.             currentLat2 = Drones[m].getCurrentLatitude()
301.             currentLon2 = Drones[m].getCurrentLongitude()
302.             destLat2 = Drones[m].getDestLatitude()
303.             destLong2 = Drones[m].getDestLongitude()
304.             line1 = LineString([(currentLat,currentLon),(destLat,destLong)])
305.             line2 = LineString([(currentLat2,currentLon2),(destLat2,destLong
2)]]
306.             print("intersection is ", line1.crosses(line2))
307.             cross = line1.crosses(line2)
308.             #checking if drone paths intersect each other
309.             if(cross):
310.                 print("Paths cross")
311.                 #getting the distances to the drones original destination
312.                 distanceToDestination1 = Drones[m].getDistance()
313.                 distanceToDestination2 = drone.getDistance()
314.                 #getting the intersection point of the drones
315.                 intersectPoint = line1.intersection(line2)
316.                 print(type(intersectPoint))
317.                 print("intersect point is ",intersectPoint)
318.                 #print(intersectPoint)
319.                 #getting the distance to the intersect point of each drone
320.                 distanceToIntersection1 = Drones[m].getDistanceToAPoint(int
ersectPoint.x,intersectPoint.y)
321.                 distanceToIntersection2 = drone.getDistanceToAPoint(interse
ctPoint.x,intersectPoint.y)
322.                 differenceInDistances = abs(distanceToIntersection1-
distanceToIntersection2)
323.                 print("difference in distance is ",differenceInDistances)
324.                 if(differenceInDistances<1):
325.                     print("differences in distance is less than 1 km",differ
enceInDistances)
326.                     if(distanceToDestination1<distanceToDestination2):
327.                         #Drones[m].setAltitude(25)
328.                         #Drones[m].setRatio(25)#increases the size of the re
ctangle to show a change in altitude
329.                         Drones[m].setSpeed(1)
330.                         print("speed has changed")
331.                         m+=1
332.
333.                     else:
334.                         #drone.setAltitude(25)
335.                         #drone.setRatio(25)#increases the size of the rectan
gle to show a change in altitude
336.                         drone.setSpeed(1)
337.                         print("speed has changed")
338.                         m+=1
339.
340.                 elif(differenceInDistances<4):
341.                     print("differences in distance is less than 3 km",differ
enceInDistances)
342.                     if(distanceToDestination1<distanceToDestination2):
343.                         Drones[m].setAltitude(25)
344.                         Drones[m].setRatio(25)#increases the size of the rec
tangle to show a change in altitude
345.                         #Drones[m].setSpeed(1)
346.                         print("atitude has changed")
347.                         m+=1
348.
349.                     else:
350.                         drone.setAltitude(25)
351.                         drone.setRatio(25)#increases the size of the rectang
le to show a change in altitude

```

