Design Specification

CrazyTrain

2022–12–09

Version 1.1

Status

Reviewed	Anton Bossen	2022-10-30
Approved	-	-

CrazyTrain
antbo569@liu.se

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

DOCUMENT HISTORY

Version	Date	Changes made	Changes by	Reviewer
0.1	2022-09-21	First draft.	Project group	Anton Bossen
0.2	2022-09-28	Second draft. Resolved comments from supervisor.	Project group	Anton Bossen
0.3	2022-10-02	Third draft. Resolved comments from supervisor.	Project group	Anton Bossen
1.0	2022-10-05	First version.	Project group	Anton Bossen
1.1	2022-10-30	First version. Resolved comments from orderer.	Project group	Anton Bossen

1 INTRODUCTION

This document contains the design specification for the project CrazyTrain in the course TSRT10. The overall design of the system is presented and then dissected into individual components. The document aims to describe how each subsystem is supposed to be created and integrated with each other.

1.1 Purpose and Goal

The aim for this project is to make a visual demonstration using formation flying drones which can be presented at Linköping University. To achieve this, the drones should be robust against disturbances and be able to follow predefined trajectories. Another goal is to build a simulation environment where future testing could be performed with software-in-the-loop, i.e. where testing of controllers and filters could be done without the need of real hardware. The project will serve as a base for further development in future projects, and the software should enable for customised flight missions to be created by other users. The purpose of this document is to, at an early state, present an overview of the different parts of the projects and break down how they are planned to be solved.

1.2 Definition of Terms

- **Crazyflie** refer to drones Crazyflie 2.0 and 2.1, developed by Bitcraze [1]. If more than one is used, the drones will be referred to as Crazyflies.
- **Crazyradio** Communication between Crazyflies and the laptop, be through Crazyradio, developed by Bitcraze [1].
- **Filter**. The filter is used to fuse data from the onboard IMU with the position from Qualisys into a state estimate of the Crazyflie.
- **Graphical User Interface (GUI)** allows the user to interact with the electronics in a simplified graphical overview.
- **Inertial Measurement Unit (IMU)** is a combination of accelerometers and gyroscopes to measure acceleration, orientation and rotation speed.
- **Path** is a spatial construct that tells the Crazyflie how to get from start to finnish in the 3D space.
- **Robot Operating System (ROS)** is the operating system that is used to divide the different processes for the Crazyflies.
- **Target** means anything with marker attached to it, Crazyflie(s) and obstacles.
- **Trajectory** is a path with timestamps which the Crazyflie will follow.
- **Qualisys Camera System** is the motion capture system that enables feedback of where the Crazyflies are positioned in the room.
- **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 local z-axis (between local x-axis and local y-axis).

- **Roll angle** is the angle around the local x-axis (between local y-axis and local z-axis).
- **Pitch angle** is the angle around the local y-axis (between local x-axis and local z-axis).

2 SYSTEM OVERVIEW

In this section an overview of the system is presented together with a short description of the different subsystems, ROS and the included hardware. The project will be based on the work from previous year's project group in the same course [2]. However, some of the main parts of the previous work will be redone and some will be further developed. A visual representation of the system components and their dependencies can be seen in Figure 1. The subsystems are briefly described below.

- **Graphical user interface**: In the interface, the user will be able to create various types of flight missions and define certain settings. The GUI then passes the information to a planner. This will be created from scratch.
- **Simulation environment**: Will be created, where the missions could be tested before real tests are conducted. The simulation should include a drone motion model which can take real control input and simulate the behaviour of real drones.
- **Motion planner**: Takes as input from the user in the form of waypoints and calculate a trajectory. The trajectory is then sent to the controller as the reference signal.
- **Controller**: Will take reference signals from the planner and position estimates from the sensor system and produces control signals which are sent to both the simulation environment and the Crazyflies or one of them.
- **State estimator**: Consists of a Qualisys motion capture system, Inertial Measurement Units in the drones and a filter where the different sensor data is fused to produce a position estimate which is sent to the controller.

