FORMATION CONTROL OF QUADCOPTERS

FORMATION CONTROL OF QUADCOPTERS

Thesis submitted to
Indian Institute of Technology Delhi
for the award of the degree

of

Master of Technology

by

Allu Sai Haneesh

under the guidance of

Dr.Shubhendu Bhasin
(Department of Electrical Engineering)

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY DELHI
July 2020

CERTIFICATE

This is to certify that the thesis titled **FORMATION CONTROL OF QUAD-COPTERS**, submitted by **ALLU SAI HANEESH**, to the Indian Institute of Technology, Delhi, for the award of the degree of **Master of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr.Shubhendu Bhasin
Assistant Professor
Dept. of Electrical Engineering
IIT-Delhi, 110016

Place: New Delhi

Date: 4th July 2020

Declaration

I certify that

a. the work contained in the thesis is original and has been done by me under the guidance of my supervisor;

b. the work has not been submitted to any other institute for any other degree or diploma;

c. I have followed the guidelines provided by the Institute in preparing the thesis;

d. I have conformed to ethical norms and guidelines while writing the thesis;

e. whenever I have used materials (data, models, figures and text) from other sources, I have given due credit to them by citing them in the text of the thesis, and giving their details in the references, and taken permission from the copyright owners of the sources, whenever necessary.

Allu Sai Haneesh

Acknowledgment

I want to thank my advisor, Dr.Shubhendu Bhasin, who introduced me to the field of multi-agent robotics and whose support lead to the development of this thesis. I'd also like to thank K.S.Santosh, Prem Sai and the Control laboratory staff for their assistance in setting up the experimental platform. Lastly, I'd like to thank my parents for their continued support of my decision to pursue a Master's degree.

Abstract

The primary purpose of the study is to investigate various formation control algorithms as well as implementing them on an experimental platform with the ultimate goal of implementing target interception by choosing the best suited among the implemented algorithms. The open source nanoquadcopter platform crazyflie 2.0 was choosen for the experimentation and ardupilot flight stack along with dronekit software in the loop were used for simulation purpose. The first phase consisted of study of virtual structure, leader-follower, a graph theoretic method of formation control. Secondly, understanding the control architecture of crazyflie 2.0, system setup and operation of optitrack motion capture system, robot operating system and dronekit Software in the loop. Up next is the controller design of the above mentioned formation control algorithms and implementation on the chosen platforms, comparing their performance in formation sustenance. Comparison shows that graph theoretic method is best suitable for formation maintenance. Finally Target interception has been simulated using the graph theoretic method and further exploitation of velocity and trajectory based formation controls are proposed as future work through optimisation techniques.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Background and Motivation

In recent years, cooperative control problems with multi-agent systems have attracted a lot of attention from many researchers. More importantly with the development of sophisticated autonomous control capabilities, multi-agent control has come to the forefront of research. Multi agent cooperation is frequently found in nature like birds flocking together, ants carrying food together etc, With increasing computing capabilities of embedded processors, cooperative control technologies are expected to be applied to actual vehicles such as Unmanned Aerial Vehicle (UAV), artifcial satellites, and autonomous mobile observation robots as well as sensor networks etc.

The possibility that multiple entity system can perform tasks more efficiently than a single entity has motivated the development of various control techniques [1]. Consensus control is a technique of bringing the multiple agents to agree on to a common state. Although it is effective, some modifications were required to perform complex tasks, the result of such modification is formation control.

Autonomous formation control is a desirable capability of many systems consisting of multiple mobile agents. Often times formation control is a necessary attribute of the system, fFor example, sensor arrays may need to maintain some formation in order to completely blanket an area with sensor coverage, or perhaps a spatial array formation is necessary to identify the direction of a propagating signal. formations are also beneficial in groups of mobile robots moving to a position target. But formation control also presents unique set of challenges like, collision detection and avoidance, maintaining the coordination and cooperation control, ability to tolerate the failure etc.

Survey on existing research shows that classification has been done based on controlability analysis, control strategies, sensed and controlled variables, group reference, and system dynamics [2] [3]. The classification based on group reference, measured and controlled variables is particularly of key interest because they can also be developed with little on the dynamics of the agents. Particularly position-based and distance-based formation control have been developed and improved extensively to suit the needs. Position-based, a centralised scheme where agents control themselves according to the targets received from a central entity, while in distance-based control, agents get into formation by sensing the relative distance of the other agents though they still present the challenge of sensing and tracking.

.

1.2 Project Objectives

This main objectives of the thesis are listed as follows:

- Development of algorithms for control and maneuvering of formation of multiple UAVs using virtual structure, leader-follower, graph theory based mechanisms
- Implement and compare various control architectures
- Implement Target interception using a suitable control technique among the above control algorithms

1.3 Thesis Organization

The rest of the thesis is structured as follows. chapter 2 describes the theoretical strategies used in the control of multiple agents and application of such strategies for target interception. chapter 3 focuses on hardware and software systems in detail that are utilised in the project along with the model and control architecture of the UAVs used in the project.

chapter 4 includes the experimental and simulated results of control strategies along with target interception application. Conclusion and further works are discussed in chapter 5

CHAPTER 2

Formation Control

This chapter begins with the introduction of well known formation control algorithms such as virtual structure, leader-follower, graph theory based and their evolutions are discussed, followed by overview of target interception problem and general low level techniques used for it are discussed.