```

352.             #drone.setSpeed(1)
353.             print("atitude and speed changed")
354.             m+=1
355.
356.         else:
357.             m+=1
358.
359.     else:
360.
361.         m+=1
362.
363.
364.
365.
366.     def on_render(self):
367.
368.         # drawing map on screen
369.         image_surf = pygame.image.load('testMaps.png').convert()
370.         self.screen.blit(image_surf,(0,0))
371.
372.         #converting drone gps to pixel values to render on map
373.         j= 0
374.         while j<len(Drones):
375.             gpsLatConvertToPixel = Drones[j].mapGpsLatToPixels(Drones[j].get
CurrentLatitude())
376.             print("latitude is ", Drones[j].getCurrentLatitude())
377.             print("lat to pixel is ",gpsLatConvertToPixel)
378.             gpsLongConvertToPixel = Drones[j].mapGpsLongToPixels(Drones[j].g
etCurrentLongitude())
379.             print("longitude is ", Drones[j].getCurrentLongitude())
380.             print("long to pixel is ",gpsLongConvertToPixel)
381.             size = Drones[j].getRatio()
382.             pygame.draw.rect(self.screen, (0, 0, 255),(gpsLongConvertToPixel
,gpsLatConvertToPixel,size,size))
383.             textSurf,textRect = self.text_objects(Drones[j].name, self.text)
384.             textRect.center = ((gpsLongConvertToPixel+(20/2)),(gpsLatConvert
ToPixel+(50/2)))
385.             self.screen.blit(textSurf,textRect)
386.             #drawing obstacle
387.             destLatGpsToPixel = int(Drones[j].mapGpsLatToPixels(Drones[j].ge
tDestLatitude()))
388.             destLongGpsToPixel = int(Drones[j].mapGpsLongToPixels(Drones[j].
getDestLongitude()))
389.             pygame.draw.circle(self.screen, (0,255,0), (destLongGpsToPixel,d
estLatGpsToPixel), 10)
390.             pygame.draw.lines(self.screen, (0,255,0),True,((gpsLongConvertTo
Pixel,gpsLatConvertToPixel),(destLongGpsToPixel,destLatGpsToPixel))
391.
392.
393.
394.             #code for rendering buttons
395.             pygame.draw.rect(self.screen,(200,0,0),(50,650,120,50))
396.             textSurf,textRect = self.text_objects("Button", self.text)
397.             textRect.center = ((50+(120/2)),(650+(50/2)))
398.             self.screen.blit(textSurf,textRect)
399.
400.             #the below code checks to see if the button was clicked
401.             mouse = pygame.mouse.get_pos()
402.             if(200+100>mouse[0]>200 and 500+50>mouse[1]>500):
403.                 print("in the box")
404.                 print(mouse)
405.
406.             #more code for rendering the other buttons
407.             pygame.draw.rect(self.screen,(0,200,0),(200,650,120,50))
408.             textSurf,textRect = self.text_objects("Button", self.text)

```

```

409.             textRect.center = ((200+(120/2)),(650+(50/2)))
410.             self.screen.blit(textSurf,textRect)
411.             pygame.draw.rect(self.screen,(0,0,200),(350,650,120,50))
412.             textSurf,textRect = self.text_objects("Button", self.text)
413.             textRect.center = ((350+(120/2)),(650+(50/2)))
414.             self.screen.blit(textSurf,textRect)
415.
416.             j+=1
417.
418.
419.             pygame.display.flip()
420.             pygame.display.update()#display all the updated values to the screen
421.
422.             self.clock.tick(2)#speed at which the values are displayed. a change
in the number
423.             #will increase the speed at which the drones move on the screen.
424.
425.         def on_cleanup(self):
426.             pygame.quit()
427.
428.         def on_execute(self):
429.             if self.on_init() == False:
430.                 self._running = False
431.
432.             while( self._running ):
433.                 for event in pygame.event.get():
434.                     self.on_event(event)
435.                 self.on_loop()
436.                 self.on_render()
437.
438.
439.
440.
441.
442.         if __name__ == "__main__" :
443.             menu()
444.             theApp = App()
445.             theApp.on_execute()

```

VirtualController.py

```

1. #Author: Brendan Mitchell
2. #Student Number: c00220212
3. #Description: This class is responsible for controlling a virtual drone. In this cl
ass a
4. #virtual drone instance is created and all the methods that are needed to
5. #control this virtual drone are listed.
6.
7.
8. #importing required modules
9. from DroneBase import DroneBase
10. #from haversine import haversine, Unit
11. import math
12. import threading
13. from threading import Timer
14. from time import time
15. import math
16. from numpy import interp
17.
18.
19.
20.

```

