Ubuntu & Gazebo with PX4 Firmware

Peyman Amiri

1 Installing Ubuntu Desktop

1.1 Downloading Ubuntu and making live USB

1) Download the latest LTS (long-term support) version of Ubuntu from https://ubuntu.com/download/desktop. Save the iso file on your computer (not on a flash memory).

2) Download Rufus to make a live USB from https://rufus.ie/en/.

3) Now, open Rufus and attach your flash memory. In Rufus, click on Select and choose the downloaded iso file. Also, select GPT for "Partition scheme" and UEFI for "Target system". Then, click on START. If you see a window asking you how to write the image, click on "Write in ISO image mode (Recommended)".

NOTE: All the data on the flash memory will be deleted through this process.

4) When you see "Ready" in Status section, you can click on "Close" and restart your computer to commence installing Ubuntu.

(Section 1.1 is inferred from https://itsfoss.com/create-live-usb-of-ubuntu-in-windows/)

1.2 Installing Ubuntu on an external hard drive

NOTE: When you install Ubuntu on an external hard drive, you can have a portable Ubuntu and just. Your external hard drive and computer should support USB3.

1) When the computer is starting, press Esc key (for ASUS laptops). Then from the "boot driver", choose "UEFI: name of your flash memory". UEFI is specification that defines a software interface between an operating system and platform firmware.

2) Follow the instructions on step-by-step (select your external hard drive instead of flash memory mentioned on this page):

https://www.fosslinux.com/10212/how-to-install-a-complete-ubuntu-on-a-usb-flash-drive.htm

Note: Don't install Ubuntu on a flash memory because it will get so hot and causes some overcurrent issues.

Note: I assigned 100 Gigabytes of my external hard to FAT32 and 100 Gigabytes of it to EXT4 for Ubuntu at the partitioning step mentioned on the above website.

3) At the end, restart the computer. When the computer is starting, press Esc key (for ASUS laptops). Then from the "boot driver", **don't choose** any "UEFI" and just choose your hard drive name to start Ubuntu.

Note 1: Ubuntu uses LibreOffice as Office in Windows. However, LibreOffice doesn't have Times New Roman and some other fonts by default. So you should add these fonts to LibreOffice. Follow the instructions of this clip to do so: https://www.youtube.com/watch?v=pi4lI5AqMww

Note 2: To number the headings of a whole document in LibreOffice, we should click on Tools then on Chapter Numbering. Then you can assign Level 1 to heading 1, Level 2 to heading to and so on. Then click on Insert then on Cross-reference to cite the headings.

1.3 Partitioning the rest of hard drive

Now we should partition the rest of the external hard drive to use it to save your files.

1) On windows, install EaseUS Partition Master software.

2) Click on “unallocated” part of your external hard and then click on Create at the right side.

3) Assign this remaining space as NTFS to store your data by choosing File system as NTFS and click on OK. Afterward, click on “Execute Operations”.

Note: Windows can just recognize FAT32 and NTFS file system to store data.

2 Installing Gazebo

1) Go to http://gazebosim.org/.

2) Click on Download icon. Then on Debian icon.

3) Copy the download script and paste it in Terminal to download and install Gazebo.

(You can find Terminal by searching it in your Applications)

3 Installing PX4 Firmware (PX4-Autopilot)

Install the firmware step-by-step according to the official clip on:

https://docs.px4.io/master/en/dev_setup/dev_env_linux_ubuntu.html

4 Running Gazebo with PX4 Firmware (PX4-Autopilot)

Now, you have Gazebo and PX4 Firmware installed on your Ubuntu.

1) Open PX4 Firmware through writing `cd Firmware` in Terminal.

(On PX4 website, it is mentioned to use `cd/path/to/PX4-Autopilot` as the PX4 repository, but be aware that `cd/path/to/` is just a symbol of a path to Firmware of PX4-Autopilot which is `cd Firmware`)

2) Then write `make px4_sitl gazebo` in Terminal.

(Section 4 is inferred from: https://docs.px4.io/master/en/simulation/gazebo.html)

5 Installing QgroundControl and Running it with Gazebo and PX4 Firmware

5.1 Installing QgroundControl

Through QgroundControl, you can plan the desired path for your drone visually and run your simulation with Gazebo and PX4 Firmware.

1) go to http://qgroundcontrol.com/ and click on download icon.

2) Write the following commands mentioned on the above website for Linux Ubuntu in Terminal, one-by-one and press Enter.

```
sudo usermod -a -G dialout $USER
sudo apt-get remove modemmanager -y
sudo apt install gstreamer1.0-plugins-bad gstreamer1.0-libav gstreamer1.0-gl -y
```

3) Click on the hyperlink of “1. Download QgroundControl.AppImage.” on the website to download the file.

4) Now, open Downloads file of your computer and check file “QgroundControl.AppImage” to be there. Right click on the environment of Downloads and select “Open in Terminal”.

5) Write the following commands mentioned on the above website in Terminal and press Enter.

```
chmod +x ./QGroundControl.AppImage
```

6) Now, you have QGroundControl installed on your computer. You can open it by double clicking on file QgroundControl.AppImage in Downloads.

(Section 5.1 is inferred from 2:40 to 4:15 of the clip on: https://www.youtube.com/watch?v=-zEddRFbMvQ)

5.2 Running QGroundControl with Gazebo and PX4 Firmware

1) Open QGoruondControl by double clicking on file QgroundControl.AppImage in Downloads.

2) Write `cd Firmware` in the terminal. Write the following codes in terminal to make the quadrotor at its home location and open Gazebo with PX4. (The following location is Manchester city)

```
export PX4_HOME_LAT=53.48
export PX4_HOME_LON=-2.24
export PX4_HOME_ALT=0
make px4_sitl gazebo
```

3) Now you should be able to see your quadrotor as a red arrow in QGroundControl and also in Gazebo. If you click on the quadrotor in Gazebo, you can see the quadrotor in a white cube to be more obvious.

