Technical Documentation

CrazyTrain

2022–12–16

Version 1.0

Status

Reviewed	Anton Bossen	2022-12-14
Approved	Filipe Barbosa	2022-12-16

Project Identity

Group E-mail:	antbo569@liu.se
Homepage:	http://www.isy.liu.se/tsrt10/group
Orderer:	Filipe Barbosa, LiU, ISY Phone: - E-mail: filipe.barbosa@liu.se
Customer:	Linköpings universitet (LiU), ISY Phone: - E-mail: -
Supervisor:	Daniel Arnström, LiU, ISY Phone: - E-mail: daniel.arnstrom@liu.se
Course Responsible:	Daniel Axehill, LiU, ISY Phone: +46 13 28 40 42 E-mail: daniel.axehill@liu.se

Participants of the group

Name	Responsibility	E-mail
Morteza Akbari	Hardware (HW)	morak752@student.liu.se
Erik Axelsson	Documentation (DOC)	eriax970@student.liu.se
Anton Bossen	Project leader (PL)	antbo569@student.liu.se
Oskar Grönlund	Simulation (SIM)	oskgr783@student.liu.se
Anton Håkansson	Design (DES)	antha842@student.liu.se
Lukas Jonsson	Software (SOF)	lukjo147@student.liu.se
Patrik Lindström	Testing (TEST)	patli821@student.liu.se
Emil Lydell	File management (FM)	emily553@student.liu.se

CONTENTS

1 Introduction 1

1.1 Partners 1

1.2 Aim and goal 1

1.3 Use 1

1.4 Project background 1

1.5 Definition of terms 1

2 System Overview 3

2.1 Hardware 3

2.2 System infrastructure 4

2.3 Crazyswarm 6

3 Graphical User Interface 7

3.1 Functions 7

4 Trajectory Building and Following 10

4.1 UAV trajectory 10

4.2 RRT star 10

4.3 Follow-leader flight 12

4.4 Trajectory following 13

4.5 Position estimation 14

5 Simulation Environment 15

5.1 Drone Simulation 15

6 Live visualisation 17

7 Safety Features 19

7.1 Collision avoidance 19

7.2 Emergency button 19

7.3 Connection loss with Qualisys or radio 20

7.4 Bad trajectory following 20

8 Conclusion 21

9 Features Developed for Future Improvements 22

9.1 Model Predictive Control 22

10 Future Improvements 25

10.1 MPC 25

10.2 Obstacle identification and Dynamic obstacles 25

10.3 Simulation environment and Motion model 25

10.4 Real-time visualisation 25

10.5 Safety features 26

10.6 Programming language 26

10.7 Autonomously and manual control 26

DOCUMENT HISTORY

Version	Date	Changes made	Changes by	Reviewer
0.1	2022-11-08	First draft.	Project group	Anton Bossen
0.2	2022-12-13	First draft.	Project group	Anton Bossen
1.0	2022-12-14	First version.	Project group	Anton Bossen

1 INTRODUCTION

This is the technical report for the project CrazyTrain in the course TSRT10, automatic control project course. In this document, all technical information about the developed system will be provided. The document contains each implemented subsystem, which will be dissected and described in detail.

1.1 Partners

Partners to this project are the following

- Daniel Axehill, customer and examiner, Automatic Control (ISY) at Linköping University.
- Filipe Marques Barbosa, orderer, Automatic Control (ISY) at Linköping University.
- Daniel Arnström, supervisor, Automatic Control (ISY) at Linköping University.

1.2 Aim and goal

This project aimed to make a visual demonstration using formation flying drones which can be presented in Visionen at LiU. To achieve this, the drones should be robust against disturbances and be able to follow predefined trajectories. This was done by extending the existing public project Crazyswarm [1] with a new planner. The planner should be able to create a path around passive obstacles. Another goal is to make the drones fly autonomously in formation, by creating a follow-the-leader formation. Additionally to this, there was a goal to build a simulation environment where future testing could be performed with software-in-the-loop.

1.3 Use

The project will serve as a base for future projects in the same course. It could also, if succeeded, be used as good advertising for LiU and ISY to attract new students.

1.4 Project background

Drones are becoming more and more popular in society, with ever-growing new application areas. One new way to utilize them is the use of multiple drones that collaborate to solve a task, such as carrying a load. To achieve good collaboration, the drones need to sufficiently be able to perform different maneuvers simultaneously without colliding with each other or their surroundings. The drones must also be able to autonomously follow trajectories to solve their tasks. In this project, the usage of several drones autonomously following pre-defined trajectories has been implemented.

1.5 Definition of terms

- **Crazyflie** refer to drones Crazyflie 2.0 and 2.1, developed by Bitcraze [2]. If more than one is used, the drones will be referred to as Crazyflies.

- **Crazyradio** Communication between Crazyflies and the laptop is conducted through a Crazyradio, developed by Bitcraze [2].
- **Filter**. The filter refers to the combination of measurements and a motion model to estimate the position of the Crazyflie.
- **Graphical User Interface (GUI)** allows the user to interact with the software in a graphical interface.
- **Inertial Measurement Unit (IMU)** is a collection of accelerometers and gyroscopes to measure acceleration, orientation, and rotation speed.
- **Path** is the track laid for the Crazyflie to follow.
- **Pitch angle** is the angle around the y-axis (between the x-axis and z-axis).
- **Qualisys Camera System** is the motion capture system that enables feedback on where the Crazyflies are positioned in the room.
- **Robot Operating System (ROS)** is the operating system that is used to divide the different processes for the Crazyflies.
- **Roll angle** is the angle around the x-axis (between the y-axis and z-axis).
- **Target** means anything with a marker attached to it, Crazyflie(s) and obstacles.
- **Trajectory** is the path of the moving object(s), the object(s) being the Crazyflie(s).
- **Visionen** is a large room at Linköping University where the Crazyflies are tested. The Qualisys Camera system is set up in this room.
- **Yaw angle** is the angle around the z-axis (between the x-axis and y-axis).

2 SYSTEM OVERVIEW

In this chapter, the hardware and overall system infrastructure of the project are presented.

2.1 Hardware

In this Section, the hardware used for the project is described. The hardware consists of the drones which are flown, a computer running the software, and a radio for communication between the two. Also, a motion capture system is used to update the actual positions of each drone.

2.1.1 *Crazyflie*

The drones used for the project are called Crazyflie 2.1, which can be seen in Figure 2. Crazyflie's are nano-quadcopters that are developed by Bitcraze and are open source [2]. The nano-quadcopters have four rotors, an internal measurement unit (IMU) and for this project an active marker deck which is described more in detail in Section 2.1.3. The Crazyflies are commonly used for research purposes as they are relatively cheap and only weighs 27 grams. The drones also come with an ecosystem of open-source code that can be used in various ways.

