DRONE AIR TRAFFIC CONTROL SYSTEM

Technical manual

By

Brendan Mitchell

Title: Drone Air Traffic Control System – Technical Manual

Author: Brendan Mitchell(c00220212)

Supervisor: Dr. Oisin 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

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

```python
#drones are rendered on the map. The destination point are alos rendered on the map. This class
# also contains the menu which the user can use to select an option such as create a virtual drone
# enter a flight path, assign a flight path, get information about a drone or information
# contained in a flight path and the user can then display the drones created on the map and
# see their journey from the starting coordinates to its destination.


#importing all the relevant modules
import pygame, sys, math, random
from pygame.locals import *
import pygame
from pygame.locals import *
from PIL import Image, ImageDraw
#from Virtual_controller import Virtual_controller
#from VirtualControllerTwo import VirtualControllerTwo
#from VirtualControllerThree import VirtualControllerThree
from VirtualController import VirtualController
from FlightPath import FlightPath
import numpy as np
from shapely.geometry import LineString
from numpy import interp

import pygame
from pygame.locals import *
from PIL import Image, ImageDraw
#from Virtual_controller import Virtual_controller

spriteSize =15

#User menu
def menu():
    loop_condition = True
    while loop_condition == True:
        print("\nWelcome to drone air traffic control system!")
        print("\nPlease enter a number for what you want to do.")
        print("Enter 1 to add drone.")
        print("Enter 2 to create a flight path.")
        print("Enter 3 to search drones")
        print("Enter 4 to search flight paths")
        print("Enter 5 to view flight path information")
        print("Enter 6 to assign a  flight path")
        print("Enter 7 to view drone information")
        print("Enter 8 to quit menu and view map .\n")

        main_input = int(input("What would you like to do? "))

        if main_input == 8:
            loop_condition = False
            break
        elif main_input == 1:
            addDrone()
        elif main_input == 2:
            createFightPath()
        elif main_input == 3:
            searchDrones()
        elif main_input == 4:
            searchFlightPaths()
        elif main_input ==5:
            viewFlightPathInformation()
        elif main_input == 6:
            assignFlightPath()
        elif main_input == 7:
            viewDroneInformation()

#creating a virtual drone
def addDrone():
    print("enter name for drone")
    name  = input()
    print("enter start latitude from 52.967109 to 52.723674")
    lat = float(input())
    print("enter start longitude from -7.159379 to -6.700014")
    lon = float(input())
    print("enter destination latitude from 52.967109 to 52.723674")
    destLat = float(input())
    print("enter destination longitude from -7.159379 to -6.700014")
    destLong = float(input())
    #self.name =  VirtualControllerThree(52.84083,-6.92611,52.735163726,-7.01833326,20,5,5,6,0,100)
    name =  VirtualController(name,lat,lon,destLat,destLong,15,5,False,6,0,15)
    Drones.append(name)

#creating a flight path
def createFightPath():
    print("enter name for flight path")
    flightPath  = input()
    print("enter start latitude from 52.967109 to 52.723674")
    lat = float(input())
    print("enter start longitude from -7.159379 to -6.700014")
    lon = float(input())
    print("enter destination latitude from 52.967109 to 52.723674")
    destLat = float(input())
    print("enter destination longitude from -7.159379 to -6.700014")
    destLong = float(input())
    #self.name =  VirtualControllerThree(52.84083,-6.92611,52.735163726,-7.01833326,20,5,5,6,0,100)
    flightPath =  FlightPath(flightPath,lat,lon,destLat,destLong,15,5)
    flight_paths .append(flightPath)

       #searching flight paths in the system
       def searchFlightPaths():
           for x in range(len(flight_paths)):
               print (flight_paths[x].getFlightPathName())

       #searching drones in the system
       def searchDrones():
            for x in range(len(Drones)):
               print (Drones[x].getName())

       #Viewing information on the chosen flight path
       def viewFlightPathInformation():
           print("enter flight path name")
           inputFlightPathName = input()
           for x in range(len(flight_paths)):
           #print(flight_paths[x].getFlightPathName)
               if(flight_paths[x].name==inputFlightPathName):
                   print("")
                   print("filght path name is ", flight_paths[x].getFlightPath()[0])
                   print("starting latitude is ", flight_paths[x].getFlightPath()[1])
                   print("starting longitude is ", flight_paths[x].getFlightPath()[2])
                   print("destination latitude is ", flight_paths[x].getFlightPath()[3])
                   print("destination longitude is ",flight_paths[x].getFlightPath()[4])
                   print("altitude is ", flight_paths[x].getFlightPath()[5])
                   print("speed is ", flight_paths[x].getFlightPath()[6])







       #Viewing information on the chosen drone
       def viewDroneInformation():
           print("enter of drone name")
           droneName = input()
           for x in range(len(Drones)):
           #print(flight_paths[x].getFlightPathName)
               if(Drones[x].name==droneName):
                   print("")
                   print("drone name is ", Drones[x].getDroneInformation()[0])
                   print("starting latitude is ", Drones[x].getDroneInformation()[1])
                   print("starting longitude is ", Drones[x].getDroneInformation()[2])
                   print("destination latitude is ", Drones[x].getDroneInformation()[3])
                   print("destination longitude is ", Drones[x].getDroneInformation()[4])
                   print("altitude is ", Drones[x].getDroneInformation()[5])
                   print("speed is ", Drones[x].getDroneInformation()[6])
                   print("status is ", Drones[x].getDroneInformation()[7])
                   print("model number is ", Drones[x].getDroneInformation()[8])
                   print("battery is ", Drones[x].getDroneInformation()[9])
                   print("current latitude is ", Drones[x].getDroneInformation()[10])