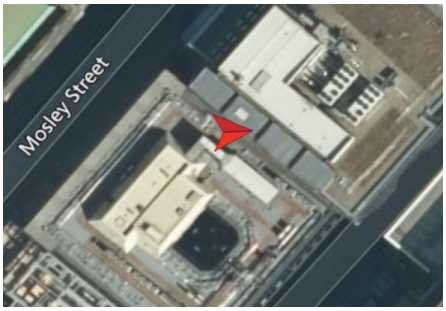

Figure 1: Arrow of the quadrotor in QGroundControl

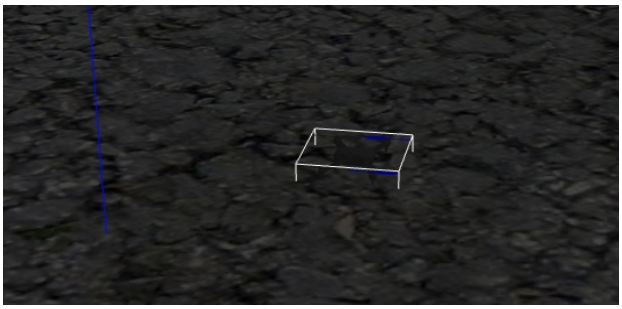

Figure 2: The quadrotor in Gazebo

4) Click on Plan in Fly tools and then on Survey from Create Pan to make your desired plan for your quadrotor.

Figure 3: Create Plan

5) Now, click on Waypoints icon in Plan tools (as you can see in the following figure) to add Waypoints. Consider the following tips to create your Plan.

Figure 4: Plan tools

Tip 1: Waypoint n is necessarily connected to Waypoint n-1 and the quadrotor flies from Waypoint n-1 to n.

Tip 2: When Waypoint icon in Plan tools is inactive, click on a Waypoint that you want to add the next Waypoint after, to make it active. For example, if you have 16 Waypoints, and click on Waypoint 10 and then click on Waypoint icon, Waypoint 11 will be added and connected to Waypoint 10, the number of Waypoints after 11 will be added by 1 and you will totally have 17 Waypoints.

Tip 3: When you make changes to your plan, "Upload Required" icon starts blinking on the top of the page. Click on it after you have completed your plan to send the new plan to the quadrotor.

Tip 4: If an item on the right list is in red, it means that the item is incomplete and you cannot save and upload your Plan. Then, you should modify the item or delete it by clicking on trash bin icon on the item. Delete Survey item on the right list because it is about some Polygon tools we do not need here.

Tip 5: You can save your plan by clicking on File in Plan tools and clicking on Save As icon.

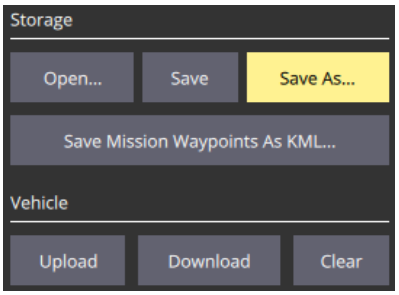

Figure 5: Save As icon to save a plan

Tip 6: if you want to reopen QgroundControl, just write commands 1 and 2 (mentioned above) in Terminal to create the quadrotor and click on Plan in Fly tools. Afterward, click on File and then click on Open to select your previously saved plan and run it. Afterward, click on "Upload Requiem" icon on the top of the page to upload the plan on the quadrotor.

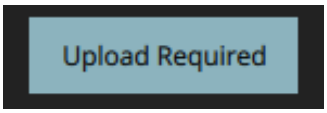

Figure 6: Upload Required icon

6) Now, click on Fly icon in Plan tools and then slide the following white arrow to right to start the mission. Also you can see the altitude, speed and the length of traveled path in this section.

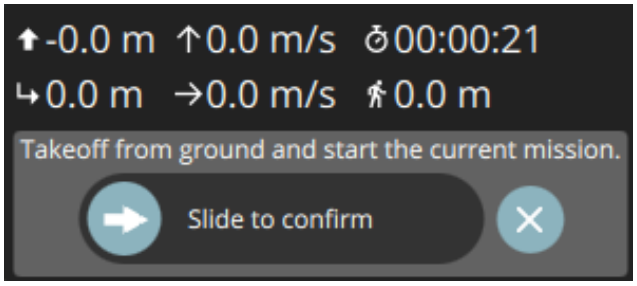

Figure 7: Confirmation to start a mission

Note: We designed a plan named Mission_1.plan and you can use it and other files in Mission_1 folder. You can see the traveled path of Mission_1 in Figure 8 (the quadrotor returned home through the upper line):

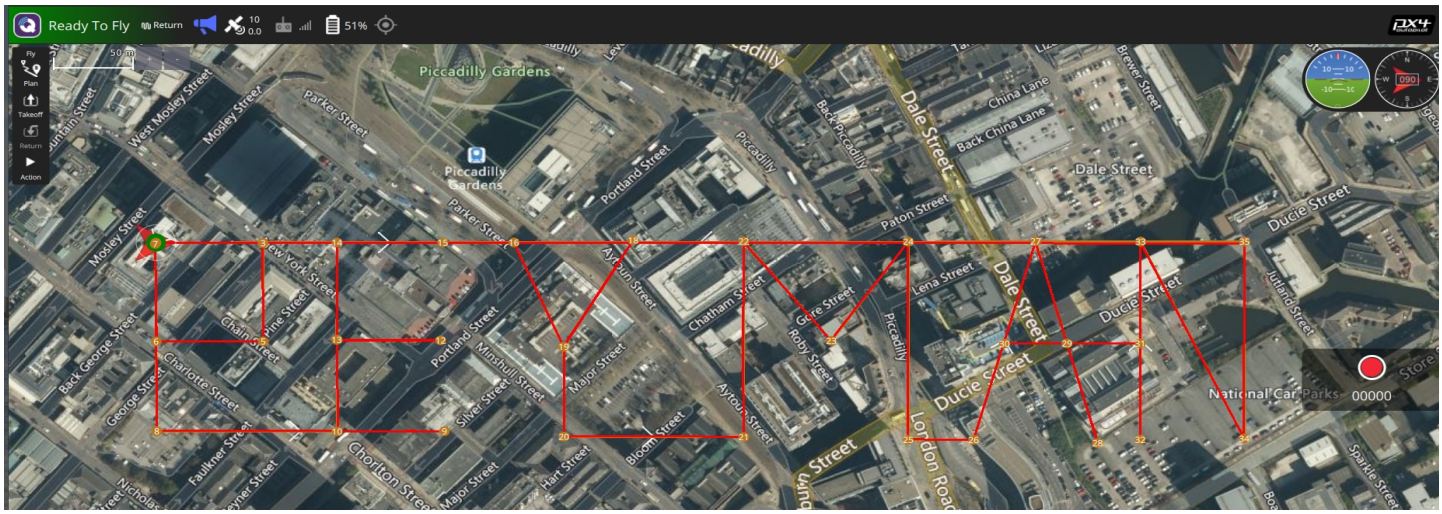

Figure 8: Traveled path of the quadrotor in Mission_1

6 Installing MAVSDK-Python and running it with Gazebo and PX4 Firmware

6.1 Installing or upgrading Pip for Python 3 (prerequisite)

Pip is a command-line program; when installed, it adds the pip command line to the system. You can use it to install and manage Python software packages.