The onboard IMU has an accelerometer, a magnetometer, and a gyroscope for measurement of acceleration in the x-, y- and z-axis and rotation in roll, pitch, and yaw, as seen in Figure 1.

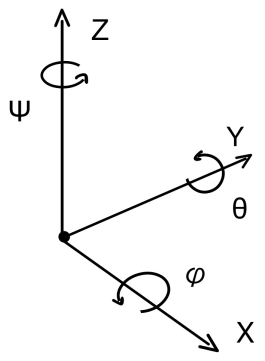

Figure 1: The coordinate system of the Crazyflie with roll, pitch and yaw depicted as ϕ, θ and Ψ.

The drones can be flown by High- or low-level command, which means that the input to the drones can either be by sending the wanted power to each rotor or by sending positions in the x-, y- and z-axis. During the project, high-level command has been used.

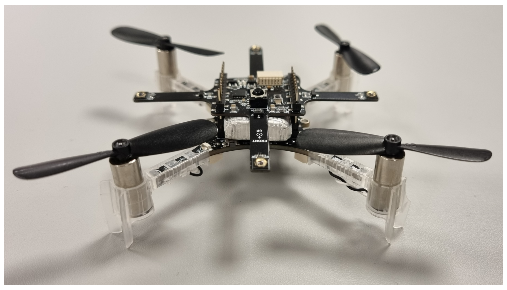

Figure 2: A Crazyflie 2.1 equipped with an active marker deck, which was used in the project.

2.1.2 *Qualisys*

Qualisys is a camera system that uses motion capture, used for positioning the Crazyflies in Visionen. The system is able to detect active or passive markers to track objects. Visionen contains 20 Qualisys cameras of two different kinds. The update speed of the Qualisys system is 100 Hz [3]. The drones are defined as separate rigid bodies in the system by their specified ID. The Qualisys system is able to keep track of the position and orientation of each drone.

2.1.3 *Active markers*

Each drone is equipped with an active markers deck. The active markers are made out of four LED lights which emit IR light to the motion capture system. Each marker can be set to a unique frequency, which makes it possible to separate the drones and assign them individual ID:s. The active markers work well with the Qualisys system.

2.1.4 *Computer*

The computer used for the project runs Ubuntu 20.04 as the operating system. To fly the drones, ROS, Crazyswarm, and CrazyTrain project need to be installed. The Crazyradio should also be connected via USB.

2.1.5 *Crazyradio*

A Crazyradio is used to connect the drones to the computer. The Crazyradio has a bandwidth of 2.4 GHz [4]. The Crazyradio is able to send signals from the computer to the drones, and receive signals from the drones to the computer. Due to bandwidth limitations, a maximum of three drones are able to send data to the computer. To fly more drones, additional Crazyradios kan be connected.

2.2 System infrastructure

The system infrastructure is shown in Figure 4. The major part of this project's implementation is the graphical user interface, which is able to control everything in the system. For more about the GUI, see Section 3. From the GUI, the desired waypoints flow down multiple scripts until a trajectory is created which can be sent to the drones via the Crazyswarm and Crazyflie projects. This project is tightly connected to the external projects Crazyswarm and Crazyflie, as seen in Figure 4. Crazyswarm and Crazyflie are used for the low-level control of the drones. This project has been mainly about adding additional features for the high-level control and providing a more user-friendly experience by connecting everything to a GUI.

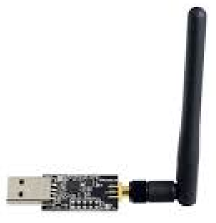
Figure 3: A Crazyradio which was used in the project to control the drones. [4]

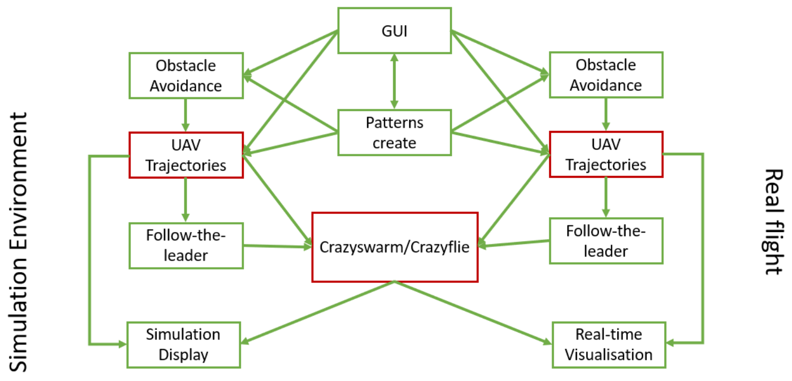

Figure 4: An overview of the system where either real flight or simulation environment is active. Green boxes represent what has been implemented by CrazyTrain while red boxes represent what has been done externally.

CrazyTrain has created a high-level commander. In this case, it means that the system calculates the reference trajectory that the Crazyflie(s) should follow. The Crazyswarm/Crazyflie project is the low-level commander which controls each individual rotor and the speed of it to follow the trajectory. Within the low-level commander, there is a Kalman filter, PID controller and pre-built collision avoidance between the drones.

The pose of the drones is received From the Qualisys system and can be plotted in the system's real-time simulation, see Section 6.

2.3 Crazyswarm

The Crazyflie(s) receive a planned trajectory, which gets processed through the controller from the CrazyTrain project. The deviation is the distance between the estimated position from the extended Kalman filter and the trajectory. Qualisys system is able to receive a pose of each drone, which is used for both the filter and CrazyTrain project. The flow chart of Crazyswarm/Crazyflie project can be seen in Figure 5.

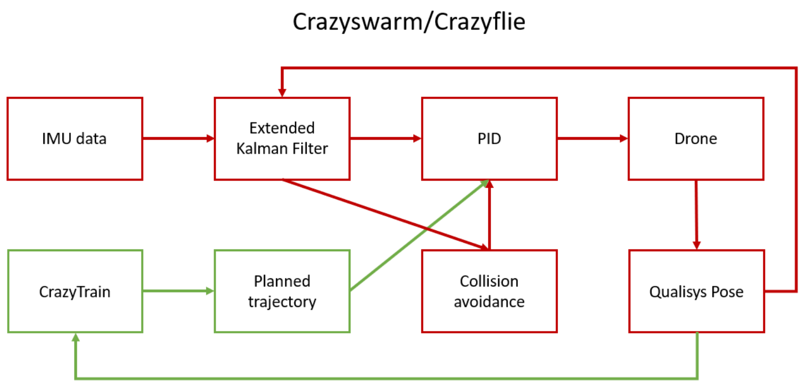

Figure 5: An overview of the Crazyswarm/Crazyflie project infrastructure.