```

21.
22.
23. class VirtualController(DroneBase):
24.
25.
26.     #variables
27.
28.     OFFSET = 268435456 # half of the earth circumference's in pixels at zoom level
29.     21
30.     RADIUS = OFFSET / math.pi
31.
32.
33.     ratio = 15
34.     status = False
35.
36.     #constructor
37.     def __init__(self,name,startLatitude,startLongitude,destLatitude,destLongitude
    ,altitude,speed,status,model,battery,ratio):
38.         self.name = name
39.         self.startLatitude = startLatitude
40.         self.startLongitude = startLongitude
41.         self.destLatitude = destLatitude
42.         self.destLongitude = destLongitude
43.         self.altitude = altitude
44.         self.speed = speed
45.         self.status = status
46.         self.model = model
47.         self.battery = battery
48.         self.currentLatitude = startLatitude
49.         self.currentLongitude = startLongitude
50.         self.ratio = ratio
51.
52.
53.
54.     #This method returns all the information about the virtual drone
55.     def getDroneInformation(self):
56.         return (self.name,
57.                 self.startLatitude,
58.                 self.startLongitude,
59.                 self.destLatitude,
60.                 self.destLongitude,
61.                 self.altitude,
62.                 self.speed,
63.                 self.status,
64.                 self.model,
65.                 self.battery,
66.                 self.currentLatitude,
67.                 self.currentLongitude)
68.
69.
70.     def takeOff(self):
71.         self.altitude = 5
72.
73.     def setAltitude(self,alt):
74.         self.altitude = alt
75.
76.
77.     def land(self):
78.         self.altitude = 0
79.         self.setStatus()
80.
81.     #to see if drone is landed. status is set to true when the drone is landed
82.     def setStatus(self):
83.         self.status = True
84.

```

```

85.     def getStatus(self):
86.         return self.status
87.
88.     def getName(self):
89.         return self.name
90.
91.     def setName(self,name):
92.         self.name = name
93.
94.
95.     def moveForward(self):
96.         self.speed = 5
97.
98.     def hover(self):
99.         self.speed = 0
100.
101.         #get the distance from the start coordinates to the destination coordina
tes
102.         def getDistance(self):
103.
104.             radius = 6371 # km
105.
106.             dlat = math.radians(self.destLatitude-
self.currentLatitude)#use currentLatitude instead of startLatitude
107.             dlon = math.radians(self.destLongitude-
self.currentLongitude)#use currentLatitude instead of startLatitude
108.             a = math.sin(dlat/2) * math.sin(dlat/2) + math.cos(math.radians(self
.currentLatitude)) \
109.             * math.cos(math.radians(self.destLatitude)) * math.sin(dlon/2) *
math.sin(dlon/2)
110.             c = 2 * math.atan2(math.sqrt(a), math.sqrt(1-a))
111.             d = radius * c
112.
113.             self.distance = d
114.             return d
115.
116.         '''this method returns the distance from current position to the new d
estination.
117.         The new destination is mainly the interesection of the drone paths.'
''
118.
119.     def getDistanceToAPoint(self,newDestinationLat,newDestinationLong):
120.
121.         radius = 6371 # km
122.
123.         dlat = math.radians(newDestinationLat-self.currentLatitude)
124.         dlon = math.radians(newDestinationLong-self.currentLongitude)
125.         a = math.sin(dlat/2) * math.sin(dlat/2) + math.cos(math.radians(self
.currentLatitude)) \
126.         * math.cos(math.radians(newDestinationLat)) * math.sin(dlon/2) *
math.sin(dlon/2)
127.         c = 2 * math.atan2(math.sqrt(a), math.sqrt(1-a))
128.         d = radius * c
129.
130.         self.distance = d
131.         return d
132.
133.
134.
135.     def returnHome(self):
136.         pass
137.
138.     def displayCameraFeed(self):
139.         pass
140.
141.         #updating telemetry

```