                   print("current longitude is ", Drones[x].getDroneInformation()[11])


       #assigning a flight path to a drone
       def assignFlightPath():
           print("Enter drone name")
           droneName = input()
           print("Enter flight path name")
           flightPathName = input()
           for x in range(len(Drones)):
               if(Drones[x].name==droneName):
                   for y in range(len(flight_paths)):
                       if(flight_paths[y].name==flightPathName):
                           Drones[x].startLatitude = flight_paths[y].startLatitude
                           Drones[x].startLongitude = flight_paths[y].startLongitude
                           Drones[x].destLatitude = flight_paths[y].destLatitude
                           Drones[x].destLongitude = flight_paths[y].destLongitude



       #setting up a flight path, flight paths are stored in a list
       flightPath1 = FlightPath("flightPath1",52.84083,-6.92611,52.735163726,-7.01833326,15,5)
       flight_paths = [flightPath1]
       flight_paths_and_Drones = {"drone1":"flightPath1"}



       #print out what drone is associated with which flight path
       for x, y in flight_paths_and_Drones.items():
           print(x, y)


       #creating a virtual drone and entering flight information
       drone1 = VirtualController("drone1",flightPath1.getFlightPath()[1],flightPath1.getFlightPath()[2],
       flightPath1.getFlightPath()[3],flightPath1.getFlightPath()[4],flightPath1.getFlightPath()[5],
       flightPath1.getFlightPath()[6],False,6,0,15)

       #self.drone1 = VirtualControllerThree(52.84083,-6.92611,52.735163726,-7.01833326,20,5,5,6,0,100)
       drone2 = VirtualController("drone2",52.87332984, -7.00666664,52.886996452, -7.1000000,15,5,False,6,0,15)
       drone3 = VirtualController("drone3",52.9366929199, -7.03684318596,52.797141, - 6.895631,15,5,False,6,0,15)

       # #carlow gps 52.827766, 6.935186
       # self.drone4 = VirtualControllerTwo( 52.827766,-6.935186,52.735163726,-7.01833326,20,5,5,6,0)

       # #array of drones
       Drones = [drone1,drone2,drone3]






       class App:
           def __init__(self):
               self._running = True
               self._display_surf = None
               self._image_surf = None
               self._map = None
               # Mouse position for testing.
               self._mouseX = 40
               self._mouseY = 50
               self.screen = pygame.display.set_mode((640, 840))
               self.clock = pygame.time.Clock()



           def on_init(self):
               pygame.init()
               pygame.font.init()
               self._clock = pygame.time.Clock()
               self._running = True
               self._running = True
               self.image_surf = pygame.image.load('testMaps.png').convert()
                #setting the font for the buttons
               self.text = pygame.font.Font("freesansbold.ttf",20)

           #code for the buttons
           def text_objects(self,text,font):
               textSurface = font.render(text,True,(0,0,0))
               return  textSurface, textSurface.get_rect()


           #This method checks if an event has been triggered.
           # eg A mouse click on the map
           def on_event(self, event):
               if event.type == QUIT:
                   self._running = False
               if event.type == pygame.MOUSEBUTTONUP:
                   self._mouseX, self._mouseY = pygame.mouse.get_pos()
                   #self._drone.processEvent(event)

           #This loop moves the drone along its flight path to its destination. it checks if it has
           # landed or not. if it is not landed, then the method checks to see if the drone is likely to
           # be involved in a collision. if the drone is on a course for a collision, then preventative
           # measures are taken to try avoid this collision.
           def on_loop(self):
               i = 0
               while i<len(Drones):
                   Drones[i].getTelemetry()
                   #checking to see if the drone has reached its destination yet
                   if(Drones[i].getDistance() <1):
                       print("drone less than 1 distance is ", Drones[i].getDistance())
                       if(Drones[i].status==True):
                           Drones[i].setCurrentLatitude(Drones[i].getDestLatitude())
                           Drones[i].setCurrentLongitude(Drones[i].getDestLongitude())
                           print("drone is landed")
                           i+=1
                           #check status(landed or not) if not landed then land else pass
                       else:
                           print("drone is about to land")
                           print("drone distance about to land", Drones[i].getDistance())
                           Drones[i].hover()
                           Drones[i].land()
                           Drones[i].setCurrentLatitude(Drones[i].getDestLatitude())
                           Drones[i].setCurrentLongitude(Drones[i].getDestLongitude())
                           Drones[i].status==True
                           i+=1




                   else:
                       if(Drones[i].status==False):
                           #pass in the drone i.e Drones[i] to the check collision function
                           self.checkCollision(Drones[i])
                           i+=1
                       else:
                           i+=1











           def checkCollision(self,drone):
               m=0

               #getting telemetry
               currentLat = drone.getCurrentLatitude()
               currentLon = drone.getCurrentLongitude()
               destLat = drone.getDestLatitude()
```

```python
                destLong = drone.getDestLongitude()

                while m<len(Drones):
                    currentLat2 = Drones[m].getCurrentLatitude()
                    currentLon2 = Drones[m].getCurrentLongitude()
                    destLat2 = Drones[m].getDestLatitude()
                    destLong2 = Drones[m].getDestLongitude()
                    line1 = LineString([(currentLat,currentLon),(destLat,destLong)])

                    line2 = LineString([(currentLat2,currentLon2),(destLat2,destLong2)])
                    print("intersection is ", line1.crosses(line2))
                    cross =  line1.crosses(line2)