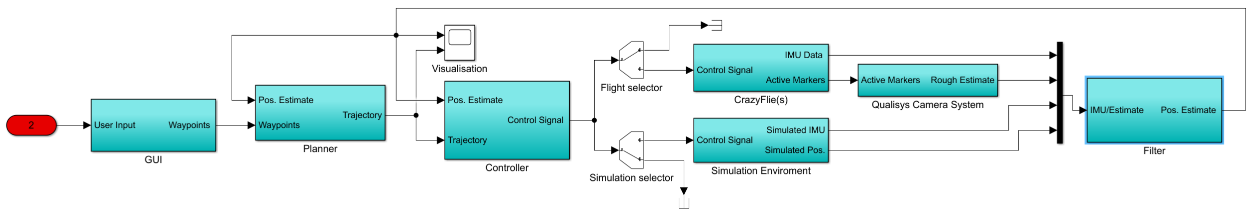

Figure 1: A simple overview of the system with its different components.

2.1 ROS Communication for the System

ROS stands for Robot Operating System and is an open source software development kit for robotics applications. It is a platform that contains a set of software libraries and tools that enables communication between computer and Crazyflie. In ROS the different modules of the system will be divided into nodes. These nodes can then communicate with each other by publishing and subscribing to different topics. The communication for each subsystem will be described in more details at each subsystem. See Table 1 for an overview of the topics

Table 1: ROS Overview

Topic	Publishers	Subscribers
/waypoints	GUI	Planner
/control_signal	Controller	CrazyFlie, Simulation
/position	Filter, Simulation	Planner, Controller
/trajectory	Planner	Simulation, Controller
/qualisys_est	Qualisys	Filter
/imu_est	IMU	Filter

2.2 Hardware

In this section, call and explain the hardware that be used in this project.

- **Crazyflies 2.1 and Crazyflies 2.0:** Crazyflies are nano quadcopters equipped with IMU. The drones only weigh 27 grams and have a battery time of approximately 7 minutes flight. See Figure 2.
- **CrazyRadio:** is a radio-based communication between Crarzflies and the laptop. The radio is connected to the laptop via an USB port. See Figure 3.
- **Laptop:** A Linux laptop with possibility to install software that are needed for the project.
- **Qualisys Camera System:** is a camera system which uses motion capture, used to positioning of the Crazyflies in Visionen. The system can detect active or passive markers to track objects. 20 of these cameras are used in Visionen for tracking. See Figure 4 and Figure 5.

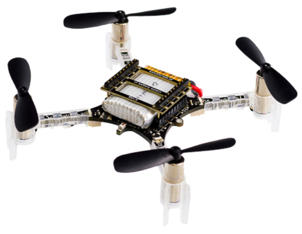

Figure 2: A crazyflie drone, version 2.0.

Figure 3: A Crazyradio, used to communicate with the drones

Figure 4: Qualisys motion capture camera.

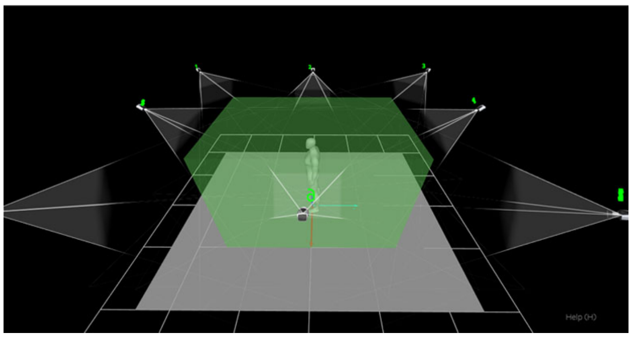

Figure 5: Qualisys motion capture system.

3 DESCRIPTION OF SUBSYSTEMS

This chapter describes the functionality of each subsystem, interaction with other subsystems and other relevant information about each subsystem.

3.1 Graphical User Interface

The GUI gives the user an intuitive way to interact with the drones. The interface itself will not perform any heavy work, its main purpose is to enable different functionalities, which will be described in more details below. The GUI takes the user defined mission as input and sends out a trajectory via ROS to the planner. Also, the simulation environment is activated from the GUI.

