Crazyflie path following with Mocap system

Laboratory of Automatic Systems

Omar Moukawim and Federico Tiberti

UnivAQ

Crazyflie path following with Mocap system

by

Omar Moukawim and Federico Tiberti

Professor: Francesco Smarra
Project Duration: about 3 months (May 2022 - Sep 2022)
Faculty: Control Systems Engineering, University of L'Aquila

Contents

1

Introduction

The goal of this project is to successfully control the trajectory of a Crazyflie 2.X drone from a target PC exploiting data of the Motion Capture system (MoCap) that is streamed by another PC; we had available four infrared cameras, these are placed in the Automation Lab in Coppito 2. Autonomous flight with the Crazyflie requires an expansion deck that can get its current position or measure relative distance so that it can move to a desired position in a desired velocity. In our case a Flow deck v2 [1.1] gives the Crazyflie the ability to sense its motion in both vertical and horizontal directions. It uses a time-of-flight laser ranging sensor to measure vertical distances up to 4m with mm precision and an optical flow sensor to detect and measure the horizontal motion of surfaces, enabling the drone to travel desired distances. The deck weighs 1.6g, measures 21x28x4mm, and is designed for mounting under the Crazyflie.

Figure 1.1: Flow deck v2

The use of this combined with data collected in real-time from the MoCap system allow the drone to follow a target trajectory starting at any given point in the range of view of the Optitrack cameras. The trajectory is generated after the drone has successfully taken off, this trajectory is based on the measured position from the cameras and a set of target points that can be fixed by the user. In our case, as soon as the drone gets to the first target point it starts flying around the column of the lab following a square path .
Some demos can be found at:

- **Demo 1** - https://youtube.com/shorts/TmJs2pTiBXY?feature=share
- **Demo 2** - https://youtube.com/shorts/N8IkSC2KowI?feature=share
- **Demo 3** - https://youtube.com/shorts/kb-wMQOnVkI?feature=share

2

Mocap System

2.1. Calibration

Like in almost every measurement systems, calibration is also essential for optical motion capture systems. During camera calibration, the system computes position and orientation of each camera and amounts of distortions in captured images, and they are used constructs a 3D capture volume in Motive. This is done by observing 2D images from multiple synchronized cameras and associating the position of known calibration markers from each camera through triangulation. Calibration can be summarized in the following steps:

- Preparation and optimization the capture volume for setting up a motion capture system.
- Application of masks to ignore existing reflections in the camera view [2.1].
- Collection of calibration samples through the wanding process.
- Review of the wanding result and application of the calibration.
- Setting of the ground plane to complete the system calibration and assign the global reference system.

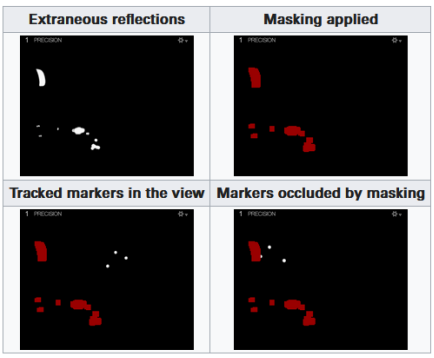

Figure 2.1: Masking process detail

The wanding process is the core pipeline that samples calibration data into Motive. A calibration wand is waved in front of the cameras repeatedly, allowing all cameras to see the markers.

Through this process, each camera captures sample frames in order to compute their respective position and orientation in the 3D space. There are a number of calibration wands suited for different capture applications. In our case we have used the CW-400 wand with configuration B (i.e. the setting for calibration of the cameras in rooms of small volumes). The Motive software allows the user to calibrate the cameras pretty easily and with different orders of precision depending on the needs of the final application. After going through the calculation, a Calibration Result Report will pop up, and detailed information regarding the calibration will be displayed. The Calibration Result is directly related to the mean error, and will update, and the calibration result tiers are: Poor, Fair, Good, Great, Excellent, and Exceptional[2.2].