```

142.         def updateTelemetry(self):
143.             newGpsCoordinates = self.gpsCoordinatesUpdate()
144.             newDistance = self.getDistance()
145.
146.
147.             return newGpsCoordinates,newDistance
148.
149.
150.         def getTelemetry(self):
151.             return self.updateTelemetry()
152.
153.         #This method creates a vector based on the coordinates of the drone
154.         def createVectorFromCoordinates(self):
155.             self.firstComponent = self.destLatitude - self.startLatitude
156.             self.secondComponent = self.destLongitude - self.startLongitude
157.
158.             return self.firstComponent, self.secondComponent
159.
160.         #returns the latitude part(first component) of vector
161.         def getVectorLatitude(self):
162.             firstComponent = self.destLatitude - self.startLatitude
163.             return firstComponent
164.
165.         #returns the longitude part(second component) of vector
166.         def getVectorLongitude(self):
167.             self.secondComponent = self.destLongitude - self.startLongitude
168.             return self.secondComponent
169.
170.         #returns the length of vector
171.         def magnitudeOfVector(self):
172.             #addComponents = ((self.firstComponent*self.firstComponent) + (self.
173. secondComponent*self.secondComponent))
174.             addComponents = ((self.getVectorLatitude()*self.getVectorLatitude())
175. + (self.getVectorLongitude()*self.getVectorLongitude()))
176.             magnitude = math.sqrt(addComponents)
177.             return magnitude
178.
179.         #updates the gps coordinates as the drone moves along vector
180.         def gpsCoordinatesUpdate(self):
181.             '''ratio was used first and then speed was used to update the coor
182. dinates along the vector'''
183.
184.             if(self.getSpeed()!=0):
185.                 self.currentLatitude = self.currentLatitude + (self.getVectorLat
186. itude()/(self.getSpeed()*60))
187.                 self.currentLongitude = self.currentLongitude + (self.getVectorL
188. ongitude()/(self.getSpeed()*60))
189.             else:
190.                 self.currentLatitude = self.currentLatitude
191.                 self.currentLongitude = self.currentLongitude
192.                 self.newGps = self.currentLatitude,self.currentLongitude
193.
194.
195.             return self.newGps
196.
197.
198.         def setSpeed(self, speed):
199.             self.speed = speed
200.
201.         def getSpeed(self):
202.             return self.speed

```

```

203.
204.         def setRatio(self,newRatio):
205.             self.ratio = newRatio
206.
207.
208.         def getRatio(self):
209.             return self.ratio
210.
211.
212.         def getState(self):
213.             if self.altitude==0:
214.                 self.state="landed"
215.             else:
216.                 self.state="flying"
217.
218.             return self.state
219.
220.
221.         def getAltitude(self):
222.             return self.altitude
223.
224.
225.         def setStartLatitude(self, latitude):
226.             self.startLatitude = latitude
227.
228.         def setStartLongitude(self, longitude):
229.             self.startLongitude = longitude
230.
231.         def setDestLatitude(self, latitude):
232.             self.destLatitude = latitude
233.
234.         def setDestLongitude(self, longitude):
235.             self.destLongitude = longitude
236.
237.         def getstartLongitude(self):
238.             return self.startLongitude
239.
240.         def getCurrentLatitude(self):
241.             return self.currentLatitude
242.
243.         def getCurrentLongitude(self):
244.             return self.currentLongitude
245.
246.         def setCurrentLatitude(self,lat):
247.             self.currentLatitude = lat
248.
249.         def setCurrentLongitude(self,lon):
250.             self.currentLongitude = lon
251.
252.         def getstartLatitude(self):
253.             return self.startLatitude
254.
255.         def getDestLongitude(self):
256.             return self.destLongitude
257.
258.         def getDestLatitude(self):
259.             return self.destLatitude
260.
261.         def getBatteryLife(self):
262.             return self.battery
263.
264.         #This method returns the bearing of the drone
265.         def getDirection(self):
266.             startLat = math.radians(self.startLatitude)
267.             startLong = math.radians(self.startLongitude)
268.             endLat = math.radians(self.destLatitude)

```



