Lund University
Automatic Control

FRTN70 Project in Systems, Control and Learning
Spring 2024

Formation Control and Coordination of Crazyflie UAVs in Dynamic Environments

Project Plan Crazyflie Group 2

Maria Tavemark[1] Harald Haglund[2] Oskar Stenberg[3] Alexandra Annedotter[4]

Project Advisor: Lara Laban

[1]ma1766ta-s@student.lu.se
[2]ha4384ha-s@student.lu.se
[3]oskar.stenberg@control.lth.se
[4]al0403ka-s@student.lu.se

1 Project Purpose

The goal of this project is to make up to four Crazyflie-UAVs fly as a swarm and keep formations whilst moving as a synchronized unit. This project is an extension to the MSc Thesis (TFRT-6181) [1] by Stevedan Ogochukwu Omodolor, done at the Control Department, Lund University. The idea is to enhance the operational efficiency and coordination of Unmanned Areal Vehicles (UAV) swarms, introducing more complex formations, adaptive real-time path planning, and improved synchronization algorithms.

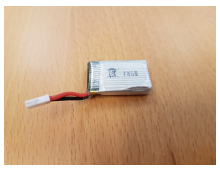
2 Equipment and material

The project will utilize the following hardware components:

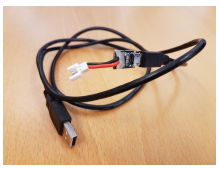
- *Crazyflie 2.1 x 4* (approx. 2200 SEK each): A lightweight open-source flying development platform.
- *Lighthouse positioning deck x 4* (approx. 1200 SEK each): An advanced positioning module that enables high-precision navigation of the UAVs using infrared light houses, crucial for maintaining tight formations and precise maneuvers.
- *Lighthouse V2 base station x 4* (approx. 2900 SEK each): Provides the infrared signals for the positioning decks, ensuring accurate and reliable spatial orientation for the UAV swarm within a designated operational area.
- *Crazyradio PA x 2* (approx. 400 SEK each): A long-range 2.4 GHz USB radio dongle that allows for enhanced communication capabilities with the UAVs, facilitating real-time control and telemetry data transmission over greater distances.
- *Crazyflie battery + charger x 10* (approx. 100 SEK each): Flight time about 20min interchangable batteries.

The appended prices are based on the Bitcraze online store. Some of the equipment can be seen in Figure 1.

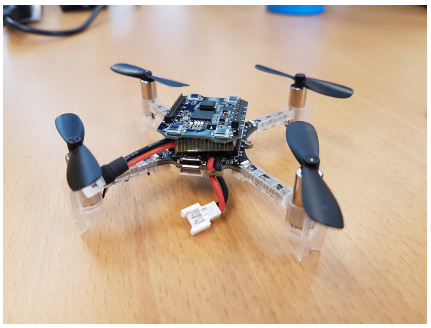
(a) Crazyradio PA.



(b) Crazyflie battery.



(c) Crazyflie charger.



(d) Crazyflie 2.1.

Figure 1: Equipment.

3 Modelling and System Design

We will be writing an implementation of a swarm-controller for the Crazyflie-UAVs, where the controller can run against Robot Operating System (ROS) (or potentially ROS2). ROS2 is a new verison

of ROS that is better for real-time systems and low-latency communication because of changes in the data middleware-layer, and ROS2 also features better multicore-performance which enables it to run more efficiently on modern computers. ROS2 also gives us more flexibility in which language the controller can be implemented in. [2]If we use ROS, we will use the officially supported operating system Ubuntu 20.04 LTS (Long Term Support). With ROS2, we will use the newer version of Ubuntu called 22.04 LTS which is mostly equivalent with Ubuntu 20.04 LTS but features newer kernel versions and upgrades to various modules that should not affect the performance of our controller. [3] The swarm-controller will send commands to individual Crazyflie-controllers that have been implemented in the Bitcraze Crazyflie Python API. The UAVs will be controlled from a central controller on the computer that is running ROS.

The swarm controller is going to be based on flocking-approaches as mentioned in the MSc Thesis (TFRT-6181 [1] and TFRT-6084 [4]) as well as the Bitcraze swarm controller library [5]. The goal is to make the UAVs follow a reference, while simultaneously avoiding other agents and matching the speed of neighbours. In case that the above works as expected, we will also try to implement a max-distance aspect in the controller where the swarm is constrained to have a specific size. This could for example mean that all UAVs in the swarm should be contained within a certain radius around a reference point.

4 Division of labour

The project is split into three main parts: Controller Implementation, Swarm Simulation, and Swarm Integration. They are described in the following subsections in detail.

4.1 Controller Implementation

The Swarm Controller is the main part of the project, and needs to be up and running in some form before the project can continue in any meaningful way. This part includes the actual flocking-based control algorithm.

One of the classic approaches to implement flocking behavior can be modeled after Reynolds' Boids algorithm, which is often used for simulating the flocking behavior of birds. The algorithm combines three simple steering behaviors - separation, alignment, and cohesion, in order to enable complex flocking dynamics.

1. Separation aims to keep simulated birds (boid) apart in order to avoid crowding,

$$F_{\text{sep}}(b) = -\sum_{b' \in N} \frac{P_{b'} - P_b}{\|P_{b'} - P_b\|^2}$$

where F_{sep} is the force of separation applied to a boid b, P_b is the position of boid b, $P_{b'}$ is the position of a neighboring boid b', and N is the set of neighboring boids within a certain distance.

2. Alignment ensures that a boid aligns its direction with the average direction of its neighbors,

$$V_{\text{align}}(b) = \frac{1}{|N|} \sum_{b' \in N} V_{b'} - V_b$$

where V_{align} is the alignment velocity for boid b, V_b is the velocity of boid b, $V_{b'}$ is the velocity of a neighboring boid b', and N is the set of neighbors.