3 GRAPHICAL USER INTERFACE

The Graphical User Interface, GUI, that was developed for the CrazyTrain project was built upon the previous year's project, Crazycrowd. It was modified and expanded upon to fit CrazyTrain's project. The GUI is programmed in Python3 with the standard Python interface library *Tkinter*.

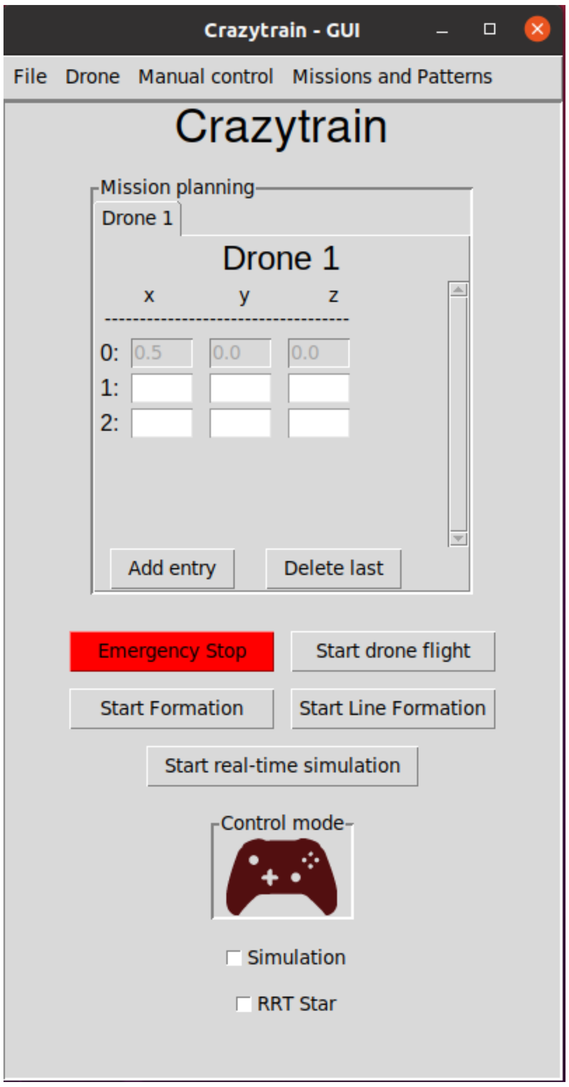

Figure 6: GUI Overview.

3.1 Functions

All the GUI functions is described in this section. A picture of the GUI can be seen in Figure 6.

3.1.1 *Waypoints*

The waypoints consist of global coordinates in the x-, y-, and z-plane together with an optional yaw angle. The starting position of each drone (seen as row zero in Figure 6) is added as an offset to the global coordinates so that the following input of waypoints are defined relative to each drone's starting position. Waypoints can be added or removed in the GUI. The waypoints will be followed in the order they are listed in, if the button *start flight* is pressed. More about this in chapter 3.1.3. The same procedure is used if the user wants to add waypoints to multiple drones. Each drone will have a separate tab in the scrollbar. Most of this is handled in the file *gui_drone_scroll_tab*. To start a flight using the defined waypoints, press the button *start flight*. The waypoints will then be stored in an array, which in turn will be converted to a CSV file using the function *save_csv_file* located in *gui_drone_scroll_tab*. This CSV file will be converted to global coordinates, which the RRT-planner and *uav_trajectories* use to create the trajectory CSV file. The trajectory CSV file will end up in the script folder `crazyswarm\ros_ws\src\crazyswarm\scripts`. The script *start_flight.py _crazytrain* will start, which first calls upon the built-in command *take-off*. Finally, the created CSV file is used and in the end, the built-in command *land* is called.

3.1.2 *Save and load*

There is a save and load feature in the GUI which can be accessed from the file menu. The save feature can be used to save waypoints. The load feature can then be used to load the saved waypoints. Waypoints are saved by pressing the *save* button located under the *file* menu. This will call upon the function *save_csv_file* located in *gui_drone_scroll_tab* which saves the waypoints as a CSV file. The button *load*, located under the menu *file* uses the saved CSV file to load in the waypoints in the GUI frame.

3.1.3 *Bash scripts*

Bash scripts, located in `GUI\bash\scripts`, were used to run the scripts located in `crazyswarm\ros_ws\src\crazyswarm\scripts` from the GUI. These are either missions, or certain simulations. These scripts either use waypoints that have already been created, such as *figure8_csv.py*, or will use the trajectory files that are created from waypoints in the GUI, see Section 3.1.1.

Another type of bash script is the emergency script, which is called when the emergency button is pressed. The script lands the drones that are being used and then shuts down every terminal to make sure all the drones are off.

To change the parameters of the controller and the filter, press the button *change parameters* under *File*. This calls on a bash script which opens up the file in which the parameters can be changed.

Under the menu *Drone*, several buttons call upon their appropriate bash script. A few of these have to be called upon before the drones can fly, see the user manual [5] for how to do this.

Two scripts can be called by pressing the button *Start Formation* and *Start Line Formation*. These buttons will start two different types of "Follow the Leader" scenarios. For more information, see Section 4.3.

The real-time simulation can be started by either pressing the *Start drone flight* button or the *Start real-time simulation* button. For more information, see Section 6.

Every mission, pattern, and waypoint script can be run in the simulation environment by first marking the simulation box in the GUI and then by using the normal procedure. For more information, see Section 5.

3.1.4 *RRT star*

The user can use RRT to avoid defined objects when using waypoints to create the trajectory. The objects can first be defined by pressing the button *Add obstacles* located under *File* in the GUI. This opens up a menu that looks similar to the waypoints menu. In the menu, each object is first defined, and then sent to the *obstacle* CSV file located in the folder `visualizer`. Most of this is handled in the *Pop_up* class and in the *Obstacle_points* class.*RRT* star can be activated by checking the *RRT star* box located on the GUI. The *RRT_on* boolean will be set to *True* and will ensure that the RRT algorithm is called before *UAV trajectory*. This is done in the function *make_trajectory*, located in the file *gui_drone _scroll _tab*). The *RRT* will take the waypoint CSV file made by waypoints (instead of *UAV trajectory*) and create a new one with obstacle avoidance in it. This file will be used by *UAV trajectory*. Then the same procedure is used as explained in Section 3.1.1.

