Design Specification

CrazyCircus-Group

December 13, 2023

Version 1.0

Status

Reviewed		
Approved		

CrazyCircus-Group
ellge955@student.liu.se

Project Identity

Group E-mail:	ellge955@student.liu.se
Homepage:	http://www.liu.se/
Orderer:	Anton Kullberg, Reglerteknik/LiU Phone: - E-mail: anton.kullberg@liu.se
Customer:	Daniel Axehill, Reglerteknik/LiU Phone: +46 13 28 40 42 E-mail: daniel.axehill@liu.se
Supervisor:	Joel Nilsson, Reglerteknik/LiU Phone: - E-mail: joel.nilsson@liu.se
Course Responsible:	Daniel Axehill, Reglerteknik/LiU Phone: +46 13 28 40 42 E-mail: daniel.axehill@liu.se

Project participants

Name	**Responsibility**	**E-mail**
Elliot Gestrin	Project manager (PM)	ellge955@student.liu.se
Martin Agebjär	Control technology (CT)	marag492@student.liu.se
Hugo Asplund	Hardware (HW)	hugas433@student.liu.se
Marcus Filipsson	Simulation (SIM)	marfi245@student.liu.se
Alvin Gustavsson Vester	Design (DES)	alvgu648@student.liu.se
Albin Helsing	Testing (TEST)	albhe896@student.liu.se
Tomas Röjder	Software (SW)	tomro614@student.liu.se
Adam Simon	GUI/Information (GUI/I)	adasi503@student.liu.se
Axel Stockhaus	Documentation (DOC)	axest416@student.liu.se

CONTENTS

DOCUMENT HISTORY

Version	Date	Changes made	Made by	Reviewed
0.1	2023-09-29	First version	CrazyCircus-Group	Alvin Gustavsson Vester
0.2	2023-10-03	Second version	CrazyCircus-Group	Alvin Gustavsson Vester
0.3	2023-10-13	Third version	CrazyCircus-Group	Elliot Gestrin
1.0	2023-10-13	Clarified communication between subsystems and extended discussion for motion planning	CrazyCircus-Group	Alvin Gustavsson Vester

1 INTRODUCTION

In this document the design specification for the project Crazyflies by CrazyCircus-Group can be found. The focus of this document will be to describe how the system and subsystems in the project are presumed to be created, implemented and integrated with each other.

1.1 Purpose and Goal

The purpose of the project is to investigate the possibilities of programming drones to perform acrobatic motions. Acrobatic motions can be flips (rotation around a horizontal axis, in place), loops (rotation around a horizontal axis, in a circular trajectory), spins (rotation around vertical axis, in place) or other nontrivially implemented or "awe-inspiring" motions.

The knowledge gained by this project can be used in future educational purposes, and will demonstrate the possibilities of regulating drones to perform specific motions.

1.2 Concept Descriptions

Table 1 and Table 2 presents definitions of terms and symbols used in this document, respectively.

Table 1: Definition of terms

Term	Description
Crazyflie	Crazyflie 2.1 drone developed by Bitzcraze [1].
Crazyradio	CrazyRadio PA developed by Bitzcraze [2].
GUI	Grafical User Interface to interact with the electronics easily.
IMU	Inertial Measurement Unit, combination of accelerometers, gyroscopes.
ROS	Robot Operating System used to build robot applications.
Visionen	Robotics lab at Linköping University.
Qualisys Camera System	A motion capture system inside Visionen, used for positioning of drones [3].
Waypoint	A specific location or point in space that is used in navigation.
Path	A route or course taken by the drone from one waypoint to another.
Trajectory	A predefined path with information of coordinates and angles for the drone to follow.
Flip	Spinning 360 degrees around its own roll and/or pitch axis in mid air.
Loop	Making a 360 degree turn with a given velocity, except it is in the vertical plane instead of the horizontal. Like a flip but in a circular motion.
Acrobatic trick	An acrobatic trick is a visually interesting motion performed by the drone, for example a loop or a flip.
Acrobatic sequence	An acrobatic sequence is a sequence of movements and tricks performed by the drone.
ROS Node / Node	A ROS node or just node is an isolated piece of Python or C++ code able to interact with other nodes and topics.
ROS Topic / Topic	A ROS topic or just topic is a channel where information is sent between ROS nodes. Each topic has a particular format for and information within the messages.

Subscriber / Subscribes	A subscriber is a term for a ROS node which receives information from a ROS topic. When new information is published to the subscriber it automatically performs some operation with this data. A subscriber is said to subscribe to the topic which it receives information from.
Publisher / Publishes	A publisher is a term for a ROS node which sends information over a ROS topic. A publisher is said to publish to the topic which it sends information to.

Table 2: Definition of symbols

Symbol	**Description**
x	Position in x-axis.
y	Position in y-axis.
z	Position in z-axis.
p	The vector $\begin{bmatrix} x & y & z \end{bmatrix}^T$
ψ	Rotation around z-axis (yaw).
θ	Rotation around y'-axis (pitch).
ϕ	Rotation around x''-axis (roll).
α	The vector $\begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^T$
ω	Angular velocity vector ($\dot{\alpha}$).
$\dot{a}$	Time derivative of a variable a.
a_x / a_y / a_z	Component of a vector a in the x / y / z -axis.