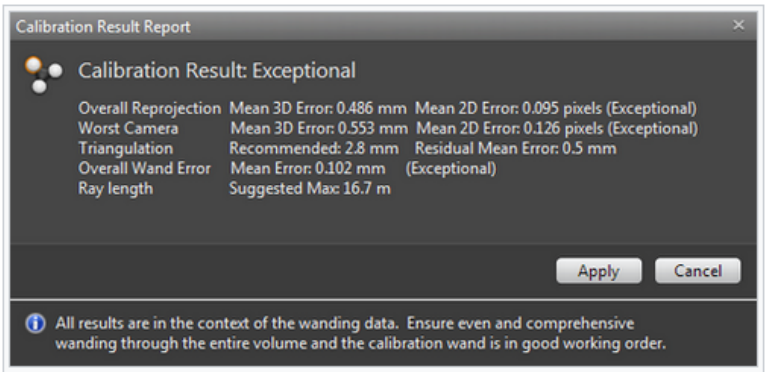

Figure 2.2: Masking process detail

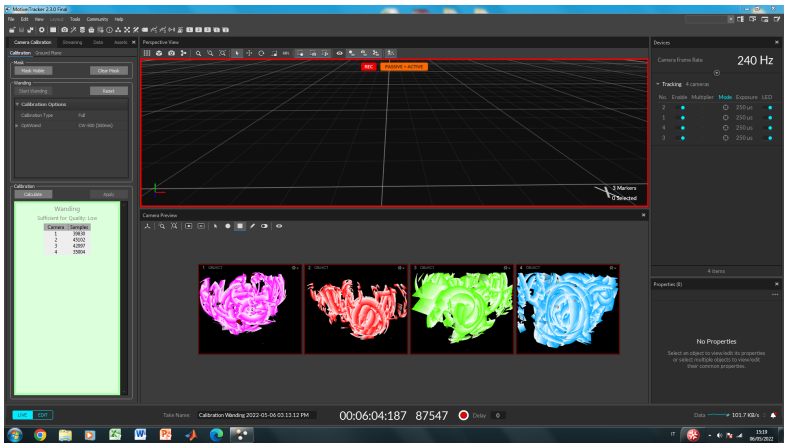

Figure 2.3: Wanding result seen on Motive

2.2. Global and Local references

After having calibrated the cameras as described in the previous section a ground plane and an origin for global references have to be set. For this a calibration square is needed and it has to be placed accordingly in the room, the longer leg on the calibration square will indicate the positive z axis, and shorter leg will indicate the direction of the positive x axis. Accordingly, the positive y axis will automatically be directed upward in a right-hand coordinate system. Another important parameter to be set in the software is the vertical offset that is used to compensate for the offset distance between the center of markers on the calibration square and the actual

ground. Defining this value takes account of the offset distance and sets the global origin slightly below the markers. Accordingly, this value should correspond to the actual distance between the center of the marker and the lowest tip at the vertex of the calibration square.

Figure 2.4: Calibration square.

2.3. Data Streaming

Common motion capture applications rely on real-time tracking, and the OptiTrack system is designed to deliver data at an extremely low latency even when streaming to third-party pipelines. Motive offers multiple options to stream tracking data onto external real-time applications, in this case we have used the standard one in the software without the addition of any other plug-in. To stream from Motive to a local interface or to another IP one can follow these steps:

- Open the Data Streaming Pane in Motive
- Select the network interface address for streaming data.
- Configure streaming settings and designate the corresponding IP address from client applications
- Stream live or playback captures

Particular attention has to be given to the selection of the network adapter for streaming data given the fact that most Motive host PC's can have multiple network adapters, for example one for the cameras and one for the local area network (LAN). In our application streaming has been done over a LAN, Motive was running on a PC and on the target PC we had a running Python code that received and processed data arriving from the host in order to correctly control the trajectory of our drone.

