# **MECH 659/666**

Aerial Robotics – Crazyflie Laboratory

Figure 1. Crazyflie 2.1 quadcopter

***Background***

In a high-rise building firefighting scenario, a team of UAVs will collaborate to autonomously extinguish secondary fires in hallways and escape routes to allow people to safely escape. The UAVs must be able to navigate through narrow corridors, passageways and fly through windows in order to reach their targets and clear the way for evacuating people.

These two labs will cover the build and testing of the Crazyflie quadcopter. In the video of lab 1, you watched how to build the quadcopter, perform some basic tests on the flight capabilities. In lab 2, you will experiment with flying the course shown in Figure 2 using open and closed loop commands with use of the flow deck and a Loco Positioning System. Once your team has selected a preferred mode of flight, each team will remotely participate in a race, completing the course in the shortest time possible. The race does not affect the grade in any way, but provides some friendly competition and motivation for the labs.

***Flight Arena***

2

+

1

+

+

1.0 m

**Take-off**

1.5m

y

x

SAFE ZONE

6m

5m

Figure 2. Flight Arena 1 (not to scale)

***Course Layout***

Table 1. Course layout for the Arena.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Area** | **X / m** | **Y / m** | **Z / m** | **Side Length/ m** |
| **Safe Zone** | -2.2 to -1.2 | -1.5 to 3.5 | 0.0 | --- |
| **Corner Base 2** | 1.20 | 0 | 0.0 | 0.6 |
| **Corner Base 1** | 1.20 | 1.30 | 0.0 | 0.6 |
| **Near side bridge pillar** | 0 to -0.35 | 1.0 to 1.35 | 0.6 to 0.7 | --- |
| **Far side**  **bridge pillar** | 0 to -0.35 | 2.0 to 2.35 | 0.6 to 0.7 | --- |

***Rules***

* Each team must program their drone to travel from the Safe Zone to the two bases and return safely to the Safe Zone as quickly as possible. These bases do not need to be reached in numerical order.
* Your drone must be facing along the positive X axis at the start and finish.
* The drone must land in each base. Propellers must stop rotating for landing to be valid.
* The drone must fly over or under the Blue block (represented by wooden chairs) and land in Base 2. Flying under the Blue block will gain you more points.
* The drone must fly around the Orange block at least once during the task.

***Scoring***

* A team’s score will be their time to complete the course, from take-off to landing, including the propellers stopping. The team with the fastest time wins.
  + It should be noted that this will not count towards your grade, although what you learn may help you in your report. (The competition is for bragging rights!)
* Failure to land at a base will result in a + 90 second penalty per base.
* Failure to fly around the Orange block will result in a +30 second penalty.
* After leaving any Base, flying your drone forwards (leading with the RED lights) between the Red blocks and under the Blue block will provide a -20 second time bonus.
* If you land your drone on top of the Orange or Blue block at any point, you will gain a -30 second time bonus. This can only be earned once per attempt.

**Build and Basic Testing**

The video of the Crazyflie build covers the building of the Crazyflie and some basic manoeuvre testing such as take-off, hover and landing. You will attempt to complete the course using an open loop system and then introduce a closed loop system using a flow deck and the LPS deck as described below. Whilst programming the Crazyflie please DO NOT attach the battery until you have checked with a demonstrator.

***The Build***

Detailed build guide videos can be found on the manufacturers website, shown here: <https://www.bitcraze.io/getting-started-with-the-crazyflie-2-0/>

The Crazyflie 2.1 packet contains the following items. Make sure that you have all of them before you start assembling.

***Package Contents:***

* 1 x Assembled Crazyflie 2.1 control board with all components mounted
* 2 x CW propellers
* 2 x CCW propellers
* 1 x Flow Deck
* 1 x Lighthouse Positioning Deck
* 1 x LiPo battery (240mAh)
* 1 x Battery holder expansion board

A close up of a logo

Description automatically generated1: Attach the propellers: There are two kinds of propellers, the clock wise (CW) and counter clock wise (CCW) propellers, each kind has their own bag in the box. The image below is for attaching the CW propellers. The image below shows the position of the CW and CCW propellers.

A picture containing person, holding, hand, indoor

Description automatically generated

A hand holding a large pair of scissors

Description automatically generated2: Attach Headers: There are two types of headers in the box, long and short ones. Insert the two long ones into the expansion connector.