```

269.         endLong = math.radians(self.destLongitude)
270.
271.         dLong = endLong - startLong
272.
273.         dPhi = math.log(math.tan(endLat/2.0+math.pi/4.0)/math.tan(startLat/2
274.         .0+math.pi/4.0))
275.         if abs(dLong) > math.pi:
276.             if dLong > 0.0:
277.                 dLong = -(2.0 * math.pi - dLong)
278.             else:
279.                 dLong = (2.0 * math.pi + dLong)
280.         bearing = (math.radians(math.atan2(dLong, dPhi)) + 360.0) % 360.0
281.
282.         return bearing #radians
283.
284.
285.     def rotateLeft(self):
286.         self.currentdirection -=1
287.
288.     def rotateRight(self):
289.         self.currentdirection +=1
290.
291.
292.     def setDirection(self, direction):
293.         self.direction = self.getDirection()
294.
295.
296.
297.
298.         '''This function maps the coordinates given in latitude and longitude
299.         and converts them to
300.         pixel positions.'''
301.     def mapGpsToPixels(self,lat,lon):
302.
303.         '''52.967109 is horizontal corner of the static map and 7.159379 i
304.         s the other corner
305.         that runs vertically.'''
306.         latDifference = (52.967109 - lat)*1000000
307.         lonDifference = (7.159379 - lon)*1000000
308.
309.         '''243435 is the difference in latitude from one side of the map t
310.         o the other.
311.         640 is the width of the static map in pixels.The interp function map
312.         s the latitude to the
313.         pixel size of the map.'''
314.         latInPixels = interp(latDifference,[0,243435],[0,640])
315.         longInPixels = interp(lonDifference,[0,459365],[0,640])
316.
317.         return longInPixels,latInPixels
318.
319.         '''This function returns the latitude coordinate in pixel positions on
320.         the map.'''
321.     def mapGpsLatToPixels(self,lat):
322.
323.         latDifference = (52.967109 - lat)*1000000
324.         latInPixels = interp(latDifference,[0,243435],[0,640])
325.
326.         return latInPixels
327.
328.         '''This function returns the longitude coordinate in pixel positions o
329.         n the map.'''
330.     def mapGpsLongToPixels(self,lon):
331.
332.         lonDifference = (7.159379 + lon)*1000000

```

```

328.         longInPixels = interp(lonDifference,[0,459365],[0,640])
329.
330.
331.
332.         return longInPixels
333.
334.
335.
336.
337.

```

BebopTestConnect.py

```

1. """
2. checks the connection to the Bebop
3.
4. Author: Brendan Mitchell
5. Student number: c00220212
6. Description: This class was used to test the connection to the bebop drone
7. """
8.
9. from pyparrot.Bebop import Bebop
10. import math
11. import threading
12. from threading import Timer
13. from time import time
14. import math
15. from numpy import interp
16.
17. #import pyparrot modules
18. import time
19. from pyparrot.networking.wifiConnection import WifiConnection
20. from pyparrot.utils.colorPrint import color_print
21. from pyparrot.commandsandsensors.DroneCommandParser import DroneCommandParser
22. from pyparrot.commandsandsensors.DroneSensorParser import DroneSensorParser
23. from datetime import datetime
24. from pyparrot.Bebop import Bebop
25. from BebopDroneController import BebopDroneController
26.
27. bebop = BebopDroneController()
28.
29. print("connecting to bebop")
30. bebop.BebopConnect()
31. print(bebop.BebopConnect())
32. bebop.startVideo()
33. print("video has started")
34. bebop.stopVideo()
35. print("video has stopped")
36. print("disconnecting bebop")
37. bebop.disconnect()

```

BebopDroneController.py

