Agricultural Drone Survey Application

Technical Manual

INSTITUTE of TECHNOLOGY CARLOW

Institiúid Teicneolaíochta Cheatharlach

Bernard Steemers
C00235159@itcarlow,ie
Supervisor: Paul Barry
paul.barry@itcarlow.ie

Table of Contents

Application Dependencies:

Linux:

Ubuntu 18.04 and Debian 9 or higher are compatible with the Olympe Python framework.

Python3.7:

```
sudo apt-get install python3.7
```

Olympe:

```
cd $HOME
mkdir -p code/parrot-groundsdk
cd code/parrot-groundsdk
pwd
repo init -u https://github.com/Parrot-Developers/groundsdk-manifest.git
repo sync
```

For further instructions please see:
https://developer.parrot.com/docs/olympe/installation.html

Flask:

```
$ pip install Flask
```

GeoPy:

```
$ pip install geopy
```

Google Maps Platform:

Please see below for set up instructions and API keys
https://cloud.google.com/maps-platform/maps

Chronological Program Progression

takeOffLand.py

```
# -*- coding:
UTF-8 -*-


import olympe
from olympe.messages.ardrone3.Piloting import TakeOff, Landing
from olympe.messages.ardrone3.PilotingState import
FlyingStateChanged


#drone = olympe.Drone("192.168.42.1")#Physical Drone IP Address
drone = olympe.Drone("10.202.0.1") #Emulated Drone IP Address

#Establish connection to physical or emulated drones
drone.connection()

#Take off
drone(
   TakeOff()
   >> FlyingStateChanged(state="hovering", _timeout=5)# wait 5
seconds for hovering state
).wait()

#Landing
drone(Landing()).wait()

#Disconnect from drone
drone.disconnection()
```

moveForward.py

```
# -*- coding:
UTF-8 -*-

                import olympe
                from olympe.messages.ardrone3.Piloting import TakeOff,
                moveBy, Landing
                from olympe.messages.ardrone3.PilotingState import
                FlyingStateChanged


                #drone = olympe.Drone("192.168.42.1")#Physical Drone IP
                Address
                drone = olympe.Drone("10.202.0.1") #Emulated Drone IP
                Address


                #Establish connection to physical or emulated drones
                drone.connection()


                #Take off
                drone(
                   TakeOff()
                   >> FlyingStateChanged(state="hovering", _timeout=5)#
                wait 5 seconds for hovering state
                ).wait()


                #Move forward by two meters
                drone(
                   moveBy(2, 0, 0, 0)# (a,b,c,d) a = X-axis, b = Z-axis,
                c = Y-axis, d = Y-axis pivot
                   >> FlyingStateChanged(state="hovering", _timeout=5)
                ).wait()



                #Landing
                drone(Landing()).wait()
```

```
#Disconnect from drone
drone.disconnection()
```

dropGPS.py

```
# -*-
coding:
UTF-8 -*-
```

```
from __future__ import print_function
import olympe
from olympe.messages.ardrone3.Piloting import TakeOff,
moveBy, Landing
from olympe.messages.ardrone3.PilotingState import
FlyingStateChanged, PositionChanged
from olympe.messages.ardrone3.GPSSettingsState import
GPSFixStateChanged, HomeChanged
```

```
#drone = olympe.Drone("192.168.42.1")#Physical Drone IP
Address
drone = olympe.Drone("10.202.0.1") #Emulated Drone IP
Address
```

```
#Establish connection to physical or emulated drone
drone.connection()
```

```
# Wait for GPS to fix state
drone(GPSFixStateChanged(_policy = 'wait'))
```

```
#Print GPS coordinates position to console before take off
print("\n\nGPS position before take-off :",
drone.get_state(HomeChanged),"\n\n")
```

```
#Take off
drone(
```

```
    TakeOff()
    >> FlyingStateChanged(state="hovering", _timeout=5)
).wait()


#Print GPS coordinates after take off
print("\n\nGPS position after Landing : ",
drone.get_state(PositionChanged),"\n\n")


#Landing
drone(Landing()).wait()


#Disconnect from drone
drone.disconnection()
```

returnHomePos.py

```
# -*- coding:
UTF-8 -*-


                from __future__ import print_function

                import olympe

                from olympe.messages.ardrone3.Piloting import TakeOff,
                moveBy, Landing,moveTo

                from olympe.messages.ardrone3.PilotingState import
                moveToChanged,FlyingStateChanged, PositionChanged

                from olympe.messages.ardrone3.GPSSettingsState import
                GPSFixStateChanged,HomeChanged

                from olympe.enums.ardrone3.Piloting import
                MoveTo_Orientation_mode



                #drone = olympe.Drone("192.168.42.1")#Physical Drone IP
                Address

                drone = olympe.Drone("10.202.0.1") #Emulated Drone IP
                Address


                #Establish connection to physical or emulated drone

                drone.connection()


                # Wait for GPS to fix state

                drone(GPSFixStateChanged(_policy = 'wait'))


                #Print GPS coordinates position to console before take
                off

                print("\n\nGPS position before take-off :",
                drone.get_state(HomeChanged),"\n\n")
```

```
#Take off
drone(
   TakeOff()
   >> FlyingStateChanged(state="hovering", _timeout=5)
).wait()


#Print GPS coordinates after take off
print("\n\nGPS position after Landing : ",
drone.get_state(PositionChanged),"\n\n")


#Set home coordinates
home_coordinates = drone.get_state(PositionChanged)


#Move forward by three meters
drone(
   moveBy(3, 0, 0, 0)# (a,b,c,d) a = X-axis, b = Z-axis,
c = Y-axis, d = Y-axis pivot
   >> FlyingStateChanged(state="hovering", _timeout=5)
).wait()


#return to home position
drone(
   moveTo(home_coordinates["latitude"],
home_coordinates["longitude"], 1.0,
MoveTo_Orientation_mode.TO_TARGET, 0.0)
   >> FlyingStateChanged(state="hovering", _timeout=5)
   >> moveToChanged(status='DONE')
```

```
    >>
moveToChanged(latitude=home_coordinates["latitude"],
longitude=home_coordinates["longitude"], altitude=1.0,
orientation_mode=MoveTo_Orientation_mode.TO_TARGET,
status='DONE', _policy='wait')
    >> FlyingStateChanged(state="hovering", _timeout=5)
).wait()


#Landing
drone(Landing()).wait()


#Disconnect from drone
drone.disconnection()
```

moveXYZPivot.py

```
# -*- coding:
UTF-8 -*-


import olympe

from olympe.messages.ardrone3.Piloting import TakeOff,
moveBy, Landing

from olympe.messages.ardrone3.PilotingState import
FlyingStateChanged


#drone = olympe.Drone("192.168.42.1")#Physical Drone IP
Address

drone = olympe.Drone("10.202.0.1") #Emulated Drone IP
Address


#Establish connection to physical or emulated drones

drone.connection()


#Take off

drone(

   TakeOff()

   >> FlyingStateChanged(state="hovering", _timeout=5)#
wait 5 seconds for hovering state

).wait()


#Move forward by two meters
```

```
drone(
   moveBy(2, 0, 0, 0)# (a,b,c,d) a = X-axis, b = Z-axis,
c = Y-axis, d = Y-axis pivot
   >> FlyingStateChanged(state="hovering", _timeout=5)
).wait()


#Move right by two meters
drone(
   moveBy(0, 2, 0, 0)
   >> FlyingStateChanged(state="hovering", _timeout=5)
).wait()


#Move backwards by two meters
drone(
   moveBy(-2, 0, 0, 0)
   >> FlyingStateChanged(state="hovering", _timeout=5)
).wait()


#Move left by two meters
drone(
   moveBy(0,-2, 0, 0)
   >> FlyingStateChanged(state="hovering", _timeout=5)
).wait()


#Pivot by 90 degrees
drone(
   moveBy(0, 0, 0, 1.5)
   >> FlyingStateChanged(state="hovering", _timeout=5)
).wait()
```

```
#Landing
drone(Landing()).wait()


#Disconnect from drone
drone.disconnection()
```

takePhoto.py

```
import
olympe
        from olympe.messages.ardrone3.Piloting import TakeOff, moveBy,
        Landing
        from olympe.messages.camera import set_camera_mode,
        set_photo_mode, take_photo, photo_progress
        from olympe.messages.ardrone3.PilotingState import
        FlyingStateChanged
        from olympe.messages import gimbal
        import os
        import re
        import requests
        import shutil
        import xml.etree.ElementTree as ET
        import time


        ##################################################################
        ###################################


        def snap_and_save(drone):


            # Drone web server URL
            DRONE_URL = "http://{}/".format(DRONE_IP)
            # Drone media web API URL
            MEDIA_API = DRONE_URL + "api/v1/media/medias/"
```

```
    TAGS = (
        "GPSLatitude",
        "GPSLongitude",
        "GPSAltitude",
    )
    # Take photo, wait for saved status and get photo ID
    print("\n\nTaking a Photo\n\n")
    save_photo = drone(photo_progress(result="photo_saved",
_policy="wait"))
    drone(take_photo(cam_id=0)).wait()
    save_photo.wait()
    photo_id =
save_photo.received_events().last().args["media_id"]
    # Create media link
    media_link = requests.get(MEDIA_API + photo_id)
    media_link.raise_for_status()
    save_to_path = '/home/bernard/Desktop/images'#change to
generic directory
    #Request photo from media link
    for photos in media_link.json()["resources"]:
        picture = requests.get(DRONE_URL + photos["url"],
stream=True)
        print("\n\nGetting Picture From: ", DRONE_URL +
photos["url"])
        #create image directory name
        image_dir = os.path.join(save_to_path,
photos["resource_id"])
        picture.raise_for_status()
        #Copy Picture to Directory
        with open(image_dir, "wb") as photo_file:
            shutil.copyfileobj(picture.raw, photo_file)
        #Open image directory
        print("\nOpening ", image_dir)
        with open(image_dir, "rb") as photo_file:
            photo_data = photo_file.read()
```

```
            xmp_photo_start = photo_data.find(b"<x:xmpmeta")
            xmp_photo_end = photo_data.find(b"</x:xmpmeta")
            #parsing xmp metadata to check for GPS coordinates
            xmp_photo =
ET.fromstring(photo_data[xmp_photo_start : xmp_photo_end + 12])
            for photo_meta in xmp_photo[0][0]:
                xmpTag = re.sub(r"{[^}]*}", "", photo_meta.tag)
                value = photo_meta.text
                if xmpTag in TAGS:
                    print("Photo ID: ", photos["resource_id"],
xmpTag, value)


################################################################
############################################################


def setup_camera(drone):
    #set camera to photo mode
    drone(set_camera_mode(cam_id=0, value="photo")).wait()
    drone(
        set_photo_mode(
            cam_id=0,
            mode="single",
            format="rectilinear",
            file_format="jpeg",
            burst="burst_14_over_1s",
            bracketing="preset_1ev",
            capture_interval=0.0,
        )
    ).wait()


    #Tilt camera 90 degrees down
    tiltCamera = drone(gimbal.set_target(
```

```
    gimbal_id=0,
    control_mode="position",
    yaw_frame_of_reference="none",
    yaw=0.0,
    pitch_frame_of_reference="absolute",
    pitch=-90.0,
    roll_frame_of_reference="none",
    roll=0.0,
    )).wait()


    #If camera tilt does not occur raise runtime error
    if not tiltCamera.success():
            raise RuntimeError("Camera tilt did not execute
correctly")
            print("\n\nConfiguring Camera Settings\n\n")


#################################################################
#######################################################


#DRONE_IP = "192.168.42.1" #Physical Drone IP Address
DRONE_IP = "10.202.0.1" #Emulated Drone IP Address


drone = olympe.Drone(DRONE_IP, loglevel=0)


#Establish connection to physical or emulated drones
drone.connection()


# Call function to tilt camera and configure settings
setup_camera(drone)
```

```
#Wait for camera configurations to finish before take off
time.sleep(5)


#Take off
drone(
   TakeOff()
   >> FlyingStateChanged(state="hovering", _timeout=5)# wait 5
seconds for hovering state
).wait()


#Take photo and save to device
snap_and_save(drone)


#Landing
drone(Landing()).wait()


drone.disconnection()
```