3.1.1 *Functionality*

The GUI will have two different frames, the main frame can be seen in Figure 6. The main frame will include six normal buttons, *start simulation*, *stop simulation*, *start automatic*, *emergency stop*, *start manual*, *stop manual*. It will also include three menu buttons, one for creating simple missions, one for loading saved missions and one for the more complicated mission.

In the second frame, the user will be able to create new missions by first choosing the number of drones and then the different waypoints for each path.

3.1.2 *Design*

The GUI design should enable a user-friendly experience, but be primarily focused on functionality. Figure 6 depicts a first sketch of the GUI. It will be based on the GUI used in Crazyswarm with minor changes to better fit this projects needs.

3.1.3 *Communication with other Subsystems*

The Graphical User Interface is responsible for taking user input data and, with it, determining what should be done. The GUI will publish to one topic, the topic in which the waypoints are published. The other buttons will only start different ROS nodes or scripts, see Table 2.

Table 2: GUI Interface

	Topic
Subscribe	
Publish	/waypoints

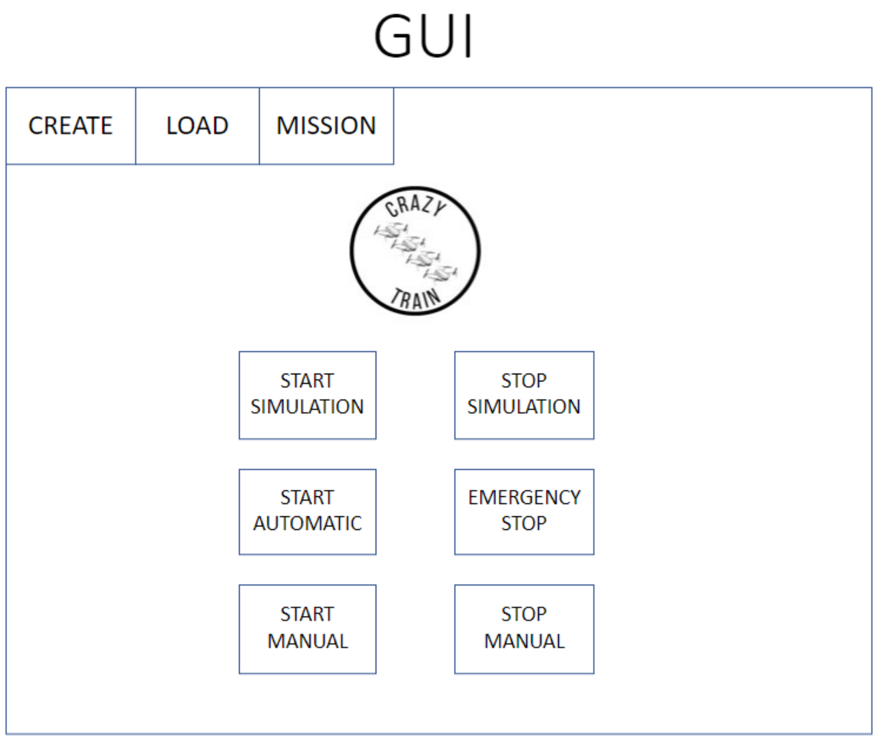

Figure 6: GUI overview

3.2 Simulation Environment

A simulation environment enables testing newly developed code without risking damaging the hardware. It also serves as a way to illustrate the trajectories of the drone(s). The simulation environment takes the control signal as input and passes the simulated states back to the controller via a filter.

3.2.1 *Functionality*

The purpose of the simulation environment is to act as a graphical representation of both the planned trajectory and the estimated position from the sensors and camera systems. Apart from giving the user an improved graphical representation of the system output, the simulation environment is meant to work as a tool for designing the different controllers and planners. By making a realistic environment using motion models for the drones, tests can be carried out in the simulation before it is applied to the actual drones. This creates a safer work approach since the risk of testing faulty programs on the hardware is significantly decreased. To achieve an environment which somewhat represents the real world, the simulation needs to be able to take real control signals as input, which requires a fairly complex model of the drone's motors and thrust as well as the air density and drone inertia.