1) Write the following code in Terminal and press enter.

```
sudo apt update
```

Then write the following code in Terminal and press enter.

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

2) Write the this code in the Terminal and press Enter to make sure that your pip version is the latest one.

```
sudo pip install --upgrade pip
```

Then you can check the version of your pip through typing this code in Terminal:

```
pip --version
```
(section 6.1 is inferred from https://phoenixnap.com/kb/how-to-install-pip-on-ubuntu)

6.2 Installing MAVSDK-Python

1) Write the following code in Terminal and press Enter.

```
pip3 install mavsdk
```

Then write the code below in Terminal and press Enter:

```
pip3 install aioconsole
```

(section 6.2 is inferred from https://mavsdk.mavlink.io/main/en/python/quickstart.html)

6.3 Running MAVSDK-Python with Gazebo, PX4 Firmware and QGroundControl

1) Open QGoruondControl by double clicking on file QgroundControl.AppImage in Downloads.

2) Write the following codes in Terminal to make the quadrotor at its home location and open Gazebo with PX4. (The following location is Manchester city)

```
cd Firmware
export PX4_HOME_LAT=53.48
export PX4_HOME_LON=-2.24
export PX4_HOME_ALT=0
make px4_sitl gazebo
```

3) Write `apython` in another Terminal and press Enter. Then copy all of the following commands and paste them in Terminal and press Enter.

```
from mavsdk import System
drone = System()
await drone.connect()
await drone.action.arm()
await drone.action.takeoff()
```

Now, quadrotor takes off and you can see and track flight of the quadrotor in Gazebo and QGroundControl.

6.4 Installing PyCharm

We will use PyCharm as a Python IDE to edit and debug out Python codes. Follow the instructions of this clip step-by-step to install PyCharm:

https://www.youtube.com/watch?v=15daSz2QExo

6.5 Running Python flight codes based on waypoints with Gazebo and PX4 Firmware

1) Go to the following website which is official MAVSDK GitHub account and click on Code icon then on Download ZIP icon.

https://github.com/mavlink/MAVSDK-Python

2) Open the downloaded file and extract file “examples” in a file. It contains some Python codes. We extracted it in Mission_2 folder. So the path of this folder on our computer is Dektop/Mission_2/examples.

3) Write `cd Firmware` in Terminal. Write the following code in Terminal to make the quadrotor in Gazebo with PX4 Firmware.

```
make px4_sitl gazebo
```

4) Open QGoruondControl by double clicking on file QgroundControl.AppImage in Downloads.

5) Open another Terminal and write the following code in it to open file takeoff_and_land.py through Python3 in example folder and press enter.

```
cd Desktop/Mission_2/examples
python3  takeoff_and_land.py
```

Then the quadrotor takes off and lands.

6) After the quadrotor lands, we can try a more advanced mission named mission.py. Write the following code in the previous Terminal to run it.

```
cd Desktop/Mission_2/examples
python3  mission.py
```

Note: In the following website, you can check the description of each function used in mission.py: http://mavsdk-python-docs.s3-website.eu-central-1.amazonaws.com/plugins/mission.html

6.6 Writing and running our Python flight code based on waypoints with Gazebo and PX4 Firmware

If you open mission.py through PyCharm, you can see that three waypoints are defined between "`mission_items = []`" and "`mission_plan = MissionPlan(mission_items)`" in order in it.

So you can define your waypoints between "mission_items = []" and "mission_plan = MissionPlan(mission_items)" in order according to the instruction bellow:

```
mission_items.append(MissionItem(lattitude of the waypoint,
                                 longitude of the waypoint,
                                 altitude of the waypoint ,
                                 speed of the quadrotor to the waypoint,
                                 True,
                                 float('nan'),
                                 float('nan'),
                                 MissionItem.CameraAction.NONE,
                                 float('nan'),
                                 float('nan'),
                                 float('nan'),
                                 float('nan')))
```

Also, we know that the quadrotor returns to the launch point because of "`await drone.mission.set_return_to_launch_after_mission(True)`" defined after "`mission_plan = MissionPlan(mission_items)`".

Note: We designed a plan named Mission_2_Peyman_Amiri.py in Mission_2 folder and you can use it. You can see the traveled path of Mission_2 containing 32 waypoints in Figure 9 (the quadrotor returned home through the upper line):

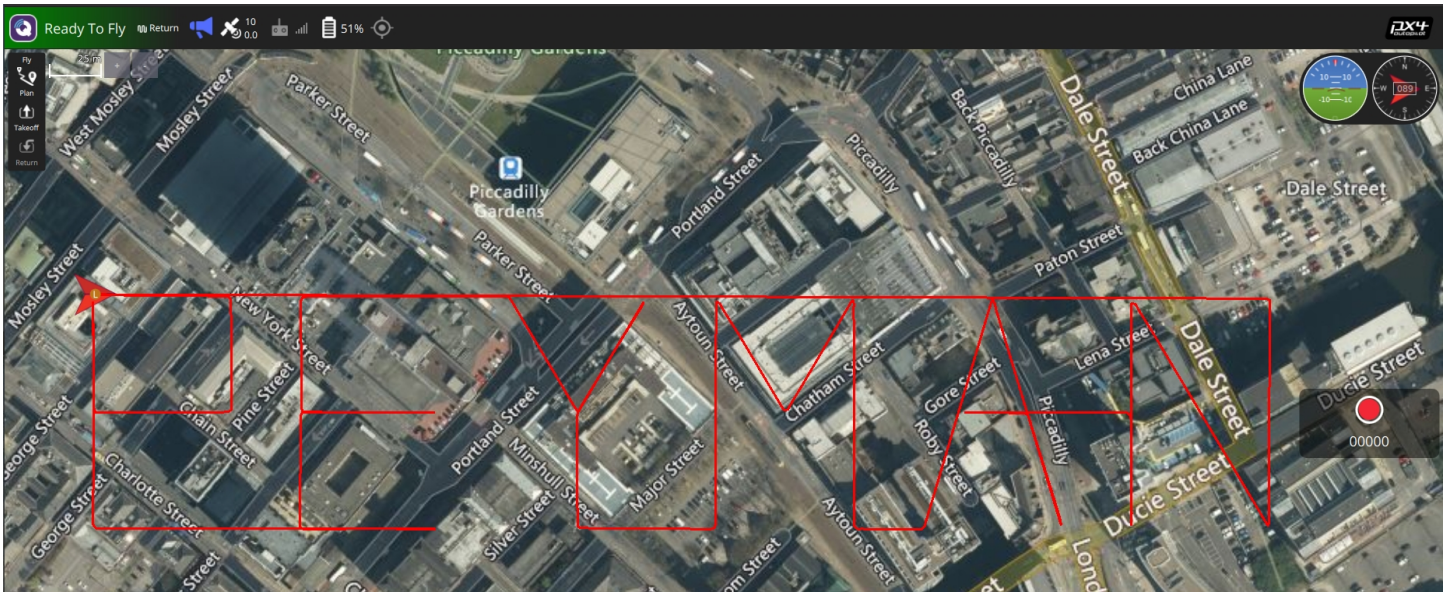

Figure 9: Traveled path of the quadrotor in Mission_2

To run this simulation and code follow the instructions bellow:

1) Write the following codes in terminal to make the quadrotor at its home location and open Gazebo with PX4. (The following location is Manchester city)

```
cd Firmware
export PX4_HOME_LAT=53.48
export PX4_HOME_LON=-2.24
export PX4_HOME_ALT=0
make px4_sitl gazebo
```

2) Open QGoruondControl by double clicking on file QgroundControl.AppImage in Downloads.

3) Open another Terminal and write the following code in it to open file takeoff_and_land.py through Python3 in example folder and press enter.

```
cd Desktop/Mission_2/examples
python3 Mission_2_Peyman_Amiri.py
```

Note: You can see the longitude and latitude of each waypoints shown in Figure 10 in file Mission_2_Peyman_Amiri.py.

Note: We assumed 0.001 difference in longitude for the length of each word and 0.0005 difference in longitude for the length of the space between them. Also, we assumes 0.001 difference in latitude for the width of the words which is twice as long as 0.001 difference in longitude (we obtained these values by trial and error). The latitude and longitude and altitude of waypoint 1 are 53.48 and -2.24 and 50 so this waypoint is exactly 50m above the initial position of the quadrotor mentioned in part 1 of section 6.6.

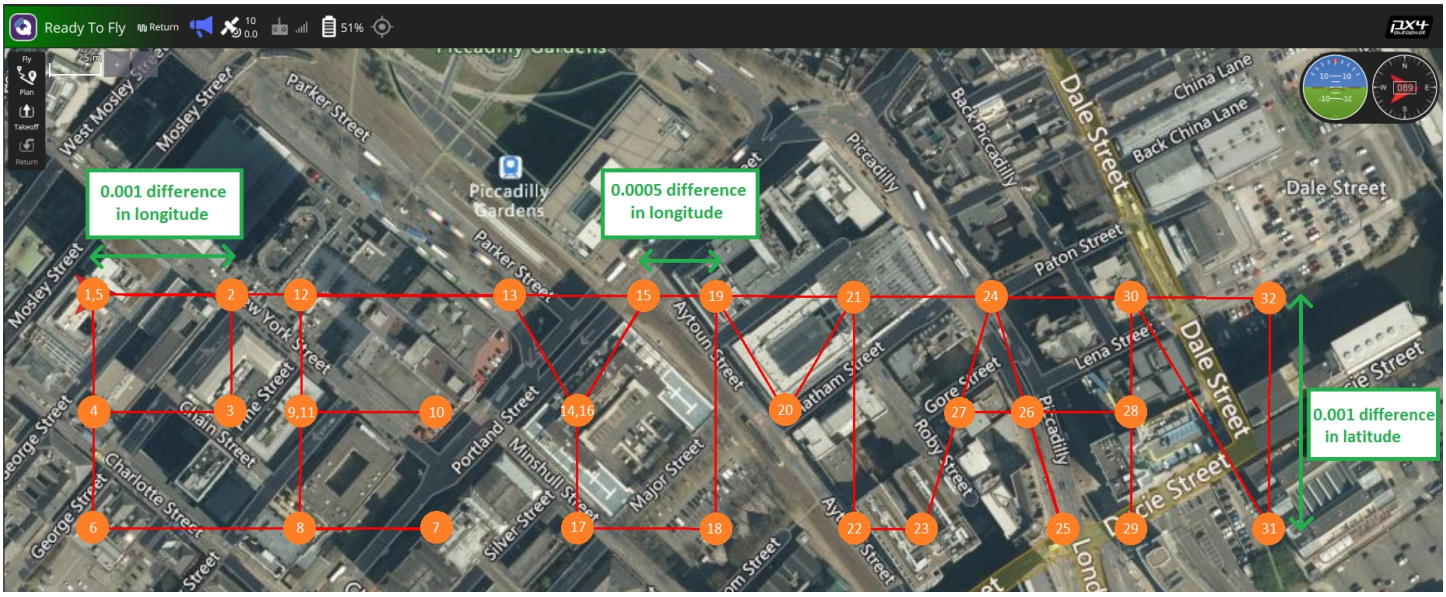

Figure 10: Edited photo of traveled path of the quadrotor in Mission_2 (waypoints and distances are added)

Note: In file python3 Mission_2_Peyman_Amiri_Modified.py we personalized and made code Mission_2_Peyman_Amiri.py easier to read but with the same performance. For example, we put all the waypoints directly in array "`mission_items`" without using "`mission_items.append`.".

6.7 Obtaining distance vector between two points using longitude and latitude (GPS outputs)

Sometimes we should obtain direct distance and vector distance between two points on the Earth using their latitude and longitude. Latitude and longitude are the outputs of GPS modules. Latitude is a decimal number between -90 and 90 and longitude is a decimal number between -180 and 180. Latitude and longitude are described in Figure 11.

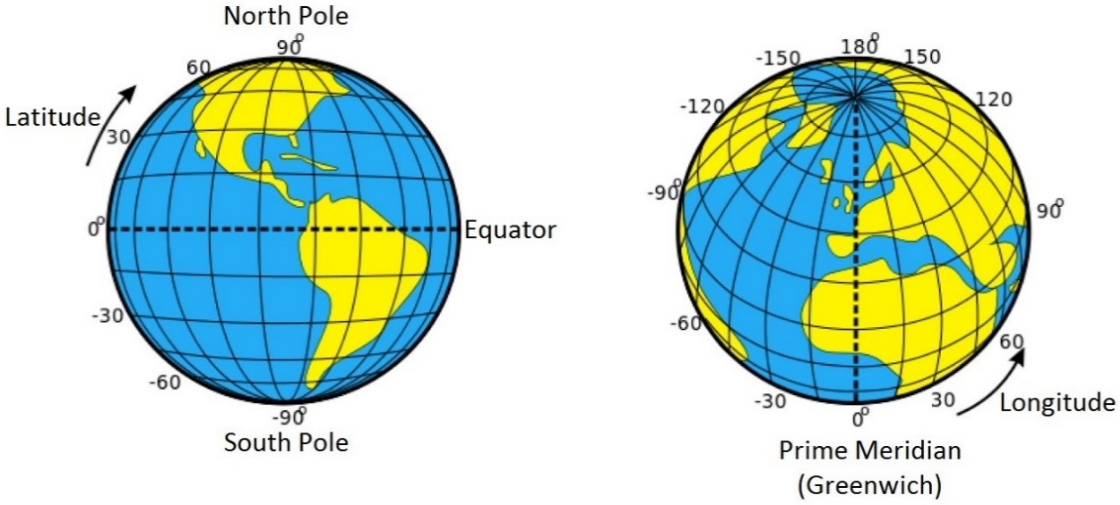

Figure 11: Position of Prime Meridian and the equator and obtaining latitude and longitude with respect to them