emergencyLand.py

```
import olympe
from olympe.messages.ardrone3.Piloting import  Landing


drone = olympe.Drone("192.168.42.1")#Physical Drone IP
Address


#Establish connection to physical or emulated drones
drone.connection()


#LANDING
drone(Landing()).wait()


#Disconnect from drone
drone.disconnection()
```

mapAll.py

```
imp
ort
oly
mpe
    from olympe.messages.ardrone3.Piloting import TakeOff, moveBy,
    Landing,moveTo
    from olympe.messages.ardrone3.PilotingState import
    moveToChanged,FlyingStateChanged, PositionChanged
    from olympe.messages.camera import set_camera_mode,
    set_photo_mode, take_photo, photo_progress
    from olympe.messages.ardrone3.GPSSettingsState import
    GPSFixStateChanged
    from olympe.enums.ardrone3.Piloting import MoveTo_Orientation_mode
    from olympe.messages import gimbal
    import os
    import re
    import requests
    import shutil
    import xml.etree.ElementTree as ET
    import time


    ##############################################################
    ##################################


    def snap_and_save(drone):
```

```
    # Drone web server URL
    DRONE_URL = "http://{}/".format(DRONE_IP)
    # Drone media web API URL
    MEDIA_API = DRONE_URL + "api/v1/media/medias/"
    TAGS = (
        "GPSLatitude",
        "GPSLongitude",
        "GPSAltitude",
    )
    # Take photo, wait for saved status and get photo ID
    print("\n\nTaking a Photo\n\n")
    save_photo = drone(photo_progress(result="photo_saved",
_policy="wait"))
    drone(take_photo(cam_id=0)).wait()
    save_photo.wait()
    photo_id = save_photo.received_events().last().args["media_id"]
    # Create media link
    media_link = requests.get(MEDIA_API + photo_id)
    media_link.raise_for_status()
    save_to_path = '/home/bernard/Desktop/images'#change to generic
directory
    #Request photo from media link
    for photos in media_link.json()["resources"]:
        picture = requests.get(DRONE_URL + photos["url"],
stream=True)
        print("\n\nGetting Picture From: ", DRONE_URL +
photos["url"])
        #create image directory name
        image_dir = os.path.join(save_to_path,
photos["resource_id"])
        picture.raise_for_status()
        #Copy Picture to Directory
        with open(image_dir, "wb") as photo_file:
            shutil.copyfileobj(picture.raw, photo_file)
```

```
        #Open image directory
        print("\nOpening ", image_dir)
        with open(image_dir, "rb") as photo_file:
            photo_data = photo_file.read()
            xmp_photo_start = photo_data.find(b"<x:xmpmeta")
            xmp_photo_end = photo_data.find(b"</x:xmpmeta")
            #parsing xmp metadata to check for GPS coordinates
            xmp_photo = ET.fromstring(photo_data[xmp_photo_start :
xmp_photo_end + 12])
            for photo_meta in xmp_photo[0][0]:
                xmpTag = re.sub(r"{[^}]*}", "", photo_meta.tag)
                value = photo_meta.text
                if xmpTag in TAGS:
                    print("Photo ID: ", photos["resource_id"],
xmpTag, value)


##################################################################
##########################################################


def setup_camera(drone):
   #set camera to photo mode
   drone(set_camera_mode(cam_id=0, value="photo")).wait()
   drone(
       set_photo_mode(
           cam_id=0,
           mode="single",
           format="rectilinear",
           file_format="jpeg",
           burst="burst_14_over_1s",
           bracketing="preset_1ev",
           capture_interval=0.0,
       )
```

```
    ).wait()


    #Tilt camera 90 degrees down
    tiltCamera = drone(gimbal.set_target(
    gimbal_id=0,
    control_mode="position",
    yaw_frame_of_reference="none",
    yaw=0.0,
    pitch_frame_of_reference="absolute",
    pitch=-90.0,
    roll_frame_of_reference="none",
    roll=0.0,
    )).wait()


    #If camera tilt does not occur raise runtime error
    if not tiltCamera.success():
            raise RuntimeError("Camera tilt did not execute
correctly")
            print("\n\nConfiguring Camera Settings\n\n")


#####################################################################
#################################################


#DRONE_IP = "192.168.42.1" #Physical Drone IP Address
DRONE_IP = "10.202.0.1" #Emulated Drone IP Address


drone = olympe.Drone(DRONE_IP, loglevel=0)


#Establish connection to physical or emulated drones
drone.connection()
```

```
# Wait for GPS to fix state
drone(GPSFixStateChanged(_policy = 'wait'))


# Call function to tilt camera and configure settings
setup_camera(drone)


#Wait for camera configurations to finish before take off
time.sleep(5)


#Take off
drone(
   TakeOff()
   >> FlyingStateChanged(state="hovering", _timeout=5)# wait 5
seconds for hovering state
).wait()


#Set home coordinates
home_coordinates = drone.get_state(PositionChanged)
#Print GPS coordinates after take off
print("\n\nGPS position after taking off : ",
drone.get_state(PositionChanged),"\n\n")



#Pivot by 90 degrees
drone(
   moveBy(0, 0, 0, 1.5)
   >> FlyingStateChanged(state="hovering", _timeout=5)
).wait()
```

```
#Print GPS coordinates after pivot

print("\n\nGPS position after pivot : ",
drone.get_state(PositionChanged),"\n\n")

#Take photo and save to device

snap_and_save(drone)
```

```
#Move forward by two meters

drone(

   moveBy(2, 0, 0, 0)# (a,b,c,d) a = X-axis, b = Z-axis, c = Y-
axis, d = Y-axis pivot

   >> FlyingStateChanged(state="hovering", _timeout=5)

).wait()

#Print GPS coordinates after moving forward

print("\n\nGPS position after moving forward : ",
drone.get_state(PositionChanged),"\n\n")

#Take photo and save to device

snap_and_save(drone)
```

```
#Move right by two meters

drone(

   moveBy(0, 2, 0, 0)

   >> FlyingStateChanged(state="hovering", _timeout=5)

).wait()

#Print GPS coordinates after moving right

print("\n\nGPS position after moving right : ",
drone.get_state(PositionChanged),"\n\n")

#Take photo and save to device

snap_and_save(drone)
```

```
#Move backwards by two meters
drone(
   moveBy(-2, 0, 0, 0)
   >> FlyingStateChanged(state="hovering", _timeout=5)
).wait()
#Print GPS coordinates after moving back
print("\n\nGPS position after moving back : ",
drone.get_state(PositionChanged),"\n\n")
#Take photo and save to device
snap_and_save(drone)
```

```
#move to altitude of 1.7meters
drone(
   moveTo(home_coordinates["latitude"],
home_coordinates["longitude"], 1.7,
MoveTo_Orientation_mode.TO_TARGET, 0.0)
   >> FlyingStateChanged(state="hovering", _timeout=5)
   >> moveToChanged(status='DONE')
   >> moveToChanged(latitude=home_coordinates["latitude"],
longitude=home_coordinates["longitude"], altitude=1.7,
orientation_mode=MoveTo_Orientation_mode.TO_TARGET, status='DONE',
_policy='wait')
   >> FlyingStateChanged(state="hovering", _timeout=5)
).wait()
```

```
#Landing
drone(Landing()).wait()
```

```
#Disconnect from drone
drone.disconnection()
```

readCoordinates.py

```
import
olympe
    from olympe.messages.ardrone3.Piloting import TakeOff, moveBy,
    Landing,moveTo
    from olympe.messages.ardrone3.PilotingState import
    moveToChanged,FlyingStateChanged, PositionChanged
    from olympe.messages.camera import set_camera_mode,
    set_photo_mode, take_photo, photo_progress
    from olympe.messages.ardrone3.GPSSettingsState import
    GPSFixStateChanged,HomeChanged
    from olympe.enums.ardrone3.Piloting import
    MoveTo_Orientation_mode
    from olympe.messages import gimbal
    import os
    import re
    import requests
    import shutil
    import xml.etree.ElementTree as ET
    import time


    from geopy.distance import geodesic
    ######################################################################
    ###################################


    def snap_and_save(drone):


       # Drone web server URL
       DRONE_URL = "http://{}/".format(DRONE_IP)
```

```
    # Drone media web API URL
    MEDIA_API = DRONE_URL + "api/v1/media/medias/"
    TAGS = (
        "GPSLatitude",
        "GPSLongitude",
        "GPSAltitude",
    )
    # Take photo, wait for saved status and get photo ID
    print("\n\nTaking a Photo\n\n")
    save_photo = drone(photo_progress(result="photo_saved",
_policy="wait"))
    drone(take_photo(cam_id=0)).wait()
    save_photo.wait()
    photo_id =
save_photo.received_events().last().args["media_id"]
    # Create media link
    media_link = requests.get(MEDIA_API + photo_id)
    media_link.raise_for_status()
    save_to_path = '/home/bernard/Desktop/images'#change to
generic directory
    #Request photo from media link
    for photos in media_link.json()["resources"]:
        picture = requests.get(DRONE_URL + photos["url"],
stream=True)
        print("\n\nGetting Picture From: ", DRONE_URL +
photos["url"])
        #create image directory name
        image_dir = os.path.join(save_to_path,
photos["resource_id"])
        picture.raise_for_status()
        #Copy Picture to Directory
        with open(image_dir, "wb") as photo_file:
            shutil.copyfileobj(picture.raw, photo_file)
        #Open image directory
        print("\nOpening ", image_dir)
```

```
        with open(image_dir, "rb") as photo_file:
            photo_data = photo_file.read()
            xmp_photo_start = photo_data.find(b"<x:xmpmeta")
            xmp_photo_end = photo_data.find(b"</x:xmpmeta")
            #parsing xmp metadata to check for GPS coordinates
            xmp_photo =
ET.fromstring(photo_data[xmp_photo_start : xmp_photo_end + 12])
            for photo_meta in xmp_photo[0][0]:
                xmpTag = re.sub(r"{[^}]*}", "", photo_meta.tag)
                value = photo_meta.text
                if xmpTag in TAGS:
                    print("Photo ID: ", photos["resource_id"],
xmpTag, value)


################################################################
############################################################


def setup_camera(drone):
   #set camera to photo mode
   drone(set_camera_mode(cam_id=0, value="photo")).wait()
   drone(
       set_photo_mode(
           cam_id=0,
           mode="single",
           format="rectilinear",
           file_format="jpeg",
           burst="burst_14_over_1s",
           bracketing="preset_1ev",
           capture_interval=0.0,
       )
   ).wait()
```

```
    #Tilt camera 90 degrees down
    tiltCamera = drone(gimbal.set_target(
    gimbal_id=0,
    control_mode="position",
    yaw_frame_of_reference="none",
    yaw=0.0,
    pitch_frame_of_reference="absolute",
    pitch=-90.0,
    roll_frame_of_reference="none",
    roll=0.0,
    )).wait()


    #If camera tilt does not occur raise runtime error
    if not tiltCamera.success():
        raise RuntimeError("Camera tilt did not execute
correctly")
    print("\n\nCamera Settings Configured\n\n")


################################################################
#####################################################


def read_In_Coordinates():
    #Open Coordinates file
    f = open("/home/bernard/Downloads/Coordinates.txt", "r")


    global altitude


    #Read in each line from file
    for line in f:
        #check for set Altitude
```

```
        if len(line) < 5:
            altitude = float(line)
        else:
            #Remove unneeded characters, add lats & longs to
lists
            gone = line.replace("(","")
            gone = gone.replace(")","")
            gone = gone.replace(" ","")

            gone = gone.split(",")
            gone[1] = gone[1].replace('\n','')
            latitudes.append(float(gone[0]))
            longitudes.append(float(gone[1]))
    f.close()
    print("\n\nMap Coordinates Have Been Successfully Read
In\n\n")



#################################################################
#######################################################

latitudes = []
longitudes = []
altitude = 0.0

closeLat = 0.0
closeLong = 0.0


#DRONE_IP = "192.168.42.1" #Physical Drone IP Address
```

```
DRONE_IP = "10.202.0.1" #Emulated Drone IP Address


#read in map chosen coordinates
read_In_Coordinates()


drone = olympe.Drone(DRONE_IP, loglevel=0)


#Establish connection to physical or emulated drones
drone.connection()


# Wait for GPS to fix state
drone(GPSFixStateChanged(_policy = 'wait'))


# Call function to tilt camera and configure settings
setup_camera(drone)


#Set home coordinates
home_coordinates = drone.get_state(HomeChanged)


#Disconnect from Drone
drone.disconnection()
```