2.1 Theoretical Strategies

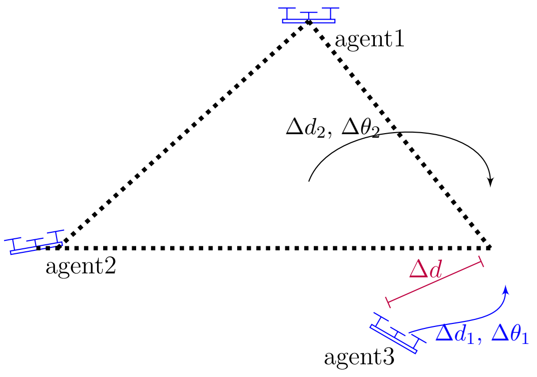
Formation control draws it's inspiration from the flocking model which can be categorised as

- **Formation Acquisition:** This implies that the agents starting from any position reach desired formation using an appropriate control algorithm.
- **Formation Maneuvering:** the entire acquired formation is navigated as desired by performing translation, rotation, scaling operations.

The formation control techniques described below discuss the formation acquisition and maneuvering and other control aspects required to achieve the goals.

2.1.1 Virtual Structure approach

Virtual structure formation control was introduced in [4]. This method is mainly used in applications that require fixed geometry shape at a time and not complex inter agent maneuvers. Key applications of this method include spacecrafts formation in deep space, laser interferometry, area mapping using unmanned robots etc. The concept of virtual structure is like that of rigid body structure except that it is not for real. Agents targets are assumed to be nodes of an imaginary structure defined and agents are expected to place themselves at the nodes. In this strategy two kinds of control are possible i.e., *unidirectional* flow and *bi-directional* flow of control.


agent1
Δd_2, $\Delta\theta_2$
agent2
Δd
Δd_1, $\Delta\theta_1$
agent3
(a) Geometrical portrayal

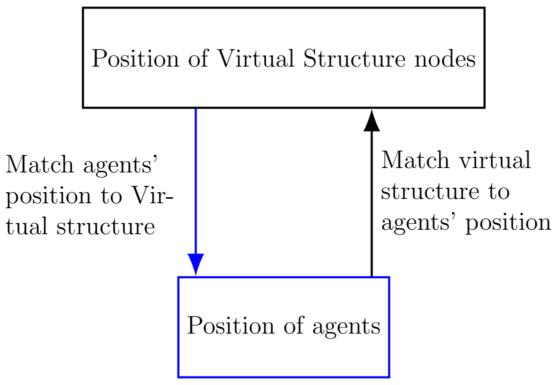
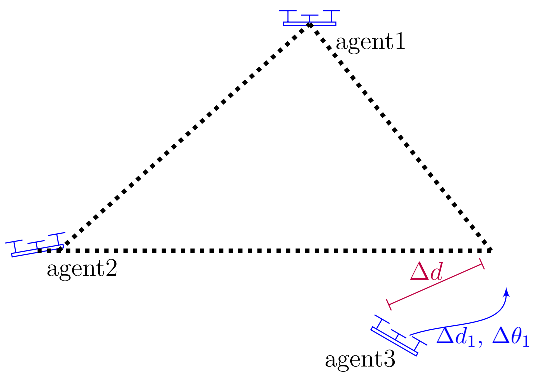

Position of Virtual Structure nodes
Match agents' position to Virtual structure
Match virtual structure to agents' position
Position of agents
(b) Block diagram

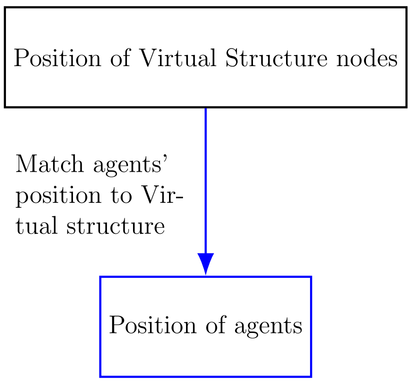
Figure 2.1: Bi-directional control of virtual structure formation

Bi-directional control

In Fig 2.1a depicted is the bi-directional flow of control or Virtual Structure with formation feedback incorporated to it. In this bi-directional control flow, agents would be in a fixed geometry and any disturbance to one agent is distributed among all the other agents since the formation feedback ensures the virtual structure adjusts itself too to match the Robots' positions as depicted in Fig 2.1b. This is similar to the unit referenced formation mentioned in [5]. Alternatively represented in Fig 2.1a whenever there occurs an error or difference of Δd in position of agents w.r.t to their target nodes, in order to zero out the error, both the virtual Structure and the out of target agent move. Depending on the Δd error, new targets are computed both for the agents having errors and as shown,the sidelined agent translates by Δd_1 and rotates by $\Delta\theta_1$. Virtual structure translates by Δd_2 and rotates by $\Delta\theta_2$ and due to this new targets are acquired for the rest of the agents and they track the target nodes accordingly.