Now we want to find the distance **vector** between two known points with known longitudes and latitudes

along the north and the east. Versine and Haversine (abbreviation of half versine) functions are important trigonometric functions in navigation. These functions are as follows:

$$versin(\Theta)=2\sin^2\left(\frac{\Theta}{2}\right) \tag{1}$$

$$hav(\Theta)=\sin^2\left(\frac{\Theta}{2}\right) \tag{2}$$

Now we consider points A and B on a sphere as Figure 12.

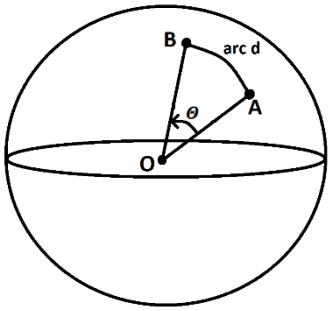

Figure 12: Two points on the surface of a sphere and central angle of Θ

So, equation 3 is valid:

$$\Theta=\frac{d}{r} \tag{3}$$

Which r is the radius of the sphere, d is the spherical distance (direct distance on the surface) between points A and B on the surface of the sphere and Θ is the central angle in front of arc d in radians.

Angle Θ of arc d for points A and B is calculated based on Haversine formula, as follows:

$$hav(\Theta)=hav\left(lat_B-lat_A\right)+\cos\left(lat_A\right)\cos\left(lat_B\right)hav\left(lon_B-lon_A\right)=h \tag{4}$$

Where lat_A and lon_A are latitude and longitude of point A, respectively, and lat_B and lon_B are latitude and longitude of point B, respectively. Therefore Θ is equal to:

$$hav(\Theta)=\sin^2\left(\frac{\Theta}{2}\right)=h\Rightarrow\Theta=2\arcsin\sqrt{h} \tag{5}$$

So, the spherical distance (the direct distance on the surface) between points A and B that is equal to the length of arc d, is as follows:

$$d=r\Theta=2r\arcsin\sqrt{h}=2r\arcsin\sqrt{hav\left(lat_B-lat_A\right)+\cos\left(lat_A\right)\cos\left(lat_B\right)hav\left(lon_B-lon_A\right)}$$

$$\Rightarrow d=2r\arcsin\sqrt{\sin^2\left(\frac{lat_B-lat_A}{2}\right)+\cos\left(lat_A\right)\cos\left(lat_B\right)\sin^2\left(\frac{lon_B-lon_A}{2}\right)} \tag{6}$$

The proof of equation 4 and more information about Haversine formula are available on:
https://en.wikipedia.org/wiki/Haversine_formula

The spherical distance (the direct distance on the surface) between point A and B was calculated as a **scalar**. But in control and navigation works, we want to know the distance vector along the north and the east. So, we should obtain the distance vector (spherical distance vector) along the *y* direction considered from the south to the north and the *x* direction considered from the west to the east. So, we devised a method to obtain the distance **vector** along the y and x directions, and we describe it in what follows:

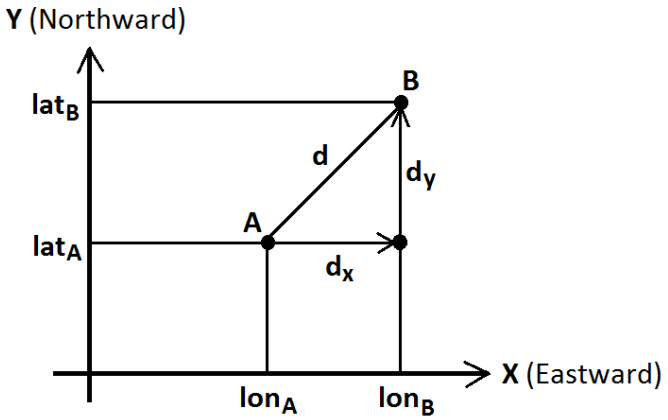

Figure 13: Distance vectors along x and y directions

According to Figure 13, to obtain the distance vector between point A and B along the *y* direction, the points' longitude must be put the same in equation 6 (we put lon_A instead of lon_B in equation 6). Also, $sign(lat_B - lat_A)$ determines the direction of this vector because if lat_B is greater than lat_A (point B is located after point A in the *y* direction), this vector is positive and if lat_A is greater than lat_B (point A is located after point B in the *y* direction), this vector is negative. Therefore, distance vector between point A and B along the *y* direction is obtained as:

$$d_y = sign(lat_B - lat_A)\, 2r \arcsin\sqrt{\sin^2\left(\frac{lat_B - lat_A}{2}\right) + \cos(lat_A)\cos(lat_B)\sin^2\left(\frac{lon_A - lon_A}{2}\right)}$$

$$\Rightarrow d_y = sign(lat_B - lat_A)\, 2r \arcsin\sqrt{\sin^2\left(\frac{lat_B - lat_A}{2}\right)}$$

$$\Rightarrow d_y = sign(lat_B - lat_A)\, 2r \arcsin\left|\sin\left(\frac{lat_B - lat_A}{2}\right)\right| \qquad (7)$$

According to Figure 13, to obtain the distance vector between point A and B along the *x* direction, the points' latitude must be put the same in equation 6 (we put lat_B instead of lat_A in equation 6). Also, $sign(lon_B - lon_A)$ determines the direction of this vector because if lon_B is greater than lon_A (point B is located after point A in the *x* direction), this vector is positive and if lon_A is greater than lon_B (point A is located after point B in the *x* direction), this vector is negative. Therefore, distance vector between point A and B along the *x* direction is obtained as:

$$d_x = sign(lon_B - lon_A)\, 2r \arcsin\sqrt{\sin^2\left(\frac{lat_B - lat_B}{2}\right) + \cos(lat_B)\cos(lat_B)\sin^2\left(\frac{lon_B - lon_A}{2}\right)}$$

$$\Rightarrow d_x = sign(lon_B - lon_A)\, 2r \arcsin\sqrt{\cos^2(lat_B)\sin^2\left(\frac{lon_B - lon_A}{2}\right)}$$
$$\Rightarrow d_x = sign(lon_B - lon_A)\, 2r \arcsin\left|\cos(lat_B)\sin\left(\frac{lon_B - lon_A}{2}\right)\right| \quad (8)$$

Some GPS devices, use the letter W (west) instead of negative sign for negative longitudes and use letter S (south) instead of negative sign for negative latitudes. In these cases, if a point is in the southern hemisphere put a negative sign before *lat* and if a point is in the western hemisphere put a negative sign before *lon* in all the equations.