3: Attach battery: Place the battery between the headers inserted into the expansion connector and insert the LPS onto the headers **(Image for step 4 shows battery holder board instead). Tapered end of LPS should point away from battery connector.** Watch out for the pins that can be a bit sharp when inserting it. The friction should hold the battery in place so tighten it until it does.

Connect the battery to finish the assembly.

A picture containing indoor, table, sitting, green

Description automatically generated4: Power on: The power button is a push button, not a sliding button. During the power-on self-test all the propellers will spin in sequence. Make sure they all spin, if they don’t then check the motor connections.

START-UP SEQUENCE:

When the Crazyflie 2.1 is powered on it will automatically go through a short sequence of events to get ready for flight.

A: Run self-tests - the Crazyflie 2.0 checks that the hardware is OK

B: Calibrate sensors - the Crazyflie 2.0 reads its sensors to get base values.

It must be absolutely still to do this, so it’s best to put it on a level surface.

3: Ready to fly!

***Flying***

Figure 4. Drone axis directions.

**Pitch**

**Yaw**

**Roll**

**Thrust**

There are four main dimensions of control:

Roll – the rotation about the longitudinal axis, located through the centre of gravity and directed towards the front of the quadcopter. Combined with the vertical thrust, a roll manoeuvre moves the Crazyflie left and right.

Pitch – the rotation about the transverse axis, perpendicular to the longitudinal axis directed towards the right side of the quadcopter. With the vertical thrust, a pitch manoeuvre moves the Crazyflie forwards or backwards.

Yaw – the rotation about the vertical axis, perpendicular to both longitudinal and transverse axis directed downwards. Yaw is used to change the flying direction by pointing the front of the Crazyflie in different directions.

Thrust – along the vertical axis, directed upwards for a quadcopter, controls the lift, altitude and the velocity as pitch or roll is also applied.

***Unbalanced Crazyflies***

If your Crazyflie drifts a lot when taking off, there are a few things you should check.

* Make sure the battery is centred. If it has slipped too far to either side, the Crazyflie would produce imbalanced lifting moments from the rotors.
* Check that the propellers are spinning freely. Blow lightly on them, one by one, and verify that they turn. A common problem is that hair is jammed between the propeller and motor. If this is the problem, pull the propeller off from the motor, remove the hair and reinstall the propeller.
* Check that the propellers are balanced, see the balancing propellers guide

***Charging the Battery***

To charge the battery of the Crazyflie 2.1, just plug in a micro USB cable. Make sure the Crazyflie is powered on. While the battery is charging, the back left blue LED will blink. When the LED is constantly lit, the battery is fully charged.

***Autonomous Flight***

Now the Crazyflie is built and you have a basic understanding of how to use one, the next section of the lab will introduce an open loop method of controlling the drone. Autonomous flight is when the Crazyflie flies without a pilot, usually using a script to control it instead. This area is out of scope for this getting started guide and only the basics will be briefly covered.

There are two ways to program the Crazyflie for autonomous flight are:

1. On board navigation command: The code is uploaded to the memory of the Crazyflie to generate navigation commands inside the micro controller.
2. Commands from computer: The navigation commands are sent through the Crazyflie radio to the Crazyflie. A log data can be sent to the computer because the Crazyflie creates 2-way communication when receiving navigation commands.

We will be using the second method where commands are sent from the computer. The easiest way to get started with autonomous flight is to use a python script running on a computer. The python script runs in a computer and sends set points to the Crazyflie using the Crazyradio. A set point contains information of where the Crazyflie should go and how fast to go there. We will be using this method during the lab.

***Open Loop Flight Commands***

You will issue commands to your Crazyflie using Python Script to fly it. These commands will be issued in sequence and should be written in the order you wish them to be executed. Command functions send instructions to the Crazyflie are based on time, thrust value, and angle. These values have limits due to limitations in the hardware and cannot be exceeded.

You should download the code from the VLE and ‘right-click’ to open and edit in Notepad. Alter the example code (using information below) to design an algorithm to complete the course in the fastest possible time. Please be very careful with the text you write – in particular make sure you don’t mix up capital letters.