                    #checking if drone paths intersect each other
                    if(cross):
                        print("Paths cross")
                        #getting the distances to the drones original destination
                        distanceToDestination1 = Drones[m].getDistance()
                        distanceToDestination2 = drone.getDistance()
                        #getting the intersection point of the drones
                        intersectPoint = line1.intersection(line2)
                        print(type(intersectPoint))
                        print("intersect point is ",intersectPoint)
                        #print(intersectPoint)
                        #getting the distance to the intersect point of each drone
                        distanceToIntersection1 =  Drones[m].getDistanceToAPoint(intersectPoint.x,intersectPoint.y)
                        distanceToIntersection2 =  drone.getDistanceToAPoint(intersectPoint.x,intersectPoint.y)
                        differenceInDistances = abs(distanceToIntersection1-distanceToIntersection2)
                        print("difference in distance is ",differenceInDistances)
                        if(differenceInDistances<1):
                            print("differences in distance is less than 1 km",differenceInDistances)
                            if(distanceToDestination1<distanceToDestination2):
                                #Drones[m].setAltitude(25)
                                #Drones[m].setRatio(25)#increases the size of the rectangle to show a change in altitude
                                Drones[m].setSpeed(1)
                                print("speed has changed")
                                m+=1

                            else:
                                #drone.setAltitude(25)
                                #drone.setRatio(25)#increases the size of the rectangle to show a change in altitude
                                drone.setSpeed(1)
                                print("speed has changed")
                                m+=1

                        elif(differenceInDistances<4):
                            print("differences in distance is less than 3 km",differenceInDistances)
                            if(distanceToDestination1<distanceToDestination2):
                                Drones[m].setAltitude(25)
                                Drones[m].setRatio(25)#increases the size of the rectangle to show a change in altitude
                                #Drones[m].setSpeed(1)
                                print("atitude has changed")
                                m+=1

                            else:
                                drone.setAltitude(25)
                                drone.setRatio(25)#increases the size of the rectangle to show a change in altitude
                        #drone.setSpeed(1)
                        print("atitude and speed changed")
                        m+=1

                else:
                    m+=1

            else:

                m+=1




    def on_render(self):

        # drawing map on screen
        image_surf = pygame.image.load('testMaps.png').convert()
        self.screen.blit(image_surf,(0,0))

        #converting drone gps to pixel values to render on map
        j= 0
        while j<len(Drones):
            gpsLatConvertToPixel = Drones[j].mapGpsLatToPixels(Drones[j].getCurrentLatitude())
            print("latitude is ", Drones[j].getCurrentLatitude())
            print("lat to pixel is ",gpsLatConvertToPixel)
            gpsLongConvertToPixel = Drones[j].mapGpsLongToPixels(Drones[j].getCurrentLongitude())
            print("longitude is ", Drones[j].getCurrentLongitude())
            print("long to pixel is ",gpsLongConvertToPixel)
            size = Drones[j].getRatio()
            pygame.draw.rect(self.screen, (0, 0, 255),(gpsLongConvertToPixel,gpsLatConvertToPixel,size,size))
            textSurf,textRect = self.text_objects(Drones[j].name, self.text)

            textRect.center = ((gpsLongConvertToPixel+(20/2)),(gpsLatConvertToPixel+(50/2)))
            self.screen.blit(textSurf,textRect)
            #drawing obstacle
            destLatGpsToPixel = int(Drones[j].mapGpsLatToPixels(Drones[j].getDestLatitude()))
            destLongGpsToPixel = int(Drones[j].mapGpsLongToPixels(Drones[j].getDestLongitude()))
            pygame.draw.circle(self.screen, (0,255,0), (destLongGpsToPixel,destLatGpsToPixel), 10)
            pygame.draw.lines(self.screen, (0,255,0),True,((gpsLongConvertToPixel,gpsLatConvertToPixel),(destLongGpsToPixel,destLatGpsToPixel)))



            #code for rendering buttons
            pygame.draw.rect(self.screen,(200,0,0),(50,650,120,50))
            textSurf,textRect = self.text_objects("Button", self.text)
            textRect.center = ((50+(120/2)),(650+(50/2)))
            self.screen.blit(textSurf,textRect)

            #the below code checks to see if the button was clicked
            mouse = pygame.mouse.get_pos()
            if(200+100>mouse[0]>200 and 500+50>mouse[1]>500):
                print("in the box")
                print(mouse)

           #more code for rendering the other buttons
            pygame.draw.rect(self.screen,(0,200,0),(200,650,120,50))
            textSurf,textRect = self.text_objects("Button", self.text)
```

```python
            textRect.center = ((200+(120/2)),(650+(50/2)))
            self.screen.blit(textSurf,textRect)
            pygame.draw.rect(self.screen,(0,0,200),(350,650,120,50))
            textSurf,textRect = self.text_objects("Button", self.text)
            textRect.center = ((350+(120/2)),(650+(50/2)))
            self.screen.blit(textSurf,textRect)

            j+=1


        pygame.display.flip()
        pygame.display.update()#display all the updated values to the screen

        self.clock.tick(2)#speed at which the values are displayed. a change in the number
        #will increase the speed at which the drones move on the screen.


    def on_cleanup(self):
        pygame.quit()

    def on_execute(self):
        if self.on_init() == False:
            self._running = False

        while( self._running ):
            for event in pygame.event.get():
                self.on_event(event)
            self.on_loop()
            self.on_render()





if __name__ == "__main__" :
    menu()
    theApp = App()
    theApp.on_execute()
```