flyToNearestPoint.py

```
impo
rt
olym
pe
        from olympe.messages.ardrone3.Piloting import TakeOff, moveBy,
        Landing,moveTo
        from olympe.messages.ardrone3.PilotingState import
        moveToChanged,FlyingStateChanged, PositionChanged
        from olympe.messages.camera import set_camera_mode,
        set_photo_mode, take_photo, photo_progress
        from olympe.messages.ardrone3.GPSSettingsState import
        GPSFixStateChanged,HomeChanged
        from olympe.enums.ardrone3.Piloting import MoveTo_Orientation_mode
        from olympe.messages import gimbal
        import os
        import re
        import requests
        import shutil
        import xml.etree.ElementTree as ET
        import time


        from geopy.distance import geodesic
        ######################################################################
        ###################################


        def snap_and_save(drone):


            # Drone web server URL
            DRONE_URL = "http://{}/".format(DRONE_IP)
            # Drone media web API URL
            MEDIA_API = DRONE_URL + "api/v1/media/medias/"
            TAGS = (
```

```
        "GPSLatitude",
        "GPSLongitude",
        "GPSAltitude",
    )
    # Take photo, wait for saved status and get photo ID
    print("\n\nTaking a Photo\n\n")
    save_photo = drone(photo_progress(result="photo_saved",
_policy="wait"))
    drone(take_photo(cam_id=0)).wait()
    save_photo.wait()
    photo_id = save_photo.received_events().last().args["media_id"]
    # Create media link
    media_link = requests.get(MEDIA_API + photo_id)
    media_link.raise_for_status()
    save_to_path = '/home/bernard/Desktop/images'#change to generic
directory
    #Request photo from media link
    for photos in media_link.json()["resources"]:
        picture = requests.get(DRONE_URL + photos["url"],
stream=True)
        print("\n\nGetting Picture From: ", DRONE_URL +
photos["url"])
        #create image directory name
        image_dir = os.path.join(save_to_path,
photos["resource_id"])
        picture.raise_for_status()
        #Copy Picture to Directory
        with open(image_dir, "wb") as photo_file:
            shutil.copyfileobj(picture.raw, photo_file)
        #Open image directory
        print("\nOpening ", image_dir)
        with open(image_dir, "rb") as photo_file:
            photo_data = photo_file.read()
            xmp_photo_start = photo_data.find(b"<x:xmpmeta")
```

```
            xmp_photo_end = photo_data.find(b"</x:xmpmeta")
            #parsing xmp metadata to check for GPS coordinates
            xmp_photo = ET.fromstring(photo_data[xmp_photo_start :
xmp_photo_end + 12])
            for photo_meta in xmp_photo[0][0]:
                xmpTag = re.sub(r"{[^}]*}", "", photo_meta.tag)
                value = photo_meta.text
                if xmpTag in TAGS:
                    print("Photo ID: ", photos["resource_id"],
xmpTag, value)


##################################################################
##########################################################


def setup_camera(drone):
   #set camera to photo mode
   drone(set_camera_mode(cam_id=0, value="photo")).wait()
   drone(
       set_photo_mode(
           cam_id=0,
           mode="single",
           format="rectilinear",
           file_format="jpeg",
           burst="burst_14_over_1s",
           bracketing="preset_1ev",
           capture_interval=0.0,
       )
   ).wait()


   #Tilt camera 90 degrees down
   tiltCamera = drone(gimbal.set_target(
   gimbal_id=0,
```

```
    control_mode="position",
    yaw_frame_of_reference="none",
    yaw=0.0,
    pitch_frame_of_reference="absolute",
    pitch=-90.0,
    roll_frame_of_reference="none",
    roll=0.0,
    )).wait()

    #If camera tilt does not occur raise runtime error
    if not tiltCamera.success():
        raise RuntimeError("Camera tilt did not execute correctly")
    print("\n\nCamera Settings Configured\n\n")

#####################################################################
#####################################################

def read_In_Coordinates():
    #Open Coordinates file
    f = open("/home/bernard/Downloads/Coordinates.txt", "r")

    global altitude

    #Read in each line from file
    for line in f:
        #check for set Altitude
        if len(line) < 5:
            altitude = float(line)
        else:
            #Remove unneeded characters, add lats & longs to lists
```

```
            gone = line.replace("(","")
            gone = gone.replace(")","")
            gone = gone.replace(" ","")

            gone = gone.split(",")
            gone[1] = gone[1].replace('\n','')
            latitudes.append(float(gone[0]))
            longitudes.append(float(gone[1]))
    f.close()
    print("\n\nMap Coordinates Have Been Successfully Read In\n\n")

#################################################################
#################################################

def find_Closest_Coordinate():

    global closeLat
    global closeLong

    #Set home position latitude and longitude
    originLat = home_coordinates['latitude']
    originLong = home_coordinates['longitude']

    #Get distance between home position and first map Coordinate
    homePosition = (originLat, originLong)
    firstPoint = (latitudes[0],longitudes[0])
    dist = geodesic(homePosition, firstPoint).meters

    #Set up other coordinates
```

```
    count = 0
    anotherPosition = (latitudes[count], longitudes[count])
    closeLat = latitudes[count]
    closeLong = longitudes[count]

    #Loop through coorinates
    for i in range(len(latitudes)):
        #Set Distance between home and another Coordinate
        anotherPosition = (latitudes[count], longitudes[count])
        anotherDist = geodesic(homePosition,
anotherPosition).meters

        #If another distance smaller than the other then replace
and increment count
        if anotherDist < dist:

            dist = anotherDist
            closeLat = float(latitudes[count])
            closeLong = float(longitudes[count])

            count +=1

        else:
            count+=1

    print("\n\nClosest Coordinate Found\n\n")

##################################################################
################################################
```

```
def fly_To_Closest():
   global closeLat
   global closeLong
   global altitude


   #Fly to the nearest coordinate from the drones home position at
selected altitude
   drone(
       moveTo(closeLat, closeLong, altitude,
MoveTo_Orientation_mode.TO_TARGET,0.0)
           >> FlyingStateChanged(state="hovering",
_timeout=5)#'latitude', 52.65485574090526), ('longitude',
-6.653909008997573)
           >> moveToChanged(status='DONE')
           >> moveToChanged(latitude=closeLat,
longitude=closeLong, altitude=altitude,
orientation_mode=MoveTo_Orientation_mode.TO_TARGET, status='DONE',
_policy='wait')
           >> FlyingStateChanged(state="hovering", _timeout=5)
   ).wait()


#################################################################
#################################################


latitudes = []
longitudes = []
altitude = 0.0


closeLat = 0.0
closeLong = 0.0
```

```
DRONE_IP = "192.168.42.1" #Physical Drone IP Address
#DRONE_IP = "10.202.0.1" #Emulated Drone IP Address


#read in map chosen coordinates
read_In_Coordinates()
print(altitude)


drone = olympe.Drone(DRONE_IP, loglevel=0)


#Establish connection to physical or emulated drones
drone.connection()


# Wait for GPS to fix state
drone(GPSFixStateChanged(_policy = 'wait'))


# Call function to tilt camera and configure settings
setup_camera(drone)


#Set home coordinates
home_coordinates = drone.get_state(HomeChanged)


#Set the nearest longitude and latitude to home position
find_Closest_Coordinate()
```

```
print(drone.get_state(HomeChanged))


#Take off
drone(
   TakeOff()
   >> FlyingStateChanged(state="hovering", _timeout=5)# wait 5
seconds for hovering state
).wait()


fly_To_Closest()


#Landing
drone(Landing()).wait()


#Disconnect from Drone
drone.disconnection()
```