3.3 Graphical design

The most important part regarding the graphical look of the simulation environment is that important data is presented in such a way that all users recognize it. This means that clarity will be of a higher importance than a "realistic" looking simulation. Possible developments towards more realistic graphics in the simulation will be made under the conditions that the current state of simulation is fully functional and presents relevant data.

3.3.1 *Communication with other Subsystems*

The main functionality in the simulation environment is to take input data from the controller and visualized the planned trajectory, furthermore the simulation environment should be fully accessible from the GUI which requires communication between these subsystems. As mentioned above the simulation environment should create a possibility of testing the controller, filter and planners before applying them to the hardware, to make this possible it is required that the simulated environment can send feedback of drone pose, acceleration and rotation speed to the filter (within the simulation). The filter will then fuse the signal together as an estimated position, which then will be sent to the controller. The simulation environment will then subscribe to the waypoints topic and the control signal topic to be able to solve the above requirements. It will also publish the simulated position to the simulated position topic, see Table 3.

Table 3: Simulation Interface

	Topic
Subscribe	/control_signal, /trajectory
Publish	/position

3.4 Planner

The planner takes in way-points from the GUI and calculates trajectories for the drones to follow through some kind of regulatory planning function. The planner passes set points to the controller, which is used as reference.

3.4.1 *Functionality*

With the help of way-points and information about the initial position of the drones, the planner creates a trajectory that tells the drones where they should be at a certain time. The trajectory, together with estimated position, is used to estimate a positioning error of the drone. This information is used in the controller to steer the drone to minimise the error.

In the case where multiple drones are flying simultaneously, the planner should be able to avoid that different paths cross each other at the same timestamp. Furthermore, the planner will take in signals from Qualisys to avoid collision with static obstacles.

3.4.2 *Communication with other subsystems*

The planner takes input data from the GUI about mission starting point and ending point and calculates a trajectory to send to the controller. Moving objects will affect, and therefore we will have input data about moving targets as position estimates from the filters. The planner will thus subscribe to the position topic and to the waypoints topic. It will publish the trajectory to a trajectory topic.

3.4.3 *Creating a trajectory*

The trajectory creation will be based on a project, uav_trajectories [3]. To create the trajectory, a seven degree polynomial function will be fitted from the waypoints. The function will give the drones smoother trajectories with the seven degrees. The function will have four variables (position in x, y, z and the yaw angle).

For the case when several drones are to be flown in formation, there are several more or less complex solutions. One simple way to achieve a formation is to create a trajectory for a given drone and then translate or rotate the trajectory so that the paths are identical but with an offset. A more complex and more autonomous way of achieving formation flight could be to feed one leader drone with a trajectory to follow, and then set constraints on other drones to follow the leader with a certain distance. The planner would then have to take the position estimates from Qualisys as input and calculate the required actions to achieve the constraints.

An algorithm can be created for the case where multiple drones are flying simultaneously and for static obstacle avoidance. Two algorithms that can be used are artificial potential fields and optimal motion planing.

Artificial potential fields works by adding to forces to calculate the movement of the drone. Repulsive forces are added to the obstacle and repulse the drone from heading on that path, while attractive forces attract the drone to its target [4].

Optimal motion planning is based on minimising a cost function, which is split into two parts. The first part is the distance between the drone and the goal position. This is added to a second part where the difference between the leader drone position and the other drones' position is added with a displacement. The displacement is the desired length between the drones. The first part also have a weight to it that can be tuned. By minimising the function, an optimal path for the drones can be calculated. To avoid crash with other (non drones obstacles) the states of the obstacle is removed from the free states for the drones [4].

3.5 Controller

The controller takes set points from the planner and the filtered position estimate from the sensor system and outputs the control signal which is sent to the Crazyflie(s) or simulation environment. It is responsible for translating the trajectory from the planner to actual driving controls for the Crazyflie(s). This can be used with several controller methods; several of them are described in [5].

3.5.1 *Functionality*

The Controller will steer the drone(s) to follow the path that have been generated by the planner.

3.5.2 *Communication with other subsystems*

The controller-block consist of two identical controllers that take input from planner and with the aid of the feedback from Crazyflie(s) and simulation environment calculate the error value. Depending on how the error vary, the output from the controller to Crazyflie(s) and simulation environment changes. The controller will subscribe to the simulated position topic, the position topic and the trajectory topic. It will publish the control signals to the control signal topic, see Table 4.