3

Crazyflie

3.1. Introduction

The Crazyflie family is a range of devices with similar hardware and open source firmware developed by BitCraze. They are indoor quadcopters, their dimension is small, they fit in the palm of an hand and for these reasons they are very used for reasearch purposes in several universities, one among all: MIT. The Crazyflie 2.X has been used for the project:

Figure 3.1: Crazyflie 2.X

It's not only a good flyer, the Crazyflie 2.X is also equipped with low-latency/long-range radio as well as Bluetooth LE. This gives you the option of, in combination with the Crazyradio PA, using your computer to display data and fly. The Crazyflie platform is much more than just a quadcopter. It consists of lots of different parts working together to complete the system. That means that people can experiment, learn and improve in various areas:

- **Flight Stabilization:**
  - **Sensors**: The platform consists of a number of different sensors;
  - **Sensor Fusion**: The outputs from all the sensors are fused together to create the best possible measurement of the platform orientation;
  - **Control Theory**: The platform consists of a number of different control-loops. Both for controlling several parameters, but especially for controlling the platform if you use external positioning system.

- **Communication:** The Crazyflie communicates via radio with the host. The protocol is very basic but has the potential to be expanded with more features.
- **Positioning:** There are many positioning systems such as the Loco Positioning System, the lighthouse system and support for MoCap systems. The last one has been chosen for this project.
  The main principle of using the MCS for positioning is that the cameras emit infrared light, which are reflected back by special reflective markers. This will enable the IR-cameras to detect the location of the marker, which the MPC software on an external computer will calculate the actual position from. This information can then send to the Crazyflie through the Crazyradio PA.

The system overview is shown:

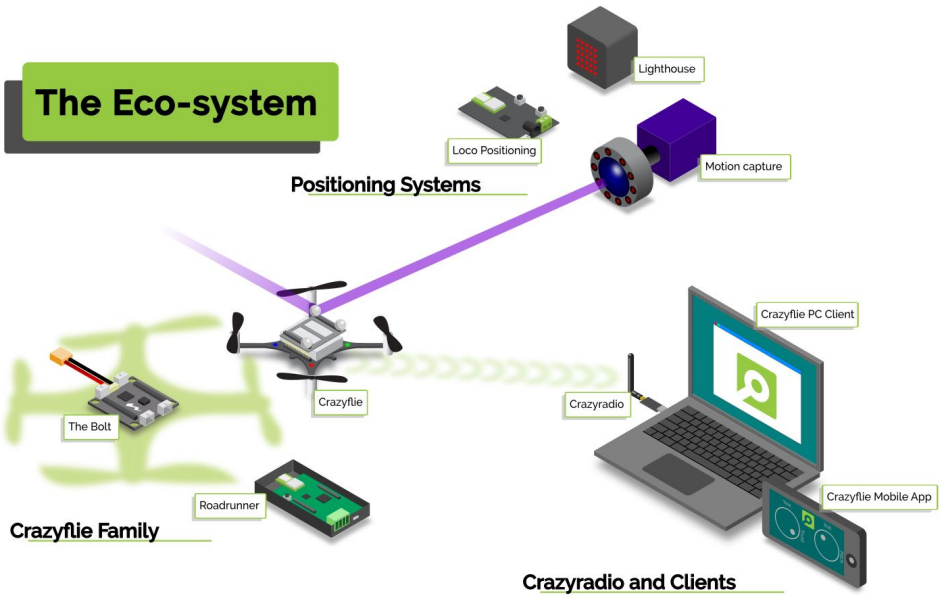

Figure 3.2: System overview

Furthermore, the Crazyflie platform has different *expansion decks* to extend the capabilities of the quadcopter. In particular, we have used an ”home-made” version of the Motion capture marker deck. It makes it easy to attach reflective markers to a Crazyflie for tracking in a motion capture system. The markers are tracked by cameras in the motion capture system and the system uses the information to calculate the current position of the drone. The position can be used for precise measurements or can be fed back to the Crazyflie for autonomous flight.