Please note that in this line ‘le = FlyPath("radio://0/90/2M")’ the ‘90’ should be the UAV number (as written in pen on the chassis). Also note that commands must be all lower case and tabulated in a straight line. The code will then be run on your UAV on the test course. You need to save onto a memory stick provided in the lab. You can reflect upon the performance of the UAV and make a few modifications to the numbers of your algorithm on the test course.

|  |  |  |
| --- | --- | --- |
| Variable | Minimum Value | Maximum Value |
| Thrust | 10001 | 60000 |
| Pitch (Forward) | 0 | 20 |
| Pitch (Reverse) | 0 | -20 |
| Yaw Rate (degrees/sec) | 0 | 90 |
| Time | 0.2 | Any multiple of 0.2 |

It should be noted that the minimum values will do nothing, except in the case of thrust where your quadcopter may produce a light breeze. In contrast the maximum values will cause your drone to move rapidly and controlled flight will become extremely difficult. You must find a balance between speed and stability when navigating the course.

All flight missions begin with unlocking the aircrafts thrust protection in the INITIALISATION set of functions which should not be altered. Here are some useful functions you may wish to use in your Crazyflie program. Variable names used in the functions can be altered in the code, use any names you like. You must decide the next course of action for your aircraft, choose wisely!

*Note: When writing in Python hashtags (#) are used to comment code.*

**COMMAND: Print to Terminal**

It is recommended that you print a line to terminal for each command issued so you can follow your drone’s actions. For example, before your first ascent you could use the print command:

print ('Commencing Take-Off...')

**COMMAND: Ascend**

This command will climb the aircraft for a defined time in seconds specified by a variable ‘Time’ (must be a multiple of 0.2). The vehicle will rise at a speed based on the ascending thrust: ‘Thrust\_X’. After ascension it will then switch into maintaining an altitude using ‘Thrust\_Y’.

self.\_Ascend(Time,Thrust\_X,Thrust\_Y)

**COMMAND: Hold**

This function is used to hold an altitude at a certain thrust level for a period of time using a value of thrust given by ‘Thrust\_Hold’, for a length of time defined by ‘Time’. This is useful for quick changes in altitude.

self.\_Hold(Hold\_Thrust,Hold\_Time)

**COMMAND: Land**

This function will descend the aircraft based on the descending thrust ‘Thrust\_Land’ for a defined time in seconds specified by ‘Time’, it will then switch off the motors.

self.\_Land(Time,Thrust\_Land)

**COMMAND: Pitch (this command tilts in the direction of flight... if you are moving forwards)**

This function allows the setting of a single action specified in the variable ‘P’ for Pitch. The action is set for a designated duration in seconds ‘Time’, and sets a certain pitch angle in degrees ‘Pitch\_Angle’. Motor thrust during this action is set using Thrust\_Pitch.

self.\_Move\_Dir('P',Time,Pitch\_Angle,Thrust\_Pitch)

**COMMAND: Roll (this is a sideways rotation... in a 2D course you probably won’t use this)**

This function allows the setting of a single action specified in the variable ‘R’ for Roll. The action is set for a designated duration in seconds ‘Time’, and sets a certain roll angle in degrees ‘Roll\_Angle’. Motor thrust during this action is set using Thrust\_Roll.

self.\_Move\_Dir('R',Time,Roll\_Angle,Thrust\_Roll)

**COMMAND: Yaw (spins on the spot and it is very difficult to control... we recommend you don’t use this... not for steering in any case)**

This function allows the setting of a single action specified in the variable ‘Y’ for Yaw. The action is set for a designated duration in seconds ‘Time’, and sets a certain yaw rate in degrees per second ‘Yaw\_Rate’. Motor thrust during this action is set using Thrust\_Yaw.

self.\_Move\_Dir('Y',Time,Yaw\_Rate,Thrust\_Yaw)

**EXAMPLE**

Here’s an example of what your code structure might look like, think carefully about which functions you should use to perform your desired flight path.

#INITIALISATION

while not self.\_newdata:

time.sleep(0.1)

# Unlock startup thrust protection

self.\_cf.commander.send\_setpoint(0, 0, 0, 0)

#Enter commands and define variables below.

#DEFINE VARIABLES Time = 2

Yaw\_Rate = 360

Thrust\_Yaw = 35000

#FLIGHT PLAN

print (‘Im spinning around...’)

self.\_Move\_Dir(‘Y’,Time,Yaw\_Rate,Thrust\_Yaw)

*Note: It is unlikely this drone would perform well on the course...*

***Closed Loop***

Now that you have attempted to fly autonomously without any position feedback, it is time to experiment with position controlled flight.

The Crazyflie has two means of identifying its postion:

1. A relative position using the flow deck
2. An absolute position using the Loco Positioning System

Since the Crazyflie continuously gets its current position from one of the two positioning methods, it has all the information it needs to fly itself.