VirtualController.py

```python
#Author: Brendan Mitchell
#Student Number: c00220212
#Description: This class is responsible for controlling a virtual drone. In this class a
#virtual drone instance is created and all the methods that are needed to
#control this virtual drone are listed.


#importing required modules
from DroneBase import DroneBase
#from haversine import haversine, Unit
import math
import threading
from threading import Timer
from time import time
import math
from numpy import interp




```

```python


class VirtualController(DroneBase):


   #variables

    OFFSET = 268435456 # half of the earth circumference's in pixels at zoom level 21
    RADIUS = OFFSET / math.pi



    ratio = 15
    status = False

    #constructor
    def  __init__(self,name,startLatitude,startLongitude,destLatitude,destLongitude,altitude,speed,status,model,battery,ratio):
        self.name = name
        self.startLatitude = startLatitude
        self.startLongitude = startLongitude
        self.destLatitude = destLatitude
        self.destLongitude = destLongitude
        self.altitude = altitude
        self.speed = speed
        self.status = status
        self.model = model
        self.battery = battery
        self.currentLatitude = startLatitude
        self.currentLongitude = startLongitude
        self.ratio = ratio



    #This method returns all the information about the virtual drone
    def getDroneInformation(self):
        return  (self.name,
        self.startLatitude,
        self.startLongitude,
        self.destLatitude,
        self.destLongitude,
        self.altitude,
        self.speed,
        self.status,
        self.model,
        self.battery,
        self.currentLatitude,
        self.currentLongitude)


    def takeOff(self):
        self.altitude  = 5

    def setAltitude(self,alt):
        self.altitude = alt


    def land(self):
        self.altitude = 0
        self.setStatus()

    #to see if drone is landed. status is set to true when the drone is landed
    def setStatus(self):
        self.status = True

    def getStatus(self):
        return self.status

    def getName(self):
        return self.name

    def setName(self,name):
        self.name = name


    def moveForward(self):
        self.speed = 5

    def hover(self):
        self.speed = 0

           #get the distance from the start coordinates to the destination coordinates
           def getDistance(self):

               radius = 6371 # km

               dlat = math.radians(self.destLatitude-self.currentLatitude)#use currentLatitude instead of startLatitude
               dlon = math.radians(self.destLongitude-self.currentLongitude)#use currentLatitude instead of startLatitude
               a = math.sin(dlat/2) * math.sin(dlat/2) + math.cos(math.radians(self.currentLatitude)) \
                   * math.cos(math.radians(self.destLatitude)) * math.sin(dlon/2) * math.sin(dlon/2)
               c = 2 * math.atan2(math.sqrt(a), math.sqrt(1-a))
               d = radius * c

               self.distance = d
               return d

           '''''this method returns the distance from current position to the new destination.
               The new destination is mainly the interesection of the drone paths.'''

           def getDistanceToAPoint(self,newDestinationLat,newDestinationLong):

               radius = 6371 # km

               dlat = math.radians(newDestinationLat-self.currentLatitude)
               dlon = math.radians(newDestinationLong-self.currentLongitude)
               a = math.sin(dlat/2) * math.sin(dlat/2) + math.cos(math.radians(self.currentLatitude)) \
                   * math.cos(math.radians(newDestinationLat)) * math.sin(dlon/2) * math.sin(dlon/2)
               c = 2 * math.atan2(math.sqrt(a), math.sqrt(1-a))
               d = radius * c

               self.distance = d
               return d



           def returnHome(self):
               pass

           def displayCameraFeed(self):
               pass

           #updating telemetry
           def updateTelemetry(self):
               newGpsCoordinates =  self.gpsCoordinatesUpdate()
               newDistance = self.getDistance()


               return newGpsCoordinates,newDistance


           def getTelemetry(self):
              return self.updateTelemetry()

           #This method creates a vector based on the coordinates of the drone
           def createVectorFromCoordinates(self):
               self.firstComponent = self.destLatitude - self.startLatitude
               self.secondComponent = self.destLongitude - self.startLongitude

               return self.firstComponent, self.secondComponent

           #returns the latitude part(first component) of vector
           def getVectorLatitude(self):
               firstComponent = self.destLatitude - self.startLatitude
               return firstComponent

           #returns the longitude part(second component) of vector
           def getVectorLongitude(self):
               self.secondComponent = self.destLongitude - self.startLongitude
               return self.secondComponent

           #returns the length of vector
           def magnitudeOfVector(self):