3.2. Marker support

In order to use the MoCap system, there is the necessity of equip the quadcopter with the markers. We initially attach them directly in the drone body, but this solution was not the most flexible, so we opted for the usage of a support. Since we didn’t want to spend money and lose time, instead of order it from the official website, we made it with a 3D printer. The result was this:

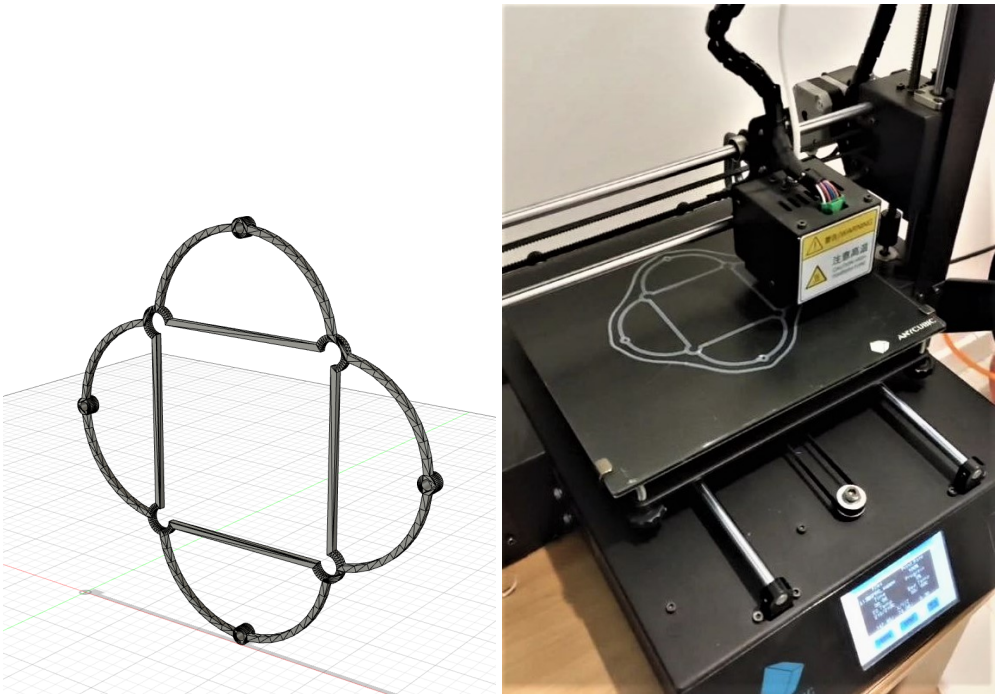

Figure 3.3: Fusion 360 project and 3d print of it.

After many flights we concluded that the above support conditioned too much the success rate of take offs and flights of the drone and so we changed our drone setup to an easier one:

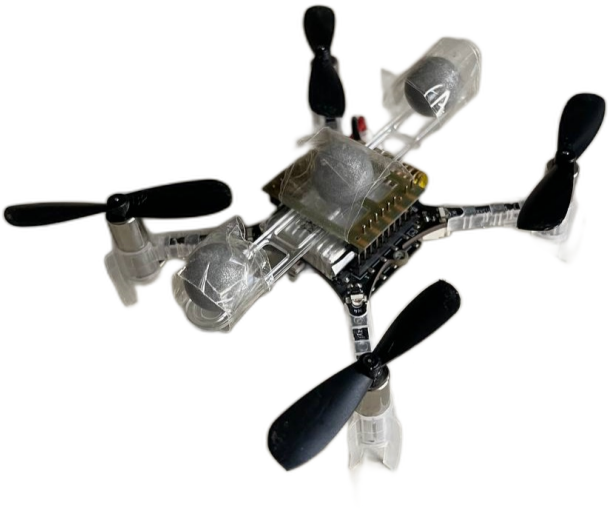

Figure 3.4

In our opinion the reason of the flight failures was due to the fact that the introduction of such 3d printed support, even thought it was only weighting 4 grams, conditioned too much the dynamics of the drone.