2 OVERVIEW OF THE SYSTEM

For a drone to successfully perform acrobatic tricks, a system capable of monitoring the drone, determining actions and performing them is needed. This makes a complex system, but the entire system does not have to reside within the drone. The drone and its movements can be monitored by external cameras, the Qualisys system. The calculations needed to determine actions can be computed with a computer, though the actions must of course be executed by the drone itself. The entire system can therefore be separated into multiple subsystems, where each subsystem is responsible for a piece of the process, and that communicates its information with the other subsystems. The system is divided into the following subsystems:

- Graphical User Interface
- Simulation Environment
- Control System
- Sensor System
  - Qualisys
  - On Board Sensor System
- Motion Planning System

2.1 Communication

The subsystems will have to communicate with each other. Some subsystems are quite different from each other and live on different hardware. Therefore, different parts of the code for the communication can be implemented in different languages (e.g. C/C++ or Python) depending on where it lies, and which subsystem that part is communicating with. For example, communication with the control system will be on the Crazyflie and will be implemented in C/C++, while communication with the simulation environment will be on the computer and likely be implemented in Python. In the later case, the API for Robot Operating System 2 (ROS2) is used. ROS2 is the framework which mainly will be used to send information between subsystems that are not located at the Crazyflie itself. The flow of information between the subsystems is visualized in Figure 1.

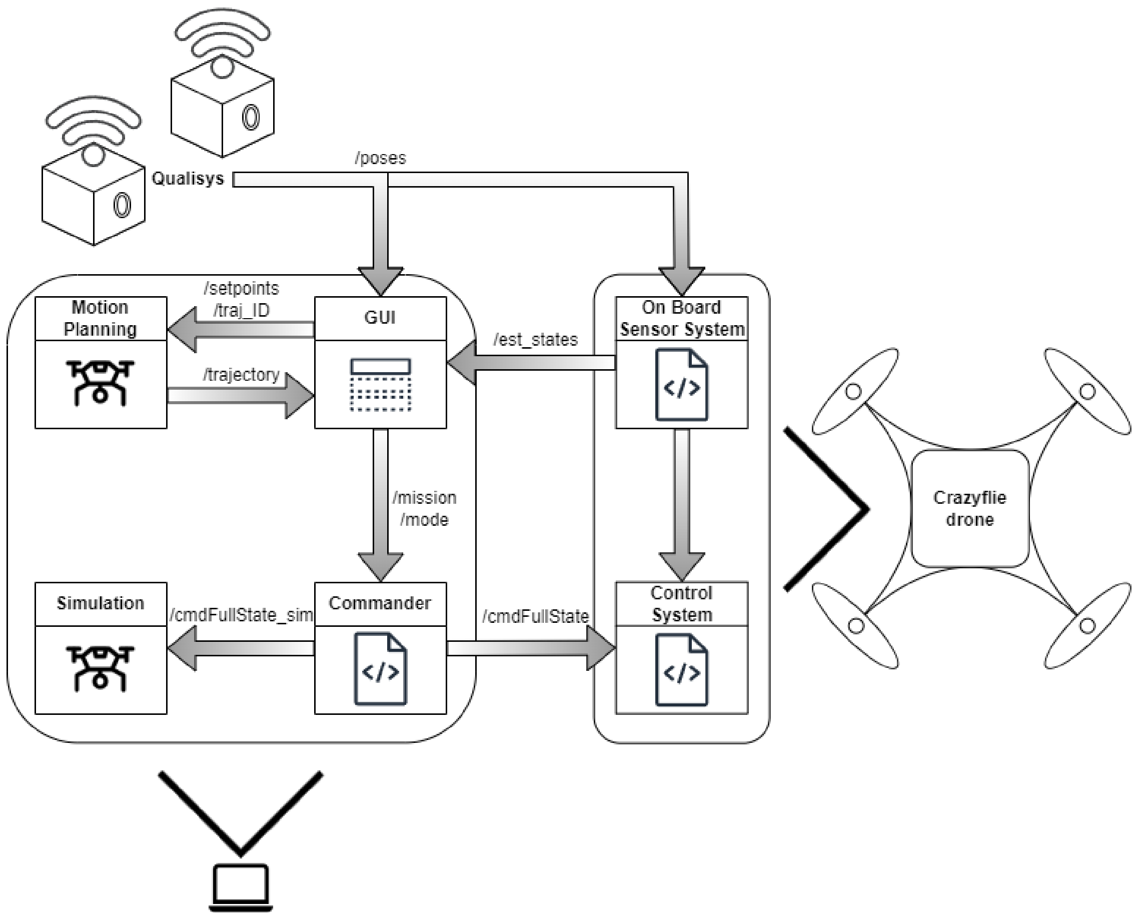

Figure 1: The flow of information between subsystems.

2.1.1 *Interface between subsystems*

This subsection defines the interface between subsystems. When a new acrobatic sequence or movement is selected in the GUI, information about the set points needs to be transferred between the GUI and the motion planner. This transfer of information is specified in Table 3.

Table 3: Package for set point

Set Point											
pos			angular pos			vel			angular vel		
x	y	z	ϕ	θ	ψ	$\dot{x}$	$\dot{y}$	$\dot{z}$	$\dot{\phi}$	$\dot{\theta}$	$\dot{\psi}$

This information is saved for each set point created in the the GUI as well as options needed for the planning, such as look-ahead horizon and sample time. This is illustrated in Table 4 below.

Table 4: Package to Motion planner

Set Point Package				
Metadata	Setpoints			
rate	sp_1	sp_2	$\dots$	sp_n

In response, the motion planner provides a sequence of full states that constitute the planned trajectory. This trajectory is saved as a CSV file. Since the motion planner plans inputs for the entire sequence, a time variable is also required. The structure of a single package can be seen in Table 5.

