# Helix Framework

*Release 0.1*

Verdon Crann, Sam Knox, Peyman Amiri

Aug 02, 2022

# CONTENTS

Welcome to the CASCADE UAV Swarming demonstration project wiki! You can find useful pages in the sidebar on the left detailing how to set up the hardware and software environments and solutions to various problems which have been encountered along the way.

**Note:** This project is under active development.

CHAPTER

ONE

# 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.

# CHAPTER TWO

# 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)

CHAPTER

THREE

# 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

# CHAPTER FOUR

# 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) It is assumed that Ubuntu 20.4 LTS is being used on the devlopment machine. This page outlines how to set up VS Code for development on drones on the local network through SSH.

CHAPTER

FIVE

# [CONTRIBUTORS] SET UP VS CODE

VS Code can be installed by following the instructions on the visual studio website, it should be as simple as downloading the .deb file and double-clicking to install.

Once Installed there are a few extensions that will make development easier and these should be installed from the extensions tab on the left side of the interface. They are as follows.

- Python
- Pylance
- Remote Development
- Remote - SSH

Once these extensions are installed and the drone has been set up with a static IP address navigate to the remote tab on the left of the interface and click on the + next to ‘ssh targets’. Now enter the following where user is the username chosen when Ubuntu was installed on the UP Core companion Computer and ip_address is the static IP address chosen during network configuration.

```
ssh user@ip_address
```

Here are a few ways to get started with this repository. There are components of this repository that are just for physical experiments and others that are for SITL (Software in the Loop), however, a majority of it applies to both.

- Creating a path by QGroundControl
- Setting up PX4 Vision
- Connecting Drones
- Multiple Drones

CHAPTER

SIX

# INSTALLING QGROUNDCONTROL AND RUNNING IT WITH GAZEBO & PX4 FIRMWARE AND CREATING A PLAN

## 6.1 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.

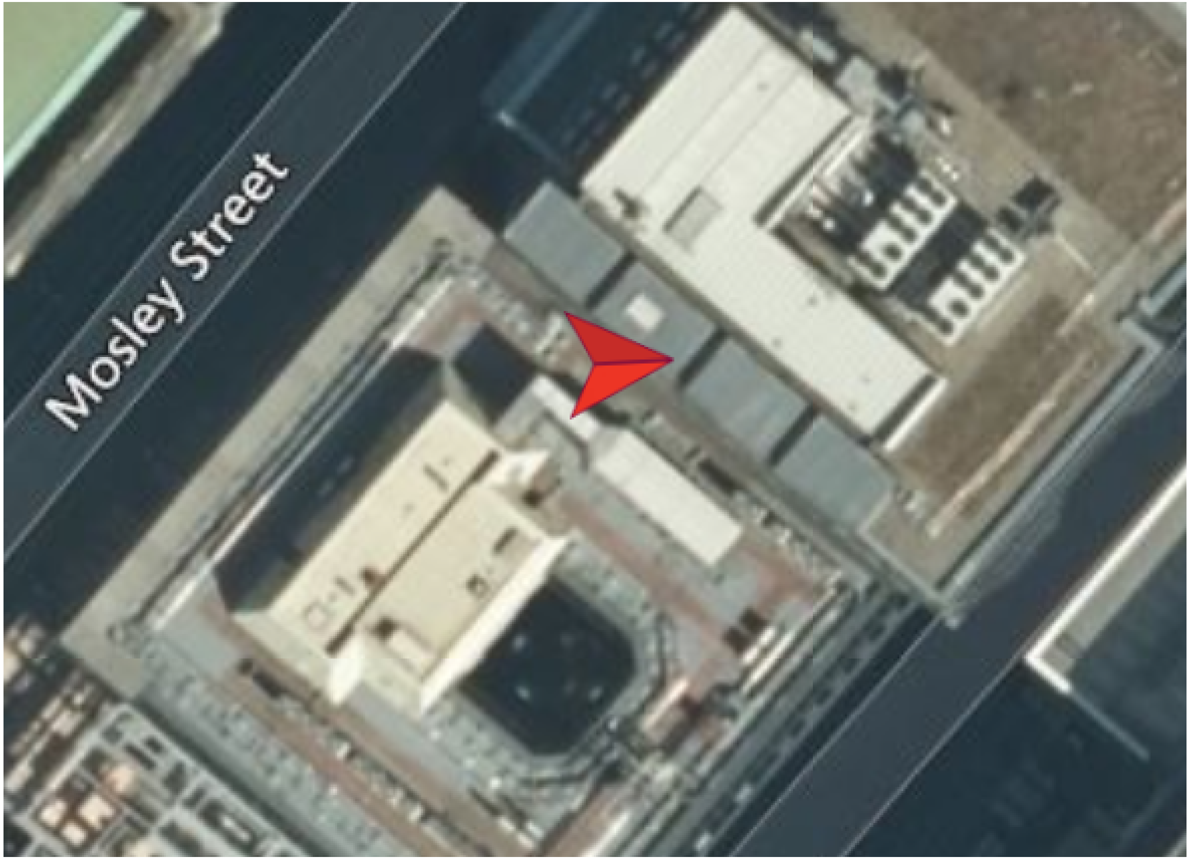
(This section is inferred from 2:40 to 4:15 of the clip on: https://www.youtube.com/watch?v=-zEddRFbMvQ)