3.1.5 *Manual control*

The user can enable manual control by pressing the *activate manual control* button located under the menu *manual control*. This sets the boolean *autonomous* to *False*, and thus the *manual_control.py* scripts will be called instead of *start_flight _Crazytrain* when the button *start flight* is pressed. To learn more about manual control, please read the *Crazycrowd* documentation.

4 TRAJECTORY BUILDING AND FOLLOWING

In this section, the process of creating reference trajectories for the drones as well as the autonomous flight programmes are explained.

4.1 UAV trajectory

In order to create smooth flight trajectories for the drones, code from a GitHub repository called UAV-trajectories has been used as an extension to this project. There is a script that takes waypoints in 3D coordinates as input and then produces 7-degree polynomials (with a specified time duration) as output, see Equation 1. The polynomial creates smooth trajectories that are fitted between each waypoint provided by the user. Embedded in the Crazyswarm code is a function for uploading the trajectory to a specific drone. Then, it is up to the onboard controller of the drone to follow this reference as well as possible. This way, the trajectory planning is conducted offline on the computer and then passed to the drones. A reason for doing it this way is to reduce the load in the radio signal, as the bandwidth of the Crazyradio is limited.

$$h(x,y,z,\psi) = \sum_{i=0}^{7} k_i x^i + l_i y^i + m_i z_i + n_i \psi^i, \tag{1}$$
$$k_i, l_i, m_i, n_i \in \mathbb{R}$$

4.2 RRT star

As mentioned previously in Section 3, the system have an RRT Star planner implemented to be able to create obstacle free paths whilst visiting the desired waypoints. Following will be a more technical description of the planner.

RRT Star is a type of motion planning algorithm used to find a path between a starting point and a goal in the world which is defined in the planner. It is an improved version of the more basic RRT Planner, which uses a random sampling method to explore the surrounding world and find an obstacle free path to the goal.

RRT Star works by first initializing a tree starting from the starting point. The algorithm will then repeatedly sample random points in the space and attempts to connect them to the tree. A step with a specified length is made from the closest point in the existing tree towards the newly created point. If the new point is deemed to be obstacle free, i.e. neither the point nor the closest way to get there intersects with any of the defined obstacles, it is connected to the closest point on the tree. The planner continues to do this until a path from the starting point to the goal is found. In this project, the parameter ϵ decides if a point is sufficiently close to the goal to deem the plan finished.

One key difference between the RRT Star and the RRT planer is that the RRT Star uses a cost function to guide the tree growth. In this case, the cost is dependent on the distance to the set goal point. This can be seen as the heuristic function for the planner. Additionally, RRT Star uses a function called *rewiring* to improve the path by connecting nearby points on the tree to their closest neighbours, even if they are not directly connected in the tree. This allows the algorithm to find shorter, more efficient paths than the basic RRT algorithm. A schematic representation of this algorithm can be seen in Figure 7. The black nodes in the figure are all part of the old exploration tree and the green node is a new, randomly sampled, node. The new node connects to the tree via the closest point, q _nearest, as seen in green. Then, the rewiring step is conducted, where all connections between nodes within the radius R _neighbours are reevaluated. The red path is removed from the tree and a new green arrow is connected to the left-most node, as this is now a more optimal path.

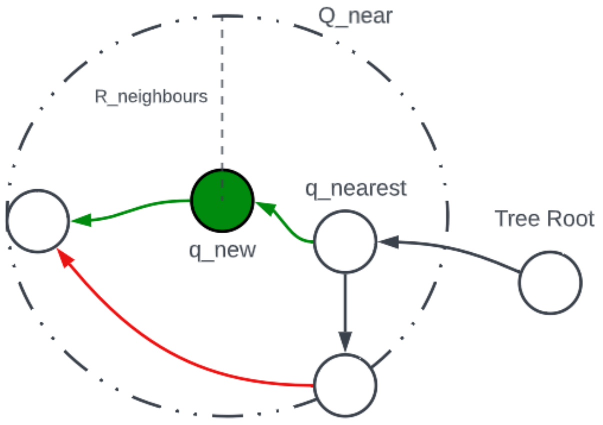

Figure 7: An illustration of how the expanding tree is rearranged when a new random node (green) is introduced among the existing tree (black). The red arrow represents a connection that was previously in the tree but is now removed in favour of the new, shorter, path created by the green arrows. Q _near represents the area defined by the radius R _neighbours from the newly created node. All previous connections between nodes within this area are reevaluated with respect to the new node and rewired if a more optimal solution is found.

The result from the implemented RRT star algorithm for a certain mission can be seen in Figure 8. One can see that a vast tree has been created in the 3D-space (grey lines in figure) and that the algorithm has found an obstacle free path from the starting point to the goal node (green line in the figure). The obstacles are defined from the GUI as blocks with start- and end coordinates.

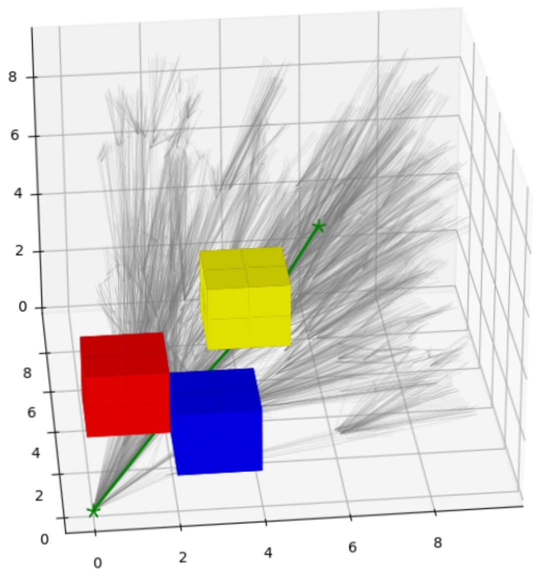

Figure 8: An example of the tree that has been created by the RRT-star algorithm. The green line represents the most optimal, obstacle free, path from the starting point to the goal.

4.3 Follow-leader flight

A follow-the-leader flight programme can be chosen from the GUI. The programme makes the drones follow a specific leader drone. The Crazyflie with the lowest ID becomes the leader. This Crazyflie is given a trajectory to fly, and the other Crazyflies try to follow the leader with a offset. This can be done either in a line formation or in an unspecified formation. In both cases of follow-the-leader only the leader drone will have a trajectory locally uploaded, the others will have calculated position inputs from Algorithm 1 or 2.