Table 5: Package from Motion planner

Full state, at time instance t																
time	position			orientation				vel			angular vel			acc		
t	x	y	z	q_1	q_2	q_3	q_4	$\dot{x}$	$\dot{y}$	$\dot{z}$	$\dot{\phi}$	$\dot{\theta}$	$\dot{\psi}$	$\ddot{x}$	$\ddot{y}$	$\ddot{z}$

A series of packages, encompassing all full states, will be stored as a CSV file. A specific trajectory with a unique ID can be loaded into the GUI by requesting that particular trajectory ID. The format of the trajectory ID is defined in Table 6.

Table 6: Request to Motion Planner

Trajectory ID
trajectory_ID

To execute a movement, the file created by the motion planner will be selected and sent as an input to a ROS node. This will then stream the information for the required duration to the Crazyflie (more exactly to the control system). Each contains the full-state package and has the form described in Table 7. Additionally, various operational modes, such as manual operation, simulation, or autonomous operation, can be selected based on the modes defined in Table 8.

Table 7: Package of full state to CF

Full state															
position			orientation				vel			angular vel			acc		
x	y	z	q_1	q_2	q_3	q_4	$\dot{x}$	$\dot{y}$	$\dot{z}$	$\dot{\phi}$	$\dot{\theta}$	$\dot{\psi}$	$\ddot{x}$	$\ddot{y}$	$\ddot{z}$

Table 8: Content of Operation Modes package

Operation Modes	
Stop/Run	*Simulation/Drone*

During operation, data from Qualisys will be streamed by a ROS node to the GUI and to the Crazyflie (more exactly to the on-board sensor system). The data is sent according to Table 9. The fused data (see Section 4.5.2) is then sent to the control system, according to Table 10.

Table 9: Package of observed positions

pos			angular pos		
x	y	z	ϕ	θ	ψ

Table 10: Package of fused estimate of state from CF

pos			vel			angular pos			angular vel		
x	y	z	$\dot{x}$	$\dot{y}$	$\dot{z}$	ϕ	θ	ψ	$\dot{\phi}$	$\dot{\theta}$	$\dot{\psi}$

2.2 Hardware

The included hardware in the project is:

- 4 Crazyflie 2.1 (Figure 2)
- 1 Crazyradio 2.0 (Figure 3)
- Qualisys motion capture system (a camera is shown in Figure 4)
- Portable computer running Ubuntu 22.04 LTS and ROS2

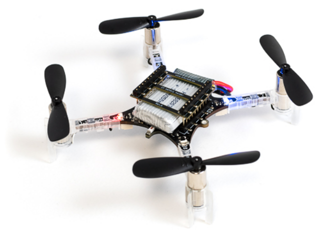

Figure 2: Crazyflie 2.1 [1].

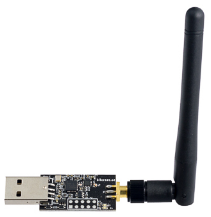

Figure 3: Crazyradio 2.0 [2].

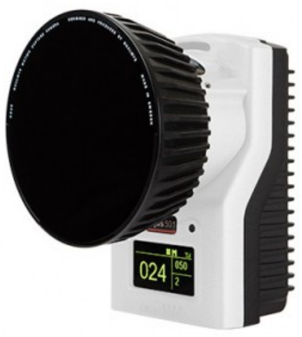

Figure 4: Qualisys motion capture camera [3].

3 MODELLING OF THE CRAZYFLIE DRONE

To successfully do acrobatics with a Crazyflie drone, it is important to have a good mathematical model of the Crazyflie. The model is the foundation for both simulation and control. In this section dynamic models of the Crazyflie, and methods of validating these models, will be presented.

3.1 Angle Representation

There are plenty of options when choosing a dynamic model of a vehicle, and one aspect of it is that there are many ways to represent three-dimensional position and rotation of a rigid body. To represent position, the obvious choice is three-dimensional coordinate systems. We can define a global coordinate system, consisting of three orthogonal (right-handed) axes x, y and z. We can also define a body fixed coordinate system, consisting of three orthogonal (right-handed) axes X, Y and Z. Origo, O, of the body fixed coordinate system is set as the body's center of mass.

3.1.1 *Euler Angles*

The orientation of the body can be represented by Euler angles, more exactly by Tait-Bryan angles or Proper Euler angles. In both cases, the complete rotation of the body is described by three sequential rotations around some axes. For Proper Euler angles, the first and third rotation is around the same axis (which can not be the same axis as for the second rotation). For Tait-Bryan angles, each of the three rotations are around a different axis. The rotations can be intrinsic or extrinsic. Intrinsic rotations are rotations around axes in the body-fixed coordinate system XYZ, and extrinsic rotations are around the fixed global coordinate system xyz. Figure 5 show an example of intrinsic Proper Euler angles.

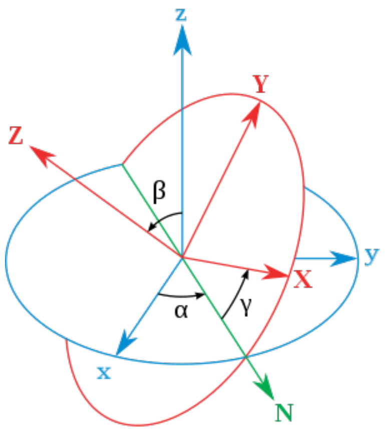

Figure 5: Intrinsic Proper Euler angles z-x'-z''. First, a rotation α around the local (which for the first rotation coincides with the global) z-axis is made. Then a rotation β around the new temporary local x-axis (denoted N in the figure) is made. Lastly a rotation γ around the local z-axis is made.

A good choice of the angles above for an aerial vehicle is the intrinsic Tait-Bryan angles z-y'-x'', meaning the first rotation *yaw* is around the z-axis, the second rotation *pitch* is around the y-axis and the third rotation *roll* is around the x-axis. This way of representing the orientation of a rigid body will be used in the project.