divideLengthBreadth.py

```
impo
rt
olym
pe
        from olympe.messages.ardrone3.Piloting import TakeOff, moveBy,
        Landing,moveTo
        from olympe.messages.ardrone3.PilotingState import
        moveToChanged,FlyingStateChanged, PositionChanged, AttitudeChanged
        from olympe.messages.camera import set_camera_mode,
        set_photo_mode, take_photo, photo_progress
        from olympe.messages.ardrone3.GPSSettingsState import
        GPSFixStateChanged,HomeChanged
        from olympe.enums.ardrone3.Piloting import MoveTo_Orientation_mode
        from olympe.messages import gimbal
        import os
        import re
        import requests
        import shutil
        import xml.etree.ElementTree as ET
        import time


        from geopy.distance import geodesic
        ###################################################################
        ##################################


        def snap_and_save(drone):


            # Drone web server URL
            DRONE_URL = "http://{}/".format(DRONE_IP)
            # Drone media web API URL
            MEDIA_API = DRONE_URL + "api/v1/media/medias/"
            TAGS = (
                "GPSLatitude",
```

```
        "GPSLongitude",
        "GPSAltitude",
    )
    # Take photo, wait for saved status and get photo ID
    print("\n\nTaking a Photo\n\n")
    save_photo = drone(photo_progress(result="photo_saved",
_policy="wait"))
    drone(take_photo(cam_id=0)).wait()
    save_photo.wait()
    photo_id = save_photo.received_events().last().args["media_id"]
    # Create media link
    media_link = requests.get(MEDIA_API + photo_id)
    media_link.raise_for_status()
    save_to_path = '/home/bernard/Desktop/images'#change to generic
directory
    #Request photo from media link
    for photos in media_link.json()["resources"]:
        picture = requests.get(DRONE_URL + photos["url"],
stream=True)
        print("\n\nGetting Picture From: ", DRONE_URL +
photos["url"])
        #create image directory name
        image_dir = os.path.join(save_to_path,
photos["resource_id"])
        picture.raise_for_status()
        #Copy Picture to Directory
        with open(image_dir, "wb") as photo_file:
            shutil.copyfileobj(picture.raw, photo_file)
        #Open image directory
        print("\nOpening ", image_dir)
        with open(image_dir, "rb") as photo_file:
            photo_data = photo_file.read()
            xmp_photo_start = photo_data.find(b"<x:xmpmeta")
            xmp_photo_end = photo_data.find(b"</x:xmpmeta")
```

```
            #parsing xmp metadata to check for GPS coordinates
            xmp_photo = ET.fromstring(photo_data[xmp_photo_start :
xmp_photo_end + 12])
            for photo_meta in xmp_photo[0][0]:
                xmpTag = re.sub(r"{[^}]*}", "", photo_meta.tag)
                value = photo_meta.text
                if xmpTag in TAGS:
                    print("Photo ID: ", photos["resource_id"],
xmpTag, value)


#####################################################################
#########################################################


def setup_camera(drone):
    #set camera to photo mode
    drone(set_camera_mode(cam_id=0, value="photo")).wait()
    drone(
        set_photo_mode(
            cam_id=0,
            mode="single",
            format="rectilinear",
            file_format="jpeg",
            burst="burst_14_over_1s",
            bracketing="preset_1ev",
            capture_interval=0.0,
        )
    ).wait()


    #Tilt camera 90 degrees down
    tiltCamera = drone(gimbal.set_target(
    gimbal_id=0,
    control_mode="position",
```

```
    yaw_frame_of_reference="none",
    yaw=0.0,
    pitch_frame_of_reference="absolute",
    pitch=-90.0,
    roll_frame_of_reference="none",
    roll=0.0,
    )).wait()


    #If camera tilt does not occur raise runtime error
    if not tiltCamera.success():
        raise RuntimeError("Camera tilt did not execute correctly")
    print("\n\nCamera Settings Configured\n\n")


#####################################################################
#################################################

def read_In_Coordinates():
    #Open Coordinates file
    f = open("/home/bernard/Downloads/Coordinates.txt", "r")


    global altitude


    #Read in each line from file
    for line in f:
        #check for set Altitude
        if len(line) < 5:
            altitude = float(line)
        else:
            #Remove unneeded characters, add lats & longs to lists
            gone = line.replace("(","")
```

```
            gone = gone.replace(")","")
            gone = gone.replace(" ","")

            gone = gone.split(",")
            gone[1] = gone[1].replace('\n','')
            latitudes.append(float(gone[0]))
            longitudes.append(float(gone[1]))
    f.close()
    print("\n\nMap Coordinates Have Been Successfully Read In\n\n")

#################################################################
#####################################################

def find_Closest_Coordinate():

    global closeLat
    global closeLong

    #Set home position latitude and longitude
    originLat = home_coordinates['latitude']
    originLong = home_coordinates['longitude']

    #Get distance between home position and first map Coordinate
    homePosition = (originLat, originLong)
    firstPoint = (latitudes[0],longitudes[0])
    dist = geodesic(homePosition, firstPoint).meters

    #Set up other coordinates
    count = 0
```

```
    anotherPosition = (latitudes[count], longitudes[count])
    closeLat = latitudes[count]
    closeLong = longitudes[count]

    #Loop through coorinates
    for i in range(len(latitudes)):
        #Set Distance between home and another Coordinate
        anotherPosition = (latitudes[count], longitudes[count])
        anotherDist = geodesic(homePosition,
anotherPosition).meters

        print(anotherPosition)

        #If another distance smaller than the other then replace
and increment count
        if anotherDist < dist:

            print(dist)
            print(anotherDist)

            dist = anotherDist
            closeLat = float(latitudes[count])
            closeLong = float(longitudes[count])

            count +=1

        else:
            print(dist)
            print(anotherDist)
```

```
            count+=1


    print("\n\n")
    print(closeLat)
    print(closeLong)
    print(altitude)
    print("Closest Coordinate Found")


#################################################################
#################################################

def fly_To_Closest():
    global closeLat
    global closeLong
    global altitude


    #Fly to the nearest coordinate from the drones home position at
selected altitude
    drone(
        moveTo(closeLat, closeLong, altitude,
MoveTo_Orientation_mode.TO_TARGET,0.0)
            >> FlyingStateChanged(state="hovering",
_timeout=5)#'latitude', 52.65485574090526), ('longitude',
-6.653909008997573)
            >> moveToChanged(status='DONE')
            >> moveToChanged(latitude=closeLat,
longitude=closeLong, altitude=altitude,
orientation_mode=MoveTo_Orientation_mode.TO_TARGET, status='DONE',
_policy='wait')
            >> FlyingStateChanged(state="hovering", _timeout=5)
    ).wait()
```

```
#####################################################################
#####################################################

def divide_Length_Breadth():


    currentPositionNum = 0
    count = 0


    #Set up coordinates and measure distance between
    currentCoord = (closeLat, closeLong)
    nextCoord = (latitudes[count],longitudes[count])
    maxDistance = geodesic(currentCoord,nextCoord)
    maxDistNum = 0


    #Loop through list of coordinate latitudes
    for l in range(len(latitudes)):


        #Set up coordinates and measure distance between
        nextCoord = (latitudes[count],longitudes[count])
        distance = geodesic(currentCoord, nextCoord).meters
        print(distance)


        #If latitude is equiv to nearest coordinate to home
position set current position count
        if latitudes[count] == closeLat:
            currentPositionNum = count
            print("Current Coordinate: ")
            print(currentPositionNum)
            count+=1
```

```
        #If another distance is larger than another reset max
Distance and save position in array
        else:
            if distance > maxDistance:
                maxDistance = distance
                maxDistNum = count
                count+=1
            else:
                count+=1


    print(maxDistNum)
    print(maxDistance)


    #ignore the current position of the drone and the furthest
point from drone and get positions and distances of remaining
    lengthBreadthCoord = []
    lengthBreadthDistances = []
    for i in range(len(latitudes)):
        if i == currentPositionNum or i == maxDistNum:
            pass
        else:
            lengthBreadthCoord.append(i)
            lbPoint = (latitudes[i],longitudes[i])
            lengthBreadth = geodesic(currentCoord,lbPoint).meters
            lengthBreadthDistances.append(lengthBreadth)
            print(i)

##################################################################
##################################################
```

```
latitudes = []
longitudes = []
altitude = 0.0


altitudeDict = {
   0.9: {
   "photoL": 1,
   "photoW": 0.7
   },
   1.35:{
   "photoL": 2,
   "photoW": 1.3
   },
   1.7:{
   "photoL": 3,
   "photoW": 2.3
   },
   3.0:{
   "photoL": 5,
   "photoW": 4
   },
   7.0:{
   "photoL": 10,
   "photoW": 7.5
   },
}



closeLat = 0.0
closeLong = 0.0
```

```
DRONE_IP = "192.168.42.1" #Physical Drone IP Address
#DRONE_IP = "10.202.0.1" #Emulated Drone IP Address



#read in map chosen coordinates
read_In_Coordinates()
print(altitude)


drone = olympe.Drone(DRONE_IP, loglevel=0)


#Establish connection to physical or emulated drones
drone.connection()


# Wait for GPS to fix state
drone(GPSFixStateChanged(_policy = 'wait'))


# Call function to tilt camera and configure settings
setup_camera(drone)


#Set home coordinates
home_coordinates = drone.get_state(HomeChanged)


#Set the nearest longitude and latitude to home position
find_Closest_Coordinate()
```

```
print(drone.get_state(HomeChanged))
print(closeLat)
print(closeLong)
print(altitude)



#Wait for camera configurations to finish before take off
#time.sleep(5)


#Take off
drone(
   TakeOff()
   >> FlyingStateChanged(state="hovering", _timeout=5)# wait 5
seconds for hovering state
).wait()


#fly_To_Closest()


divide_Length_Breadth()




#Landing
drone(Landing()).wait()


#Disconnect from Drone
drone.disconnection()
```