Table 4: Controller Interface

	Topic
Subscribe	/position, /trajectory
Publish	/control_signal

3.5.3 *PID*

The Proportional-Integral-Derivative Controller, or simply PID controller, is the most commonly used controller in industrial control systems and other applications. The controller continuously takes the error value e(t) = r(t) - y(t) (reference signal minus measured signal) and applies correction based on the proportional gain (K), and time constants of the integral (T_i) and derivative (T_d) terms.

$$u(t) = K(e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d\frac{de(t)}{dt}) \tag{1}$$

This equation can be described using Laplace transform as

$$U(s) = F(s)E(s) = K(1 + \frac{1}{T_i s} + T_d s)E(s) \tag{2}$$

but we might instead use parallel form

$$F(s) = K(1 + \frac{1}{T_i s} + \frac{T_d s}{\mu T_d s + 1}) \tag{3}$$

with an extra variable μ set to a low value (e.g. 0.1) for continuity reasons in implementation.

For a visual representation of the system, see Figure 7. F is the controller for the system G we want to control. The block diagram represents the closed loop transfer function:

$$G_c = \frac{GF}{1+GF} \tag{4}$$

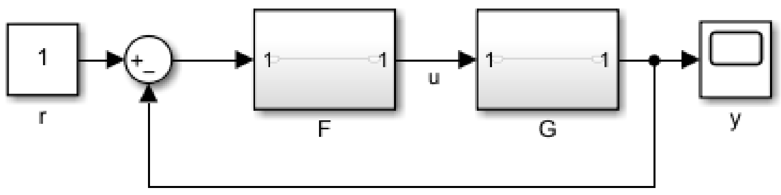

Figure 7: PID controller

In order to get the controller to discrete time, Euler's method will be implemented on the PID controller, which means that one use the following approximations:

$$\int_{t_0}^{t} e(\tau)d\tau \approx T_s(e(t) + e(t-T_s) + e(t-2T_s) + ...) \tag{5}$$

and

$$\frac{de(t)}{dt} \approx \frac{e(t) - e(t-T_s)}{T_s} \tag{6}$$

T_s is sampling time. Equation (5) represents the sum of the historical error and the Equation 6 is the slope between the last two time steps. It is possible to calculate the slope between different time steps by replacing t with $t - nT_s$, where n is the number of time steps back. In a practical implementation, it is recommended to limit the number of calculation by only taking the N last data points in order to decrease the number of calculations and therefore increase performance.

The Time discrete equation of the PID controller shows below.

$$u_k = K(e_k + \frac{T_s}{T_i}\sum_{j=0}^{k} e_j + T_d\frac{e_k - e_{k-1}}{T_s}) \tag{7}$$

This model should be modify in order to consider to saturation of the control signal, an anti-wind-up could be implemented by limiting the integration.

$$I_k = I_{k-1} + K\frac{T_s}{T_i}e_k \tag{8}$$

$$v_k = Ke_k + I_k + K\frac{T_d}{T_S}(e_k - e_{k-1}) \tag{9}$$

$$u_k \begin{cases} u_{max}, & \text{if } v_k > u_{max} \\ v_k, & \text{if } u_{min} \le v_k \le u_{max} \\ u_{min}, & \text{if } v_k < u_{min} \end{cases} \tag{10}$$

$$I_k := I_k + \frac{T_s}{T_t}(u_k - v_k) \tag{11}$$

If the time step T_s is too small, then noise will be amplified greatly. The continuous function for derivation

$$\frac{T_d s}{\mu T_d s + 1} \tag{12}$$

is approximately s when s is small and approximately $\frac{1}{\mu}$ when s is large. This means that small frequencies will follow almost exactly, and high frequencies will get $\frac{1}{\mu}$ gain, i.e. a high pass filter. This will give us good derivation for higher frequencies (smaller sampling time T_s), but high frequency noise will still not be suppressed that well.