3.1.2 *Quaternions*

Another way to represent the orientation of a rigid body is with quaternions. Quaternions are an extension of imaginary numbers to four dimensions. The subset consisting of quaternions of length one (four dimensions reduced to three) can be used to represent orientation in three dimensions. The representation is not as intuitive as Euler angles, but has its benefits. Computations using quaternions are cheaper than computations using rotation matrices with Euler angles. Also, in difference to the Euler angle representation, the quaternion representation does not suffer from Gimbal lock. Therefore the quaternion representation might be used in the project.

3.2 State Space Model

The following model is based on existing work [4][5]. The external force of gravity on the drone is given by

$$F_g = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} \tag{1}$$

where F_g is a force vector in the negative z-direction, m is the weight of the drone and g is the gravity acceleration. The upward thrust force in Newtons generated by each propeller is given by

$$f_i = C_T\omega_i^2, i \in \{1,2,3,4\} \tag{2}$$

where C_T is a thrust coefficient and ω_i is the rotation speed of the i-th motor in revolutions per minute. Assuming the drone is in an upright position, one can sum up the thrust generated from all four propellers, f_i, to form a vector F_{tot} in the Z-axis,

$$F_{tot} = \begin{bmatrix} 0 \\ 0 \\ \sum_{i=1}^{4} f_i \end{bmatrix}. \tag{3}$$

However the thrust generated from the propellers are not always in z-axis. The angles of the drone needs to be accounted for. The transformation matrix that connects the moving frame to the fixed frame can be found in three steps. First, a rotation around the z-axis by an angle ψ, followed by a rotation around the y-axis by an angle θ, and finally, a rotation around the x-axis by an angle ϕ. These rotations are defined as three rotation matrices,

$$R_z(\psi) = \begin{bmatrix} cos\psi & sin\psi & 0 \\ -sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}, \; R_y(\theta) = \begin{bmatrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ -sin\theta & 0 & cos\theta \end{bmatrix}, \; R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi & sin\phi \\ 0 & -sin\phi & cos\phi \end{bmatrix}. \tag{4}$$

The resulting transformation matrix R is defined as

$$R = R_z(\psi) \times R_y(\theta) \times R_x(\phi). \tag{5}$$

Conclusively equations 1, 2, 3 and 5 can be combined using Euler's first law, which results in

$$m\ddot{p} = RF_{tot} + F_g, \tag{6}$$

where $\ddot{p}$ is the drones acceleration vector in the global coordinate system. The drones translation has now been dealt with and the next stage is calculating the angular acceleration for the different angles. The momentum of the drone can be represented as

$$M = \begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix} = \begin{bmatrix} \frac{d}{\sqrt{2}} C_T \left(-\omega_1^2 - \omega_2^2 + \omega_3^2 + \omega_4^2\right) \\ \frac{d}{\sqrt{2}} C_T \left(-\omega_1^2 + \omega_2^2 + \omega_3^2 - \omega_4^2\right) \\ C_D \left(-\omega_1^2 + \omega_2^2 - \omega_3^2 + \omega_4^2\right) \end{bmatrix}, \tag{7}$$

where M is the resulting momentum vector, d denotes the distance from the drones center of gravity to the center of each motor and C_D denotes the aerodynamic drag coefficient. The inertia matrix for the drone can be represented and hopefully approximated as

$$I = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix} \approx \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}, \tag{8}$$

this matrix is in local coordinates and needs to be converted to global coordinates. The relation can be described as

$$W = \begin{bmatrix} 1 & 0 & -sin(\theta) \\ 0 & cos(\phi) & cos(\theta)sin(\phi) \\ 0 & -sin(\phi) & cos(\theta)cos(\phi) \end{bmatrix}. \tag{9}$$

The final the inertia matrix can now be represented as

$$J = W^T \times I \times W, \tag{10}$$

and the Coriolis matrix as

$$C = \frac{d}{dt} J - \frac{1}{2} \frac{d}{d\alpha} \left(\alpha^T \cdot J\right), \quad \alpha = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}. \tag{11}$$

The angular acceleration can then be calculated using Euler's second law and equation 7, 10 and 11,

$$J \cdot \ddot{\alpha} = (M - C \cdot \dot{\alpha}). \tag{12}$$

Lastly the state space vector $x = \begin{bmatrix} p & \alpha & \dot{p} & \dot{\alpha} \end{bmatrix}^T$, can be constructed and updated using our known values $\dot{p}$ and $\dot{\alpha}$ joint with the calculated values $\ddot{p}$ and $\ddot{\alpha}$. The non linear state space vector is portrayed in equation 13 below,

$$\dot{X} = \begin{bmatrix} \dot{p} \\ \dot{\alpha} \\ \ddot{p} \\ \ddot{\alpha} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ \frac{RF_{tot}}{m} + \frac{F_g}{m} \\ J^{-1} \cdot (M - C \cdot \dot{\alpha}) \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ \frac{C_T \left(sin(\phi)sin(\psi) + cos(\phi)cos(\psi)sin(\theta)\right) \cdot (\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2))}{m} \\ \frac{C_T \left(cos(\psi)sin(\phi) - cos(\phi)sin(\psi)sin(\theta)\right) \cdot (\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2))}{m} \\ \frac{-gm + C_T cos(\phi)cos(\theta) \cdot (\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2))}{m} \\ J^{-1} \cdot (M - C \cdot \dot{\alpha}) \end{bmatrix}. \tag{13}$$

The resulting state space vector is in continuous form and needs to be discretized for application to a real system. There are several discretization methods that can be used, such as zero-order-hold (ZOH), forward Euler or Runge-Kutta. For this application the Runge-Kutta method will be used since it offers a trade-off between accuracy and computational complexity, which allows for more precise solutions compared to simpler methods like Euler's and ZOH.