***The Flow Deck***

Figure 5. Flow Deck

The Flow Deck helps the CrazyFlie 2.1 to understand the path it is moving towards through optical motion detection and will serve as a closed loop system for the rest of lab 1. It has two sensors; a laser sensor (VL53L0X) ToF (Time-of-Flight) that measures the distance to the floor and the ‘PMW3901’optical flow sensor that measures the movement in relation to the ground.

These two sensors make scripting of the fly path possible and flying the Crazyflie 2.1 autonomously or follow a pre-programmed trajectory. The flow deck is attached under the Crazyflie and automatically detected (the four propellers will spin consecutively. The sensors allow the program to know how it flies.

***Measurements***

When the VL53L0X ToF sensor measures the distance to the ground it will not use a single point. Instead the detection will use a cone (see below) where the sensor will report the closest detection. This means that the higher the Crazyflie is, the larger the area of detection. For instance, this could be an issue if the Crazyflie gets close to a wall, then it will report the distance to the wall instead of the floor.

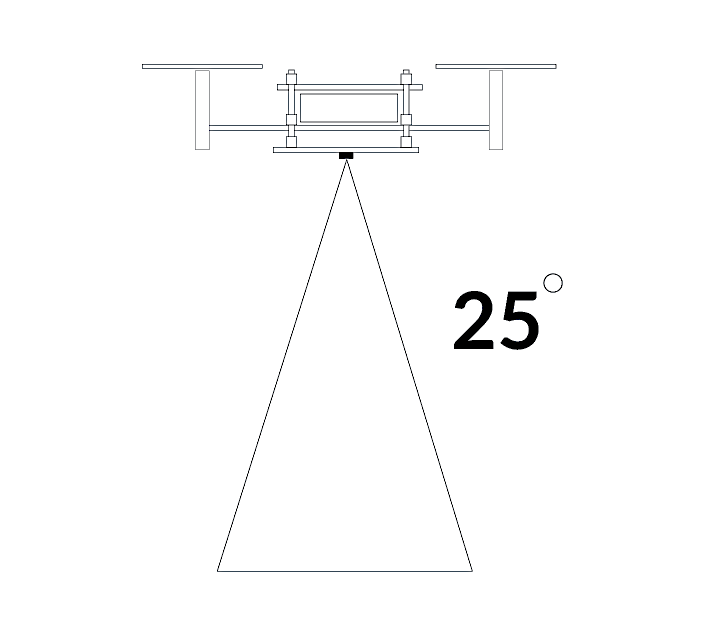


Figure 6. Sensor Measurement

Similarly, the optical flow sensor assumes it is working over flat ground. If you are flying close to wall or obstacle the tracking might be affected. The optical flow sensor is a camera and works by tracking how patterns are moving. This means that it will have a hard time operating over a blank surface.

***Flying Environment***

If the floor below the flow deck is completely black, the light will be absorbed and the measurements will be inaccurate because the visual sensors determine the height and movement of the Crazyflie 2.0. Inaccurate readings will also occur if the flow deck is flying above a plain ground. An appropriate fix for this would be to use a patterned floor.

***Relative Motion Commands***

This offers a good start for learning programming Autonomous actions. Below are some of the main commands that can be used to program an autonomous movement. With a relative positioning system, it can be expected to have significant drift after prolonged use. This can upon in this lab with a flow deck, as explained in the next section.

More commands can be found in “motion\_commander.py” example code can be found in “Relative\_Flow\_Seq.py” sample exercise on the following link <https://www.bitcraze.io/getting-started-with-stem-drone-bundle/>

1. take\_off(self, height=None, velocity=VELOCITY):

"""

Takes off, starts the motors, goes straigt up and hovers.

Do not call this function if you use the with keyword. Take off is

done automatically when the context is created.

:param height: the height (meters) to hover at. None uses the default

height set when constructed.

:param velocity: the velocity (meters/second) when taking off

:return:

"""

1. land(self, velocity=VELOCITY):

"""

Go straight down and turn off the motors.

Do not call this function if you use the with keyword. Landing is

done automatically when the context goes out of scope.