3. Cohesion moves boids towards the average position of their neighbors to keep the flock together,

$$P_{\text{coh}}(b) = \frac{1}{|N|} \sum_{b' \in N} P_{b'} - P_b$$

where P_{coh} represents the cohesion position towards which boid b should move, P_b is the position of boid b, $P_{b'}$ is the position of a neighboring boid b', and N is the set of neighboring boids.

The final steering force applied to each boid can be a weighted sum of these three behaviors,

$$F(b) = w_{\text{sep}} \cdot F_{\text{sep}}(b) + w_{\text{align}} \cdot V_{\text{align}}(b) + w_{\text{coh}} \cdot P_{\text{coh}}(b)$$

where w_{sep}, w_{align}, and w_{coh} are weights that determine the relative importance of separation, alignment, and cohesion, respectively [6].

4.2 Swarm Simulation

During the following part of the project, we will try to integrate and run the Swarm Controller in ROS and visualize using ROS Visualization (Rviz) to see how the controller is behaving. We will spend a lot of time debugging the controller and changing the behaviour of the swarm during this phase. The work here can start at the same time as the controller implementation as we need to get a working ROS and Rviz-system that can simulate the behaviour of the UAVs before the controller is ready for testing.

4.3 Swarm Integration

Finally, after the previously described integration is completed, we can start integrating the swarm controller with the physical UAVs. We still plan to have ROS and rviz running as a base-layer, but to integrate the physical radios and UAVs into the system. This might require some additional coding, but should not be advanced since the Bitcraze library already offers most (if not all) of the functionality we need.

5 Time Plan

5.1 Subtasks

- Code centralized controller in python using Bitcraze library. **Estimated deadline: April 26**
- Simulate one agent in ROS. **Estimated deadline: April 26**
- Fly one agent in lighthouse using ROS. **Estimated deadline: May 3**
- Simulate all agents in ROS. **Estimated deadline: May 17**
- Fly all agents in lighthouse using ROS. **Estimated deadline: May 24**

5.2 Important dates

- Mar 25 - Hand in project plan.
- Mar 27 - Git repo in shape.
- Apr 15 - Self-evaluation 1.
- May 3 - Preliminary report.
- May 7 - Preliminary report - peer review.
- May 27 - Final report.
- May 29 - Self-evaluation 2.
- May 29 - Final report - peer review.
- Jun 4 - Revised final report.

5.3 Gantt Chart

We have formalized the time plan as a Gantt chart. The major tasks can be seen plotted in Figure 2.

Fixed tasks:	Finished:	Vecka 12	Vecka 13	Vecka 14	Vecka 15	Vecka 16	Vecka 17	Vecka 18	Vecka 19	Vecka 20	Vecka 21	Vecka 22	(Vecka 23)
Preliminary Project Plan	☑	March 22											
Revised project plan (25/3)	☐		March 25										
Git repo in shape (27/3)	☑		March 27										
Self evaluation 1 (1/4)	☐			April 1									
Preliminary report (3/5)	☐							May 3					
Preliminary report per review (7/5)	☐								May 7				
Final report (27/5)	☐											May 27	
Self evaluation 2 (29/5)	☐											May 29	
Final report peer review (29/5)	☐											May 29	
Revised final report (4/6)	☐												June 4
Subtasks:													
Code centralized controller in python using Bitcraze library (26 April)	☐												
Simulate one agent in ROS. (26 April)	☐												
Fly one agent in lighthouse using ROS.	☐												
Simulate all agents in ROS.	☐												
Fly all agents in lighthouse using ROS.	☐												
				Omtenta-P	Omtenta-P								

Figure 2: The Gantt chart describing the work flow of our project.

5.4 Risk Assessment

- Since we are bound to a small amount of UAVs and spare parts, a risk that follows is that the destruction of UAVs or components can lead to limited (or no) testing. We are approaching this risk by being extra careful when flying the UAVs and simulating the flights in a virtual environment (rviz) before performing any real flights.
- Another risk with this project is that we might have trouble synchronizing ROS with our controller. We believe this could be a risk specifically because not all group members have previous experience in ROS. This risk could lead to delays in the time-plan and insufficient time to finish the other sub-tasks. We are tackling this by trying to stick to our deadlines in the gantt chart as much as possible.

References

[1] Stevedan Ogochukwu Omodolor. *Distance and orientation-based formation control of UAVs and coordination with UGVs*. URL: https://lup.lub.lu.se/luur/download?func=downloadFile&recordOId=9096642&fileOId=9096646.

[2] Arun Venkatadri. *Ros 1 vs Ros 2 what are the biggest differences?* Apr. 2023. URL: https://www.model-prime.com/blog/ros-1-vs-ros-2-what-are-the-biggest-differences.

[3] Lukasz Zemczak. Feb. 2024. URL: https://discourse.ubuntu.com/t/jammy-jellyfish-release-notes/24668.

[4] Pontus Månsson Sebastian Green. *Autonomous control of unmanned aerial multi-agent networks in confined spaces*. URL: https://lup.lub.lu.se/luur/download?func=downloadFile&recordOId=8991431&fileOId=8991432.

[5] *Bitcraze swarm documentation*. URL: https://www.bitcraze.io/documentation/repository/crazyflie-lib-python/master/api/cflib/crazyflie/swarm/.

[6] Craig W. Reynolds. "Flocks, Herds and Schools: A Distributed Behavioral Model". In: *SIGGRAPH Comput. Graph.* 21.4 (1987), pp. 25–34. DOI: 10.1145/37402.37406. URL: https://dl.acm.org/doi/10.1145/37402.37406.