(a) Geometrical portrayal


Position of Virtual Structure nodes
Match agents' position to Virtual structure
Position of agents

(b) Block diagram

Figure 2.2: Uni-directional control of virtual structure formation

Uni-directional control

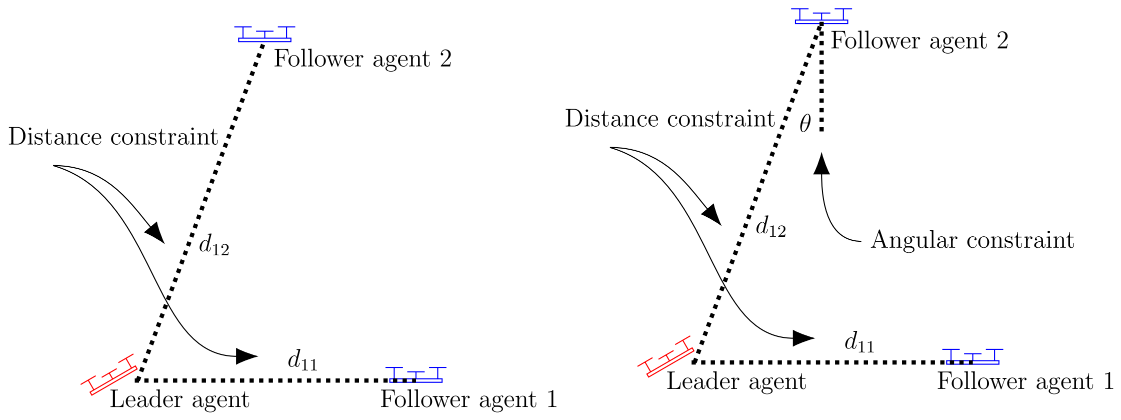
Represented in 2.2 is uni-directional flow of virtual structure formation control. Although the agents is this strategy be in fixed geometry, disturbance to agent is limited to itself due to lack of formation feedback to virtual structure. Movement of agents is similar to that of in the bi-directional control, except that virtual structure remains fixed unless altered by the user or externally generated inputs. Further insights are given in [6]. The uni-directional flow model of the virtual structure is similar to the **Formation control via Generalized Coordinates** which is a position-based control [2] where each agents desired position is the position of the node of the structure realized with respect to a reference point.
In Virtual Structure Formation control design, following tasks are involved:

i. Defining virtual structure and computing its target trajectories or goals.
ii. Computing node positions of the Virtual structure at the given time and processing them as inputs for each agent.
iii. Tracking controller design for each agent to track the received inputs.

iv. Formation feedback back propagated to goal/target computation depending on the kind of control flow

2.1.2 Leader Follower approach

As the name suggests, the formation comprises of lead agents(s) and follower agents. The target trajectories or goals are computed for the lead agents while the follower agents try to position themselves w.r.t to the lead agent by virtue of predefined constraints. In [3] two kinds of formation control strategies are proposed for this approach



(a) Distance constrained formation

(b) Distance-bearing constrained formation

Figure 2.3: Leader-follower formation control

Distance constrained formation

In this approach each follower agents are expected to maintain predefined distance d_{11}, d_{12} as shown in Fig 2.3a between itself and the leader agent(s). In the presence of availability of global information, by sensing the leader's position distance between itself and lead agent(s) are found, and followers position is adjusted accordingly by the control commands generated. In the presence of sensory equipment, the distance is measured directly without the need for knowledge of global Information.

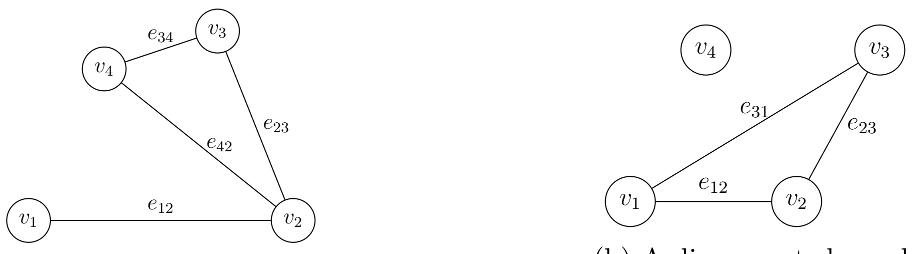
Distance-bearing constrained formation

The term bearing implies the angle of line of sight between two points. In this approach each follower agent is expected to maintain predefined distance constraint d_{11}, d_{12} as well as bearing constraint θ to be maintained between itself and the leader agent(s). similar to distance constrained formation, requirement of sensory equipment is dependent on whether agents can get global information or not. But this approach using the distance and bearing constraints, if by resolving the distance component along the body x, y, z axes, gets modified instead of distance bearing but as **Position based approach** where each follower agent is required to maintain x, y, z difference with respect to the lead agent. Fig 2.3b shows the distance bearing approach.
In the control design of leader-follower formation, following tasks are involved

i. Defining distance and/or bearing constraints for the formation.
ii. Computing targets for the follower agents w.r.t to leader agent.
iii. Design of tracking controllers for both leader and follower agents for target position tracking.