```
1. #Author: Brendan Mitchell
2. #Student Number: c00220212
3. #Description: This class is responsible for controlling a bebop drone. This class
4. #inherits from the Bebop class which contains all the methods to create and control
5. #a parrot Bebop2 drone.
6.
7.
8.
9. from DroneBase import DroneBase
10. #from haversine import haversine, Unit
11. import math
12. import threading
13. from threading import Timer
14. from time import time
15. import math
16. from numpy import interp
17.
18. #import pyparrot modules
19. import time
20. from pyparrot.networking.wifiConnection import WifiConnection
21. from pyparrot.utils.colorPrint import color_print
22. from pyparrot.commandsandsensors.DroneCommandParser import DroneCommandParser
23. from pyparrot.commandsandsensors.DroneSensorParser import DroneSensorParser
24. from datetime import datetime
25. from pyparrot.Bebop import Bebop
26.
27.
28.
29.
30. #inherits from the Bebop class
31. class BebopDroneController(Bebop):
32.     def BebopConnect(self):
33.         self.connect(5)
34.
35.     def BebopDisconnect(self):
36.         self.disconnect()
37.
38.     def moveForward(self):
39.         print("Flying direct: going forward (positive pitch)")
40.         self.fly_direct(roll=0, pitch=50, yaw=0, vertical_movement=0, duration=1)
41.
42.     def takeOff(self):
43.         self.safe_takeoff(5)
44.
45.     def land(self):
46.         self.safe_land(5)
```

```

47.
48.     def hover(self):
49.         self.fly_direct(roll=0, pitch=0, yaw=0, vertical_movement=0, duration=1)
50.
51.     def rotateLeft(self):
52.         pass
53.
54.     def startVideo(self):
55.         self.start_video_stream()
56.
57.     def stopVideo(self):
58.         self.stop_video_stream()
59.
60.
61.

```

DroneBase.py

```

1. #Author: Brendan Mitchell
2. #Student Number: c00220212
3. #Description: This class is an abstract class. All other classes that inherit from
   this
4. #class must use the abstract methods below.
5.
6.
7.
8. from abc import ABC, abstractmethod
9. class DroneBase(ABC):
10.
11.     #initialising variables
12.     startLatitude=0
13.     startLongitude = 0
14.     destLatitude = 0
15.     destLongitude = 0
16.     altitude = 0
17.     battery = 100
18.     speed = 0
19.     status = 0
20.     model = 0
21.     direction = 0
22.     currentdirection = 0
23.     distance = 0
24.     currentTime = 0
25.
26.
27.
28.     @abstractmethod
29.     def takeOff(self):
30.         pass
31.
32.     @abstractmethod
33.     def land(self):
34.         pass
35.     @abstractmethod
36.     def moveForward(self):
37.         pass

```

```

38.
39.     @abstractmethod
40.     def hover(self):
41.         pass
42.
43.     @abstractmethod
44.     def setDirection(self):
45.         pass
46.
47.     @abstractmethod
48.     def rotateLeft(self):
49.         pass
50.
51.     @abstractmethod
52.     def rotateRight(self):
53.         pass
54.
55.     @abstractmethod
56.     def getDistance(self):
57.         pass
58.
59.     @abstractmethod
60.     def getDistanceToPoint(self):
61.         pass
62.
63.     @abstractmethod
64.     def returnHome(self):
65.         pass
66.
67.     @abstractmethod
68.     def displayCameraFeed(self):
69.         pass
70.
71.     @abstractmethod
72.     def getTelemetry(self):
73.         pass
74.
75.     @abstractmethod
76.     def setSpeed(self, speed):
77.         pass
78.
79.     @abstractmethod
80.     def getSpeed(self):
81.         pass
82.
83.     @abstractmethod
84.     def setState(self, state):
85.         pass
86.
87.     @abstractmethod
88.     def getState(self):
89.         pass
90.
91.     @abstractmethod
92.     def getAltitude(self):
93.         pass
94.
95.     @abstractmethod
96.     def setStartLatitude(self, latitude):
97.         pass
98.
99.     @abstractmethod
100.        def setStartLongitude(self, longitude):
101.            pass
102.
103.        @abstractmethod

```

```

104.         def gpsCoordinatesUpdate(self):
105.             pass
106.
107.
108.         @abstractmethod
109.         def setDestLatitude(self, latitude):
110.             pass
111.
112.         @abstractmethod
113.         def setDestLongitude(self, longitude):
114.             pass
115.
116.         @abstractmethod
117.         def getstartLongitude(self):
118.             pass
119.
120.
121.
122.         @abstractmethod
123.         def mapGpsToPixels(self, lat, long):
124.             pass
125.
126.
127.         @abstractmethod
128.         def mapGpsLatToPixels(self, lat):
129.             pass
130.
131.
132.         @abstractmethod
133.         def mapGpsLongToPixels(self, long):
134.             pass
135.
136.
137.         @abstractmethod
138.         def createVectorFromCoordinates(self):
139.             pass

```

FlightPath.py