3.3. Software tools

Crazyflie is not only a platform, it is also a big community! Since all their projects are open source, they can easily distribute them among the users. In addition, many people have contributed to the development thanks to their projects pubblications on platforms like GitHub. There are different available libraries, one among all: **crazyflie-lib-python**. This GitHub repository contains the source code for the python library client to control the Crazyflie from the PC. The general SW architecture is this:

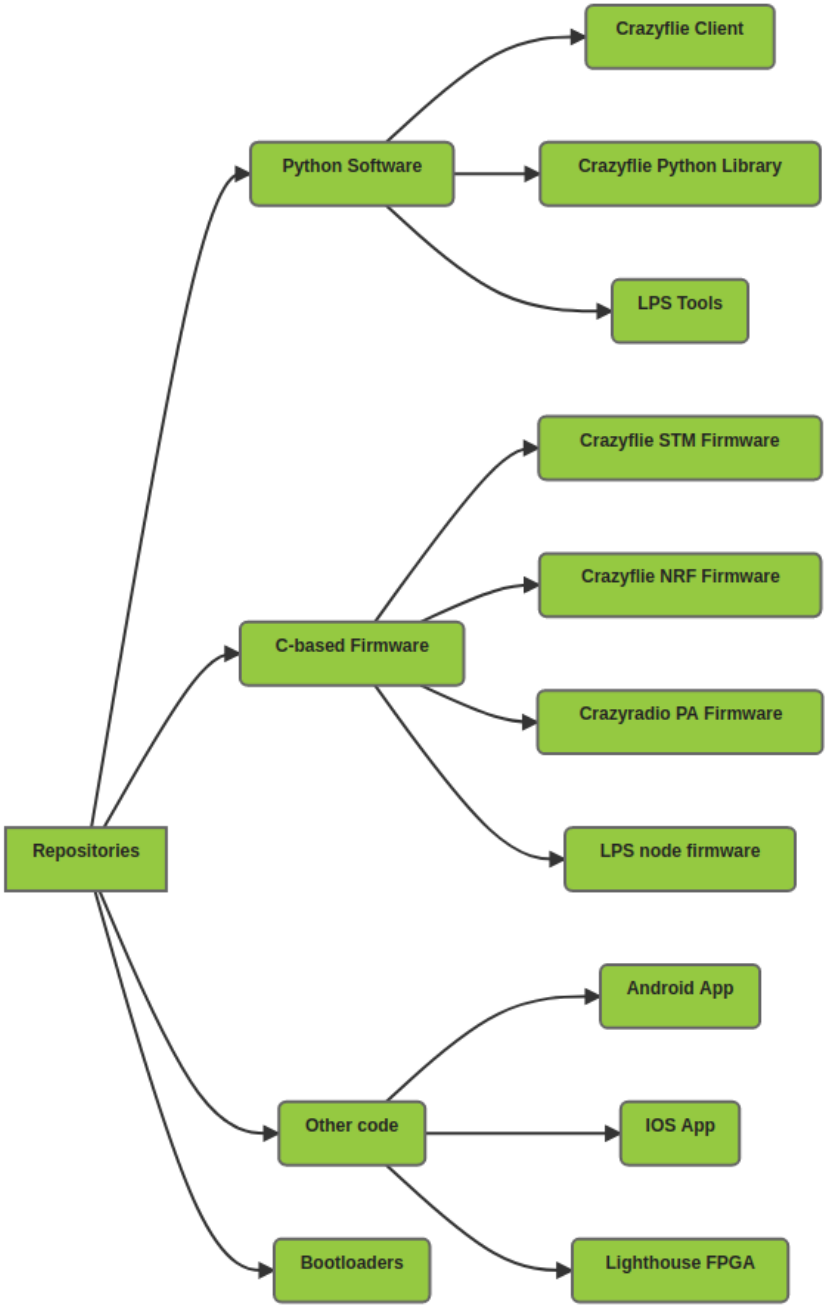

Figure 3.5: SW structure

4

Realization

4.1. Joint usage of two frameworks

For this project we have been involved in using two SW tools:

1. NatNet SDK: for cameras data management
2. Crazyflie-lib-python (GitHub repository): for controlling the quadcopter

The first one is a client/server networking SDK for streaming motion tracking data across networks. NatNet's client/server architecture allows client applications to run on the same system as the tracking software (Motive), on separate system(s), or both. In our case, we ran it on separate system (our pc). The SDK integrates seamlessly with standard APIs (C/C++/.NET), tools (Microsoft Visual Studio) and protocols (UDP/Unicast/Multicast). Using the NatNet SDK, it has been possible to quickly integrate OptiTrack motion tracking data into new and existing applications. In particular, we have used all the scripts present in this folder:
NatNet_SDK_4.0/NatNetSDK/Samples/PythonClient, that are:
PythonSample.py, MoCapData.py, NatNetClient.py and DataDescription.py. The first one has been useful to establish a connection (by creating a NatNetClient object) and receive data via a NatNet connection and decode it using the NatNetClient library. The others have been used to clearly read and print the data.