In discrete form, this equation will be transformed to:

$$d_k = \frac{\mu T_d}{\mu T_d + T_s} d_{k-1} + \frac{1}{\mu T_d + T_s}(e_k - e_{k-1}) \tag{13}$$

This means we have to modify Equation (9) to the following equation:

$$v_k = K e_k + I_k + K T_d d_k \tag{14}$$

3.5.4 *Cascade*

Cascade control is an extension of PID that is using one slow outer regulator and a fast inner regulator. For a block diagram, see Figure 8. The slow outer regulator takes care of the rough regulation but is normally not good enough, which leaves detail regulation to the fast inner loop, which regulate the final bit to the desired output signal. Normally, both inner and outer loops are described as PID regulators.

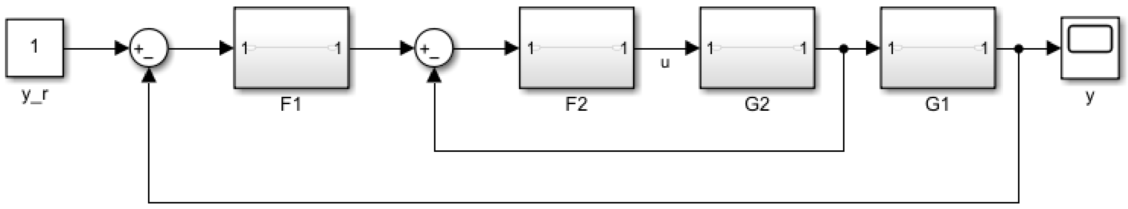

Figure 8: Cascade controller

The inner loop described by F_2 and G_2 can be described by the closed loop transfer function

$$G_{c,2} = \frac{G_2F_2}{1+G_2F_2} \tag{15}$$

This leaves us with the closed loop of the entire system, described by the closed loop transfer function:

$$G_c = \frac{G_1G_{c,2}F_1}{1+G_1G_{c,2}F_1} \tag{16}$$

If the inner loop is much faster than the outer loop, then we see the inner loop as static gain and simply describe the system as:

$$G_{c,1} = \frac{G_1F_1}{1+G_1F_1} \tag{17}$$

3.5.5 *Collision avoidance*

Collision avoidance for multiple moving objects is important for safety reasons, ensuring that no collision occurs. This can be done in many ways, and here is one possible solution.

$\mathbf{r_i}$ is the vector from origin to object i.
$\mathbf{r_j}$ is the vector from origin to object j.
$\mathbf{v_i}$ is the velocity vector for a drone i. Note, negative velocity is velocity in the opposite direction.
$\mathbf{v_j}$ is the velocity vector for a drone j. Note, negative velocity is velocity in the opposite direction.
$\mathbf{u_i}$ is a throttle signal for each axis direction for drone i. Throttle is defined as 1 is full throttle along axis direction, 0 is no throttle and -1 is full throttle opposite axis direction.
d_{safe} is the given safety distance.

Then, for every drone i, calculate the distance to all the other neighbours j and regulate throttle u_i depending on the eight cases that can be possible for each axis. In six of those cases a collision is possible and of those, four cases should be acted upon until the planner has a new route. In the pseudocode, below is one possible implementation of a function returning steering signals for drone i. See Algorithm 1 for one possible implementation in pseudocode.

Algorithm 1 Collision Avoidance

```
    Input: $r_i, r_j, v_i, v_j, u_i, d_{safe}$
    Output: $u_i$
function COLLISION_AVOIDANCE
    $r_{ij} \leftarrow r_j - r_i$                                        ▷ vector from i to j
    $r \leftarrow \sqrt{r_{ij}.x^2 + r_{ij}.y^2 + r_{ij}.z^2}$
    if r < $d_{safe}$ then
        for each axis in [x,y,z] do
            if $r_{ij}.axis > 0$ then
                if $v_i.axis > 0$ AND $v_j.axis > 0$ AND $v_i.axis > v_j.axis$ then
                                                        ▷ i will eventually collide with j →
                                                        ▷ decrease velocity of i
                    $u_i.axis \leftarrow 0$
                else if $v_i.axis > 0$ AND $v_j.axis < 0$ then
                                                        ▷ Immediate collision incoming →
                                  ▷ try to avoid collision by changing direction of throttle for i
                    $u_i.axis \leftarrow -1$
                end if
                                                        ▷ Other cases, do not change $u_i$
            else if $r_{ij}.axis < 0$ then
                if $v_i.axis < 0$ AND $v_j.axis < 0$ AND $v_i.axis < v_j.axis$ then
                                                        ▷ i will eventually collide with j →
                                                        ▷ decrease velocity of i
                    $u_i.axis \leftarrow 0$
                else if $v_i.axis < 0$ AND $v_j.axis > 0$ then
                                                        ▷ Immediate collision incoming →
                                  ▷ try to avoid collision by changing direction of throttle for i
                    $u_i.axis \leftarrow -1$
                end if
                                                        ▷ Other cases, do not change $u_i$
            end if
        end for
    end if
end function
```