6.8 Running Python flight codes based on velocity with Gazebo and PX4 Firmware

1) To run the offboard codes we should deactivate failsafe. We have two methods to do so:

A. Temporary: Writing `export COM_RCL_EXCEPT=4` in Terminal to set parameter `COM_RCL_EXCEPT` as 4.

B. Permanent: Open file px4_parameters.hpp manually using this path:

Firmware/build/px4_sitl_default/src/lib/parameters/

Then search COM_RCL_EXCEPT in the code and change its value from 0 to 4 as bellow:

"COM_RCL_EXCEPT",

.val = { .i = 4},

2) Write `cd Firmware` in Terminal. Write the following code in Terminal to make the quadrotor in Gazebo with PX4 Firmware.

```
make px4_sitl gazebo
```

3) Open QGoruondControl by double clicking on file QgroundControl.AppImage in Downloads.

4) Open another Terminal and write the following code in it to open file takeoff_and_land.py through Python3 in example folder and press enter.

```
cd Desktop/Mission_3/examples
python3  offboard_velocity_ned.py
```

Then the quadrotor will have the desired speed in each direction.

Note: In the following MAVSDK-Python website, you can check the description of each function used in offboard_velocity_ned.py:

http://mavsdk-python-docs.s3-website.eu-central-1.amazonaws.com/plugins/offboard.html

Note: The module is called offboard because the commands can be sent from external sources as opposed to onboard control right inside the autopilot “board”. Also, NED stands for North, East and Down coordinate (not body coordinate).

6.9 Writing and running our Python code based on velocity with Gazebo and PX4 Firmware

Now, we describe our flight code for the drone to follow a desired path through sending target velocities to the drone. The code named Mission_3_PID_NED_Peyman_Verdon_Sam.py and is located in folder Desktop/Mission_3/examples.

```
import time
from math import (cos,sin,sqrt)
import asyncio
from mavsdk import System
from mavsdk.offboard import (OffboardError, VelocityNedYaw)
import pymap3d as pm

# defining calss of drone telemetry-----------------------------------------------------------------------------------
class drone_telemetry:
    qx = 0          # position of the quadrotor in x direction with respect to initial point
    qy = 0          # position of the quadrotor in y direction with respect to initial point
    qz = 0          # position of the quadrotor in z direction with respect to initial point

# run function (main code) --------------------------------------------------------------------------------------------
async def run():
    # Init the drone
    drone = System()
    await drone.connect(system_address="udp://:14540")
    print("Waiting for drone to connect...")
    async for state in drone.core.connection_state():
        if state.is_connected:
            print(f"Drone discovered!")
            break

    print("-- Arming")
    await drone.action.arm()

    print("-- Setting initial setpoint")
    await drone.offboard.set_velocity_ned(VelocityNedYaw(0.0, 0.0, 0.0, 0.0))

    print("-- Starting offboard")
    try:
        await drone.offboard.start()
    except OffboardError as error:
        print(f"Starting offboard mode failed with error code: \
              {error._result.result}")
        print("-- Disarming")
        await drone.action.disarm()
        return
    #Start background task
    #below are reference GPS coordinates used as the origin of the NED coordinate system
    ref_lat = 53.48
    ref_long = -2.24
    ref_alt = 0
    asyncio.ensure_future(get_position(drone, ref_lat, ref_long, ref_alt))
    # End of Init the drone  ---------------------------------------------------------------------------------------------
    kp=2
    ki=1
    dt=0.1                          # duration of each loop
    ex=0                            # error in x direction
    Integ_x=0                       # Integral of error in x direction
    ey=0                            # error in y direction
    Integ_y=0                       # Integral of error in y direction
    ez=0                            # error in z direction
    Integ_z=0                       # Integral of error in z direction
    maxSpeed = 5                    # maximum magnitude of the speed vector
    # take off the quadrotor ---------------------------------------------------------------------------------------------
    await drone.offboard.set_velocity_ned(VelocityNedYaw(0.0, 0.0, -2.0, 0.0))
    await asyncio.sleep(5)

    Mission_start_time=time.time()  # start time of the mission
    # Endless loop (Mission)----------------------------------------------------------------------------------------------
    while 1:
        loop_start_time=time.time()

        #Printing the poition of the quadrotor in NED coordinates ------------------------------------------------------
        print("--qx=",drone_telemetry.qx) # drone_telemetry.qx is the most updated position of the quadrotor in x direction
        print("--qy=",drone_telemetry.qy) # drone_telemetry.qy is the most updated position of the quadrotor in y direction
        print("--qz=",drone_telemetry.qz) # drone_telemetry.qz is the most updated position of the quadrotor in z direction