2.1.3 Graph theory based approach

As a way of describing the interaction among the agents in a network, graph theory has become popular. In this section several key concepts of graph theory related to this project are introduced, followed by the consensus equation which forms the base for several formation control methods and adding to it two known formation control methods. The circle representations in the Fig 2.4 are the *Nodes*



(a) An example graph (connected)

(b) A disconnected graph

Figure 2.4: Sample graph representation

and the line representation between two nodes are the *Edges*. These two are the fundamental objects that makeup a graph. Particularly in this control context, nodes can be treated as robots and the edges between them can be treated as constraint to be maintained. Collection of vertices, edges and neighbourhood of a node are described below

$$\text{Vertex set } V := \{v_1, v_2 \cdots v_n\} \tag{2.1}$$

$$\text{Edge set } E := \{e, e_2 \cdots e_m\} \tag{2.2}$$

$$\text{Neighbourhood } N(i) := \{v_j \in v/e_j \in E\} \tag{2.3}$$

A graph is said to be connected if there exists a path connecting two nodes not necessarily direct but by tracing along other nodes as well. Requirement of complete graph is necessary for the consensus described later in this section. Related to the graphs are some matrices which describe the properties of the graph.

Adjacency matrix [2] describes the connectivity of constraint existence between the nodes. Adjacency matrix of graph in Fig 2.4a is derived below. If the graph is weighted by the virtue of constraints, the $1's$ are replaced by their respective weights.

$$\text{Adjacency matrix A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \tag{2.4}$$

The key to any control design based on the graph theory is the consensus protocol[7]. Agents starting from states of different value agree to converge to a common state of position or velocity etc, depending on the model provided the their graph representation is connected [7]. The consensus equation is described in Eq 2.5 where x_i is the agent's state i, and $j \epsilon N_i$ denotes all agents, j, in the neighbourhood of i.

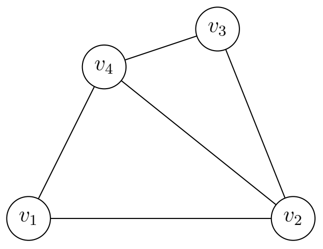
$$\dot{x}_i(t) = k \sum_{j \in N_i} (x_j(t) - x_i(t)) \tag{2.5}$$

Proof for state agreement of agents using Eq 2.5 can be found in [7]. If agents starting from various positions agree upon the position consensus, then they all reach a common position in due time provided the graph connectivity. But if we impose the constraints on the edges, it is required that the agents maintain the constraints. Hence modifications are done to the consensus law in Eq 2.5. The result of modification is the distance based formation control where the network of agents represented as nodes and the distance constraint is the weight of the edge [2] [8]. The graph is realised by the set of points $y_1 \dots y_n$, and points $x_1 \dots x_n$ denote agent positions. The distance constraint d_{ij} is calculated as $y_i - y_j$. Agent x_i is assigned to point y_i. The modified consensus law [9] is

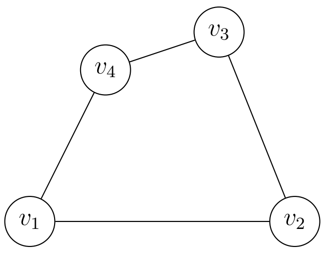
$$\dot{x}_i(t) = k \sum_{j \in N_i} \omega_{ij} \left(x_j(t) - x_i(t)\right) \tag{2.6}$$

$$\text{where } \omega_{ij} = \left(\|x_j(t) - x_i(t)\| - d_{ij} \right) \tag{2.7}$$

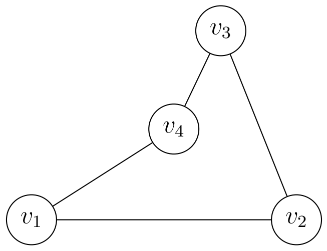
Proof of formation using control law in Eq 2.6,2.7 can be found in [9]. The converge to desired formation depends on whether the graph is rigid or not. Rigid graph means the graph when realised by bar and hinge model, is non deformable. It is



(a) An example Rigid graph (2D)



(b) An example Flexible graph (2D)



(c) Isomorphic representation of the flexible graph

Figure 2.5: Rigid and Flexible graph representations

required that the graph is rigid because there can be many isomorphic realisations of a non-rigid graph as shown in Fig 2.5 and that the formation with desired constraints would not not be as expected. Hence graph rigidity guarantees that formation takes place as represented by the graph. But too many edges would jam up the graph and increases number of constraints. Hence minimally rigid graphs are used. A rigid graph in which, when an edge is taken out is not a rigid graph, then such graphs are called *Minimally rigid* graph. Minimal rigidity is a necessary condition, while rigidity is sufficient condition [8].

2.2 Target Interception

Target interception is the problem of persuading a target object. In general using a single entity, target tracking is done using line of sight techniques such as [10], and several other meta heuristic methods have been used. In these, guidance logic based on persuader dynamics and target velocity profile is designed. Utilisation of the multi agent techniques for target interception have been presented in [11],[8]. In the multi agent techniques, group of persuaders are assigned target locally to intercept the target while referencing with other agents in the system.