faceDirection.py

```
# -*-
coding:
UTF-8 -
*-
```

```
import olympe
from olympe.messages.ardrone3.Piloting import TakeOff, moveBy,
Landing,moveTo
```

```
from olympe.messages.ardrone3.PilotingState import
moveToChanged,FlyingStateChanged, PositionChanged,
AttitudeChanged
from olympe.messages.camera import set_camera_mode,
set_photo_mode, take_photo, photo_progress
from olympe.messages.ardrone3.GPSSettingsState import
GPSFixStateChanged,HomeChanged
from olympe.enums.ardrone3.Piloting import
MoveTo_Orientation_mode
from olympe.messages import gimbal
import os
import re
import requests
import shutil
import xml.etree.ElementTree as ET
import time
import math


from geopy.distance import geodesic
#####################################################################
#####################################


def snap_and_save(drone):


    # Drone web server URL
    DRONE_URL = "http://{}/".format(DRONE_IP)
    # Drone media web API URL
    MEDIA_API = DRONE_URL + "api/v1/media/medias/"
    TAGS = (
        "GPSLatitude",
        "GPSLongitude",
        "GPSAltitude",
    )
```

```
    # Take photo, wait for saved status and get photo ID
    print("\n\nTaking a Photo\n\n")
    save_photo = drone(photo_progress(result="photo_saved",
_policy="wait"))
    drone(take_photo(cam_id=0)).wait()
    save_photo.wait()
    photo_id =
save_photo.received_events().last().args["media_id"]
    # Create media link
    media_link = requests.get(MEDIA_API + photo_id)
    media_link.raise_for_status()
    save_to_path = '/home/bernard/Desktop/images'#change to
generic directory
    #Request photo from media link
    for photos in media_link.json()["resources"]:
        picture = requests.get(DRONE_URL + photos["url"],
stream=True)
        print("\n\nGetting Picture From: ", DRONE_URL +
photos["url"])
        #create image directory name
        image_dir = os.path.join(save_to_path,
photos["resource_id"])
        picture.raise_for_status()
        #Copy Picture to Directory
        with open(image_dir, "wb") as photo_file:
            shutil.copyfileobj(picture.raw, photo_file)
        #Open image directory
        print("\nOpening ", image_dir)
        with open(image_dir, "rb") as photo_file:
            photo_data = photo_file.read()
            xmp_photo_start = photo_data.find(b"<x:xmpmeta")
            xmp_photo_end = photo_data.find(b"</x:xmpmeta")
            #parsing xmp metadata to check for GPS coordinates
```

```
            xmp_photo =
ET.fromstring(photo_data[xmp_photo_start : xmp_photo_end +
12])
            for photo_meta in xmp_photo[0][0]:
                xmpTag = re.sub(r"{[^}]*}", "", photo_meta.tag)
                value = photo_meta.text
                if xmpTag in TAGS:
                    print("Photo ID: ", photos["resource_id"],
xmpTag, value)


################################################################
################################################################


def setup_camera(drone):
    #set camera to photo mode
    drone(set_camera_mode(cam_id=0, value="photo")).wait()
    drone(
        set_photo_mode(
            cam_id=0,
            mode="single",
            format="rectilinear",
            file_format="jpeg",
            burst="burst_14_over_1s",
            bracketing="preset_1ev",
            capture_interval=0.0,
        )
    ).wait()


    #Tilt camera 90 degrees down
    tiltCamera = drone(gimbal.set_target(
    gimbal_id=0,
    control_mode="position",
```

```
    yaw_frame_of_reference="none",
    yaw=0.0,
    pitch_frame_of_reference="absolute",
    pitch=-90.0,
    roll_frame_of_reference="none",
    roll=0.0,
    )).wait()


    #If camera tilt does not occur raise runtime error
    if not tiltCamera.success():
        raise RuntimeError("Camera tilt did not execute
correctly")
    print("\n\nCamera Settings Configured\n\n")


############################################################
#########################################################


def read_In_Coordinates():
    #Open Coordinates file
    f = open("/home/bernard/Downloads/Coordinates.txt", "r")


    global altitude


    #Read in each line from file
    for line in f:
        #check for set Altitude
        if len(line) < 5:
            altitude = float(line)
        else:
            #Remove unneeded characters, add lats & longs to
lists
```

```
            gone = line.replace("(","")
            gone = gone.replace(")","")
            gone = gone.replace(" ","")

            gone = gone.split(",")
            gone[1] = gone[1].replace('\n','')
            latitudes.append(float(gone[0]))
            longitudes.append(float(gone[1]))
    f.close()
    print("\n\nMap Coordinates Have Been Successfully Read
In\n\n")

############################################################
#########################################################

def find_Closest_Coordinate():

    global closeLat
    global closeLong

    #Set home position latitude and longitude
    originLat = home_coordinates['latitude']
    originLong = home_coordinates['longitude']

    #Get distance between home position and first map
Coordinate
    homePosition = (originLat, originLong)
    aCoord = (latitudes[0], longitudes[0])
    dist = geodesic(homePosition, aCoord).meters
```

```
    #Set up other coordinates
    count = 0
    closeLat = latitudes[count]
    closeLong = longitudes[count]


    #Loop through coorinates
    for i in range(len(latitudes)):
        #Set Distance between home and another Coordinate
        anotherPosition = (latitudes[count], longitudes[count])
        anotherDist = geodesic(homePosition,
anotherPosition).meters


        print(anotherPosition)



        #If another distance smaller than the other then
replace and increment count
        if anotherDist < dist:
            print(dist)
            print(anotherDist)


            dist = anotherDist
            closeLat = float(latitudes[count])
            closeLong = float(longitudes[count])
            count +=1
        else:
            print(dist)
            print(anotherDist)
```

```
            count+=1


    print("\n\n")
    print(closeLat)
    print(closeLong)
    print(altitude)


    print("Closest Coordinate Found")


#############################################################
#########################################################


def fly_To_Closest():
    global closeLat
    global closeLong
    global altitude


    #Fly to the nearest coordinate from the drones home
position at selected altitude
    drone(
        moveTo(closeLat, closeLong, altitude,
MoveTo_Orientation_mode.TO_TARGET,0.0)
            >> FlyingStateChanged(state="hovering",
_timeout=5)#'latitude', 52.65485574090526), ('longitude',
-6.653909008997573)
            >> moveToChanged(status='DONE')
            >> moveToChanged(latitude=closeLat,
longitude=closeLong, altitude=altitude,
orientation_mode=MoveTo_Orientation_mode.TO_TARGET,
status='DONE', _policy='wait')
```

```
            >> FlyingStateChanged(state="hovering", _timeout=5)
    ).wait()


##############################################################
#########################################################

def get_Length_Breadth():


    currentPositionNum = 0
    count = 0


    #Set up coordinates and measure distance between
    currentCoord = (closeLat, closeLong)
    nextCoord = (latitudes[count],longitudes[count])
    maxDistance = geodesic(currentCoord,nextCoord)
    maxDistPos = 0


    #Loop through list of coordinate latitudes
    for l in range(len(latitudes)):


        #Set up coordinates and measure distance between
        nextCoord = (latitudes[count],longitudes[count])
        distance = geodesic(currentCoord, nextCoord).meters
        print(distance)


        #If latitude is equiv to nearest coordinate to home
position set current position count
        if latitudes[count] == closeLat:
            currentPositionNum = count
```

```
            print("Current Coordinate: ")
            print(currentPositionNum)


            count+=1


        #If another distance is larger than another reset max
Distance and save position in array
        else:
            if distance > maxDistance:
                maxDistance = distance
                maxDistPos = count
                count+=1
            else:
                count+=1


    print(maxDistPos)
    print(maxDistance)



    #ignore the current position of the drone and the furthest
point from drone and get positions and distances of remaining
    lBArrayPos = []
    lbDistances = []
    for i in range(len(latitudes)):
        if i == currentPositionNum or i == maxDistPos:
            pass
        else:
            lBArrayPos.append(i)
            lbPoint = (latitudes[i],longitudes[i])
```

```
            lengthBreadth =
geodesic(currentCoord,lbPoint).meters
            lbDistances.append(lengthBreadth)


            print(i)


    #Save length and breadth positions in array
    breadthNum = lBArrayPos[0]
    lengthNum = lBArrayPos[1]
    if lbDistances[1] < lbDistances[0]:
        breadthNum = lBArrayPos[1]
        lengthNum = lBArrayPos[0]


    #Call method to face drone towards breadth point
    face_Breadth(breadthNum)


##############################################################
#########################################################

def face_Breadth(breadthNum):


    global drone


    #Set points to get angle between given
    firstPoint = (closeLat, closeLong)
    secondPoint = (latitudes[breadthNum],
longitudes[breadthNum])
```

```
    if (type(firstPoint) != tuple) or (type(secondPoint) !=
tuple):
        raise TypeError("Data type is not a tuple")


    lat1 = math.radians(firstPoint[0])
    lat2 = math.radians(secondPoint[0])


    diffLong = math.radians(secondPoint[1] - firstPoint[1])


    #θ = atan2(sin(Δlong).cos(lat2),
                #cos(lat1).sin(lat2) – sin(lat1).cos(lat2).cos(Δlong))
    x = math.sin(diffLong) * math.cos(lat2)
    y = math.cos(lat1) * math.sin(lat2) - (math.sin(lat1)
            * math.cos(lat2) * math.cos(diffLong))


    bearing = math.atan2(x, y)


    print("\n\nBearing: ",bearing)


    #Get yaw(direction) of drone
    yaw = drone.get_state(AttitudeChanged)
    print("Yaw: ",yaw['yaw'])


    #take yaw from bearing of given points
    turn = (bearing - yaw['yaw'])


    #Pivot drone by difference
    drone(
```

```
        moveBy(0, 0, 0, turn)
        >> FlyingStateChanged(state="hovering", _timeout=5)
    ).wait()



##############################################################
############################################################


latitudes = []
longitudes = []
altitude = 0.0


altitudeDict = {
    0.9: {
    "photoL": 1,
    "photoW": 0.7
    },
    1.35:{
    "photoL": 2,
    "photoW": 1.3
    },
    1.7:{
    "photoL": 3,
    "photoW": 2.3
    },
    3.0:{
    "photoL": 5,
    "photoW": 4
    },
    7.0:{
```

```
    "photoL": 10,
    "photoW": 7.5
    },
}



closeLat = 0.0
closeLong = 0.0



#DRONE_IP = "192.168.42.1" #Physical Drone IP Address
DRONE_IP = "10.202.0.1" #Emulated Drone IP Address



#read in map chosen coordinates
read_In_Coordinates()
print(altitude)


drone = olympe.Drone(DRONE_IP, loglevel=0)


#Establish connection to physical or emulated drones
drone.connection()


# Wait for GPS to fix state
drone(GPSFixStateChanged(_policy = 'wait'))
```

```
#Wait for camera configurations to finish before take off
#time.sleep(5)


# Call function to tilt camera and configure settings
setup_camera(drone)


#Set home coordinates
home_coordinates = drone.get_state(HomeChanged)
print(drone.get_state(HomeChanged))




#Set the nearest longitude and latitude to home position
find_Closest_Coordinate()


print(drone.get_state(HomeChanged))
print(closeLat)
print(closeLong)
print(altitude)




#Wait for camera configurations to finish before take off
#time.sleep(5)


#Take off
drone(
   TakeOff()
   >> FlyingStateChanged(state="hovering", _timeout=5)# wait 5
seconds for hovering state
```

```
).wait()



#fly_To_Closest()



get_Length_Breadth()



#Landing
drone(Landing()).wait()

#Disconnect from Drone
drone.disconnection()
```