3.6 Filter system

The filter takes IMU and Qualisys data as input and passes an estimated position to the planner and controller. The filter system is essential for the position tracking of each drone. The filter is used to fuse different sensory data together in order to get a position estimate as accurate as possible. In this project, the sensor data that will be used is the position estimate provided by the Qualisys motion capture system and the data from the onboard IMU, which senses accelerations and angular rates of the drone.

3.6.1 *Functionality*

The filter system will contain two different input signals. The first will come from the Qualisys system in Visionen and the second will come from the drones IMU. Since the output data from Qualisys is likely rather accurate, the IMU data will mostly be used as backup for cases when the Qualisys signal gets too poor or unreliable. Previous tests have shown that the system in Visionen have various performance in different parts of the room. One way to solve this could be to take the IMU data into account to a greater extent in the poorer areas of the room.

3.6.2 *Qualisys Sensor System*

Qualisys is a motion capture system which will be used for determining the position of the drones within Visionen. Inside Visionen there are 20 Qualisys cameras, which have a latency of 4 ms. The Qualisys system works both with passive and active markers. This feature is advantageous for this project since Crazyflies 2.0 uses passive markers while Crazyflies 2.1 uses active markers [6].

The real-time signals from the tracking will go into the ROS, where it will be filtered with signals from the IMU to give state estimates of the drones.

3.6.3 *Markers*

There are two types of markers that will be used during the project, passive and active.

The passive markers are reflective spheres which the Qualisys system can detect by lighting up Visionen with IR-flashes and then identify the reflection. These markers will be attached to the Crazyflies 2.0.

The active markers work by emitting LED-lights from the drones, which is directly detected by the Qualisys system. These markers will be on Crazyflies 2.1. The active markers use sequential coding of the lights to identify each drone from each other. This means that by assigning the markers individual ID's, the Qualisys system can keep track of each individual drone. Furthermore, the Crazyflie 2.1 version is equipped with four active markers which are mounted on a four-armed deck placed on top of the drone. By assigning each marker individual ID's, the orientation of the drone can easily be identified by the system.

In this project, the Crazyflies 2.1 with the active markers will mostly be used. The Crazyflies 2.0 with passive markers will be used in the beginning since they are cheaper. If an accident occurs in initial testing, it is better to risk a Crazyflie 2.0 than a Crazyflie 2.1.

3.6.4 *IMU Sensors*

Each drone has an inertial measurement unit which includes an accelerometer, gyroscope and magnetometer. These are, together with Qualisys, used to create the estimated states of the system. The states will consist of positioning in the global coordinates and potentially the attitude controls and velocity.

3.6.5 *Filter*

The main filter will be an Extended Kalman filter (EKF), which uses weighted matrices in order to prioritise which data should be taken mostly into account. Possibly, these weights could be dynamic and dependent on the drone's position in the room in order to compensate for the varying performance of the Qualisys data, as mentioned above. Another advantage with fusing these sensors together is to reduce the communication bandwidth by placing the filter onboard the drone, letting the IMU update at a higher frequency than that provided by the Qualisys estimate. By doing this, the drones have a way of roughly estimating their states even though the Qualisys signal is temporarily lost.