CHAPTER 3

Experimental setup

In this chapter the hardware as well as software technology requirements for the experimentation and simulation process are discussed. While the hardware include UAV which is the quadrotor aircraft, motion capture camera technology for the positioning, software requirements include software-in-the-loop(SITL) technology for simulations, robot operation system (ROS) for commanding the UAV, Motion capture software for data processing and streaming.

3.1 Crazyflie Quadcopter (UAV)

The Crazyflie 2.0, a nano Quad-copter is a versatile flying development platform developed by Bitcraze [12]. It is considered ideal for many areas of research and so is very popular in the robotics community due to their small size, quiet operation, and high maneuverability. Also, due to their low inertia, they pose minimal harm to their surroundings if system failures occur. Their small size and robust nature are well suited for flying in an indoor. The source code and hardware are open, making it possible to tap into any part of the system for complete control and full flexibility. It also enables new hardware or sensors to be added through the versatile expansion ports. The crazyflie 2.0 is represented in Fig 3.1. Its specification are described in next section.

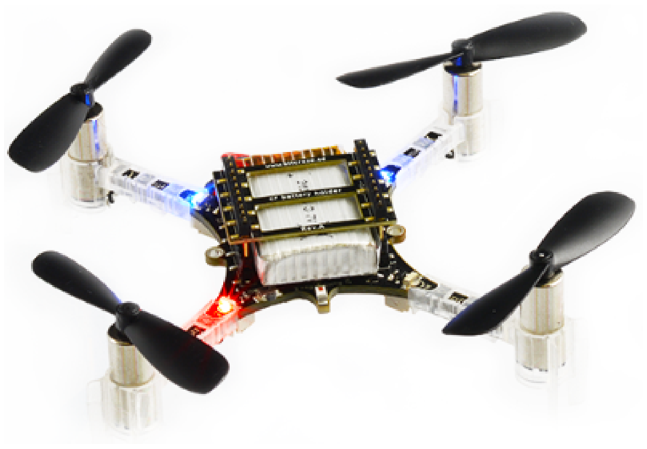


Figure 3.1: Crazyflie 2.0 Quad-rotor aircraft

3.1.1 Specifications

This section includes several hardware, electronics and communication protocol specifications of the crazyflie 2.0 [12] that form the core part of it's operation and the on-board electronic architecture is also presented.

i. Crazyflie 2.0 weighs only $27g$ and is of $92x92x29\,mm$ size. It has a maximum flight time of 7 minutes, charging time of atleast 40 minutes with a payload capacity of 15g. It is fitted with coreless DC motors with a motor constant of $14000\,rpm/volt$.

ii. Crazyflie 2.0 works on two microcontrollers that are placed on-board. STM32F405 is the main application MCU $[Cortex{-}M4, 168MHz, 192kbSRAM, 1Mbflash]$ while the other nRF51822, is a radio communications and power management MCU [Cortex-M0, 32Mhz, 16kb SRAM, 128kb flash] [13]

iii. It also has a 10-DoF inertial measurement Unit (IMU) which contains 3 axis gyro, 3 axis accelerometer, 3 axis magnetometer and a high precision pressure sensor. This unit forms an essential feedback component in flight stabilisation and control

iv. A 2.4 GHz radio USB dongle called the Crazyradio allows communication between the host and the crazyflie. The crazyradio has a 20 dbM power

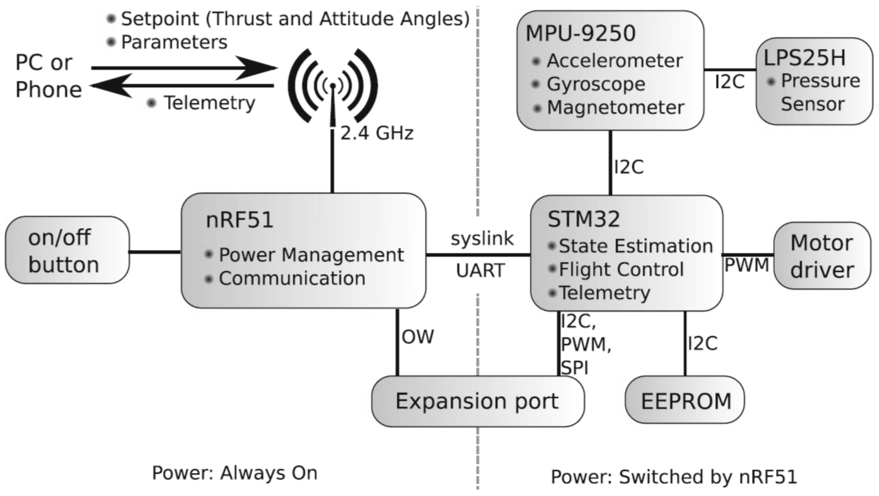


Figure 3.2: On board system architecture

amplifier which can increase the range of communication upto 1 km line-of-sight. In addition to this, crazyflie can also be controlled from an android or iOS device using bluetooth LE.

v. Commands and messages are packed using *Crazy Real Time Protocol* (CRTP) which enables the crazyflie to accept external as well as internal commands