               #addComponents = ((self.firstComponent*self.firstComponent) + (self.secondComponent*self.secondComponent))
               addComponents = ((self.getVectorLatitude()*self.getVectorLatitude()) + (self.getVectorLongitude()*self.getVectorLongitude()))
               magnitude = math.sqrt(addComponents)
               return magnitude

           #updates the gps coordinates as the drone moves along vector
           def gpsCoordinatesUpdate(self):
               '''''ratio was used first and then speed was used to update the coordinates along the vector'''

               if(self.getSpeed()!=0):
                   self.currentLatitude = self.currentLatitude + (self.getVectorLatitude()/(self.getSpeed()*60))
                   self.currentLongitude = self.currentLongitude + (self.getVectorLongitude()/(self.getSpeed()*60))
               else:
                    self.currentLatitude =  self.currentLatitude
                    self.currentLongitude = self.currentLongitude
               self.newGps = self.currentLatitude,self.currentLongitude



               return self.newGps



           def setSpeed(self,speed):
               self.speed = speed

           def getSpeed(self):
               return self.speed

           def setState(self, state):
               self.state=state
```

```

            def setRatio(self,newRatio):
                self.ratio = newRatio


            def getRatio(self):
                return self.ratio


            def getState(self):
                if self.altitude==0:
                    self.state="landed"
                else:
                    self.state="flying"

                return self.state


            def getAltitude(self):
                return self.altitude


            def setStartLatitude(self, latitude):
                self.startLatitude = latitude

            def setStartLongitude(self, longitude):
                self.startLongitude = longitude

            def setDestLatitude(self, latitude):
                self.destLatitude = latitude

            def setDestLongitude(self, longitude):
                self.destLongitude = longitude

            def getstartLongitude(self):
                return self.startLongitude

            def getCurrentLatitude(self):
                return self.currentLatitude

            def getCurrentLongitude(self):
                return self.currentLongitude

            def setCurrentLatitude(self,lat):
                self.currentLatitude = lat

            def setCurrentLongitude(self,lon):
                self.currentLongitude = lon

            def getstartLatitude(self):
                return self.startLatitude

            def getDestLongitude(self):
                return self.destLongitude

            def getDestLatitude(self):
                return self.destLatitude

            def getBatteryLife(self):
                return self.battery

            #This method returns the bearing of the drone
            def getDirection(self):
                startLat  = math.radians(self.startLatitude)
                startLong = math.radians(self.startLongitude)
                endLat = math.radians(self.destLatitude)
```

```python
        endLong = math.radians(self.destLongitude)

        dLong = endLong - startLong

        dPhi = math.log(math.tan(endLat/2.0+math.pi/4.0)/math.tan(startLat/2.0+math.pi/4.0))
        if abs(dLong) > math.pi:
            if dLong > 0.0:
                dLong = -(2.0 * math.pi - dLong)
            else:
                dLong = (2.0 * math.pi + dLong)

        bearing = (math.radians(math.atan2(dLong, dPhi)) + 360.0) % 360.0

        return bearing #radians


    def rotateLeft(self):
        self.currentdirection -=1

    def rotateRight(self):
        self.currentdirection +=1


    def setDirection(self, direction):
        self.direction = self.getDirection()




    '''''This function maps the coordinates given in latitude and longitude and converts them to
     pixel positions.'''

    def mapGpsToPixels(self,lat,lon):

        '''''52.967109 is horizontal corner of the static map and 7.159379 is the other corner
        that runs vertically.'''
        latDifference = (52.967109 - lat)*1000000
        lonDifference = (7.159379 - lon)*1000000

        '''''243435 is the difference in latitude from one side of the map to the other.
        640 is the width of the static map in pixels.The interp function maps the latitude to the
        pixel size of the map.'''
        latInPixels = interp(latDifference,[0,243435],[0,640])
        longInPixels = interp(lonDifference,[0,459365],[0,640])

        return longInPixels,latInPixels

    '''''This function returns the latitude coordinate in pixel positions on the map.'''
    def mapGpsLatToPixels(self,lat):

        latDifference = (52.967109 - lat)*1000000
        latInPixels = interp(latDifference,[0,243435],[0,640])

        return latInPixels

    '''''This function returns the longitude coordinate in pixel positions on the map.'''
    def mapGpsLongToPixels(self,lon):

        lonDifference = (7.159379 + lon)*1000000
```

```
                longInPixels = interp(lonDifference,[0,459365],[0,640])


                return longInPixels