movementInstructions.py

```
# -*-
coding:
UTF-8 -
*-

import olympe
```

```
from olympe.messages.ardrone3.Piloting import TakeOff, moveBy,
Landing,moveTo

from olympe.messages.ardrone3.PilotingState import
moveToChanged,FlyingStateChanged, PositionChanged,
AttitudeChanged

from olympe.messages.camera import set_camera_mode,
set_photo_mode, take_photo, photo_progress

from olympe.messages.ardrone3.GPSSettingsState import
GPSFixStateChanged,HomeChanged

from olympe.enums.ardrone3.Piloting import
MoveTo_Orientation_mode

from olympe.messages import gimbal

import os

import re

import requests

import shutil

import xml.etree.ElementTree as ET

import time

import math



from geopy.distance import geodesic

#####################################################################
#######################################


def snap_and_save(drone):


    # Drone web server URL

    DRONE_URL = "http://{}/".format(DRONE_IP)

    # Drone media web API URL

    MEDIA_API = DRONE_URL + "api/v1/media/medias/"

    TAGS = (

        "GPSLatitude",

        "GPSLongitude",
```

```
        "GPSAltitude",
    )
    # Take photo, wait for saved status and get photo ID
    print("\n\nTaking a Photo\n\n")
    save_photo = drone(photo_progress(result="photo_saved",
_policy="wait"))
    drone(take_photo(cam_id=0)).wait()
    save_photo.wait()
    photo_id =
save_photo.received_events().last().args["media_id"]
    # Create media link
    media_link = requests.get(MEDIA_API + photo_id)
    media_link.raise_for_status()
    save_to_path = '/home/bernard/Desktop/images'#change to
generic directory
    #Request photo from media link
    for photos in media_link.json()["resources"]:
        picture = requests.get(DRONE_URL + photos["url"],
stream=True)
        print("\n\nGetting Picture From: ", DRONE_URL +
photos["url"])
        #create image directory name
        image_dir = os.path.join(save_to_path,
photos["resource_id"])
        picture.raise_for_status()
        #Copy Picture to Directory
        with open(image_dir, "wb") as photo_file:
            shutil.copyfileobj(picture.raw, photo_file)
        #Open image directory
        print("\nOpening ", image_dir)
        with open(image_dir, "rb") as photo_file:
            photo_data = photo_file.read()
            xmp_photo_start = photo_data.find(b"<x:xmpmeta")
            xmp_photo_end = photo_data.find(b"</x:xmpmeta")
            #parsing xmp metadata to check for GPS coordinates
```

```
            xmp_photo =
ET.fromstring(photo_data[xmp_photo_start : xmp_photo_end +
12])
            for photo_meta in xmp_photo[0][0]:
                xmpTag = re.sub(r"{[^}]*}", "", photo_meta.tag)
                value = photo_meta.text
                if xmpTag in TAGS:
                    print("Photo ID: ", photos["resource_id"],
xmpTag, value)


################################################################
################################################################


def setup_camera(drone):


    #set camera to photo mode
    drone(set_camera_mode(cam_id=0, value="photo")).wait()
    drone(
        set_photo_mode(
            cam_id=0,
            mode="single",
            format="rectilinear",
            file_format="jpeg",
            burst="burst_14_over_1s",
            bracketing="preset_1ev",
            capture_interval=0.0,
        )
    ).wait()


    #Tilt camera 90 degrees down
    tiltCamera = drone(gimbal.set_target(
```

```
    gimbal_id=0,
    control_mode="position",
    yaw_frame_of_reference="none",
    yaw=0.0,
    pitch_frame_of_reference="absolute",
    pitch=-90.0,
    roll_frame_of_reference="none",
    roll=0.0,
    )).wait()


    #If camera tilt does not occur raise runtime error
    if not tiltCamera.success():
        raise RuntimeError("Camera tilt did not execute
correctly")
    print("\n\nCamera Settings Configured\n\n")


#################################################################
#########################################################


def read_In_Coordinates():


    #Open Coordinates file
    f = open("/home/bernard/Downloads/Coordinates.txt", "r")


    global altitude


    #Read in each line from file
    for line in f:
        #check for set Altitude
        if len(line) < 5:
```

```
            altitude = float(line)
        else:
            #Remove unneeded characters, add lats & longs to
lists
            gone = line.replace("(","")
            gone = gone.replace(")","")
            gone = gone.replace(" ","")

            gone = gone.split(",")
            gone[1] = gone[1].replace('\n','')
            latitudes.append(float(gone[0]))
            longitudes.append(float(gone[1]))
    f.close()
    print("\n\nMap Coordinates Have Been Successfully Read
In\n\n")

############################################################
#########################################################

def find_Closest_Coordinate():

    global closeLat
    global closeLong

    #Set home position latitude and longitude
    originLat = home_coordinates['latitude']
    originLong = home_coordinates['longitude']

    #Get distance between home position and first map
Coordinate
```

```
    homePosition = (originLat, originLong)
    aCoord = (latitudes[0], longitudes[0])
    dist = geodesic(homePosition, aCoord).meters


    #Set up other coordinates
    count = 0
    closeLat = latitudes[count]
    closeLong = longitudes[count]


    #Loop through coorinates
    for i in range(len(latitudes)):
        #Set Distance between home and another Coordinate
        anotherPosition = (latitudes[count], longitudes[count])
        anotherDist = geodesic(homePosition,
anotherPosition).meters


        print(anotherPosition)




        #If another distance smaller than the other then
replace and increment count
        if anotherDist < dist:
            print(dist)
            print(anotherDist)


            dist = anotherDist
            closeLat = float(latitudes[count])
            closeLong = float(longitudes[count])
            count +=1
        else:
```

```
            print(dist)
            print(anotherDist)


            count+=1


    print("\n\n")
    print(closeLat)
    print(closeLong)
    print(altitude)


    print("Closest Coordinate Found")


#############################################################
#########################################################


def fly_To_Closest():


    global closeLat
    global closeLong
    global altitude


    #Fly to the nearest coordinate from the drones home
position at selected altitude
    drone(
        moveTo(closeLat, closeLong, altitude,
MoveTo_Orientation_mode.TO_TARGET,0.0)
            >> FlyingStateChanged(state="hovering",
_timeout=5)#'latitude', 52.65485574090526), ('longitude',
-6.653909008997573)
            >> moveToChanged(status='DONE')
```

```
            >> moveToChanged(latitude=closeLat,
longitude=closeLong, altitude=altitude,
orientation_mode=MoveTo_Orientation_mode.TO_TARGET,
status='DONE', _policy='wait')
            >> FlyingStateChanged(state="hovering", _timeout=5)
    ).wait()


#############################################################
#############################################################


def get_Length_Breadth():


    global drone
    currentPositionNum = 0
    currentCoord = (closeLat, closeLong)
    count = 0


    #Set up coordinates and measure distance between
    nextCoord = (latitudes[count],longitudes[count])
    maxDistance = geodesic(currentCoord,nextCoord)
    maxDistPos = 0


    #Loop through list of coordinate latitudes
    for l in range(len(latitudes)):


        #Set up coordinates and measure distance between
        nextCoord = (latitudes[count],longitudes[count])
        distance = geodesic(currentCoord, nextCoord).meters
        print(distance)
```

```
        #If latitude is equiv to nearest coordinate to home
position set current position count
        if latitudes[count] == closeLat:
            currentPositionNum = count


            print("Current Coordinate: ")
            print(currentPositionNum)


            count+=1
        else:
            if distance > maxDistance:
                maxDistance = distance
                maxDistPos = count
                count+=1
            else:
                count+=1


    print("Max Distance Position: ", maxDistPos)
    print("Max Distance: ", maxDistance)


    #ignore the current position of the drone and the furthest
point from drone and get positions and distances of remaining
    lBArrayPos = []
    lbDistances = []
    for i in range(len(latitudes)):
        if i == currentPositionNum or i == maxDistPos:
            pass
        else:
            lBArrayPos.append(i)
            lbPoint = (latitudes[i],longitudes[i])
```

```
            lengthBreadth =
geodesic(currentCoord,lbPoint).meters
            lbDistances.append(lengthBreadth)


            print(i)



    #Save length and breadth positions in array
    breadthNum = lBArrayPos[0]
    lengthNum = lBArrayPos[1]
    bDistPos = 0
    lDistPos = 1
    if lbDistances[1] < lbDistances[0]:
        breadthNum = lBArrayPos[1]
        lengthNum = lBArrayPos[0]
        bDistPos = 1
        lDistPos = 0


    print("Breadth Distance: ", lbDistances[bDistPos])
    print("Length Distance: ", lbDistances[lDistPos])


    #Get the yaw of the drone
    yaw = drone.get_state(AttitudeChanged)
    print("\n\nDrone Yaw: ",yaw['yaw'])


    #calculate the bearing to the breadth coordinate
    bBearing = calculate_Direction(breadthNum)
```

```
    #Set turn to the amount the drone needs to rotate by to
face the breadth coordinate
    turn = (bBearing - yaw['yaw'])
    print("Turn: ", turn)


    #Call method to face drone towards breadth point
    face_Breadth(turn)


    #Calculate the bearing of the length coordinate
    lBearing = calculate_Direction(lengthNum)
    print("Length Bearing", lBearing)


    #Find out if length coordinate is on the left(false) or the
right(true) handside of the drone
    leftRight = left_Or_Right(bBearing,lBearing)
    print("left or right: ", leftRight)


    #Divide the breadth and length distance by the the breadth
and length of a photo captured at chosen altitude
    stopsForward = lbDistances[bDistPos] /
altitudeDict.get(altitude,{}).get('photoW')
    print(stopsForward)
    stopsLR = lbDistances[lDistPos] /
altitudeDict.get(altitude,{}).get('photoL')
    print(stopsLR)


    #Send length and breadth movements with left or right
boolean
    movement_Instructions(stopsForward,stopsLR,leftRight)
```

```
#############################################################
#########################################################


def calculate_Direction(breadthLengthNum):


    #set coordinate points to get bearing of
    pointA = (closeLat, closeLong)
    pointB = (latitudes[breadthLengthNum],
longitudes[breadthLengthNum])


    #check for data type equals tuple, error if not
    if (type(pointA) != tuple) or (type(pointB) != tuple):
        raise TypeError("Only tuples are supported as
arguments")


    #Get points in radians
    lat1 = math.radians(pointA[0])
    lat2 = math.radians(pointB[0])


    diffLong = math.radians(pointB[1] - pointA[1])


    #θ = atan2(sin(Δlong).cos(lat2),
                #cos(lat1).sin(lat2) – sin(lat1).cos(lat2).cos(Δlong))
    x = math.sin(diffLong) * math.cos(lat2)
    y = math.cos(lat1) * math.sin(lat2) - (math.sin(lat1)
            * math.cos(lat2) * math.cos(diffLong))


    bearing = math.atan2(x, y)
    print("\n\nBearing: ",bearing)
```

```
    return bearing


#############################################################
#########################################################
def face_Breadth(turn):


    global drone


     #Pivot to face toward breadth coordinate
    drone(
        moveBy(0, 0, 0, turn)
        >> FlyingStateChanged(state="hovering", _timeout=5)
    ).wait()


#############################################################
#########################################################
def left_Or_Right(yaw,lengthBearing):


    #If drone yaw > -1.57 and Length bearing < yaw then length
cordinate is on the LHS of drone
    if yaw > -1.57 and lengthBearing < yaw:
        return False
    #length cordinate is on the LHS of drone
    elif yaw < -1.57 and lengthBearing > yaw:
        return False
    #length cordinate is on the RHS of drone
    elif yaw < 1.57 and yaw < lengthBearing:
        return True
    #length cordinate is on the RHS of drone
    elif yaw > 1.57 and yaw > lengthBearing:
        return True
```

```
#############################################################
#########################################################
def movement_Instructions(stopsForward, stopsLR,leftRight):


    global drone
    print(int(stopsForward))
    print(int(stopsLR))


    #Increment each by one to count for 50% offset of breadth
and length
    sF = int(stopsForward) + 1
    sLR = int(stopsLR) + 1


    # move forward and capture image if lr is even number
    for lr in range(sLR):


        #Take photo and save to device
        snap_and_save(drone)
        #Print GPS coordinates after moving right
        print("\n\nGPS position after moving right : ",
drone.get_state(PositionChanged),"\n\n")


        for fm in range(sF):


            if lr % 2 ==0:
                drone(
```

```
                    moveBy(altitudeDict.get(altitude,
{}).get('photoW'), 0, 0, 0)
                    >> FlyingStateChanged(state="hovering",
_timeout=5)
                ).wait()


                #Take photo and save to device
                snap_and_save(drone)
                #Print GPS coordinates after moving right
                print("\n\nGPS position after moving right : ",
drone.get_state(PositionChanged),"\n\n")


            #move backward and capture image if lr is odd
number
            else:


                drone(
                    moveBy((-altitudeDict.get(altitude,
{}).get('photoW')), 0, 0, 0)
                    >> FlyingStateChanged(state="hovering",
_timeout=5)
                ).wait()


                #Take photo and save to device
                snap_and_save(drone)
                #Print GPS coordinates after moving right
                print("\n\nGPS position after moving right : ",
drone.get_state(PositionChanged),"\n\n")


        #bank left or right depnding on the the position of the
length coordinate in relation to drones facing direction
        if leftRight == True:
```

```
            drone(
                moveBy(0, altitudeDict.get(altitude,
{}).get('photoL'), 0, 0)
                >> FlyingStateChanged(state="hovering",
_timeout=5)
            ).wait()


        elif leftRight == False:
            drone(
                moveBy(0, -altitudeDict.get(altitude,
{}).get('photoL'), 0, 0)
                >> FlyingStateChanged(state="hovering",
_timeout=5)
            ).wait()




##############################################################
#########################################################


latitudes = []
longitudes = []
altitude = 0.0


closeLat = 0.0
closeLong = 0.0


altitudeDict = {
   0.9: {
   "photoL": 1,
   "photoW": 0.7
   },
```

```
    1.35:{
    "photoL": 2,
    "photoW": 1.3
    },
    1.7:{
    "photoL": 3,
    "photoW": 2.3
    },
    3.0:{
    "photoL": 5,
    "photoW": 4
    },
    7.0:{
    "photoL": 10,
    "photoW": 7.5
    },
}


DRONE_IP = "192.168.42.1" #Physical Drone IP Address
#DRONE_IP = "10.202.0.1" #Emulated Drone IP Address




#read in map chosen coordinates
read_In_Coordinates()
print(altitude)


drone = olympe.Drone(DRONE_IP, loglevel=0)


#Establish connection to physical or emulated drones
```

```
drone.connection()


# Wait for GPS to fix state
drone(GPSFixStateChanged(_policy = 'wait'))


# Call function to tilt camera and configure settings
setup_camera(drone)


#Set home coordinates
home_coordinates = drone.get_state(HomeChanged)
print(drone.get_state(HomeChanged))


#Set the nearest longitude and latitude to home position
find_Closest_Coordinate()




print(drone.get_state(HomeChanged))
print(closeLat)
print(closeLong)
print(altitude)




#Take off
drone(
   TakeOff()
   >> FlyingStateChanged(state="hovering", _timeout=5)# wait 5
seconds for hovering state
).wait()
```

```
#Fly to the nearest coordinate
fly_To_Closest()



#Find the length and breath of the area
get_Length_Breadth()



#Landing
drone(Landing()).wait()


#Disconnect from Drone
drone.disconnection()
```