The GitHub repository instead, gave us the possibility to send commands to the Crazyfie. The most important code, for our project, is: "high_level_commander.py" which is intended to work with an absolute positioning system (as our MoCap system is). Its main power is the function "go_to", that takes a set of points as input, and it is in charge of sending the drone the right commands in order to move itself to the set point.
Putting all these libraries together, the final "main" script has been created. All can be schematized in the following steps:

- Connection of our host with the cameras;

```
1    optionsDict = {}
2    optionsDict["clientAddress"] = "10.105.204.117"
3    optionsDict["serverAddress"] = "10.132.42.98"
```

```
    optionsDict["use_multicast"] = False

    optionsDict = my_parse_args(sys.argv, optionsDict)
    streaming_client = NatNetClient()
    streaming_client.set_client_address(optionsDict["clientAddress"])
    streaming_client.set_server_address(optionsDict["serverAddress"])
    streaming_client.set_use_multicast(optionsDict["use_multicast"])
```

- Connection of our host PC with the Crazyflie;

```
    URI = 'radio://0/60/2M/E7E7E7E7E7'
    cflib.crtp.init_drivers(enable_debug_driver=False)

    with SyncCrazyflie(URI, cf=Crazyflie(rw_cache='./cache')) as scf:
        cf = scf.cf
        cflib.crtp.init_drivers()
```

- Trajectory definition, in our case we defined a square measuring four points around the column in the lab. The measures are taken with respect to the origin of the Optitrack system:

```
    # 1st_point
    x1 = 0.5
    y1 = 0.6
    z1 = 0.25
    # 2nd_point
    x2 = 1.5
    y2 = 0.6
    z2 = 0.25
    # 3rd_point
    x3 = 1.5
    y3 = 0.6
    z3 = -0.75
    # 4th_point
    x4 = 0.5
    y4 = 0.6
    z4 = -0.75
```

- Main loop in which we first enable the high_level_commander library in order to control the drone through it, then we send a take off command and wait for 5 seconds for the data from the cameras. This waiting time is necessary since we have noticed that some lag is introduced by the blocking properties of the Crazyflie libraries. After that a command is sent in order to bring the drone to the correct starting point (first point of the square), this command clearly takes into account the position measured from the cameras; lastly the crazyflie follows a square path around the column:

```
    cf.param.set_value('commander.enHighLevel', '1')
    time.sleep(0.1)

    commander = cf.high_level_commander
    commander.takeoff(0.6, 3.0)
    time.sleep(5)

    # Read the crazyflie position from the camera
    drone = streaming_client.get_rigid_body()
    pos = drone.getPos()

```

```
    # To initial point
    commander.go_to(x1 - pos[0], -z1 + pos[2], y1, 0, 3)
    time.sleep(5)

    # Trajectory starts here:
    commander.go_to(x2 - pos[0], -z2 + pos[2], y2, 0, 3)
    time.sleep(5)
    commander.go_to(x3 - pos[0], -z3 + pos[2], y3, 0, 3)
    time.sleep(5)
    commander.go_to(x4 - pos[0], -z4 + pos[2], y4, 0, 3)
    time.sleep(5)
    commander.go_to(x1 - pos[0], -z1 + pos[2], y4, 0, 3)
    time.sleep(5)

    # Landing:
    commander.go_to(x1 - pos[0], -z1 + pos[2], 0, 0, 3)
    time.sleep(5)

    commander.stop()
```

where:

- **drone** refers to the rigid body created in Motive that represents the real life quadcopter. The software streams all of its measured attributes to our **drone** object in the python script.
- **pos** is one of the attributes and it is a position vector (x, y, z). It refers to the rigid body center of mass.
- **scf**, **cf**, and **commander** are respectively SyncCrazyflie, Crazyflie and HighLevelCommander objects.