## 6.2 Running QGroundControl with Gazebo and PX4 Firmware to create a plan based on waypoints

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 in Manchester city)

```
export PX4_HOME_LAT=53.4800000000000000
export PX4_HOME_LON=-2.2400000000000000
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.



The hardware being used for this demonstration is the PX4 Vision Autonomy Development Kit. This platform provides a ready to fly quadcopter with a pixhawk 4 flight controller and an on board UP Core companion computer. For the purposes of the demonstration some additional setup is required which is detailed in this document.

CHAPTER

SEVEN

# HARDWARE SETUP

The only hardware setup required is to install the TBS Crossfire Nano Rx.

## 7.1 Receiver Installation

1. Solder Header pins to ground, 5V and CH1 pins.
2. Connect Rx header pins to port marked DSM/SBUS RC on Pixhawk 4 with included cable.
3. Bind Rx to radio.
4. Run crossfire lua script on radio, select Nano Rx and change CH1 to sbus.

## 7.2 WiFi Telemetry Setup

By default the PX4 Vision Autonomy Devlopment Kit comes with an ESP8266 WiFi Telemetry module which is set to access point mode. This will broadcast a network which the ground station can connect to to receive telemetry. For the purposes of the demonstration the ESP8266 module needs to be in station mode instead. When in station mode the module will connect to an existing wifi network and telemetry can be passed to the ground control station through the network. This means one ground station can receive telemetry from multiple drones through the same WiFi network. Though QGroundcontrol offers a setup inteface for the module it is not functional, therefore the browser setup interface must be used.

1. Connect to the wireless access point created by the ESP8266 module, in the case of the PX4 Vision Autonomy Development Kit the SSID and password are both pixhawk4.
2. Navigate to http://192.168.4.1/ in a browser.
3. click on ‘Setup’.
4. Set the ‘WiFi Mode’ to ‘Station’, set the ‘WiFi Channel’ to 6, enter the SSID and password of the existing wifi network into ‘Station SSID’ and ‘Station Password’, all other settings can be left as default. Save these settings and reboot the drone.
5. The WiFi Module should now reboot in station mode and connect to the existing network, telemetry should now be visible in port 14550 in QGroundcontrol.
6. Assign a static IP to the WiFi Telemetry module following the instructions in the Setting up the Local Network page. The setup page can now be accessed by navigating to the new static IP in the browser.

CHAPTER

EIGHT

# SOFTWARE SETUP

Both the Pixhawk and the UP Core companion computer require some software setup for the demonstration.

## 8.1 Ubuntu Installation on UP Core

1. Follow the tutorial on the Ubuntu website to create a bootable Ubuntu 20.4 LTS USB Stick.
2. Connect a keyboard, mouse and monitor to the UP Core and insert the USB stick then connect the computer to power.
3. The computer will boot and give the option to try Ubuntu or install it, select install and follow the on screen setup process. Make sure to select ‘Minimal Installation’ and ‘Erase disk and install Ubuntu’.

## 8.2 Enabling use of the serial port in Ubuntu (IMPORTANT)

By default Ubuntu includes a modem manager which blocks the use of the serial port to communicate with the Pixhawk. This must be removed with the following terminal commands.