3.1.2 On-board Control Architecture

In understanding the onboard control architecture, the following representation and their necessity is known. The twist about the y axis is the pitch motion, twist about the x axis is the roll motion and twist about the z axis is the yaw motion. A typical body axis representation of the roll,pitch yaw is shown in 3.3. When a motor rotates, the propeller attached to it also rotates in the same direction. Due to rotation, aerodynamic lift is generated perpendicular to the axis of rotation [14]. The aerodynamic lift force produced due to each propeller spinning is $F = k\omega^2$ where ω is propeller angular speed, k is force constant. In a well-balanced condition i.e.., all the motors running at the same speed, the net force produced due to rotation of propellers is along the **z** axis and acts against the gravity, and the net force about **x** and **y** axis are 0. As seen from Fig 3.3 in order to obtain

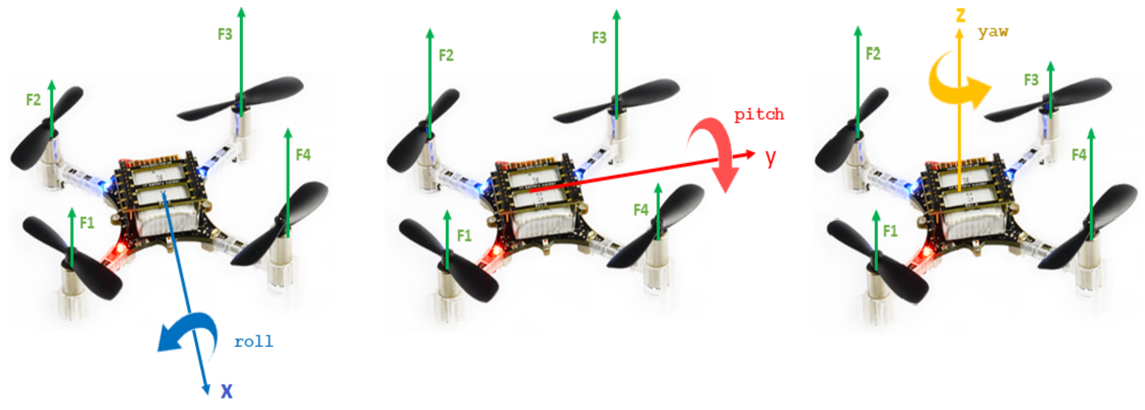


Figure 3.3: Roll,Pitch,Yaw moments of Crazyflie

either pitch, roll or yaw movement there needs to be a torque generated due to difference in the forces about the their respective axis. Table 3.1 describes the contribution of forces to roll,pitch, yaw moments.

Forces	**Roll**		**Pitch**		**Yaw**	
	+ve	**-ve**	**+ve**	**-ve**	**+ve**	**-ve**
F_1	↓	↑	↑	↓	↑	↓
F_2	↓	↑	↓	↑	↓	↑
F_3	↑	↓	↓	↑	↑	↓
F_4	↑	↓	↑	↓	↓	↑

Table 3.1: Forces combination for Roll, Pitch, Yaw moments

The onboard controller is a cascaded PID controller[15] for controlling pitch and roll angles. It consists of angular velocity PID controller which regulates the plant that is the quadcopter. It is followed by the angular PID controller which regulates the angular velocity PID loop. Fig 3.4 is the representation of the cascaded control structure.

The **inner rate control** calculates the input deviation from the equilibrium point of the motors in order to maintain the angular momentum required. For that, three independent controllers are used. This loop controls the quickness of the quadrotor. The **outer attitude control** loop act as a regulator of the rate controller loop. The attitude controller calculates the appropriate setpoints for the angular velocities around the **x** and **y** axis, in order to stabilize the quadcopter