```

1. #Author: Brendan Mitchell
2. #Student Number: c00220212
3. #Description: This class sets up flight paths that the drone can use to navigate fr
   om a starting
4. # point to a destination.
5.
6.
7. #importing necessary modules
8. import pygame, sys, math, random
9. from pygame.locals import *
10. import pygame
11. from pygame.locals import *
12. from PIL import Image, ImageDraw
13. #from Virtual_controller import Virtual_controller
14. #from VirtualControllerTwo import VirtualControllerTwo
15. #from VirtualControllerThree import VirtualControllerThree
16. from VirtualController import VirtualController
17. import numpy as np
18. from shapely.geometry import LineString

```

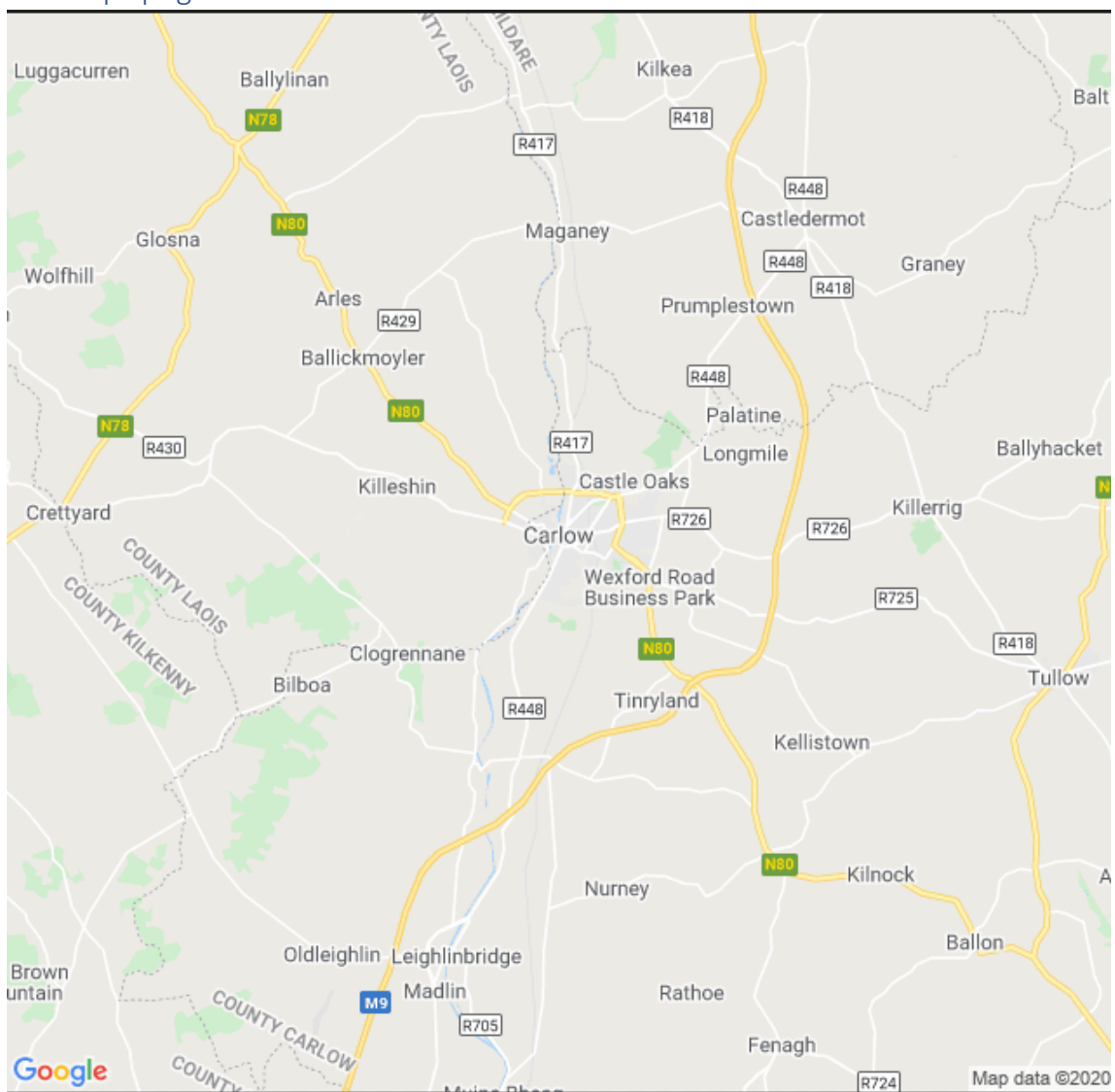
```