```
sudo usermod -a -G dialout $USER
sudo apt-get remove modemmanager -y
```

## 8.3 Enabling SSH

By default ssh is not installed on Ubuntu 20.4 LTS. In order to avoid the need for a monitor keyboard and mouse for each drone in the swarm during development it can be installed and enabled with the following terminal commands.

```
sudo apt update
sudo apt install openssh-server
sudo ufw allow ssh
```

During the demonstration the drones communicate with each other through a local wireless network. The network needs to be set up so that the DHCP server assigns static IP addresses to the drones and the ground station. This is necessary for routing the MQTT messages which are used to pass telemetry data around the network.

CHAPTER

NINE

# ROUTER CONFIGURATION

The router used in this project is manufactured by TPLink however the process should be similar for other routers.

## 9.1 Assign Static IP Addresses

1. Navigate to the router configuration page in a browser, usually found at http://192.168.0.1.
2. Login with the details found on the router.
3. Navigate to the 'Advanced' tab and then to 'DHCP Server'
4. Under 'Address Reservation' click 'Add' and then enter the MAC address of the drone or ground station and a permanent static IP address.

CHAPTER

TEN

# GROUND STATION NETWORK SETTINGS

The ground station is connected via ethernet to the router and this connection can be configured to enable the ground station to access the internet through a seperate wifi connection while still being able to communicate on the local network. It is assumed the ground station is running Ubuntu 20.4 LTS

1. Navigate to the ‘Network’ tab in the ubuntu settings and click the gear next to ‘Ethernet’.
2. Navigate to the ‘IPv4’ tab and select ‘Automatic (DHCP)’ for ‘IPv4 Method’.
3. Check ‘Use this connection only for resources on its network’ at the bottom of the page.

Flocking algorithms can be tested in SITL by simulating multiple drones which are each controlled by an instance of the same flocking script. The PX4 Firmware includes a shell script for spawning multiple drones in Gazebo, however each one requires an instance of the Mavsdk server and flocking script. This document outlines how to set up a multi drone simulation and run a basic flocking algorithm.

CHAPTER

ELEVEN

# SPAWNING MULTIPLE DRONES IN GAZEBO

1. Navigate to the directory where the PX4 Firmware is stored in terminal.
2. Navigate into the Firmware/Tools directory, inside is a script called “gazebo_sitl_multiple_run.sh”.