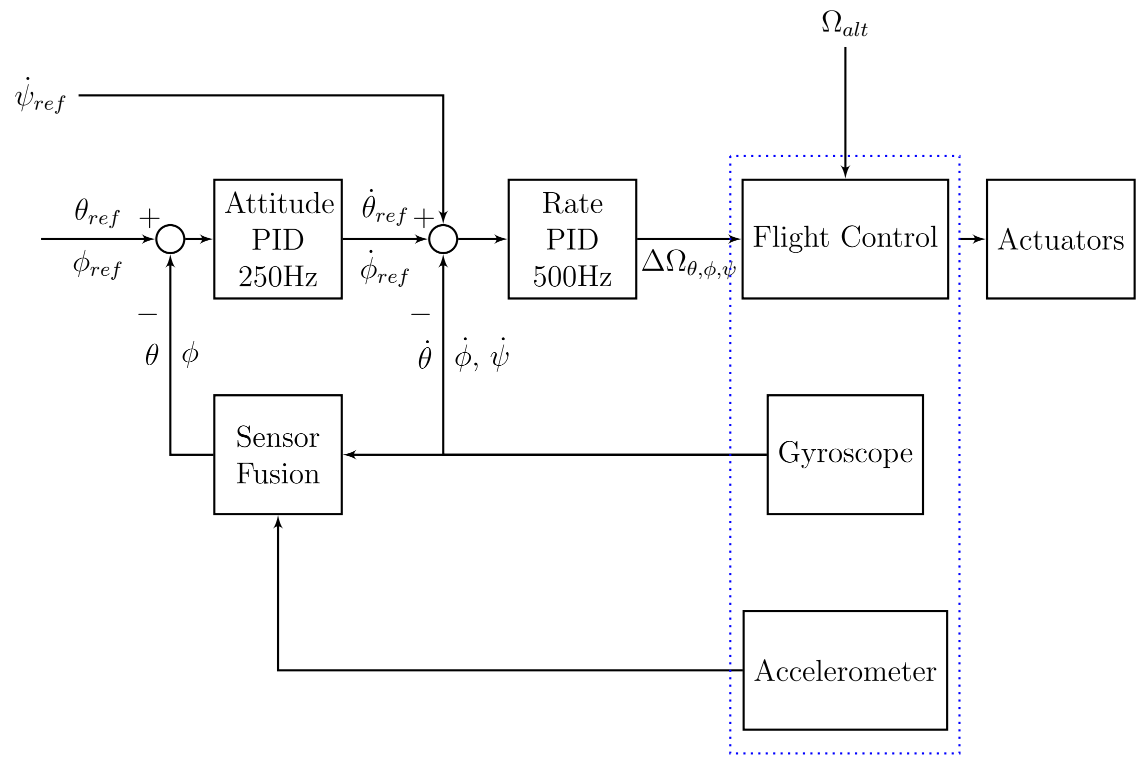


Figure 3.4: Onboard control architecture of crazyflie 2.0

Notation	**Description**
$\phi_{ref}, \theta_{ref}, \psi_{ref}$	reference roll, pitch, yaw angles in rad
ϕ, θ, ψ	roll, pitch, yaw angles in rad
$\dot{\phi}_{ref}, \dot{\theta}_{ref}, \dot{\psi}_{ref}$	reference roll, pitch, yaw angular velocity in rad/s
$\dot{\phi}, \dot{\theta}, \dot{\psi}$	roll, pitch, yaw angular velocity in rad/s
$\Delta\Omega$	thrust deviation/correction input
Ω_{alt}	thrust input

Table 3.2: Control parameters and Input representation

at a certain desired angular position. The attitude controller uses the pitch and roll angles from data that is fed back by the onboard IMU, compares them to the external commands θ_{ref} and ϕ_{ref} (coming from teleoperation, off-board controller, etc,) and feeds them to the controller that calculates the desired angular velocities $\dot{\theta}_{ref}$ and $\dot{\phi}_{ref}$. But in this yaw control is not present.The $\dot{\psi}_{ref}$ (yaw rate reference) commands coming off board are directly sent to the flight control. Yaw control needs to be implemented off board.

As mentioned, output of the rate controller is the total input deviation of the motors from the nominal state required to generate a torque in the desired direction of movement. This input deviation has to be distributed to the motors in the same fashion as in Fig 3.3 for it to maneuver appropriately using the PWM input [15]. Given that the quadcopter is in X configuration, the motor effort has to be distributed halfway in each motor for a desired torque around the X or Y axis. The distribution is depicted in equation 3.1 that is implemented on board of the crazyflie 2.0, known as *Control Mixer.*

$$\begin{cases} PWM_{\text{motor}_1} = \Omega_{alt} - \omega_{\phi}/2 - \omega_{\theta}/2 - \omega_{\psi} \\ PWM_{\text{motor}_2} = \Omega_{alt} + \omega_{\phi}/2 - \omega_{\theta}/2 + \omega_{\psi} \\ PWM_{\text{motor}_3} = \Omega_{alt} + \omega_{\phi}/2 + \omega_{\theta}/2 - \omega_{\psi} \\ PWM_{\text{motor}_4} = \Omega_{alt} - \omega_{\phi}/2 + \omega_{\theta}/2 + \omega_{\psi} \end{cases} \tag{3.1}$$

Post this calculation, the PWM signals are then fed to the Electronic Speed Controllers (ESCs) which then applies appropriate voltage to the motors.

3.2 Local Positioning System

Positioning system can be defined as a navigation system which provides location information about the object in interest at possible places. Famous positioning system NAVSTAR GPS is widely used in outdoor environments. But in the indoor or dense regions, where the line of sight between satellites and the recipient is not clear, reliable information cannot be obtained. To tackle this, particularly in indoor environments, local positioning systems are used. These local positing systems provide the location information of the object locally in the desired region. Key Local positioning systems like motion capture technology, ultrawide band technology, wifi and other technologies are in use. In the current project motion capture system provided by Optitrack is used as local positioning system.

3.2.1 Optitrack Motion Capture Technology

"Motion Capture" (sometimes referred as mo-cap in short) is the technique of digitizing or recording the movement of the subject in interest[16]. It is used in entertainment, sports, medical applications, and also for validation of computer vision and robotics etc.,[17] Commonly, retro reflective markers are attached to the subject's body and the data is captured by performing a range of movements in front of an series of motion capture cameras. The Optitrack motion capture system which is used in this project, provides the data with the use of Flex13 camera and Motive software.