19.
20. import pygame
21. from pygame.locals import *
22. from PIL import Image, ImageDraw
23. #from Virtual_controller import Virtual_controller
24.
25.
26.
27. class FlightPath:
28.
29.
30.     #constuctor
31.     def __init__(self,name,startLatitude,startLongitude,destLatitude,destLongitude
,altitude,speed):
32.         self.name = name
33.         self.startLatitude = startLatitude
34.         self.startLongitude = startLongitude
35.         self.destLatitude = destLatitude
36.         self.destLongitude = destLongitude
37.         self.altitude = altitude
38.         self.speed = speed
39.
40.     #This method returns all the relevant attributes
41.     def getFlightPath(self):
42.         return (self.name,
43.                 self.startLatitude,
44.                 self.startLongitude,
45.                 self.destLatitude,
46.                 self.destLongitude,
47.                 self.altitude,
48.                 self.speed)
49.
50.     #Below are the getters and setters for the class
51.     def getFlightPathName(self):
52.         return self.name
53.
54.     def getStartLat(self):
55.         return self.startLatitude
56.
57.     def getStartLon(self):
58.         return self.startLongitude
59.
60.     def getDestLat(self):
61.         return self.destLatitude
62.
63.     def getDestLon(self):
64.         return self.destLongitude
65.
66.     def getAltitude(self):
67.         return self.altitude
68.
69.     def getSpeed(self):
70.         return self.speed
71.
72.     def setStartLat(self,lat):
73.         self.startLatitude = lat
74.
75.     def setStartLon(self,lon):
76.         self.startLongitude = lon
77.
78.     def setDestLat(self,Dlat):
79.         self.destLatitude = Dlat
80.
81.     def setDestLon(self,Dlon):
82.         self.destLongitude = Dlon
83.

```

```
84. def setAltitude(self, alt):
85.     self.altitude = alt
86.
87. def setSpeed(self, spd):
88.     self.speed = spd
89.
```

testMaps.png



Plagiarism form



Work submitted for assessment which does not include this declaration will not be assessed.

DECLARATION

*I declare that all material in this submission e.g. thesis/essay/project/assignment is entirely my/our own work except where duly acknowledged.

*I have cited the sources of all quotations, paraphrases, summaries of information, tables, diagrams or other material, including software and other electronic media in which intellectual property rights may reside.

*I have provided a complete bibliography of all works and sources used in the preparation of this submission.

*I understand that failure to comply with the Institute's regulations governing plagiarism constitutes a serious offence.

Student Name: (Printed) BRENDAN MITCHELL
Student Number(s): 00 220 212
Signature(s): Brendan Mitchell
Date: 20/4/19

Please note:

- a) * Individual declaration is required by each student for joint projects.
- b) Where projects are submitted electronically, students are required to type their name under signature.
- c) The Institute regulations on plagiarism are set out in Section 10 of Examination and Assessment Regulations published each year in the Student Handbook.