The EKF is split into two parts, which are time update and measurement update [7]. The time update will create a prediction of the next movements using a motion model, while the measurement update will update the current estimation with real measurements. In the case of the filter, the IMU will be the time update and Qualisys will be the measurement update. The data from Qualisys and IMU might be pre-filtered before the entire data is filtered with the EKF.

The IMU data comes in the form of acceleration, a_m, and rotational rate, ω_m, and needs to be transformed into state estimates $x = (p, v, q)$ using motion models. This could be achieved according to (18), (19) and (20) which is presented in the Crazyswarm project [8]. Here, q is the unit quaternion which transforms the local coordinates into global and $\Omega(\omega)$ is the quaternion multiplication matrix of ω. $\odot$ denotes quaternion-vector rotation.

$$\dot{p} = v \tag{18}$$

$$\dot{v} = q \cdot \odot a_m - g \tag{19}$$

$$\dot{q} = \frac{1}{2}\Omega(\omega_m)q \tag{20}$$

The Qualisys system has a pose as an output. The pose consist of the position in global Cartesian coordinates, p, and the quaternion rotation, q. The measurement update is written in the form $y = Hx$ and can be described with the three states (p, v, q) as:

$$y = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} p & v & q \end{bmatrix}^T \tag{21}$$

3.6.6 *Communication with other subsystems*

The Filter system takes input data from Qualisys and IMU and export filtered data to the planner and controller. This is done by both subscribing to the Qualisys topic and the IMU topic and publishing to the position topic, see Table 5.

Table 5: Filter Interface

	Topic
Subscribe	/qualisys_est, /imu_est
Publish	/position

4 FURTHER DEVELOPMENT

In every project there is room for further development. One possible extension of the project is to introduce Model Predictive Control in the controller.

4.1 Model Predictive Control

Model Predictive Control (MPC) is an advanced method to control a process while satisfying a set of constraints. The basic idea of MPC is to predict the future behaviour of the controlled system over a limited time horizon in order to calculate the optimal control signal.

$$J_N(x(k)) = \sum_{j=0}^{N-1} ||z(k+j)||^2_{Q_1} + ||u(k+j)||^2_{Q_2} \tag{22}$$

$$z(k+1) = f(z(k), u(k)) \tag{23}$$

In Equation (22) we have a limited prediction horizon N, which is an extra design variable that is around the desired settling time of the system. This gives us a limited amount of control signals u. By calculating the predictions of z using a function (23), we can get an prediction of where the drone(s) will go to in the next N time steps. In addition to this, we might have conditional equations of how close the drones can be to each other and where they can go, etc. Using these equations, we can find an optimal solution by using quadratic programming from optimisation theory. Q_1 and Q_2 are diagonal weight matrices.

This can be expanded upon with by adding a reference signal r:

$$J_N(x(k)) = \sum_{j=0}^{N-1} ||z(k+j) - r(k+j)||^2_{Q_1} + ||u(k+j)||^2_{Q_2} \tag{24}$$

and by integrating the control signal u:

$$J_N(x(k)) = \sum_{j=0}^{N-1} ||z(k+j) - r(k+j)||^2_{Q_1} + ||u(k+j) - u(k+j-1)||^2_{Q_2} \tag{25}$$

REFERENCES

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

[2] Ryman, V., et al. “Crazycrowd documentation.” (2021), [Online]. Available: http://www.isy.liu.se/edu/projekt/tsrt10/2021/visionen/. (accessed: 2022-09-16).

[3] J. Danielsson. “Uav_trajectories.” (2022), [Online]. Available: https://github.com/whoenig/uav_trajectories. (accessed: 2022-09-28).

[4] A. Åsbrink, H. Gunnarsson. “Intelligent drone swarms.” (2022), [Online]. Available: http://www.diva-portal.org/smash/get/diva2:1677350/FULLTEXT01.pdf. accessed: 2022-09-26.

[5] I. f. s. Reglerteknik, *Industriell reglerteknik Kurskompendium*, First. Linköpings universitet, 581 83 Linköping, 2014, www.control.isy.liu.se.

[6] 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).

[7] G. Fredrik, *Statistical Sensor Fusion*, Third. Studentlitteratur AB, Lund, 2018.

[8] 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.