Project Files

index.html

```
<!
DOCTYPE
html>
        <html>
```

```
<head>
   <title>Agricultural Drone Survey System</title>
   <meta name="viewport" content="initial-scale=1.0">
   <meta charset="utf-8">
   <link rel="stylesheet" href="https://
stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/
bootstrap.min.css"
         integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/
Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh"
crossorigin="anonymous">
</head>
<body style="zoom: 0.9;">

<nav class="navbar navbar-dark fixed-top bg-dark flex-md-nowrap
p-0 shadow">
   <a class="navbar-brand col-sm-3 col-md-2 mr-0"
href="#">Agricultural Drone Survey System</a>
</nav>

<div class="mx-2 mt-5">
   <div class="row">

       <main role="main" class="col-sm-11 mx-auto">

           <div class="d-flex justify-content-between flex-wrap
flex-md-nowrap align-items-center pt-3 pb-2 mb-3 border-
bottom">
               <h1 class="h2">Choose Survey Area</h1>
           </div>

           <div class="row ">
```

```
                <div class="col-8">
                    <div class="w-100" style="height: 75vh"
id="map"></div>
                </div>
                <div class="col-4">
                    <div style="max-height: 300px; overflow-x:
unset; overflow-y: scroll">
                        <table class="table"
id="coordinatesTable">
                            <thead>
                            <tr>
                                <td>Latitude</td>
                                <td>Longitude</td>
                            </tr>
                            </thead>
                            <tbody>
                            </tbody>
                        </table>
                    </div>
                    <br>
                    <p><b>Altitudes correspond directly to the
Area Captured within a Photo (Photo Width  & Height)</b></p>
                    <label id="altLabel"
for="groundCover">Please Select an Altitude / Photo Width</
label>
                    <select class="w-100" id="groundCover">
                        <option value="0.9">Altitude: .9
Meters / Width: 1 Meter</option>
                        <option value="1.35">Altitude 1.4 Meters
/ Width: 2 Meters</option>
                        <option value="1.7">Altitude: 1.7 Meters
/ Width: 3 Meters</option>
                        <option value="3.0">Altitude: 3 Meters /
Width: 5 Meters</option>
```

```
                        <option value="7.0">Altitude: 7 Meters /
Width: 10 Meters</option>
                    </select>
                    <br>
                    <button onclick="run_drone_mapper()">Run</
button>
                    <button onclick="reset()">Reset</button>
                </div>


            </div>


            <div id="images_container" class="row"></div>


        </main>
    </div>
</div>


<script src="https://code.jquery.com/jquery-3.4.1.min.js"></
script>
<script src="https://maps.googleapis.com/maps/api/js?
key=AIzaSyDzFXegu2gBl6Nxp9Z4iB1pphHDm_ppm4Q&callback=initMap"
async
        defer></script>
<script>
    var map;
    var area;
    var path


    function initMap()
    {
        map = new
google.maps.Map(document.getElementById('map'),
```

```
        {
            center: {lat: 52.654, lng: -6.654},
            zoom: 16,
            mapTypeId: 'satellite'
        });

    area = new google.maps.Polygon({
        map: map,
        strokeColor: "#FF0000",
        strokeOpacity: 1.0,
        strokeWeight: 2,
        fillColor: "#FF0000",
        fillOpacity: 0.10,
        editable: true,
        draggable: false,
        clickable: true
    });
    area.setMap(map);

    map.addListener('click', addLatLng);
}

function addLatLng(event)
{
    path = area.getPath();

    if(path.length > 3)
    {
        alert('You Can Only Select Four Coordinates');
    }
```

```
    else
    {

    let latAndLong = event.latLng;
    path.push(latAndLong);

    $('#coordinatesTable tbody').append(`
                        <tr>
                            <td>${latAndLong.lat()}</td>
                            <td>${latAndLong.lng()}</td>
                        </tr>`);
    }
}

function reset() {
    $('#coordinatesTable tbody').empty();
    while(area.getPath().length > 0) {
        area.getPath().pop();
    }
}

function run_drone_mapper()
{

    if(path.length < 4)
    {
        alert('You Must Select Four Coordinates');
    }
```

```
else
{


    let path = area.getPath();
    let coords = [];
    for(let index in path["i"]) {
        let co = path["i"][index];
        let lat = co.lat();
        let lng = co.lng();


        let coord = {
            "lat": lat,
            "lng": lng
        };
        coords.push(coord)
    }


    let alt = $('#groundCover').val();


    let data = {};
    data['coordinates'] = coords;
    data['altitude'] = alt;
    $.ajax({
        url: "run",
        data: JSON.stringify(data),
        contentType: "json",
        type: "POST"
    })
```

```
        }
    }

    function check_for_images() {
        var intervalID = window.setInterval(myCallback, 5000);

        function myCallback() {
         $.ajax({
            url: "images",
            dataType: "json",
            type: "GET",
            success: function(images) {
                display_images(images);
            },
            error: function(err) {
                alert(err);
            }
        });
        }
    }

    check_for_images();

    function display_images(img_data) {
        let images_container =
document.getElementById('images_container');

        let images = "";
        for(let index in img_data) {
            let img_url = "static/images/" + img_data[index];
```

```
            let image_template = `<div class="col-3"><img
class="img-thumbnail" src="${img_url}"/></div>`;
            images += image_template;
        }
        images_container.innerHTML = images;
    }


</script>


</body>
</html>
```

app.py

```
from flask import
Flask, request
                    import droneMapper
                    import json
                    import os
                    app = Flask(__name__)
```

```
@app.route('/')
def display_html_page():
   with open('index.html') as file:
       page = file.read()
   return page



@app.route('/run', methods=["POST"])
def run_drone_mapper():


   # Write Co-ordinates to file


   req = request
   content = json.loads(req.data)
   with open("Coordinates.txt", "w+") as file:
       for coord in content['coordinates']:
           file.write("({},
{})".format(coord['lat'], coord['lng']))
           file.write('\n')
       file.write(content['altitude'])


   # run drone script
   droneMapper.main()
   return content


@app.route('/images')
def get_images():
```

```
    images_urls = {}
    count = 0


    path_to_images = 'static/images/'
    for entry in os.listdir(path_to_images):
        if
os.path.isfile(os.path.join(path_to_images,
entry)):
            print(entry)
            images_urls[count] = entry
            count += 1


    return images_urls




if __name__ == '__main__':
    app.run()
```