3. Run the script using the -n flag to specify the number of drones to spawn, for example to spawn 6 drones: .. code:: sh

   > bash gazebo_sitl_multiple_run.sh -n 6

This will spawn the speficied number of drones in the Gazebo simulation. Each will communicate on UDP port 14550 meaning that they will all be visible in QGroundcontrol, the first drone will also communicate on UDP port 14540 with each following drone communicating on the next port in sequence, 14541, 14542 etc.

CHAPTER

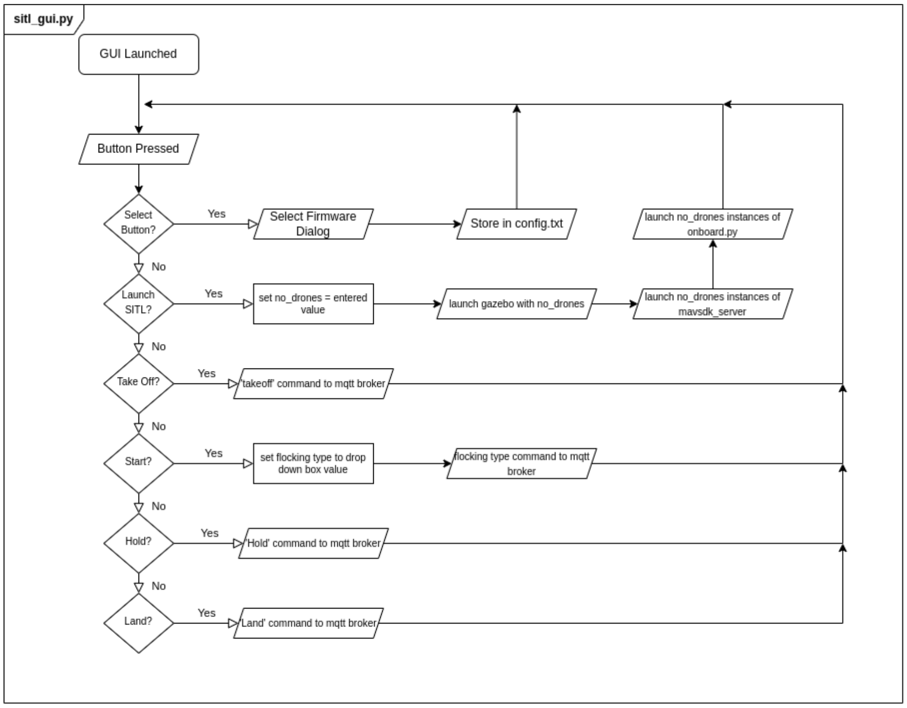
TWELVE

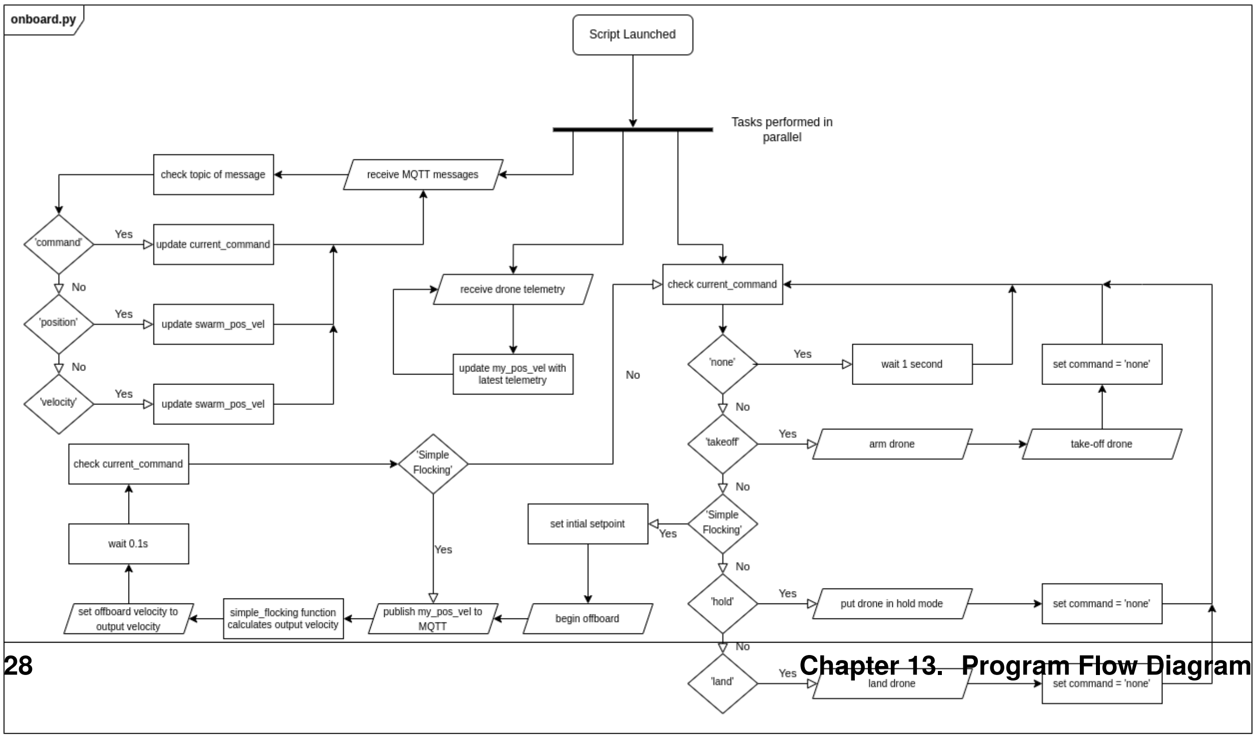
# CONNECTING TO MULTIPLE DRONES WITH MAVSDK

CHAPTER

THIRTEEN

# PROGRAM FLOW DIAGRAM


sitl_gui.py
GUI Launched
Button Pressed
Select Button?
Yes
Select Firmware Dialog
Store in config.txt
launch no_drones instances of onboard.py
No
Launch SITL?
Yes
set no_drones = entered value
launch gazebo with no_drones
launch no_drones instances of mavsdk_server
No
Take Off?
Yes
'takeoff' command to mqtt broker
No
Start?
Yes
set flocking type to drop down box value
flocking type command to mqtt broker
No
Hold?
Yes
'Hold' command to mqtt broker
No
Land?
Yes
'Land' command to mqtt broker



View the full resolution zoomable diagram on diagrams.net

CHAPTER

# FOURTEEN

# 1. CENTERED CORRIDORS

We define centered corridors for our drones through our Ground Control Station. The drones should decide how to move in these corridors without collision in a decentralized manner. These centered corridors have two specifications, their center points and directions. In what follows they are described.

## 14.1 1.1. Points of Corridors

We define a list of ordered points as center of a corridor in which each drone should fly without collision based on their decentralized swarming rules. It is noteworthy that the points are not time-based, but they are ordered which means that when a drone is flying based on its current point of the list, tries to pass the next point. When it has passed the next point, the point is considered as the drone's current one. The list containing the points of each drone's corridor is sent to each drone thorough our Ground Control Station. For instance, the equation of center points of a corridor as an ellipse with origin point of (x0, y0, z0) and width of 2a and length of 2b is: (x-x0)2/a2 +(y-y0)2/b2 = 1, z=z0. The list containing coordination of the points is created through the following piece of code in Python:

```
Points=[]
for i in range(points_number):
    Points.append(numpy.array([a*math.sin(i*6.28/points_number) + x_0, b*math.cos(i*6.28/
↪points_number)+y_0, z_0]))
```

If we consider x0 as 25, y0 as 50, z0 as 20, a (width) as 25 and b (length) as 50, we have an ellipse as shown below:

## 14.2 1.2. Directions of Corridors

A direction vector at a point of a corridor, is a vector that a drone should fly and move forward along with in the corridor. Therefore, we create a list of directions, in which nth element (nth direction) is a normalized vector from point nth to (n+1)th. The list containing vector of directions is obtained through the following piece of code in Python:

```
Directions=[]
for i in range(len(Points)):
    if i==len(Points)-1: # for the last point
        Directions.append((Points[0]-Points[i])/numpy.linalg.norm(Points[0]-Points[i]))
↪#normalized direction vector
    else:
        Directions.append((Points[i+1]-Points[i])/numpy.linalg.norm(Points[i+1]-
↪Points[i])) #normalized direction vector
```

We use these direction vectors as the direction of the migration velocity, and to obtain lane cohesion error vector described in what follows.

CHAPTER

FIFTEEN

# 2. VELOCITIES & SWARMING RULES DURING A MISSION

There are four target velocities to make drones move forward and rotate at the right distance from the direction vectors without collision in corridors. These velocities are described in what follows.

## 15.1 2.1. Lane Cohesion Velocity

We want each drone to rotate around the line from its current point to the next point at a radius called lane radius. So we should have a velocity/force to make the drone at the right distance (lane radius) from the line. Therefore, we call this velocity/force as lane cohesion velocity/force that provides centripetal force to keep the drone at the right distance from the line. To obtain the magnitude and direction of this velocity, at first, we define lane cohesion error as shown in the following figure.

As it seems from the previous figure, lane cohesion error is the shortest vector from the line to the drone. So, magnitude of lane cohesion velocity should be proportional to magnitude of lane cohesion error minus lane radius, and its direction should be along with opposite direction of the normalized lane cohesion error vector. Therefore, the code to obtain it in Python is:

```
P=Drone.position-Points[current_point]
lane_cohesion_error=P-numpy.dot(P,Directions[current_point])*Directions[current_point]
lane_cohesion_error_magnitude=np.linalg.norm(lane_cohesion_error)

if np.linalg.norm(lane_cohesion_position_error) != 0:
    v_lane_cohesion = -1*(lane_cohesion_error_magnitude -lane_radius)*lane_cohesion_
↪error/np.linalg.norm(lane_cohesion_error)
```

We will use lane cohesion error to obtain velocity of rotation in the following sections.

## 15.2 2.2. Migration Velocity

Migration velocity/force is along with the direction of the current point (a normalized vector from the previous point of current point to the current point) to make drones move forwad. Keep in mind that this velocity is perpendicular to lane cohesion error and velocity. The magnitude of this velocity is 1 that we can adjust it by gains assigned by a user.