4.2. Trajectory planning

The aim of our project is to follow a global trajectory that turns around the column (which is present in the Lab). After the definition of a global reference system, we defined a mathematical trajectory with respect to it. The final goal is to follow it, whatever the drone starting point is. An example of a complex trajectory that could be performed is:

```
import math
import numpy as np
import matplotlib.pyplot as plt

r = 0.5
x_offset = 1
y_offset = 0.4
z_offset = -0.25geo
t = np.linspace(0.1, 10, 100)
x = x_offset + r*(np.cos(2*math.pi*t))
z = z_offset + r*(np.sin(2*math.pi*t))
y = y_offset + 0.05*t

A_row = np.array([x, z, y])
A = A_row.T

fig = plt.figure()
ax = plt.axes(projection='3d')
```

```
19      ax.set_title("Spiral")
20      ax.plot3D(A[:,0], A[:,1], A[:,2])
21      B0 = line(0.3, 0.8, 0.0, 0.3, 0.8, 0.4)
22      B1 = line(0.3, 0.8, 0.4, A[0,0], A[0,1], 0.4)
23      ax.plot3D(B0[:, 0], B0[:, 1], B0[:, 2])
24      ax.plot3D(B1[:, 0], B1[:, 1], B1[:, 2])
25      plt.show()
```

Where **line** is a function we have created to generate points (straight line) from the initial point to the final one.
The plot is:

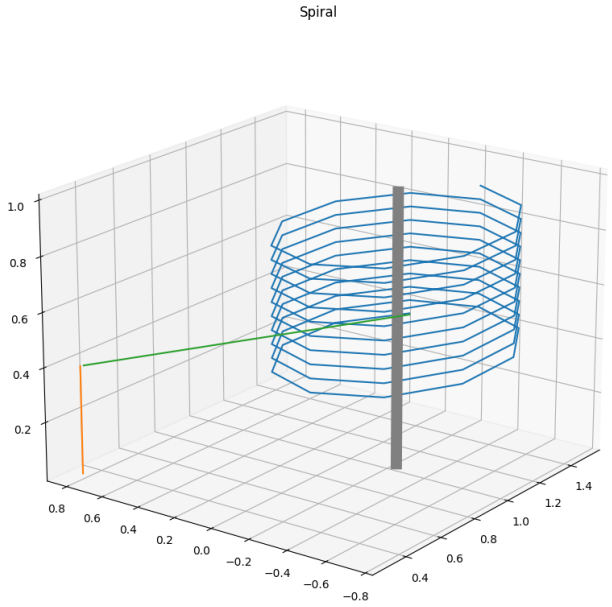

Figure 4.1: Spiral trajectory

In our case for practical and time reasons we opted for a square trajectory:

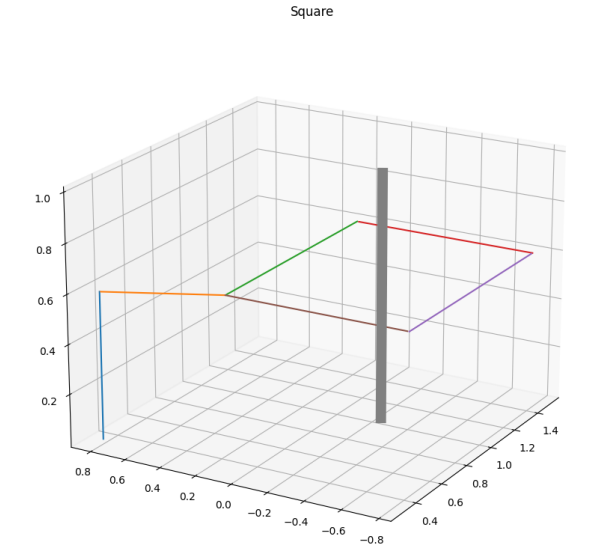

Figure 4.2: Square trajectory

4.3. Encountered issues

In this section we are going to explain some issues we got, in order to help future students or anyone else who wants to work on it to solve them quickly. To be more clear, we list them:

- **ERROR: Could not connect properly. Check that Motive streaming is on.**
  This problem can occur if the network between the client and the server has some problem, or if the streaming is not set up correctly in Motive. We have dealt with it several times, mostly because of connection problems. Fortunately, Professor Smarra and Dr. D'Ortenzio helped us by providing a good network to the lab.
  P.S. Anyone who wants to come across this project should no longer have this problem, don't worry!
- **Mismatch between real and cameras measurements**
  This difficulty arose from the calibration. When you have to do it, you have to specify if the markers on the wand are in position A or B. The total distance between the initial and final marker will be 250mm if they are in position B or 500mm if in A. This difference lead to a double or half mapping of the room if you do not set the length correctly on Motive. In our case, we did not know this at first, so we left the default setting of 250 mm on Motive but, instead, we put the marker in position A. Because of this, a movement that should be 1.5 m turns out to be 0.75 m! We realized this by doing several experiments, with a real-time data stream. Obviously real-time streaming itself might be responsible for this mismatch too since data has to travel from the host to target PC and be processed on python.
- **Make data accessible**
  We face this issue in order to use the camera measurments outside their environment. This was a necessity for us, since we had to work with NatNet and Crazyflie libraries together, so we need to ”export” data from their native codes. In particular, after hours of searching, we realised where the rigid body object, with its position and orientation, was. Once we found it, we became aware of the fact that it was hidden in its library: NatNetClient.py. That's why in this code we added a variable, a copy of the existing object handle and made it an attribute, so that it is accessible from the outside with a simple ”get method” that we implemented.
- **Lack of simulation platform**
  Since BitCraze doesn't offer a simulation environment (it is working on it), to test the code we had to make a flight, no way out. When you are testing something you are not sure on the result of course and, furthermore, these quadcopters sometimes have a random behavoiur, so that we end up with several unavoidable crashes. In addition, our drone has a mount (shown in Fig.3.3) and is even more vulnerable. In fact, during our testing, a breakage of the stand occurred and unfortunately we had to reprint it.
- **Drone not flying as we wanted**
  Almost every time after adding some functionalities to our code we went to try it on the actual drone. To our surprise we found out that it responded differently to the same commands and the flight usually ended with an unexpected crash. This behaviour was especially enhanced with our support (Fig. 3.3, the heavier) for this reason we have chosen to change it. After this change things went definitely better even though we didn't get 100% reliability.

5

Conclusion

After many failed attempts and hours in the laboratory we have managed to successfully control the trajectory of the Crazyflie 2.X with the combined measurements of the MoCap system. In particular, we managed to make it follow a square trajectory around the column present in the Automation Coppito 2 laboratory. Overall it has been a really interesting and stimulating experience. It has given us the opportunity to ramp up on some hands on experience with the MoCap system that with difficulty could be acquired elsewhere, moreover it has given us an idea of what working across multiple complex frameworks looks like. Hopefully this project will be the foundation of future projects for students on the Master's Programme in Control Sytstems Engineering to learn more about autonomously controlled drones and optimize what we have started.
Finally we thank Prof. Francesco Smarra and Dott. Alessandro D'Ortenzio for their availability and useful insights that helped us solve some of the issues we had.

Future works

- Continuously check the camera positions with the ones acquired by the flowdeck and command a correction in position whenever necessary, this would ensure a really high precision given the accuracy of measurements ensured by the Optitrack system.
- Control the drone without the need of the flowdeck and the associated measures, this could be done by considering the camera positions as state variables for the drone. In this way, the precision will be ideally of the order of millimeters.
- Implement relative movements in order to correct the global position error without modifying the state variables. E.g. if the drone sensor measures x = 0.5m the cameras measure x=0.4m, one can send a command to the drone of going backwards of 0.1m along the x axis.