:param velocity: The velocity (meters/second) when going down

:return:

"""

1. left(self, distance\_m, velocity=VELOCITY):

"""

Go left

:param distance\_m: the distance to travel (meters)

:param velocity: the velocity of the motion (meters/second)

:return:

"""

1. right(self, distance\_m, velocity=VELOCITY):

"""

Go right

:param distance\_m: the distance to travel (meters)

:param velocity: the velocity of the motion (meters/second)

:return:

"""

1. forward(self, distance\_m, velocity=VELOCITY):

"""

Go forward

:param distance\_m: the distance to travel (meters)

:param velocity: the velocity of the motion (meters/second)

:return:

"""

1. back(self, distance\_m, velocity=VELOCITY):

"""

Go backwards

:param distance\_m: the distance to travel (meters)

:param velocity: the velocity of the motion (meters/second)

:return:

"""

1. up(self, distance\_m, velocity=VELOCITY):

"""

Go up

:param distance\_m: the distance to travel (meters)

:param velocity: the velocity of the motion (meters/second)

:return:

"""

1. down(self, distance\_m, velocity=VELOCITY):

"""

Go down

:param distance\_m: the distance to travel (meters)

:param velocity: the velocity of the motion (meters/second)

:return:

"""

1. turn\_left(self, angle\_degrees, rate=RATE):

"""

Turn to the left, staying on the spot

:param angle\_degrees: How far to turn (degrees)

:param rate: The trurning speed (degrees/second)

:return:

"""

1. turn\_right(self, angle\_degrees, rate=RATE):

"""

Turn to the right, staying on the spot

:param angle\_degrees: How far to turn (degrees)

:param rate: The trurning speed (degrees/second)

:return:

"""

1. circle\_left(self, radius\_m, velocity=VELOCITY, angle\_degrees=360.0):

"""

Go in circle, counter clock wise

:param radius\_m: The radius of the circle (meters)

:param velocity: The velocity along the circle (meters/second)

:param angle\_degrees: How far to go in the circle (degrees)

:return:

"""

1. circle\_right(self, radius\_m, velocity=VELOCITY, angle\_degrees=360.0):

"""

Go in circle, clock wise

:param radius\_m: The radius of the circle (meters)

:param velocity: The velocity along the circle (meters/second)

:param angle\_degrees: How far to go in the circle (degrees)

:return:

"""

***Example Script “Relative\_Flow\_Seq.py”***

The sample code provided (Relative\_Flow\_Seq.py) can be opened and edited in notepad++ please use spaces for indentation. The sample code has a few areas which will allow you to control the performance of the robot.

The ID of the Crazyflie marked on your drone must be inserted on line 49:

uri = 'radio://0/id/2M'

Lines 52 – 68: Relative\_sequence():

def Relative\_sequence():

with SyncCrazyflie(uri, cf=Crazyflie(rw\_cache='./cache')) as scf:

# We take off when the commander is created

with MotionCommander(scf) as mc:

time.sleep(1)

# Go to target land and come back

mc.up(0.5)

time.sleep(1)

mc.forward(2.8,velocity=1.0)

time.sleep(1)

mc.land()

mc.take\_off(1.0)

mc.turn\_left(180)

mc.forward(2.8)

time.sleep(1)

mc.land()

This allows the drone to fly in relative motion to its initial position.

Lines 71 – 75: Main Program

if \_\_name\_\_ == '\_\_main\_\_':

cflib.crtp.init\_drivers(enable\_debug\_driver=False)

Relative\_sequence()

This is where all the functions are called from.

**LPS and Absolute position navigation.**

You will now be introduced to the Lighthouse Positioning System, which enables the Crazyflie to navigate using an absolute spatial reference frame. Once you have tested this system, your team must decide which navigation mode and configuration will be used in the final competition.

***The Lighthouse Positioning System***

The Crazyflie 2.1 by default has only a limited perception of its movement, relying on onboard sensors such as the IMU and barometer. While these provide relative motion and orientation data, they are not sufficient for precise, long-term position tracking or autonomous flight. To achieve absolute position awareness indoors, an external positioning system such as the Lighthouse Positioning System (LPS) can be used. Outside GPS can be used, but indoors the options are limited, often complex, expensive or both.

The Lighthouse Positioning System is an optical tracking method that provides accurate 3D position estimates without the need for radio communication between multiple anchors. This is like a miniature GPS system. It operates using base stations, similar to those used in virtual reality systems, which emit structured infrared light patterns across the environment. Each base station projects a sweeping infrared laser signal combined with synchronization pulses that define a global timing reference.