```

```
        #Creating desired path ---------------------------------------------------------------------------------------
        t=time.time()-Mission_start_time   # current time of the mission
        x_desired=5*sin(0.5*t)             # position of the desired position in x direction
        Vx_desired=0.5*5*cos(0.5*t)        # velocity of the desired position in x direction
        y_desired=5*cos(0.5*t)             # position of the desired position in y direction
        Vy_desired=0.5*-5*sin(0.5*t)       # velocity of the desired position in y direction
        z_desired=-20                      # position of the desired position in z direction
        Vz_desired=0                       # velocity of the desired position in z direction

        #PID controller ----------------------------------------------------------------------------------------------

        # x direction -------------------------------------------------------------------------------------------------
        previous_ex=ex
        ex=x_desired-drone_telemetry.qx
        Integ_x=((ex+previous_ex)/2)*dt +Integ_x # Trapezoidal Integration
        if Integ_x >= 1:  # limiting the accumulator of integral term
            Integ_x=1
        if Integ_x <= -1:
            Integ_x=-1

        target_vx=kp*(ex)+ki*Integ_x+Vx_desired

        print("--ex=",ex)
        print("--Integ_x=",Integ_x)

        # y direction -------------------------------------------------------------------------------------------------
        previous_ey=ey
        ey=y_desired-drone_telemetry.qy
        Integ_y=((ey+previous_ey)/2)*dt +Integ_y # Trapezoidal Integration

        if Integ_y >= 1:  # limiting the accumulator of integral term
            Integ_y=1
        if Integ_y <= -1:
            Integ_y=-1

        target_vy=kp*(ey)+ki*Integ_y+Vy_desired

        print("--ey=",ey)
        print("--Integ_y=",Integ_y)

        #z direction -------------------------------------------------------------------------------------------------
        previous_ez=ez
        ez=z_desired-drone_telemetry.qz
        Integ_z=((ez+previous_ez)/2)*dt +Integ_z # Trapezoidal Integration

        if Integ_z >= 1:  # limiting the accumulator of integral term
            Integ_z=1
        if Integ_z <= -1:
            Integ_z=-1

        target_vz=kp*(ez)+ki*Integ_z+Vz_desired

        print("--ez=",ez)
        print("--Integ_z=",Integ_z)

        #limiting and normalizing the speed of the drone -------------------------------------------------------------
        v = sqrt(target_vx**2 + target_vy**2 + target_vz**2)
        if v > maxSpeed:
            normFactor = maxSpeed / v
            target_vx = target_vx * normFactor
            target_vy = target_vy * normFactor
            target_vz = target_vz * normFactor

        print("--target_vx=",target_vx)
        print("--target_vy=",target_vy)
        print("--target_vz=",target_vz)

        #Sending the target velocities to the quadrotor --------------------------------------------------------------
        await drone.offboard.set_velocity_ned(VelocityNedYaw(target_vx, target_vy, target_vz, 0.0))

        #Checking frequency of the loop
        await asyncio.sleep(0.1-(time.time()-loop_start_time)) # to make while 1 work at 10 Hz
        print("loop duration=",(time.time()-loop_start_time))
    # End of Endless loop ---------------------------------------------------------------------------------------------
# End of run function (main code) ---------------------------------------------------------------------------------

#runs in background and upates state class with latest telemetry ---------------------------------------------------
async def get_position(drone,ref_lat,ref_long,ref_alt):
    async for position in drone.telemetry.position():
            drone_telemetry.qx, drone_telemetry.qy, drone_telemetry.qz = pm.geodetic2ned\
                (position.latitude_deg, position.longitude_deg, position.absolute_altitude_m, ref_lat, ref_long, ref_alt)

if __name__ == "__main__":
    # Start the main function
    asyncio.ensure_future(run())

    # Runs the event loop until the program is canceled with e.g. CTRL-C
    asyncio.get_event_loop().run_forever()
```

At first install pymap3d package by typing the following code in Terminal and pressing Enter.

```
python3 -m pip install pymap3d
```

We can find the position of the drone in NED coordinate using latitude, longitude and altitude of the drone and the reference point (origin of NED coordinate) through this package.

In line 1 to 6, all the libraries (packages) we want are imported. In the following, they are explained more.

In line 8 to 12, class of drone_telemetry is defined to prepare parameters qx, qy and qz to get the position of the drone in NED coordinate whose origin is the reference point.

The main code should be defined in function run, so `async def run():` is written in line 15 of our code. If you open offboard_velocity_ned.py, you can see that the main flight codes are written in run() function, and the codes from line 11 to 36 are to initiate the drone. Therefore, they are copied and pasted into our code from line 17 to 39 to initiate the drone.

In line 42 to 44, the latitude, longitude and altitude of the reference point are defined. In line 45, function `get_position` is called to run in background and the reference position is sent to it. This function is defined in line 149 to 151 to calculate the position of the drone in NED coordinates whose origin is reference point. This function runs simultaneously with function run() and puts the position of the drone along z, y and z directions of NED coordinates in `drone_telemetry.qx`, `drone_telemetry.qy` and `drone_telemetry.qy`, respectively.

In line 47 to 55, value of parameters of the PID controller are defined. In line 56, the maximum allowed magnitude of the drone speed is defined in m/s unit.

In line 39 to 40 of offboard_velocity_ned.py code, you can see the following code to send the target velocities to the quadrotor:

```
await drone.offboard.set_velocity_ned(VelocityNedYaw(0.0, 0.0, -2.0, 0.0))
await asyncio.sleep(4)
```

The first argument of function `VelocityNedYaw` is target velocity in North direction, the second argument is target velocity in East direction and the third one is target velocity in Down direction. You can see more information about this function in offboard plugin on MAVSDK-Python website. Afterward, `await asyncio.sleep(4)`adds a delay of 4 seconds to the code.

Therefore, in line 58 and 59, target speed of -2.0 m/s in Down direction is sent to the drone and a delay of 5 seconds is added in order to take the drone off directly to prevent any crash with ground.

Note: `await asyncio.sleep(x)`: waits for x seconds (like delay function in Arduino). You can use decimal numbers with it, as well. To use it, we should import `asyncio` at the beginning of the code before definition of all the functions (like what was done in our code).

In line 61, the time in which the mission starts is put in `Mission_start_time`. Function time.time() shows the current time in seconds. To use it, we should import `time` at the beginning of the code before definition of all the functions (like what was done in our code).

In line 63, an endless loop is defined to accomplish the mission. Then, in line 64, the start time of each loop is put in `loop_start_time`. This parameter will be used to measure the duration of each loop. In line 67 to 69, the current position of the drone along x, y and z directions are printed (they are printed in Terminal).

In line 72, parameter `t`, showing the current time of the mission, is calculated by subtracting the start time of the mission from the current time. In line 73 to 78, the desired path is created based on parameter `t` and

mathematical functions imported from `math` library. Parameters `x_desired`, `y_desired` and `z_desired` are desired position of the drone based on `t`, and parameters `Vx_desired`, `Vx_desired` and `Vx_desired` are the derivatives of the desired positions with respect to time (`t`).

Note: The desired positions can be static setpoints. If you consider the desired positions as static setpoints (not a path), set the derivatives of them as zero.

In line 80 to 124, the PID controller is defined to calculate the target velocities. In line 83, the position error of the drone along the x direction in the previous loop is put in parameter `previous_ex`. Afterward, in line 84, the position error of the drone along the x direction in the current loop, `ex`, is calculated by subtracting the current position of the drone from the desired position of the drone along the x direction. This error of the current loop, `ex`, will be `previous_ex` of the next loop.

In line 85, integral of the error is calculated based on trapezoidal numerical integration method. This method is shown and compared with rectangular method in Figure 14. Then, in line 86 to 89, the integral term is limited to range of [-1, 1].