droneMapper.py

```
imp
ort
oly
mpe
    from olympe.messages.ardrone3.Piloting import TakeOff, moveBy,
    Landing,moveTo
    from olympe.messages.ardrone3.PilotingState import
    moveToChanged,FlyingStateChanged, PositionChanged, AttitudeChanged
    from olympe.messages.camera import set_camera_mode, set_photo_mode,
    take_photo, photo_progress
    from olympe.messages.ardrone3.GPSSettingsState import
    GPSFixStateChanged,HomeChanged
    from olympe.enums.ardrone3.Piloting import MoveTo_Orientation_mode
    from olympe.messages import gimbal
    import os,webbrowser
    import re
    import shutil
    import xml.etree.ElementTree as ET
    import time
    import math
    import sys
    import glob
    from PIL import Image
    from geopy.distance import geodesic
    import requests
    from requests.exceptions import HTTPError
```

```
#DRONE_IP = "192.168.42.1" #Physical Drone IP Address
DRONE_IP = "10.202.0.1" #Emulated Drone IP Address



latitudes = []
longitudes = []
altitude = 0.0


closeLat = 0.0
closeLong = 0.0


altitudeDict = {
   0.9: {
   "photoL": 1,
   "photoW": 0.7
   },
   1.35:{
   "photoL": 2,
   "photoW": 1.3
   },
   1.7:{
   "photoL": 3,
   "photoW": 2.3
   },
   3.0:{
   "photoL": 5,
   "photoW": 4
```

```
    },
    7.0:{
    "photoL": 10,
    "photoW": 7.5
    },
}
```

```
##################################################################
################################
```

```
def snap_and_save(drone):
```

```
    # Drone web server URL
    DRONE_URL = "http://{}/".format(DRONE_IP)
    # Drone media web API URL
    MEDIA_API = DRONE_URL + "api/v1/media/medias/"
    TAGS = (
        "GPSLatitude",
        "GPSLongitude",
        "GPSAltitude",
    )
    # Take photo, wait for saved status and get photo ID
    print("\n\nTaking a Photo\n\n")
    save_photo = drone(photo_progress(result="photo_saved",
_policy="wait"))
    drone(take_photo(cam_id=0)).wait()
    save_photo.wait()
    photo_id = save_photo.received_events().last().args["media_id"]
```

```
    # Create media link
    media_link = requests.get(MEDIA_API + photo_id)


    success = False
    while success == False:
        try:


            media_link.raise_for_status()


        except HTTPError:
            delay = 5
            print('\t --CONNECTION ERROR--',
                  '\n\t Sleeping for {} seconds.'.format(delay))
            #time.sleep(delay)
            media_link = requests.get(MEDIA_API + photo_id)
        else:
            success = True
            print('Success')


    save_to_path = 'static/images'#change to generic directory
    #Request photo from media link
    for photos in media_link.json()["resources"]:
        picture = requests.get(DRONE_URL + photos["url"],
stream=True)
        print("\n\nGetting Picture From: ", DRONE_URL +
photos["url"])
        #create image directory name
        image_dir = os.path.join(save_to_path,
photos["resource_id"])
        picture.raise_for_status()
```

```
        #Copy Picture to Directory
        with open(image_dir, "wb") as photo_file:
            shutil.copyfileobj(picture.raw, photo_file)
        #Open image directory


        print("\nOpening ", image_dir)
        with open(image_dir, "rb") as photo_file:
            photo_data = photo_file.read()
            xmp_photo_start = photo_data.find(b"<x:xmpmeta")
            xmp_photo_end = photo_data.find(b"</x:xmpmeta")


            #parsing xmp metadata to check for GPS coordinates
            xmp_photo = ET.fromstring(photo_data[xmp_photo_start :
xmp_photo_end + 12])
            for photo_meta in xmp_photo[0][0]:
                xmpTag = re.sub(r"{[^}]*}", "", photo_meta.tag)
                value = photo_meta.text
                if xmpTag in TAGS:
                    print("Photo ID: ", photos["resource_id"],
xmpTag, value)


#####################################################################
#########################################################


def setup_camera(drone):


   #set camera to photo mode
   drone(set_camera_mode(cam_id=0, value="photo")).wait()
   drone(
       set_photo_mode(
           cam_id=0,
```

```
            mode="single",
            format="rectilinear",
            file_format="jpeg",
            burst="burst_14_over_1s",
            bracketing="preset_1ev",
            capture_interval=0.0,
        )
    ).wait()

    #Tilt camera 90 degrees down
    tiltCamera = drone(gimbal.set_target(
    gimbal_id=0,
    control_mode="position",
    yaw_frame_of_reference="none",
    yaw=0.0,
    pitch_frame_of_reference="absolute",
    pitch=-90.0,
    roll_frame_of_reference="none",
    roll=0.0,
    )).wait()


    #If camera tilt does not occur raise runtime error
    if not tiltCamera.success():
        raise RuntimeError("Camera tilt did not execute correctly")
    print("\n\nCamera Settings Configured\n\n")


#####################################################################
#################################################

def read_In_Coordinates():
```

```
    #Open Coordinates file
    f = open("Coordinates.txt", "r")



    #Read in each line from file
    for line in f:
        #check for set Altitude
        if len(line) < 5:
            altitude = float(line)
        else:
            #Remove unneeded characters, add lats & longs to lists
            gone = line.replace("(","")
            gone = gone.replace(")","")
            gone = gone.replace(" ","")


            gone = gone.split(",")
            gone[1] = gone[1].replace('\n','')
            latitudes.append(float(gone[0]))
            longitudes.append(float(gone[1]))
    f.close()
    print("\n\nMap Coordinates Have Been Successfully Read In\n\n")


#################################################################
################################################


def find_Closest_Coordinate(home_coordinates):
```

```
    #Set home position latitude and longitude
    originLat = home_coordinates['latitude']
    originLong = home_coordinates['longitude']


    #Get distance between home position and first map Coordinate
    homePosition = (originLat, originLong)
    aCoord = (latitudes[0], longitudes[0])
    dist = geodesic(homePosition, aCoord).meters


    #Set up other coordinates
    count = 0
    closeLat = latitudes[count]
    closeLong = longitudes[count]


    #Loop through coorinates
    for i in range(len(latitudes)):
        #Set Distance between home and another Coordinate
        anotherPosition = (latitudes[count], longitudes[count])
        anotherDist = geodesic(homePosition, anotherPosition).meters




        #If another distance smaller than the other then replace and
increment count
        if anotherDist < dist:


            dist = anotherDist
            closeLat = float(latitudes[count])
            closeLong = float(longitudes[count])
```

```
            count +=1
        else:
            count+=1


    print("Closest Coordinate Found")


#####################################################################
################################################

def fly_To_Closest(closeLat, closeLong, altitude):


    #Fly to the nearest coordinate from the drones home position at
selected altitude
    drone(
        moveTo(closeLat, closeLong, altitude,
MoveTo_Orientation_mode.TO_TARGET,0.0)
            >> FlyingStateChanged(state="hovering", _timeout=5)
            >> moveToChanged(status='DONE')
            >> moveToChanged(latitude=closeLat, longitude=closeLong,
altitude=altitude,
orientation_mode=MoveTo_Orientation_mode.TO_TARGET, status='DONE',
_policy='wait')
            >> FlyingStateChanged(state="hovering", _timeout=5)
    ).wait()


#####################################################################
################################################

def get_Length_Breadth():


    currentPositionNum = 0
```

```
    currentCoord = (closeLat, closeLong)
    count = 0


    #Set up coordinates and measure distance between
    nextCoord = (latitudes[count],longitudes[count])
    maxDistance = geodesic(currentCoord,nextCoord)
    maxDistPos = 0


    #Loop through list of coordinate latitudes
    for l in range(len(latitudes)):


        nextCoord = (latitudes[count],longitudes[count])
        distance = geodesic(currentCoord, nextCoord).meters


        #If latitude is equiv to nearest coordinate to home position
set current position count
        if latitudes[count] == closeLat:
            currentPositionNum = count


            count+=1
        else:
            if distance > maxDistance:
                maxDistance = distance
                maxDistPos = count
                count+=1
            else:
                count+=1


    #ignore the current position of the drone and the furthest point
from drone and get positions and distances of remaining
```

```
lBArrayPos = []
lbDistances = []
for i in range(len(latitudes)):
    if i == currentPositionNum or i == maxDistPos:
        pass
    else:
        lBArrayPos.append(i)
        lbPoint = (latitudes[i],longitudes[i])
        lengthBreadth = geodesic(currentCoord,lbPoint).meters
        lbDistances.append(lengthBreadth)

#Save length and breadth positions in array
breadthNum = lBArrayPos[0]
lengthNum = lBArrayPos[1]
bDistPos = 0
lDistPos = 1
if lbDistances[1] < lbDistances[0]:
    breadthNum = lBArrayPos[1]
    lengthNum = lBArrayPos[0]
    bDistPos = 1
    lDistPos = 0

print("Breadth Distance: ", lbDistances[bDistPos])
print("Length Distance: ", lbDistances[lDistPos])

#Get the yaw of the drone
yaw = drone.get_state(AttitudeChanged)
print("\n\nDrone Yaw: ",yaw['yaw'])

#calculate the bearing to the breadth coordinate
```

```
    bBearing = calculate_Direction(breadthNum)


    #Set turn to the amount the drone needs to rotate by to face the
breadth coordinate
    turn = (bBearing - yaw['yaw'])


    #Call method to face drone towards breadth point
    face_Breadth(turn)


    #Calculate the bearing of the length coordinate
    lBearing = calculate_Direction(lengthNum)
    print("Length Bearing", lBearing)


    #Find out if length coordinate is on the left(false) or the
right(true) handside of the drone
    leftRight = left_Or_Right(bBearing,lBearing)
    print("left or right: ", leftRight)


    #Divide the breadth and length distance by the the breadth and
length of a photo captured at chosen altitude
    stopsForward = lbDistances[bDistPos] /
altitudeDict.get(altitude,{}).get('photoW')
    stopsLR = lbDistances[lDistPos] / altitudeDict.get(altitude,
{}).get('photoL')


    #Send length and breadth movements with left or right boolean
    movement_Instructions(stopsForward,stopsLR,leftRight)


#####################################################################
################################################
```

```
def calculate_Direction(breadthLengthNum):



    #set coordinate points to get bearing of
    pointA = (closeLat, closeLong)
    pointB = (latitudes[breadthLengthNum],
longitudes[breadthLengthNum])


    #check for data type equals tuple, error if not
    if (type(pointA) != tuple) or (type(pointB) != tuple):
        raise TypeError("Only tuples are supported as arguments")


    #Get points in radians
    lat1 = math.radians(pointA[0])
    lat2 = math.radians(pointB[0])


    diffLong = math.radians(pointB[1] - pointA[1])


    #θ = atan2(sin(Δlong).cos(lat2),
                #cos(lat1).sin(lat2) – sin(lat1).cos(lat2).cos(Δlong))
    x = math.sin(diffLong) * math.cos(lat2)
    y = math.cos(lat1) * math.sin(lat2) - (math.sin(lat1)
            * math.cos(lat2) * math.cos(diffLong))


    bearing = math.atan2(x, y)
    print("\n\nBearing: ",bearing)
```

```
    return bearing


#####################################################################
#################################################
def face_Breadth(turn):


     #Pivot to face toward breadth coordinate
    drone(
        moveBy(0, 0, 0, turn)
        >> FlyingStateChanged(state="hovering", _timeout=5)
    ).wait()


#####################################################################
#################################################
def left_Or_Right(yaw,lengthBearing):


    #If drone yaw > -1.57 and Length bearing < yaw then length
cordinate is on the LHS of drone
    if yaw > -1.57 and lengthBearing < yaw:
        return False


    #length cordinate is on the LHS of drone
    elif yaw < -1.57 and lengthBearing > yaw:
        return False


    #length cordinate is on the RHS of drone
    elif yaw < 1.57 and yaw < lengthBearing:
        return True
```

```
    #length cordinate is on the RHS of drone
    elif yaw > 1.57 and yaw > lengthBearing:
        return True



#####################################################################
#################################################
def movement_Instructions(stopsForward, stopsLR,leftRight):


    #Increment each by one to count for 50% offset of breadth and
length
    sF = int(stopsForward) + 1
    sLR = int(stopsLR) + 2




    #Set image size
    photo = Image.open('test1.JPG')
    photo_Length = (photo.width * sLR)
    photo_Breadth = (photo.height * sF) + photo.height
    new_map = Image.new('RGB', (photo_Length,photo_Breadth))


    if leftRight == True:
        photo_X = 0
    else:
        photo_X = photo_Length - photo.width
```

```
    photo_Y = photo_Breadth - photo.height



    # move forward and capture image if lr is even number
    for lr in range(sLR):


        #Take photo and save to device
        snap_and_save(drone)


        list_of_files = glob.glob('static/images/*')
        latest_file = max(list_of_files, key=os.path.getctime)
        latest_image = Image.open(latest_file)


        #stitch image
        new_map.paste(latest_image,(photo_X,photo_Y))






        for fm in range(sF):


            if lr % 2 ==0:
                drone(
                    moveBy(altitudeDict.get(altitude,
{}).get('photoW'), 0, 0, 0)
                    >> FlyingStateChanged(state="hovering",
_timeout=5)
                ).wait()
```

```
                #Take photo and save to device
                snap_and_save(drone)


                #take away breadth
                #stitch image
                photo_Y -= photo.height


                list_of_files = glob.glob('static/images/*')
                latest_file = max(list_of_files,
key=os.path.getctime)
                latest_image = Image.open(latest_file)


                #stitch image
                new_map.paste(latest_image,(photo_X,photo_Y))









            #move backward and capture image if lr is odd number
            else:


                drone(
                    moveBy((-altitudeDict.get(altitude,
{}).get('photoW')), 0, 0, 0)
                    >> FlyingStateChanged(state="hovering",
_timeout=5)
                ).wait()
```

```
                #Take photo and save to device
                snap_and_save(drone)


                #add breadth
                #stitch image
                photo_Y += photo.height


                list_of_files = glob.glob('static/images/*')
                latest_file = max(list_of_files,
key=os.path.getctime)
                latest_image = Image.open(latest_file)


                #stitch image
                new_map.paste(latest_image,(photo_X,photo_Y))




        #bank left or right depnding on the the position of the
length coordinate in relation to drones facing direction
        if leftRight == True:
            drone(
                moveBy(0, altitudeDict.get(altitude,
{}).get('photoL'), 0, 0)
                >> FlyingStateChanged(state="hovering", _timeout=5)
            ).wait()
            photo_X += photo.width
```

```
        elif leftRight == False:
            drone(
                moveBy(0, -altitudeDict.get(altitude,
{}).get('photoL'), 0, 0)
                >> FlyingStateChanged(state="hovering", _timeout=5)
            ).wait()
            photo_X -= photo.width


    new_map.save('test.jpg')




#####################################################################
#################################################

def return_Home(home_coordinates,altitude):


    #return to home position
    drone(
        moveTo(home_coordinates["latitude"],
home_coordinates["longitude"], altitude,
MoveTo_Orientation_mode.TO_TARGET, 0.0)
        >> FlyingStateChanged(state="hovering", _timeout=5)
        >> moveToChanged(status='DONE')
        >> moveToChanged(latitude=home_coordinates["latitude"],
longitude=home_coordinates["longitude"], altitude=altitude,
orientation_mode=MoveTo_Orientation_mode.TO_TARGET, status='DONE',
_policy='wait')
        >> FlyingStateChanged(state="hovering", _timeout=5)
    ).wait()
```

```
#########################################################################
#################################################
def main():

    global drone
    drone = olympe.Drone(DRONE_IP, loglevel=0)


    #read in map chosen coordinates
    read_In_Coordinates()




    #Establish connection to physical or emulated drones
    drone.connection()



    # Wait for GPS to fix state
    drone(GPSFixStateChanged(_policy = 'wait'))



    # Call function to tilt camera and configure settings
    setup_camera(drone)


    #Set home coordinates
```

```
    home_coordinates = drone.get_state(HomeChanged)



    #Set the nearest longitude and latitude to home position
    find_Closest_Coordinate(home_coordinates)



    #Take off
    drone(
        TakeOff()
        >> FlyingStateChanged(state="hovering", _timeout=5)# wait 5
seconds for hovering state
    ).wait()



    #Fly to the nearest coordinate
    fly_To_Closest(closeLat, closeLong, altitude)



    #Find the length and breath of the area
    get_Length_Breadth()



    #Return home
    return_Home(home_coordinates,altitude)
```

```
browser = webbrowser.get("firefox")
browser.open("test.jpg")


#Landing
drone(Landing()).wait()




#Disconnect from Drone
drone.disconnection()
```

References:

Renard, J., 2020. *Compass Bearing Between Two Points In Python*. [online] GitHub Gist. Available at: <https://gist.github.com/jeromer/2005586> [Accessed 1 April 2020].

INSTITUTE of TECHNOLOGY CARLOW

Institiúid Teicneolaíochta Cheatharlach

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 offense.

Student Name: Bernard Steemers

Student Number: C00235159

Signature:

Date: 20/04/2020