3.3 Parameter Estimation and Model Validation

Given a physical model of the Crazyflie with parameters Θ, there are multiple ways to determine these parameters. One way is to perform physical experiments where the parameters are measured. Such experiments can be to directly measure the weight of the Crazyflie, or to indirectly measure the components of the inertia matrix by rotating the body around certain angles by applying certain momentum. This has already been done in other studies of the Crazyflie, and the hope is to be able to apply these parameters to our model.

Another way to determine the parameters is by collecting data of direct use of the system. That is, by collecting data of the state of the Crazyflie while it is operating. The hope is then to be able to estimate the parameters using machine learning techniques. The challenge with this method is instead to properly excite the system, and for the Crazyflie model this would mean that experiments containing fast movements and flips would have to be designed. Since the plan is to use existing parameters, these experiments can hopefully be avoided.

Although the plan is to use parameters from other papers, it is still important to validate the obtained model. This is especially true since the parameters might have to be tweaked a little, since the Crazyflies in the project have active markers-shields on them and therefore should have somewhat different dynamics than just the Crazyflies themselves. The data driven method described above might be good enough for validation of the model. One idea is to let the Crazyflie fly around, giving it known inputs. For example by operating it manually. The states of the Crazyflie during operation, $\bar{X}(t)$, is to be measured. The Crazyflie is then simulated for the known input and the parameters. The simulated states, $\tilde{X}(t;\Theta)$, can then be compared to the measured states, the goal being to get the simulated states as close to the measured states as possible. In other words, the goal is to get the error $\bar{e}(t;\Theta) = \tilde{X}(t;\Theta) - \bar{X}(t)$ close to zero for all t, or to minimize the cost function

$$I(\Theta) = \int_{t_0}^{t_f} \bar{e}(t;\Theta)^2 \, dt = \int_{t_0}^{t_f} (\tilde{X}(t;\Theta) - \bar{X}(t))^2 \, dt \tag{14}$$

where t_0 and t_f is the start time and stop (final) time measuring/simulating. In practise, both measured and simulated states are given in discrete time t_k, $k = 1, 2, 3..., n$, where n is the number of measurements (and simulated states). The cost function becomes

$$J(\Theta) = \sum_{k=1}^{n} \bar{e}(t_k;\Theta)^2 = \sum_{k=1}^{n} (\tilde{X}(t_k;\Theta) - \bar{X}(t_k))^2. \tag{15}$$

When using this method to find better parameters, this function is to be minimized with respect to the parameters Θ. If it is possible to find the gradient of J with respect to Θ, gradient-search can be used. The method iteratively search for more exact parameters. The evaluation of $J(\Theta)$ can serve as a measurement of the accuracy of the simulation and the parameters Θ. This is the primary intent of the method in this case, to validate the parameters. Other loss functions can be investigated when necessary.

4 DESCRIPTION OF SUBSYSTEMS

The following section describes the different subsystems and the implementation plan.

4.1 Graphical User Interface

The Graphical User Interface (GUI) will be where the user controls the Crazyflie drone. Everything from start, take off and selecting tricks to landing and shutdown will be done via the GUI. More about its functionality can be read below. The design of the GUI is meant to be user-friendly, however the priority is functionality. The GUI will be implemented in Rviz, a 3D visualization tool for ROS2.

4.1.1 *Functionality*

The GUI will have two different frames, one for autonomous mode and one for manual mode. It will be possible to switch between them in a tab in the top left corner of the GUI. There will also be a third tab named "File" where it will be possible to save and load acrobatic sequences. The manual mode frame will look and work similar to the Crazyflie client, where for example the orientation, thrust, battery status and flight data are shown. The autonomous mode frame will mostly be used to start an acrobatic sequence with a button. There will also be a button to do an emergency landing as well as a button to do an emergency stop, shutting the drone off completely. There will also be a button to switch simulation on and off. Starting an acrobatic sequence with the corresponding button together with simulation being switched on will start the simulation. Lastly there will also be some kind of indicator of the state for the Crazyflie.

4.1.2 *Communication with other subsystems*

The GUI serves as the primary interface between the user and the system, responsible for capturing user input and translating it into actions. Through the GUI, the user can select set points, i.e. states, that the drone should pass through in its final trajectory. These set points are published on /setpoints and received by the Motion Planning system.

By submit a requested trajectory ID on the topic /traj_ID, the GUI obtains a generated trajectory from the Motion Planning module on the /trajectory topic. Once a trajectory is received, the GUI can visualize the planned trajectory.

The GUI is also responsible for initiating an additional ROS node called Commander. Commander is an auxiliary node used when the GUI sends a mission to either the Simulation Environment or the drone. The GUI initiates a mission by simultaneously publishing a stream of full states on the /mission topic and publishing the desired mode on the /mode topic. The Commander subscribes to both of these topics and processes the stream of full states differently depending on the selected mode. The Commander can, for example, choose operational modes such as manual operation, simulation, or autonomous operation. If simulation is selected, based on the chosen mode, the Commander forwards the stream of full states by publishing them on the topic /cmdFullState_sim. If autonomous trajectory following is chosen, the stream of full states is published on the topic /cmdFullState instead.