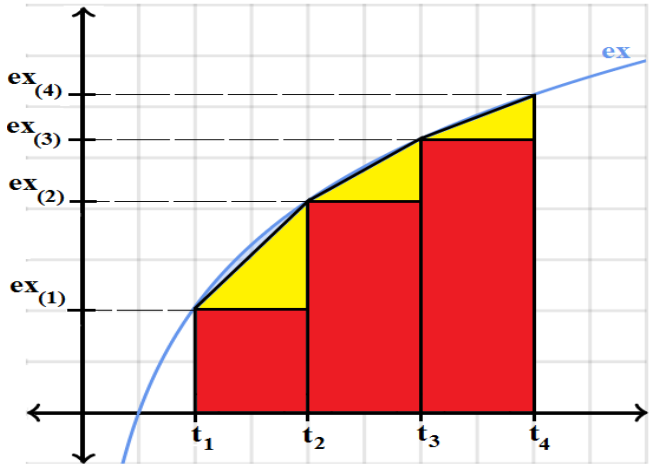

Figure 14: Trapezoidal (yellow plus red area) and rectangular (red area) numerical integration method

In line 91, the target velocity along the x direction is calculate based on PID control rule. Then, in line 93 and 94, position error and integral of error in x direction are printed to provide us a feedback of the controller.

In line 96 to 124, PID controllers of y and z directions are defined the same as x direction.

In line 127, the magnitude of the total speed of the drone is calculated as `v`. In line 128, the magnitude of this vector is checked and if it is more than the maximum allowed magnitude, all the components of the vector which are target velocities are normalized in order to decrease the magnitude to the maximum allowed one, `maxSpeed`. In line 134 to 136, all these components (target velocities) are printed.

In line 139, all of the calculated and checked target velocities are sent to the drone.

We want the loop to work at frequency of 10 Hz. So in line 142, a delay is added to the end of the loop to make sure that the loop duration is exactly 0.1 second. Therefore, the amount of the delay is: 0.1 – (the duration of the loop until now). In line 143, the duration of the whole loop is printed to provide us a feedback of it.

As it is mentioned before, function `get_position` is defined in line 149 to 151 to calculate the position of the drone in NED coordinates whose origin is reference point. This function runs simultaneously along with function `run()` and puts the position of the drone along z, y and z directions of NED coordinates in `drone_telemetry.qx`, `drone_telemetry.qy` and `drone_telemetry.qy`, respectively.

In line 153 to 158, function `run()` is made run forever.

To run flight code Mission_3_PID_NED_Peyman_Verdon_Sam.py, follow the steps below:

1) Write the following codes in terminal to make the quadrotor at its home location and open Gazebo with PX4. (The following location is Manchester city)

```
cd Firmware

export PX4_HOME_LAT=53.48

export PX4_HOME_LON=-2.24

export PX4_HOME_ALT=0

make px4_sitl gazebo
```

2) Open QGoruondControl by double clicking on file QgroundControl.AppImage in Downloads.

3) Open another Terminal and write the following code in it to open file Mission_3_PID_NED_Peyman_Verdon_Sam.py through Python3 in example folder and press enter.

```
cd Desktop/Mission_3/examples

python3  Mission_3_PID_NED_Peyman_Verdon_Sam.py
```

You can see the traveled path of Mission_3 in Figure 15.

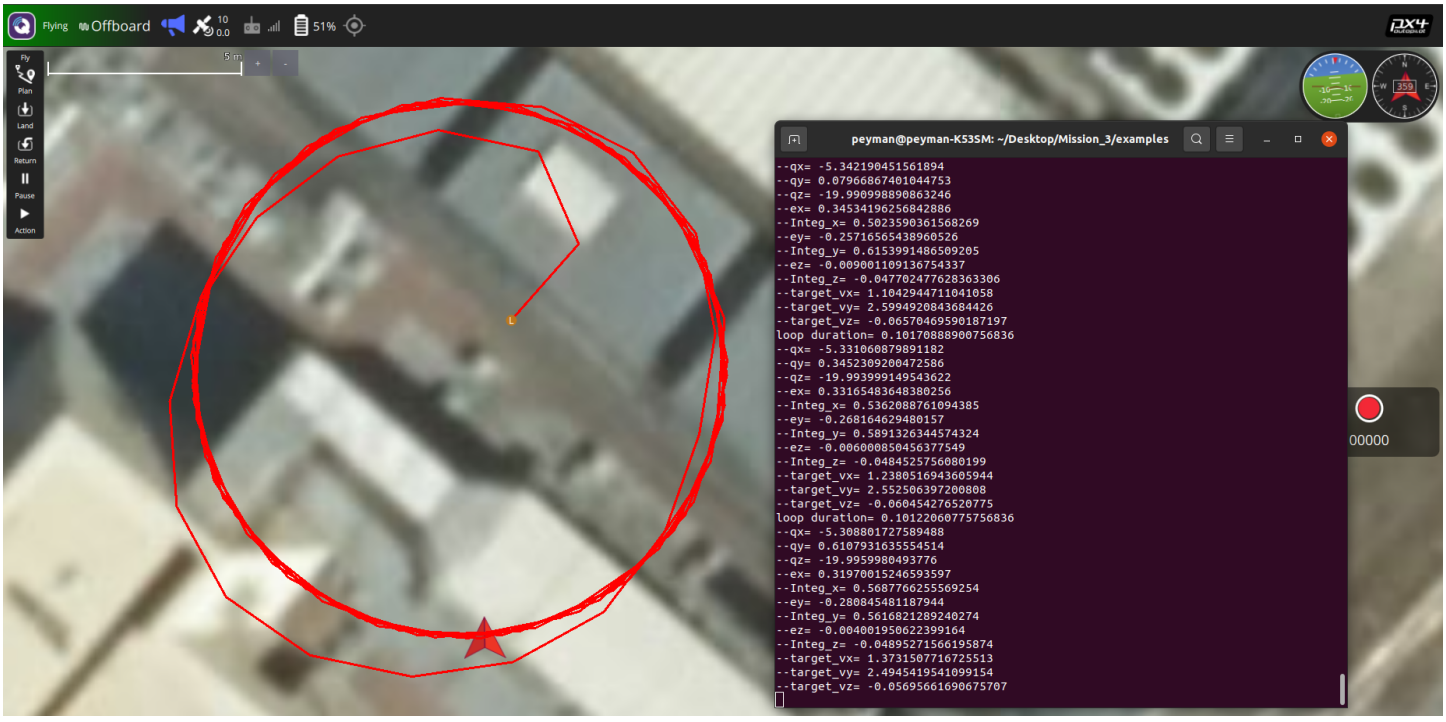

Figure 15: Traveled path of the quadrotor in Mission_3