The line formation is implemented using the position of the drone ahead as the goal position and the desired offset between drones. The position of each drone is broadcasted by default via ROS and is obtained via the command, *cf.position()*. If the distance between the Crazyflies is smaller than the set safety distance, then they will keep their position until it is safe to continue. For example, if the leader drone turn 180 degrees or the drones for some other reason come too close to each other, then the collision avoidance will activate. The programme starts by positioning all the drones in a line, and then it will send the command *cmdPosition()* with a rate of 10 Hz to each follower drone. The height of the following drones will always be set as the height of the leader drone. This is because the collision avoidance can give some undesired height changes to the sub leader drone. See Algorithm 1 for the specific algorithm that makes follow-the-leader possible.

Algorithm 1 Follow-the-leader line

```
    Input: $p_i, j_j, d_{offset}, z_{leader}$

1:
2: if $||p_i - p_j|| < d_{offset}$ then
3:     $drone_j.goTo(p_j.x, p_j.y, z_{leader})$        ▷ The follower will stay put
4: else
5:     $drone_j.goTo(p_i.x, p_i.y, z_{leader})$        ▷ The follower will go to $drone_i$
6: end if
```

The unspecified formation works similarly. In this case, every Crazyflie tries to follow the leader drone instead of the drone ahead. This makes the pattern of the formation unpredictable. The algorithm for the formation flying is shown in Algorithm 2.

Algorithm 2 Follow-the-leader formation

```
    Input: $p_L, p_j, d_{offset}$

1:
2: if $||p_L - p_j|| < d_{safe}$ then
3:     $drone_j.goTo(p_j)$        ▷ The follower will stay put
4: else
5:     $drone_j.goTo(p_L.x, p_L.y, p_L.z)$        ▷ The follower will go to the leader
6: end if
```

4.4 Trajectory following

Onboard each of the Crazyflies, there is a controller implemented which has the purpose of ensuring proper following of the commands that the high-level commander has received. Apart from adjusting the controller parameters, the controller structure has been created and implemented to its full extent within the *crazyflie-firmware* which is created by Bitcraze.

The controller structure that has been implemented is a Cascade-PID-controller. More precisely there are four levels of control for each of the Crazyflies, the different levels are position, velocity, attitude, and attitude rate. For a visual representation of the controller structure, see Figure 9. For further reading, see [2].

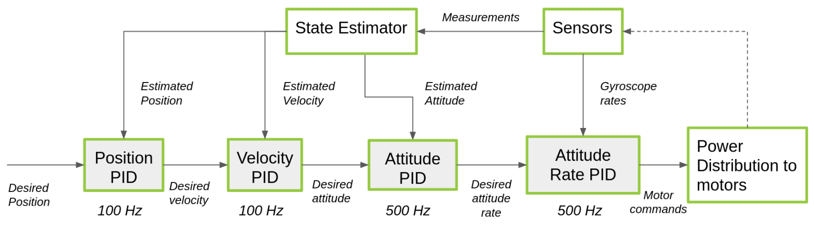

Figure 9: An overview of the cascaded PID controller, developed by Bitcraze, which is used on the Crazyflies. [2]

4.5 Position estimation

For the position estimation, an Extended Kalman filter (EKF) is used. The EKF is local to the drone and is created by Bitcraze [2]. The filter fuses the IMU data with the pose of the Qualisys system. For further reading, see [2].

5 SIMULATION ENVIRONMENT

The simulation environment is used to simulate and visualise the drones' position and planned path. The simulation environment consists of several callable functions which simulate with different methods. Additionally, a separate simulation environment exists for the Model Predictive Controller (MPC) that has been researched during the project. The MPC simulation environment is separate because it is not currently integrated with the CrazyTrain GUI as it does not, as of now, work with the other integrated parts. Thus, it did not make sense to integrate it within the GUI.

5.1 Drone Simulation

Simulations of the chosen drone trajectories can be called upon via the Graphical User Interface (GUI), for reference, see Figure 6. By selecting the simulation mode, the existing flight modes, represented by the different buttons in the GUI, will be presented in animated Matplotlib graphs rather than actual drone flights. All missions that can be created for the real drones can also be tested in the simulation environment before the real-life testing. This is a good way to evaluate the performance of a flight before risking the hardware.

5.1.1 *Software implementation*

Simulations of the different flight programs in the systems are animated with Matplotlib [6], which is described more in detail in the below section. How the simulation works is that the same signals that are sent to the high-level commander whilst flying the drones are mirrored on a "virtual drone", which is represented by a point in the Matplotlib animation. An example of the animation is shown in Figure 10.

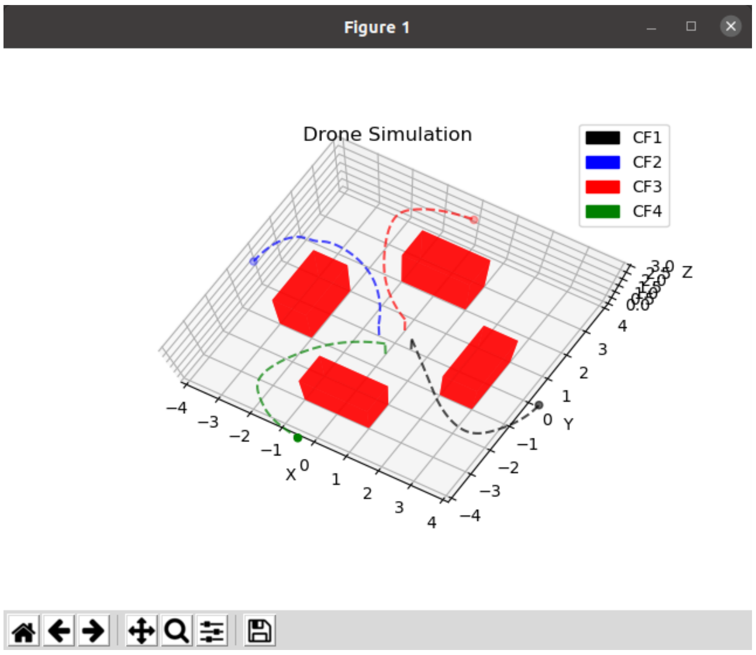

Figure 10: Example of a finished simulation using the RRT* planner.

This way of simulating flight programs comes with both advantages and disadvantages. Firstly, by mirroring the high-level commands on the "virtual drone" it can be ensured that the simulation works for multiple different flight programs. This could otherwise be a problem since the Crazyswarm module includes a plethora of different ways to send high-level commands, ranging from set-points to full-state commands. This could present problems for a simulation environment built for more of a low-level control approach, where it is presumed that the input to the system would be in the same format for all different flight programs.

The biggest downsize with this way of simulating the systems is the lack of an implementation of a proper motion model, to take the dynamics of each Crazyflie into consideration. This type of implementation would require a more low-level control approach, whereas the simulation would have to take into consideration how the onboard Crazyflie controllers convert the high-level commands to rotor speed inputs to each drone.