The Crazyflie Lighthouse deck, mounted on the drone, is equipped with multiple photodiode sensors that detect these infrared sweeps. By measuring the exact time intervals between the synchronization pulse and the horizontal and vertical laser sweeps received from each base station, the Crazyflie determines the angles from which the signals arrived. With data from two or more base stations, these angular measurements are combined to triangulate the drone’s precise 3D position in real time.

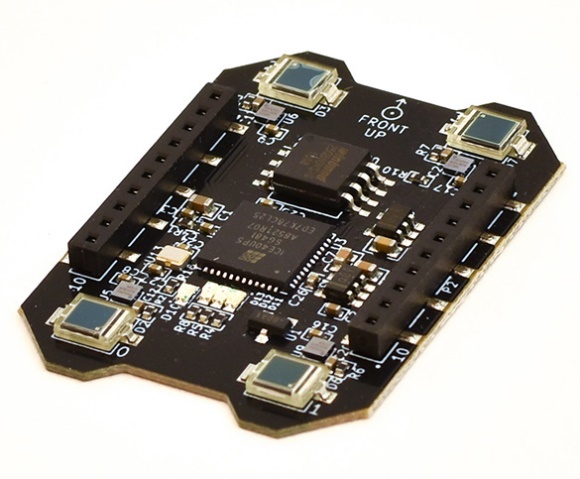


Figure 7. Lighthouse positioning deck

Unlike other positioning systems, the Lighthouse solution performs **all position calculations onboard the Crazyflie**, reducing latency and allowing for fully autonomous indoor flight without requiring an external computer. This self-contained positioning capability makes it ideal for research, education, and robotics applications that demand high accuracy and minimal infrastructure complexity.

Once set up, the system allows the Crazyflie to navigate freely within the area covered by the base stations, maintaining accurate position estimates with minimal drift. The setup is simple: the base stations are placed at known fixed locations, and the drone’s onboard firmware handles synchronization and position estimation automatically.

The Lighthouse Positioning System provides a robust, scalable, and low-maintenance method for **absolute position-based navigation** indoors, unlocking advanced flight control modes such as trajectory tracking, mapping, and cooperative multi-drone operations.

The Lighthouse Base Station is the core component of the system, responsible for projecting structured infrared light across the environment. Each base station emits timed laser sweeps in both horizontal and vertical directions, along with synchronization flashes that define a common time reference for all receivers. By detecting these sweeping light patterns, the Crazyflie’s onboard sensors can determine the angles to each base station. When signals from at least two base stations are received, the Crazyflie can accurately triangulate its absolute position in 3D space. Proper placement and orientation of the base stations are essential for achieving optimal coverage and positioning accuracy within the flight area.

A picture of the base station is shown below:

Figure 8. Lighthouse V2 base station

More information about positioning modes in Appendix A.

***Waypoint Navigation***

This method provides the Crazyflie with a set of waypoints to follow. This utilizes the LPS and is a new method of autonomous movement. Students will learn to create a path and the drone will follow it. Linear interpolation is used between waypoints so if step path is required, this has to be taken into account in the path planning phase.

Example can be found in “Absolute\_LPS\_Seq.py”

Below are the main commands needed to follow one waypoint.

cf.param.set\_value('flightmode.posSet', '1')

cf.commander. send\_position\_setpoint(X in m, Y in m , Altitude Z in m, Yaw Angle in Degrees)

***Example Script*** ***“Absolute\_LPS\_Seq.py”***

The sample code provided (Absolute\_LPS\_Seq.py) can be opened and edited in notepad++ please use spaces for indentation. The sample code has a few areas which will allow you to control the performance of the robot.

The ID of the Crazyflie marked on your drone must be inserted on line 50:

uri = 'radio://0/id/2M'

Below is the main command needed to follow one waypoint.

cf.commander.send\_position\_setpoint (X in m, Y in m , Altitude Z in m, Yaw Angle in Degrees)

0° Yaw is pointing in the X direction. (+) Yaw values will turn the drone CCW.

Lines 63 – 67

sequence0\_a1 = [

(offset\_x + 1.8, 1.0, 1.0, 0),

(offset\_x + 1.8, 2.0, 1.0, 90),

(offset\_x + 1.8, 3.0, 1.0, 90),

]

This displays a matrix of co-ordinates that allows you to determine the points of trajectory of the drone. This is in the order: [x , y, z, yaw]. You can define as many sequences as you need. offset\_x is used to allow you to fly in any side of the flying area, Arena 1 and Arena 2. When flying in Arena 1, offset\_x=0 whereas when flying in Arena 2 offset\_x=3.58.