```
v_migration = Directions[current_point]
```

## 15.3 2.3. Rotation Velocity

Each drone should rotate around its current direction vector. The direction of this rotation is perpendicular to both the direction vector and lane cohesion error vector. So the direction of this velocity should be the cross product of them. We define the magnitude of rotation velocity as the magnitude of lane cohesion error over lane radius if lane cohesion error is less than lane radius, and define it as lane radius over the magnitude of lane cohesion error if lane cohesion error is equal or greater than lane radius. Therefore, the magnitude is always less than or equal to 1, and it decreases as a drone gets farther from the lane radius around the direction vector of its current point.

```
if lane_cohesion_error_magnitude < lane_radius:
    v_rotation_magnitude = (lane_cohesion_error_magnitude /lane_radius)
else:
    v_rotation_magnitude = (lane_radius / lane_cohesion_error_magnitude)
cross_prod=numpy.cross(lane_cohesion_error, directions[current_point]
if (np.linalg.norm(cross_prod)) != 0):
    v_rotation = v_rotation_magnitude*cross_prod/numpy.linalg.norm(cross_prod))
else:
    v_rotation = numpy.array([0, 0, 0])
```

## 15.4 2.4. Separation Velocity

To prevent collision of drones in swarming, we define a velocity/force called separation velocity to separate them. Each drone has a feedback of other drones' velocities, so each drone can calculate its own separation velocity. The whole separation velocity for a drone, is the sum of separation velocities between the drone and other drones. The direction of each separation velocity is along with a vector from another drone to the drone (to push the drone away), and its magnitude is based on their distance. The figure of the magnitude of separation velocity between two drones are shown below:

And the equation of separation velocity between two drone based on the above figure is:

In the above equation, rconf, rcoll and vsep refer to radius of conflict area, radius of collision area and separation velocity, respectively. The Python code to calculate total separation velocity is:

```
v_separation = np.array([0, 0, 0])
for drone in swarm:
    if drone == Drone:
        continue
    x = Drone.position - numpy.array(drone.position)
    d = np.linalg.norm(x)
    if d <= r_conflict and d > r_collision and d != 0:
        v_separation = v_separation + ((x / d) * (r_conflict - d / r_conflict - r_
↪collision))
    if d <= r_collision and d != 0:
        v_separation = v_separation + 1 * (x / d)
```

Note that object Drone in the above code refers to the drone currently running the code, and swarm contains objects of all the drones.

## 15.5 2.5. Passing Points

We should define a mechanism to check if the drone has passed the points next to its current point or not. If we just consider closest distance to determine the drone's current point, we will face a problem as shown in the photo below:

As we can see, when the drone was in the middle of its way from point 0 to 1, it changed its current point to 1, and changed its path and direction to point 1 to 2 without completing its path from point 0 to 1. This problem is more sensible when the number of the points are not high. To solve this problem and make the movement of the drones smoother, we define a mechanism based on dot product (not closest distance) to determine and find out when a drone has passed a point next to its current point. The mechanism explains that a drone has passed point m if:

1. It has passed all points from the current point to m-1.
2. The dot product of the vector from point m to drone and direction vector from point m-1 to m is equal or greater than zero. The dot product is shown as follows:

CHAPTER

SIXTEEN

# 3. PREPARATIONS AFTER AND BEFORE A MISSION

There are two conditions triggering failsafe mode. The conditions are as follows:

1. Connection between the onboard computer and the flight controller is lost (loss of offboard connection)
2. Telemetry connection between the drone and the Ground station is lost (loss of MQTT connection)

loss of offboard connection is more dangerous than loss of MQTT connection, because in the case of loosing offboard connection we do not have a computer on the drone to run our emergency code. Therefore, the safest and only decision we can make is to land the drone. We should keep in mind that landing on slopes may be challenging for the drone losing its onboard computer.

The frequency of loosing offboard connection is so rare because it uses cable connection. But the frequency of loosing MQTT connection is often because the connection protocol is WiFi.

| Type of connection loss | Cause | Frequency | Measure | Challenge of Measure |
|---|---|---|---|---|
| loss of offboard | loss of cable connection | rarely | landing immediately | landing on slopes |
| loss of MQTT | loss of WiFi connection | often | triggering return home after a few seconds | not knowing altitude of other drones |

COMPLEX AUTONOMOUS AIRCRAFT SYSTEMS CONFIGURATION, ANALYSIS AND DESIGN EXPLORATORY
CASCADE