Furthermore, the simulations include some graphical features to assist with assessing the simulation results. Firstly, the trajectory which has been calculated via the use of UAV-Trajectories is visible within the animation, with the requirement that the high-level command that is sent includes waypoints. Otherwise, when sending reference velocities to the high-level commander, for example, there is no available trajectory for the simulation to present. In addition to the trajectories, a graphical representation of the obstacles is implemented, where the obstacles are defined from the GUI and saved in a CSV file. These graphical features have been implemented in such a way that they can be removed or added via the GUI to ensure that the information which is of importance to the current flight program is clearly shown.

5.1.2 *Matplotlib*

All simulation and visualisation windows use the Python library *Matplotlib* [6] for graphical visualisation. This is an object-oriented API that can be used to easily represent data in graphs. The usage of matplotlib means that the simulation does not visually represent a model of the actual drones, but is rather a tool for evaluating flight behaviour and performance.

6 LIVE VISUALISATION

A real-time visualisation of the drone positions is provided through the GUI, where the live positions of each drone are plotted together with their planned trajectories. The Crazyflie positions are logged and can be saved to CSV files for further processing. The visualisation gets its data from a ROS node which subscribes to a ROS topic with the estimated pose from Qualisys. The pose is sent from Qualisys via the Visionen Wi-Fi connection at a frequency of 10 Hz. The Euclidean distance between the estimated positions and the corresponding reference positions in the trajectories is continuously calculated, and if the position error becomes greater than 1 m for any drone, the emergency function is called so that all drones land. This is a safety measure that is used to make sure that if a drone is faulty and wanders off its trajectory, the program is terminated to protect the drones and their surroundings. The error measurements can also act as an evaluation basis for the performance of a certain controller or filter. The real-time visualisation is operated through a pop-up GUI, as seen in Figure 11, where there are options such as visualisation of virtual obstacles which the drones should avoid when obstacle avoidance is used. An example of the visualisation of a mission can be seen in 12 and the corresponding plot of position error during the flight can be seen in Figure 13.

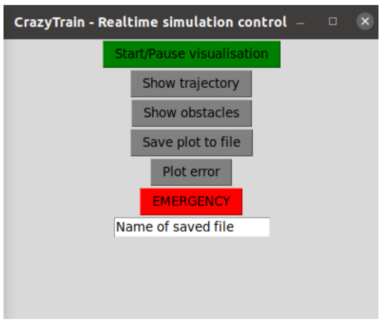

Figure 11: The pop-up GUI is used to control the real-time visualisation plot.

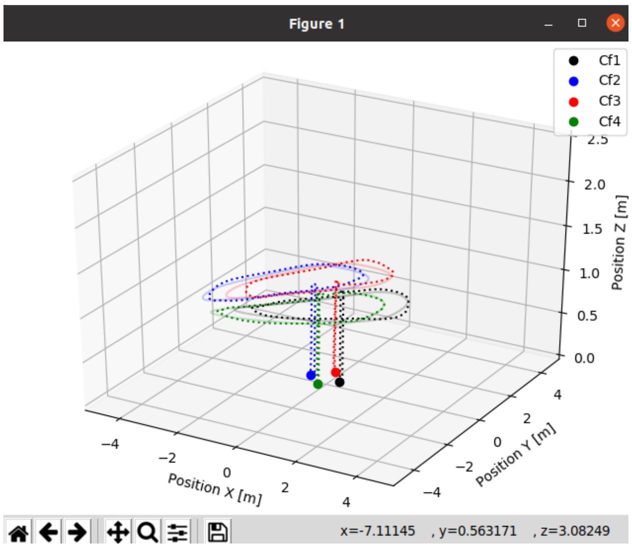

Figure 12: A screenshot of the output from the real-time visualisation after completing a mission. The current position of each drone is depicted as a big point, the travelled path as a dotted line and the reference trajectory as a faded line.

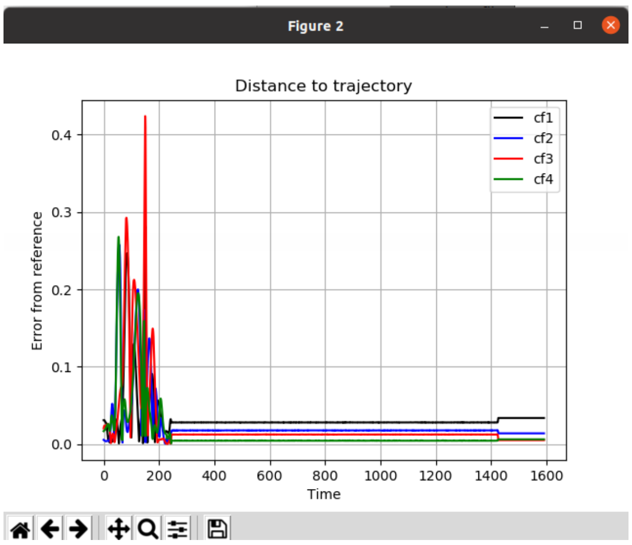

Figure 13: A screenshot of the output from the error plot after completing the mission, as depicted in Figure 12. The error is measured in the xy-plane and shows how many meters each drone is from its corresponding trajectory.

7 SAFETY FEATURES

This chapter details the different safety features of the project.

7.1 Collision avoidance

The collision avoidance between drones only needs to use the position of the drones and does not need any information on the trajectory to avoid a collision. The collision avoidance uses buffered Voronoi cells which the drone is only allowed to move inside to guarantee a collision-free environment. For further reading on collision avoidance between drones, see [7].

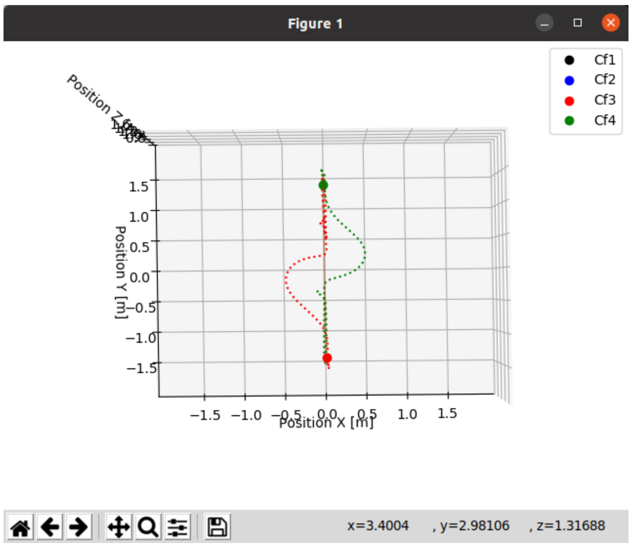

Figure 14: A screenshot of a figure that shows a real-time simulation of two drones that have trajectories through each other.