```

BebopTestConnect.py

```
"""
checks the connection to the Bebop

Author: Brendan Mitchell
Student number: c00220212
Description: This class was used to test the connection to the bebop drone
"""

from pyparrot.Bebop import Bebop
import math
import threading
from threading import Timer
from time import time
import math
from numpy import interp

#import pyparrot modules
import time
from pyparrot.networking.wifiConnection import WifiConnection
from pyparrot.utils.colorPrint import color_print
from pyparrot.commandsandsensors.DroneCommandParser import DroneCommandParser
from pyparrot.commandsandsensors.DroneSensorParser import DroneSensorParser
from datetime import datetime
from pyparrot.Bebop import Bebop
from BebopDroneController import BebopDroneController

bebop = BebopDroneController()

print("connecting to bebop")
bebop.BebopConnect()
print(bebop.BebopConnect())
bebop.startVideo()
print("video has started")
bebop.stopVideo()
print("video has stopped")
print("disconnecting bebop")
bebop.disconnect()
```

BebopDroneController.py

```python
#Author: Brendan Mitchell
#Student Number: c00220212
#Description: This class is responsible for controlling a bebop drone. This class
#inherits from the Bebop class which contains all the methods to create and control
#a parrot Bebop2 drone.



from DroneBase import DroneBase
#from haversine import haversine, Unit
import math
import threading
from threading import Timer
from time import time
import math
from numpy import interp

#import pyparrot modules
import time
from pyparrot.networking.wifiConnection import WifiConnection
from pyparrot.utils.colorPrint import color_print
from pyparrot.commandsandsensors.DroneCommandParser import DroneCommandParser
from pyparrot.commandsandsensors.DroneSensorParser import DroneSensorParser
from datetime import datetime
from pyparrot.Bebop import Bebop




#inherits from the Bebop class
class BebopDroneController(Bebop):
    def BebopConnect(self):
        self.connect(5)

    def BebopDisconnect(self):
        self.disconnect()

    def moveForward(self):
        print("Flying direct: going forward (positive pitch)")
        self.fly_direct(roll=0, pitch=50, yaw=0, vertical_movement=0, duration=1)

    def takeOff(self):
        self.safe_takeoff(5)

    def land(self):
        self.safe_land(5)

    def hover(self):
        self.fly_direct(roll=0, pitch=0, yaw=0, vertical_movement=0, duration=1)

    def rotateLeft(self):
        pass

    def startVideo(self):
        self.start_video_stream()

    def stopVideo(self):
        self.stop_video_stream()



```

DroneBase.py

```python
#Author: Brendan Mitchell
#Student Number: c00220212
#Description: This class is an abstract class. All other classes that inherit from this
#class must use the abstract methods below.



from abc import ABC, abstractmethod
class DroneBase(ABC):

    #initialising variables
    startLatitude=0
    startLongitude = 0
    destLatitude = 0
    destLongitude = 0
    altitude = 0
    battery = 100
    speed = 0
    status = 0
    model = 0
    direction = 0
    currentdirection = 0
    distance = 0
    currentTime = 0



    @abstractmethod
    def takeOff(self):
        pass

    @abstractmethod
    def land(self):
        pass
    @abstractmethod
    def moveForward(self):
        pass

    @abstractmethod
    def hover(self):
        pass

    @abstractmethod
    def setDirection(self):
        pass

    @abstractmethod
    def rotateLeft(self):
        pass

    @abstractmethod
    def rotateRight(self):
        pass

    @abstractmethod
    def getDistance(self):
        pass

    @abstractmethod
    def getDistanceToAPoint(self):
        pass

    @abstractmethod
    def returnHome(self):
        pass

    @abstractmethod
    def displayCameraFeed(self):
        pass

    @abstractmethod
    def getTelemetry(self):
        pass

    @abstractmethod
    def setSpeed(self,speed):
        pass

    @abstractmethod
    def getSpeed(self):
        pass

    @abstractmethod
    def setState(self, state):
        pass

    @abstractmethod
    def getState(self):
        pass

    @abstractmethod
    def getAltitude(self):
        pass

    @abstractmethod
    def setStartLatitude(self, latitude):
        pass

    @abstractmethod
        def setStartLongitude(self, longitude):
            pass

        @abstractmethod
```