During the flight, the GUI also subscribes to the /poses and /est_states topic, which displays the current measured position by Qualisys and the current state estimation by the drone. This data is used for visualization. The system is also responsible for starting ROS nodes that publish measured positions for the drone's control system, and Qualisys will continuously stream position measurements to the drone until the mission is completed. The ROS topics and services can be seen in Table 11.

Table 11: GUI interface

	Topic/Service	**Format**
Subscribe	/trajectory	Table 5
Subscribe	/poses	Table 9
Subscribe	/est_states	Table 10
Publish	/setpoints	Table 4
Publish	/traj_ID	Table 6
Publish	/mission	Table 7
Publish	/mode	Table 8

4.1.3 *Design*

A simple design of what the GUI can possibly look like can be seen in Figure 6.

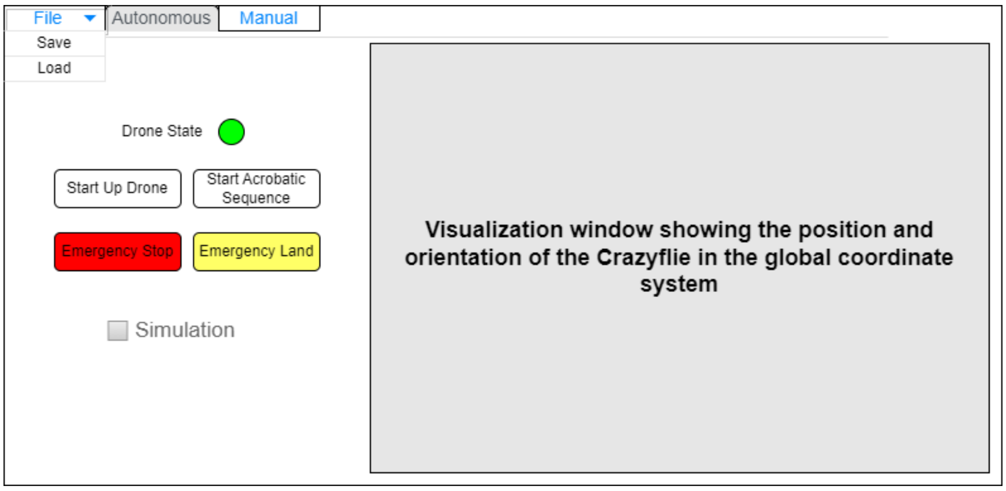

Figure 6: Design of the Graphical User Interface.

4.2 Simulation Environment

The simulation for the Crazyflie drone will run in the Gazebo simulation environment. The simulation will be used to evaluate all the subsystems of the drone, such as the controller, before being deployed on actual hardware. Gazebo will be used in conjunction with ROS2. This enables the same software to be used in the simulation as in real life, so called software in the loop. To interact with the Gazebo simulation python scripts will be used together with provided API's. The controller used will be identical to the one used on the CF.

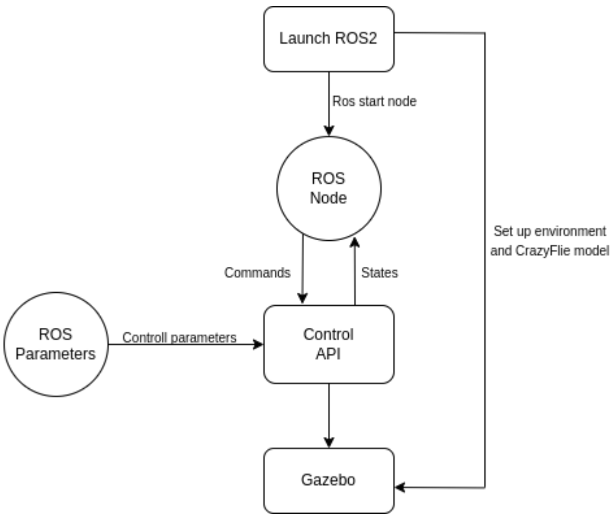

Figure 7: Simulation system overview.

Figure 7 illustrates the simulation system in its highest level. This is to show the planned interaction between ROS2 and Gazebo.

4.2.1 *Communication with other subsystems*

The simulation environment will subscribe to the topic /cmdFullState_sim and, as a result, receive the trajectory as a stream of full states. The simulation will use the same controller as the drone and the same target setpoints. The ROS topic is listed in Table 12.

Table 12: Simulation interface

	Topic	**Format**
Subscribe	/cmdFullState_sim	Table 7

4.3 Motion Planning System

By modeling the Crazyflies' physical characteristics, feasible trajectories can be calculated, along with corresponding control signals. Substantial work has been done regarding aggressive trajectory planning for Crazyflies [4][5]. Although the existing work has to be investigated further, some general things regarding the modeling of physical characteristics are worth mentioning, see Section 3.2.

To simulate an acrobatic trajectory, such as a loop, a few target states are chosen along the trajectory. For example, target states are chosen target in a sequence as follows, $[x, y, z, \phi, \theta, \psi, \dot{p}, \dot{\alpha}] = [3, 0, 2, 0, 0, 0, \dot{p}, \dot{\alpha}]$, $[2, 0, 3, \pi, 0, 0, \dot{p}, \dot{\alpha}]$, $[2, 0, 1, 2\pi, 0, 0, \dot{p}, \dot{\alpha}]$, where $\dot{p}$ and $\dot{\alpha}$ are what they need to be in order to achieve the desired positions and angles. Subsequently, there is an optimization problem intended to result in a trajectory passing through the target states mentioned above, with a chosen tolerance of accuracy. The final trajectory should be smooth between the target states and achievable based on the drone's mechanical characteristics as stated in Equation (13). The optimization problem is divided into multiple sub-optimization problems, each corresponding to a target state. Each sub-problem optimizes the trajectory from one target state to the next target state in the sequence. The sub-optimization problem aims to minimize the cost function,