Lines 145 – 154: run\_sequence

def run\_sequence(cf, sequence):

for position in sequence:

print('Setting position {}'.format(position))

for i in range(50):

cf.commander.send\_position\_setpoint(position[0],

position[1],

position[2],

position[3])

# Must send once every 100ms for crazyflie to think it is still connected.

time.sleep(0.1)

This is the sequence that allows the drone to fly to a set of co-ordinates. It allows for the drone to fly to the co-ordinates defined in a previous matrix sequence, and must be followed by other commands so that the drone doesn’t turn off and drop.

Line 159-175: land

def land(cf, position):

print("Landing...")

step\_height\_m = 0.05

start\_height\_m = position[2]

num\_steps = int((start\_height\_m / step\_height\_m)) + 1

new\_height\_m = start\_height\_m - step\_height\_m

# Must send once every 100ms for crazyflie to think it is still connected.

for i in range(0, num\_steps):

print("Landing height", new\_height\_m)

cf.commander.send\_position\_setpoint(position[0],

position[1],

new\_height\_m,

position[3])

new\_height\_m -= step\_height\_m

time.sleep(0.2)

print("Landed")

This function will allow the drone to reduce its height progressively until reaching a height of 0. At which point the drone would turn off its motors.

Lines 191 – 195: run\_sequence\_and\_land

def run\_sequence\_and\_land(scf, sequence):

cf = scf.cf

run\_sequence(cf, sequence)

land(cf, sequence[-1])

stop(cf)

The first part allows the drone to travel to a sequence of matrix co-ordinates. After the last co-ordinate has been reached, the drone will land in approximately 4 seconds at the last x – y co-ordinates.

Lines 198 – 214: Main Program

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

# Main Program Entry Point

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

if \_\_name\_\_ == '\_\_main\_\_':

cflib.crtp.init\_drivers(enable\_debug\_driver=False)

with SyncCrazyflie(uri, cf=Crazyflie(rw\_cache='./cache')) as scf:

reset\_estimator(scf)

start\_position\_printing(scf)

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

# Modify the code for each arena between here...

run\_sequence\_and\_land(scf, sequence0\_a1)

run\_sequence\_and\_land(scf, sequence1\_a1)

# run\_sequence\_and\_land(scf, sequence2\_a1)

# and here

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

print("Finished!")

scf.cf.close\_link()

This is where all the functions are called from. Note how the run\_sequence\_and\_land function is called. You can see the commented calls for subsequent sequences.

**Appendix A**

The **Lighthouse Positioning System** determines the Crazyflie’s absolute 3D position using **optical angle measurements** rather than radio-based ranging. It operates by tracking the angular position of infrared laser sweeps emitted by one or more **Lighthouse Base Stations**. Each base station projects structured infrared light that sweeps horizontally and vertically across the environment, accompanied by synchronization flashes that define a global time reference.

The **Lighthouse deck** mounted on the Crazyflie is equipped with an array of **photodiode sensors** that detect these sweeps. By precisely measuring the time between the synchronization pulse and each sweep detected on multiple sensors, the Crazyflie can determine the **angle of arrival** of the signal from each base station. With data from two or more base stations, the onboard firmware triangulates these angles to estimate the drone’s **absolute position and orientation in 3D space**.

**Characteristics**

**Advantages:**

* High positional accuracy (typically within a few millimetres) in small indoor environments.
* Fully self-contained; no external computer or network communication is required for position estimation.
* Low latency and robust performance once base stations are correctly calibrated.
* Scalable for multi-drone operations, as all Crazyflies can independently compute their positions using the same base stations.

**Limitations:**

* Requires direct line-of-sight to at least two base stations for accurate positioning.
* Coverage area is limited to the optical field of view of the base stations (typically a few metres).
* Base station placement and calibration are critical; reflective surfaces or obstacles can reduce accuracy.

**Performance**

The Lighthouse system provides **high-precision indoor positioning** with sub-centimetre accuracy under optimal conditions. The position is estimated onboard the Crazyflie at up to **100 Hz**, allowing smooth autonomous flight and precise trajectory tracking. Accuracy depends on base-station alignment, lighting conditions, and the number of active stations (two or more are recommended for stable 3D tracking).

For further information, search the bitcraze website: www.bitcraze.io