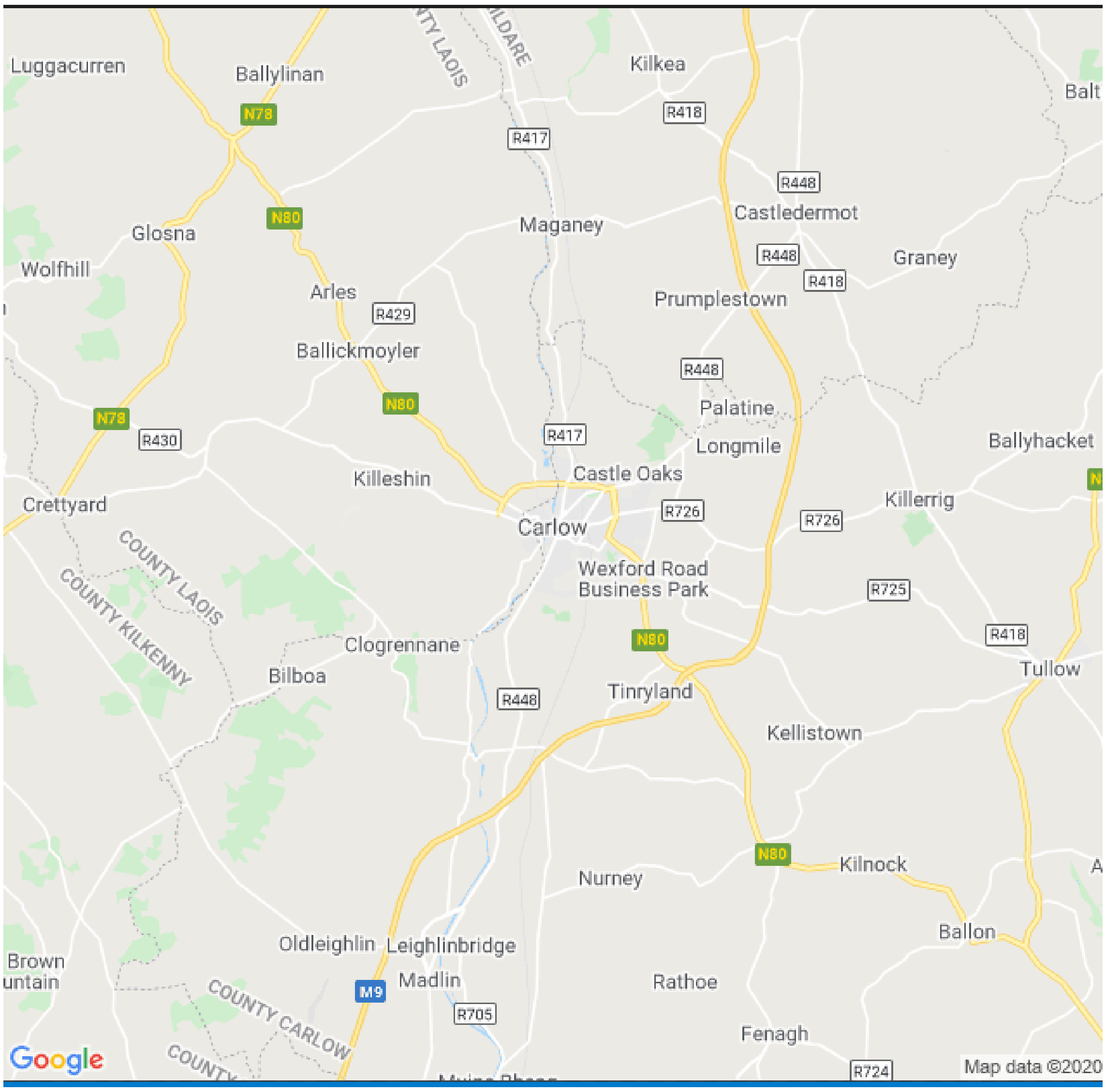
```python
        def  gpsCoordinatesUpdate(self):
            pass


        @abstractmethod
        def setDestLatitude(self, latitude):
            pass

        @abstractmethod
        def setDestLongitude(self, longitude):
            pass

        @abstractmethod
        def getstartLongitude(self):
            pass



        @abstractmethod
        def mapGpsToPixels(self,lat,long):
            pass


        @abstractmethod
        def mapGpsLatToPixels(self,lat):
            pass


        @abstractmethod
        def mapGpsLongToPixels(self,long):
            pass


        @abstractmethod
        def createVectorFromCoordinates(self):
            pass
```