$$\begin{aligned} &\sum_{\forall k} \left[(X_k - X_{target})^T Q (X_k - X_{target}) + (U_k^T - U_{hover}) R (U_k - U_{hover}) \right] \\ &\text{subject to } X_{k+1} = f(X_k,\, U_k), \\ &\qquad\qquad X_k \in X_c,\ \forall k \\ &\qquad\qquad U_k \in U_c,\ \forall k \end{aligned} \tag{16}$$

where k represents the time instances within the sub-optimization problem, X_k denotes the drone's state at time instance k, X_{target} is the target state specific to the sub-optimization problem, U_k represents the control signals applied to the drone, U_{hover} is the calculated control signals required for hovering, Q and R are penalty matrices that assign different penalties to different drone states and control inputs respectively, f represents the state transition function that describes how the drone's state evolves from one time instance to the next and X_c and U_c are sets of constraints for the states and the control input. When an achievable trajectory is calculated, all states and control signals are returned in every time instance.

However, there are certain aspects of the optimization problem stated in Equation (16) that should be discussed. In the context of aggressive trajectory planning for Crazyflies, the primary objective is not necessarily to conserve energy, as implied by the second part of the cost function containing the penalty matrix R. While matrix R might encourage smoother, more energy-efficient maneuvers, it's worth exploring whether the penalties on control inputs should be reduced or set to zero to grant the system more freedom to execute quicker and more aggressive movements.

It's also important to note that the optimization problem is approached sequentially. Each sub-optimization problem aims to optimize the trajectory between two consecutive target states. However, this approach introduces a risk of the drone ending up in unfavorable states due to a lack of foresight. As previously mentioned, $\dot{p}$ and $\dot{\alpha}$ are set to achieve the desired positions and angles. For instance, the velocity may be optimal for reaching one target state, but not suitable for reaching the subsequent target state. If the optimization problem were solved as a single global optimization problem, unfavorable states could be eliminated by considering the entire sequence of maneuvers. This approach might allow the drone to perform even more challenging maneuvers and optimize the trajectory as a whole. Nevertheless, adopting the global approach comes with potential challenges, such as a greater computational burden or the risk of encountering such a complex optimization problem that no feasible solution can be found. The optimization problem needs further investigation within the project.

4.3.1 *Communication with other subsystems*

Motion planning communicates with the GUI trough ROS. Motion planning response to the service /setpoints and, upon a request from the GUI (i.e., when the GUI request on /setpoints), a trajectory is generated. The trajectory is based on a number of setpoints and metadata as specified in Table 4.

Once a trajectory is generated, it is saved to a CSV file with a unique ID, containing the entire trajectory, including all full states. This CSV file is stored in a specified database/directory. Each row in the CSV file corresponds to a package with full states as specified in Table 5. The GUI can request the completed trajectory by request on the topic traj_ID, which triggers the corresponding trajectory to be published on the topic /trajectory. By request on /trajectory, the GUI can access the completed trajectory. The ROS services are listed in Table 13.

Table 13: Motion planing interface

	Service	**Format**
Subscribe	/setpoint	Table 4
Subscribe	/traj_ID	Table 6
Publish	/trajectory	Table 5

4.4 Control System

The control system will regulate the Crazyflie drone to follow predefined trajectories and be in certain states, in terms of position and/or velocities.

4.4.1 *On board controller*

The default controller on the Crazyflie is a cascaded Proportional-Integral-Derivative (PID) controller. Four types of states are measured and used in the control: position, velocity, attitude, and attitude rate. These are controlled in cascade, where the innermost and fastest loop controls the attitude rates, resulting in the desired thrusts for roll, pitch, yaw, and height. These thrusts are handled by the power distribution of the motors, as shown in Figure 8.

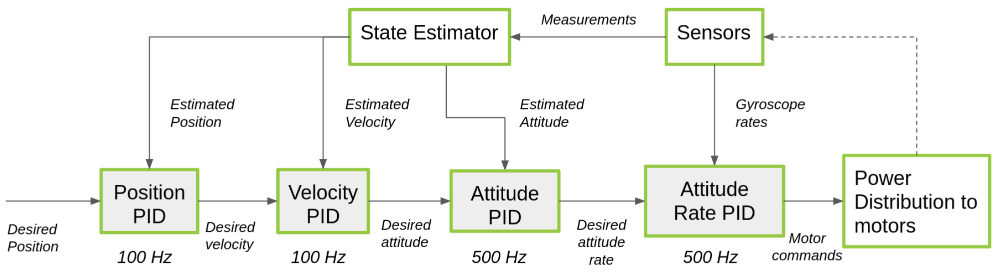

Figure 8: Cascaded PID controller.

Desired state setpoints will be sent to the onboard controller. Different types of setpoints can be sent, entering different parts of the cascaded PID. One can set a high-level setpoint which plans a smooth trajectory from the current state to the goal position. The goal position will be sent to the position PID as the desired position. One can also send a streaming full-state setpoint. The full-state setpoint is particularly feasible for aggressive maneuvers where feedforward inputs for acceleration and angular velocity play a vital role in achieving precise tracking performance. Full-state setpoints can be derived from trajectory parameterizations obtained from piecewise polynomials.

The goal is to take the states returned from the motion planning system described in section 4.3 and format them to be sent as a streaming full-state setpoint to the cascaded PID controller. This will enable the controller to handle drone control during acrobatic maneuvers.

4.4.2 *Communication with other subsystems*

The controller requires knowledge of the current position and the target state to calculate a control signal. During real-time flight, the controller runs on the Crazyflie and receives its current measured state directly from the on board sensor system, see Section 4.5.2.