The collision avoidance is fast and has a smooth transition between following the trajectory and avoiding collisions. An example run of the collision avoidance can be seen in Figure 14, where the mission is for the drones to fly to each other's starting points, with the collision avoidance of the drones their paths will not intersect.

7.2 Emergency button

The emergency button is found in the GUI, and the purpose of this button is that the user should be able to stop the flight if something goes wrong. When the emergency button is pressed, it will call the land function on each drone to land the drone/drones safely. It will also terminate all active programs and ROS terminals to ensure that no further signal is sent to the drones.

7.3 Connection loss with Qualisys or radio

If the drones lose connection with the radio, the emergency stop button activates and the drones fall slowly to the ground. If Qualisys lose track of the Crazyflies, then the Crazyflies will start to fall slowly to the ground and eventually land. If Qualisys yet again get track of the Crazyflies, then they will return to the mission.

7.4 Bad trajectory following

A safety feature has been implemented that will call on the emergency land function if the drone position is more than one meter off the course of the trajectory. This is done in the real-time simulation script and uses the same position estimate as the plot of the real-time simulation. The position is compared with the closest point on the trajectory, if it is larger than one meter the emergency land function is called. All drones will then land, even if some are following their trajectory.

8 CONCLUSION

The final product is an extension of the project by the previous LiU project *Crazycrowd* and the project *Crazyswarm*. The extension includes a GUI from which all the functions that have been implemented can be controlled. The functions are an RRT star planner, a simulation environment, a real-time simulation environment, missions, and patterns which all work for 4 drones. There are also two types of Follow-The-Leader missions, where three drones autonomously follows a leader drone. Further, one drone can be flown manually by switch to manual control in the GUI.

The project has been created with modularity in mind. All controller and filter parameters can be changed in the GUI. The submodules are also changeable, it is possible to toggle some submodules on and off. Further development can conveniently be added to the project and some existing modules can be removed.

9 FEATURES DEVELOPED FOR FUTURE IMPROVEMENTS

This section includes features developed, but not implemented in the final project. These features could be implemented in the future in another project.

9.1 Model Predictive Control

Model Predictive Control (MPC) simulations can be called upon from the Graphical User Interface (GUI). In this environment, the user can simulate different predefined missions or create their own for the scenario *Follow the leader*. The motion model used for the drones is described in Section 9.1.1. The MPC is implemented with the package *do-mpc* which is described in Section 9.1.2. Mission selection is described in Section 9.1.3. Visualisation of the path is described in Section 9.1.4.

9.1.1 *Motion Model*

Each drone has the simplified motion model using the state space form of Equation 2, where $\mathbf{X}$ is the state vector, $\dot{\mathbf{X}}$ is the time derivative of the state vector, i.e. $\frac{d\mathbf{X}}{dt}$, $\mathbf{U}$ is the control vector, $\mathbf{A}$ is the states' matrix, $\mathbf{B}$ is the input matrix of the model and $\mathbf{C}$ is the disturbance vector.

$$\dot{\mathbf{X}} = \mathbf{AX} + \mathbf{BU} + \mathbf{C} \tag{2}$$

For one single drone the state space model is using Equations 3 – 8.

$$\dot{\mathbf{X}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} \tag{3}$$

$$\mathbf{X} = \begin{bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} \tag{4}$$

$$\mathbf{U} = \begin{bmatrix} \theta \\ \varphi \\ T \end{bmatrix} \tag{5}$$

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{6}$$

$$\mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & -g & 0 \\ g & 0 & 0 \\ 0 & 0 & \frac{1}{m} \end{bmatrix} \tag{7}$$

$$\mathbf{C} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -g \end{bmatrix} \tag{8}$$

Where x, y and z are the Cartesian coordinates, $\dot{x}$, $\dot{y}$, $\dot{z}$ are the velocity in each direction, $\ddot{x}$, $\ddot{y}$, $\ddot{z}$ is the acceleration in each direction, g is the gravitational constant, m is the mass of the drone, θ is the pitch (angle around the local x-axis), φ is the roll (angle around local y-axis) and T is the thrust force from the propellers.

Each state and control signal has limitations with minimum and maximum values, the same for each drone. x, y, z is limited by the size of the testing location called Visionen. $\dot{x}$, $\dot{y}$, $\dot{z}$ is limited by the maximum velocity v_{max} of the drones, which is set to 1 in the implementation. θ, φ is limited by the maximum allowed angle for safety reasons, and T is limited by the maximum thrust of the drones. Explicitly, stated as the constraints 9.

$$\begin{array}{ccccc} -5 & \leq & x & \leq & 5 \\ -5 & \leq & y & \leq & 5 \\ 0 & \leq & z & \leq & 10 \\ 0 & \leq & \dot{x} & \leq & v_{max} \\ 0 & \leq & \dot{y} & \leq & v_{max} \\ 0 & \leq & \dot{z} & \leq & v_{max} \\ -\frac{\pi}{4} & \leq & \theta & \leq & \frac{\pi}{4} \\ -\frac{\pi}{4} & \leq & \theta & \leq & \frac{\pi}{4} \\ T_{min} & \leq & T & \leq & T_{max} \end{array} \tag{9}$$

For four drones, this model is expanded with states for each drone. Coordinates along the global x-axis for each drone are denoted x_1, x_2, x_3, x_4. The same principle is used for all other states and control signals. This is implemented by simply expanding the equations for one drone, see Equations 10 – 15.

$$\dot{\mathbf{X}} = \begin{bmatrix} \dot{\mathbf{X}}_1 \\ \dot{\mathbf{X}}_2 \\ \dot{\mathbf{X}}_3 \\ \dot{\mathbf{X}}_4 \end{bmatrix} \tag{10}$$

$$\mathbf{X} = \begin{bmatrix} \mathbf{X}_1 \\ \mathbf{X}_2 \\ \mathbf{X}_3 \\ \mathbf{X}_4 \end{bmatrix} \tag{11}$$

$$\mathbf{U} = \begin{bmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \\ \mathbf{U}_3 \\ \mathbf{U}_4 \end{bmatrix} \tag{12}$$

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_2 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{A}_3 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{A}_4 \end{bmatrix} \tag{13}$$

$$\mathbf{B} = \begin{bmatrix} \mathbf{B}_1 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{B}_2 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{B}_3 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{B}_4 \end{bmatrix} \tag{14}$$

$$\mathbf{C} = \begin{bmatrix} \mathbf{C}_1 \\ \mathbf{C}_2 \\ \mathbf{C}_3 \\ \mathbf{C}_4 \end{bmatrix} \tag{15}$$