FlightPath.py

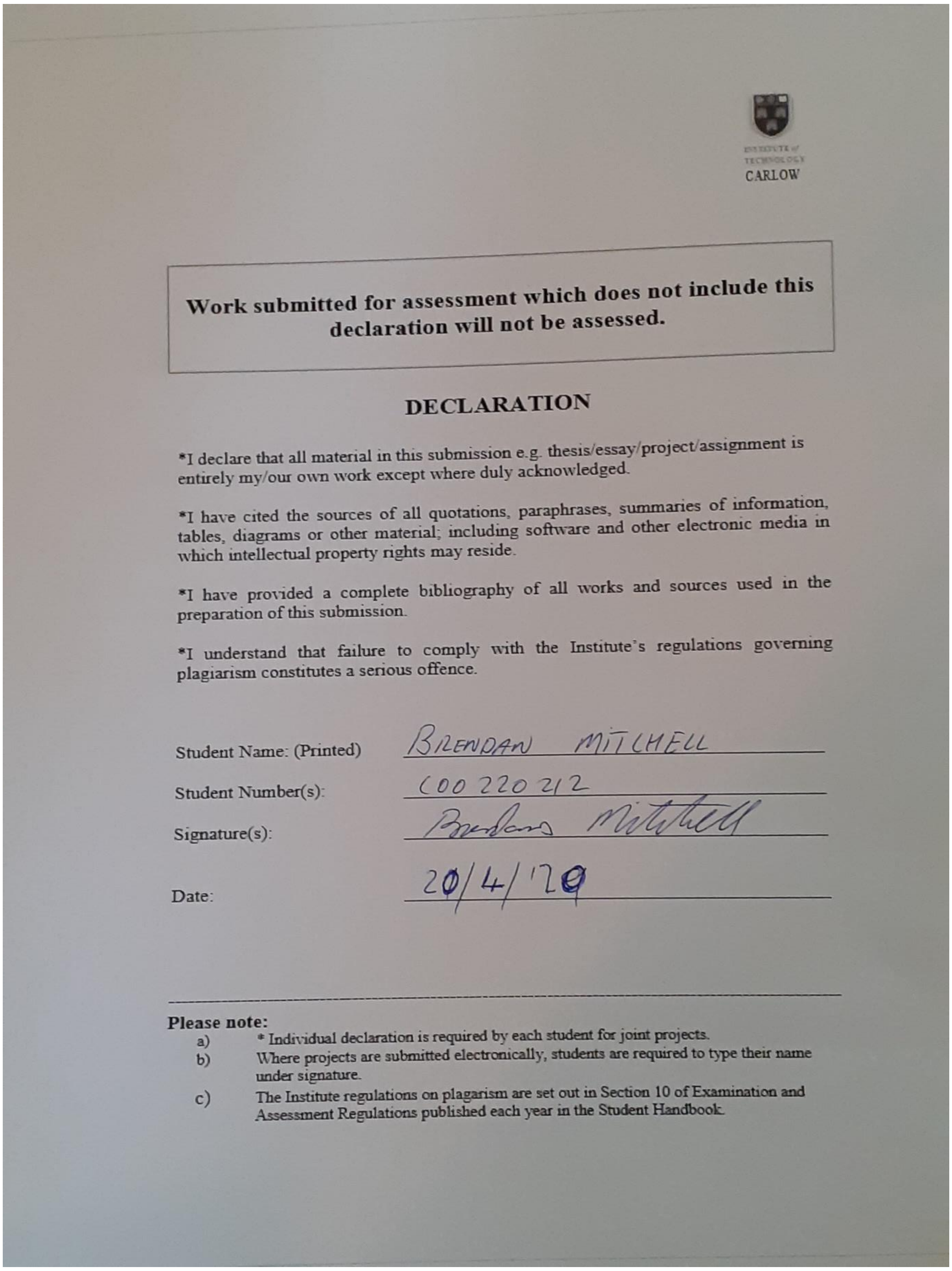
```python
#Author: Brendan Mitchell
#Student Number: c00220212
#Description: This class sets up flight paths that the drone can use to navigate from a starting
# point to a destination.


#importing neccessary modules
import pygame, sys, math, random
from pygame.locals import *
import pygame
from pygame.locals import *
from PIL import Image, ImageDraw
#from Virtual_controller import Virtual_controller
#from VirtualControllerTwo import VirtualControllerTwo
#from VirtualControllerThree import VirtualControllerThree
from VirtualController import VirtualController
import numpy as np
from shapely.geometry import LineString
```

```python

import pygame
from pygame.locals import *
from PIL import Image, ImageDraw
#from Virtual_controller import Virtual_controller



class FlightPath:


    #constuctor
    def  __init__(self,name,startLatitude,startLongitude,destLatitude,destLongitude
,altitude,speed):
        self.name = name
        self.startLatitude = startLatitude
        self.startLongitude = startLongitude
        self.destLatitude = destLatitude
        self.destLongitude = destLongitude
        self.altitude = altitude
        self.speed = speed

    #This method returns all the relevant attributes
    def getFlightPath(self):
        return  (self.name,
        self.startLatitude,
        self.startLongitude,
        self.destLatitude,
        self.destLongitude,
        self.altitude,
        self.speed)

    #Below are the getters and setters for the class
    def getFlightPathName(self):
        return self.name

    def getStartLat(self):
        return self.startLatitude

    def getStartLon(self):
        return self.startLongitude

    def getDestLat(self):
        return self.destLatitude

    def getDestLon(self):
        return self.destLongitude

    def getAltitude(self):
        return self.altitude

    def getSpeed(self):
        return self.speed

    def setStartLat(self,lat):
        self.startLatitude = lat

    def setStartLon(self,lon):
        self.startLongitude = lon

    def setDestLat(self,Dlat):
        self.destLatitude = Dlat

    def setDestLon(self,Dlon):
        self.destLongitude = Dlon

    def setAltitude(self, alt):
        self.altitude = alt

    def setSpeed(self,spd):
        self.speed = spd

```

testMaps.png



Plagiarism form



INSTITUTE of TECHNOLOGY CARLOW

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):	C00220212
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.