While following a trajectory, the control system also receives a continuous stream of full states by subscribing to the topic /cmdFullState. This stream provides the current reference position. The ROS topics can be found in Table 14.

Table 14: Controller interface

	Topic	**Format**
Subscribe	/cmdFullState	Table 7

4.5 Sensor System

The sensor system is split into two parts, the Qualisys system (the outer sensor system) and the on board sensor system (the inner sensor system).

4.5.1 *Qualisys*

The system that will be used for measuring the position of the drone is a Qualisys Motion Capture system. The Qualisys system used in Visionen uses 20 Qualisys IR cameras. The Crazyflie is equipped with active IR markers which emit LED light. These markers are designed to be detected by the Qualisys cameras and the Crazyflie can, if needed, also be equipped with passive markers. The passive markers work by reflecting IR flashes from the cameras which can then be detected in order to estimate position and orientation. The Qualisys system is capable of capturing low-latency 6DOF positioning data in real-time to estimate the state of the Crazyflie. The data will be streamed in real time using the Qualisys Track Manger software and published on the topic /poses. The measured position is received by the GUI as well as the on board control system on the Crazyflie by subscribing on /poses. There are two ways to obtain the pose from the Qualisys. One with a rotation matrix and the other one that represents the orientation as Euler angles. Both have their advantages and disadvantages. Euler angles are intuitive and provides uncomplicated representation of orientation, however they have problem with singularity. Rotation matrices are a better choice for computational purposes since it does not have problems with singularity.

The Qualisys-node will publish its measured positions to a ROS topic according to Table 15.

Table 15: Qualisys interface

	Topic	Format
Subscribe	/poses	Table 9

4.5.2 *On Board Sensor System*

The on board sensor system will fetch measurements from the Inertial Measurement Unit (IMU) located on the Crazyflie. It will fuse the received position estimate (acquired from Qualisys through /poses) with the sensor data from the IMU to make a final estimate of its state. It will then send that estimate directly to the control system, and to the GUI for visualization by submitting on the topic /est_states with 1 Hz. The ROS topics can be found in Table 16.

Table 16: On board sensor system interface

	Topic	Format
Subscribe	/poses	Table 9
Publish	/est_states	Table 10

4.5.3 *Sensor Fusion*

Qualisys is an exceptionally accurate and reliable system, and therefore the plan is to prioritize, or have greater trust, for these estimates in the sensor fusion. However, in situations where the data from Qualisys is likely to be flawed the IMU data will be prioritized. This can for example be when the Crazyflie is upside down, since the Qualisys cameras are placed high and the markers could be out of view from the cameras. The filter that will be responsible for the fusion of the IMU data and the state estimation from the Qualisys system is an Extended Kalman filter. The extended Kalman filter is an improved version of the traditional Kalman filter that can handle non-linear filtering. The extended Kalman filter linearize non-linear functions using taylor series around the mean. The IMU data consists of acceleration and angular velocity which will be used together with a dynamic model of the drone to get estimated position and orientation which can then be fused with the measurements from the Qualisys system. The procedure is shown in Figure 9.

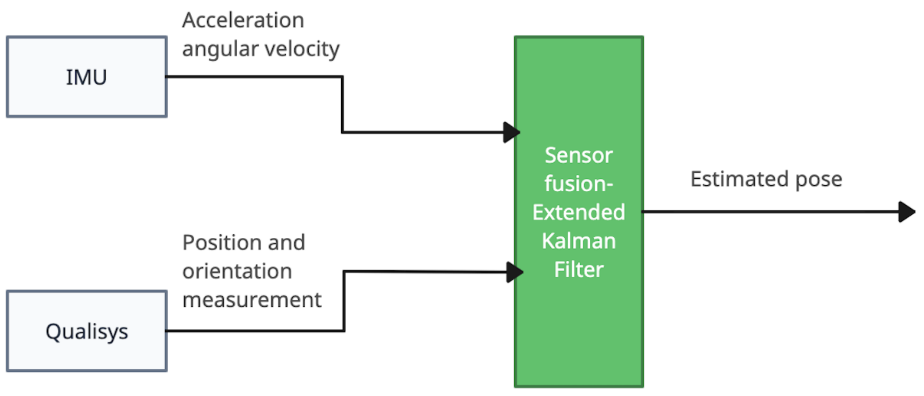
IMU
Acceleration angular velocity
Qualisys
Position and orientation measurement
Sensor fusion-Extended Kalman Filter
Estimated pose

Figure 9: The sensor fusion process.

5 FURTHER DEVELOPMENT

Since the project focuses on one single Crazyflie drone, further development could be made by extending the project with several drones. It could for example be multiple drones doing synchronized flips and loops as part of an acrobatic performance.

REFERENCES

[1] Bitcraze, “Crazyflie 2.1,” https://www.bitcraze.io/products/crazyflie-2-1/, 2023, [Online; accessed September 12, 2023].

[2] ——, “Crazyradio pa,” https://www.bitcraze.io/products/crazyradio-pa/, 2023, [Online; accessed September 12, 2023].

[3] Qualisys, “Motion capture camera for mri scanners,” https://www.qualisys.com/cameras/oqus-mri/, 2023, [Online; accessed September 20, 2023].

[4] O. Ljungberg, “Crazyflie accrobatics,” https://gitlab.liu.se/olilj86/crazyflie_accrobatics, 2023, [Online; accessed September 14, 2023].

[5] C. Luis, “Design of a trajectory tracking ccontroller for a nanoquadcopter,” 2016, [Online; accessed October 4, 2023].