The limitations of each drone are the same, but there are further limitations for safety reasons, namely the minimum distance allowed between the drones. Explicitly, stated as in Constraints 16.

$$\begin{array}{ccccc} d_{min} & \leq & d_{12} & \leq & d_{max} \\ d_{min} & \leq & d_{23} & \leq & d_{max} \\ d_{min} & \leq & d_{34} & \leq & d_{max} \\ 2d_{min} & \leq & d_{13} & \leq & 2d_{max} \\ 2d_{min} & \leq & d_{24} & \leq & 2d_{max} \\ 3d_{min} & \leq & d_{14} & \leq & 3d_{max} \end{array} \tag{16}$$

The distance between drone 1 and 2 is denoted d_{12} and defined as in Equation 17. The same principle is used between the other drones. d_{min} is set to 0.3 meters and d_{max} is set to 1 meter.

$$d_{12} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{17}$$

9.1.2 *Implementation*

The MPC was implemented by using *do-mpc* package using Python. Because of the limitations of the package, the distance between the drones is implemented as states, further expanding the model. The structure of the program using the MPC is separated into several files:

1. *animations.py* – Animates the trajectories in 3D-plots for the MPC.
2. *mission_select.py* – Select a predefined mission or create a custom mission to simulate in the MPC. Returns mission goals and break_dist used in mpc_main.py for running the main loop.
3. *mpc.py* – Runs one iteration for the MPC.
4. *mpc_controller.py* – Controller setup for the MPC.
5. *mpc_main.py* – Main loop for the MPC. Returns the trajectory of the states.
6. *mpc_model.py* – Model setup for the MPC.
7. *mpc_simulator.py* – Simulator setup for the MPC.

9.1.3 *Mission Selection*

The user can create a custom mission in the MPC simulation. The program assumes the drones start at the point of origin in a line and the user types in the goal point the drones will go to within the allowed three-dimensional space. The user input is handled to avoid program crashes by only allowing valid user input otherwise, the program will repeat until it gets valid input. It is also possible for the user to select two pre-defined missions.

9.1.4 *Visualisation*

Each drone's projected trajectory is plotted in three-dimensional plots in Python using the package *Matplotlib*. The animation file will produce one static plot with the complete path and one plot with an animated path in three dimensions.

10 FUTURE IMPROVEMENTS

Parts of this project can be improved for next year's group.

10.1 MPC

The model used in the MPC could be improved. The software package used has flaws, especially in performance. Next year's group can improve performance dramatically by using another package.

10.2 Obstacle identification and Dynamic obstacles

The obstacles can now only be defined manually in the GUI. One improvement could be to let the Qualisys system automatically define them through passive markers. This will also open up the use of dynamic obstacles in which active markers have to be used instead. The obstacle avoidance that is already implemented between drones will not however work on other things than drones so some implementation is needed here.

10.3 Simulation environment and Motion model

The simulation environment in its current version does not take the dynamics of the drones into consideration. If one were to create a different simulation environment where this can be taken into account, it would improve the accuracy of the simulation by quite a bit.

The current system setup works with high-level commands, hence the simulation environment would need to take this into account as well. One possible implementation that would be able to both work with high-level commands and provide a way to simulate the dynamics of the drone would be to create a structure which would make use of "software-in-the-loop". This would be done by creating a virtual model of the drone structure, meaning that the model would have the high-level commands as input and then process these commands as the drone would in real life.

With this structure of the system, it would be possible to extract the low-level control signals to the virtual drone and together with an appropriate motion model simulate this in either a simple visual tool like Matplotlib, or even better something more advanced like Gazebo.

10.4 Real-time visualisation

There are some improvements which can be made on the real time visualisation code. The error distance function is currently only calculating the offset from the trajectories in the xy-plane, as a measure to get around the fact that all trajectories that are sent to the Crazyflies are defined in the air, when the drones have made their take-off routine. Thus, the position error will always be approximately one meter in the z-direction from the start, which would trigger the emergency function immediately. Further, the visualisation is currently calculating the error distances as the minimum distance from the position of the drone to all the points in the trajectory. This is inefficient from a computational point of view. This could possibly cause problems if a very long or complicated trajectory is being used.

10.5 Safety features

The emergency land is called when the controller is working poorly. An extension to this could be to call upon the emergency stop when the battery is too low.

10.6 Programming language

Improvements in performance can be made by switching programming language to anything faster than Python. Since the performance of the system is key, switching to e.g. C/C++ is recommended. The drones are coded with C.

10.7 Autonomously and manual control

The system can fly drones either autonomously or manually. The transition between them is only possible while the drone is not flying. Something that could be improved is to be able to switch them during flight. A combination could also be added to the project. That one drone is flying manually, meanwhile the other drones are flying autonomously. This function could be adapted into the follow-the-leader function, in that the leader is controlled manually while the other drones follow it autonomously.

REFERENCES

[1] J. A. Preiss*, W. Hönig*, G. S. Sukhatme, and N. Ayanian, "Crazyswarm: A large nano-quadcopter swarm," in *IEEE International Conference on Robotics and Automation (ICRA)*, Software available at https://github.com/USC-ACTLab/crazyswarm, IEEE, 2017, pp. 3299–3304. DOI: 10.1109/ICRA.2017.7989376. [Online]. Available: https://doi.org/10.1109/ICRA.2017.7989376.

[2] Bitcraze AB. "Crazyflie documentation." (2022), [Online]. Available: https://www.bitcraze.io/. (accessed: 2022-09-15).

[3] Qualisys AB. "Qualisys motion capture system." (2022), [Online]. Available: https://www.ntnu.edu/documents/221360533/1261844700/Qualysis_Oqus_QTM_brief2014.pdf/aab4b882-43ac-4524-818a-9e853df7479e. (accessed: 2022-09-19).

[4] Bitcraze. "Crazyradio pa." (2022), [Online]. Available: https://www.bitcraze.io/products/crazyradio-pa/. accessed: 2022-11-22.

[5] CrazyTrain, "User manual," 2022.

[6] Matplotlib. "Matplotlib: Visualization with python." (), [Online]. Available: https://matplotlib.org/. (accessed: 2022-12-14).

[7] Dingjiang Zhou, Zijian Wang, Saptarshi Bandyopadhyay, Mac Schwager, "Fast, on-line collision avoidance for dynamic vehicles using buffered voronoi cells," *IEEE ROBOTICS AND AUTOMATION LETTERS*, vol. 2, 2017.