Flex-13 Camera

Flex 13 camera[18] as seen in Fig 3.5 is specifically designed for the motion capture system. With its 56^o field of view, it can track several complex movements of the object. It is equipped with a ring of 28 LEDs which produce strobe or continuous IR rays (illumination). This illumination when incident on retro reflective markers that are placed in its field of view, gets reflected and when also well within field of view, they get captured by the camera image plane and the 2D point of the marker is formed. It has a removable front plate, which helps in accessing the lens for adjusting the focus. In order to obtain 3D data, several such cameras are used and their data is collectively sent to the motive software through the hubs as depicted below and through the process of triangulation 3D pose estimate is formed. Brief discussion on 3D reconstruction is discussed in sub section below. 3.6 shows the data transfer connection from the cameras to the Motive software. Cameras are primarily connected via USB cables to the OptiHub. Each hub requires power supply and each can accommodate maximum of 6 camera

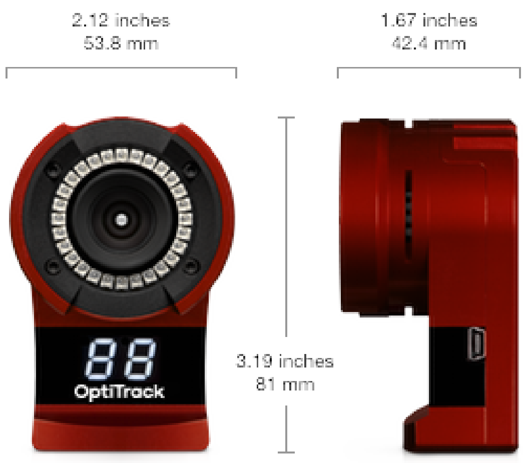


Figure 3.5: Flex-13 Camera

connections. For more connections incoming, additional hubs are used and the camera connection incoming must be uniformly distributed for better processing of data and all such hubs are interconnected via sync cable to synchronise the data transfer. These hubs are the connected to the personal computer in which motive software is being utilised.

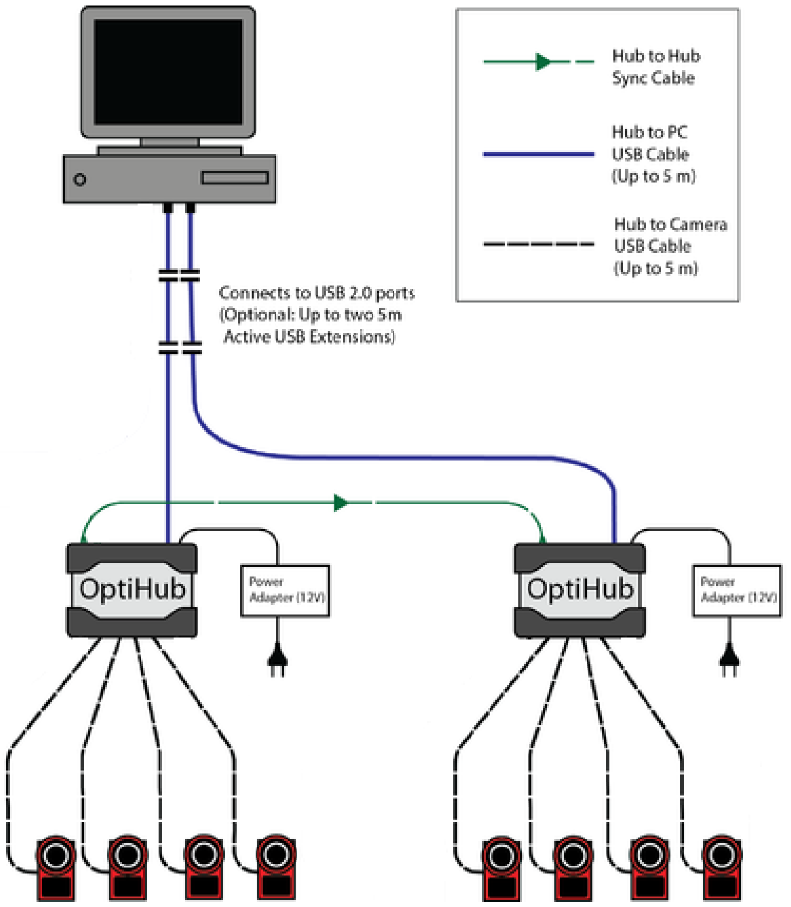

Hub to Hub Sync Cable
Hub to PC USB Cable (Up to 5 m)
Hub to Camera USB Cable (Up to 5 m)
Connects to USB 2.0 ports (Optional: Up to two 5m Active USB Extensions)
OptiHub
Power Adapter (12V)
OptiHub
Power Adapter (12V)

Figure 3.6: Connection arrangement for data transfer to Motive software

Motive Software

Motive is a software platform designed to control motion capture systems for various tracking applications. Motive not only allows the user to calibrate and configure the system, but it also provides interfaces for both capturing and processing of 3D data. The captured data can be both recorded or live-streamed to other pipelines. Motive obtains 3D information via Reconstruction, which is the process of compiling multiple 2D images of markers to obtain 3D coordinates. But for the system to do 3D reconstruction, it should have been calibrated so that the exact scaling of the area and orientation of the cameras, which are used to perform the experiment would be known to the software [19].

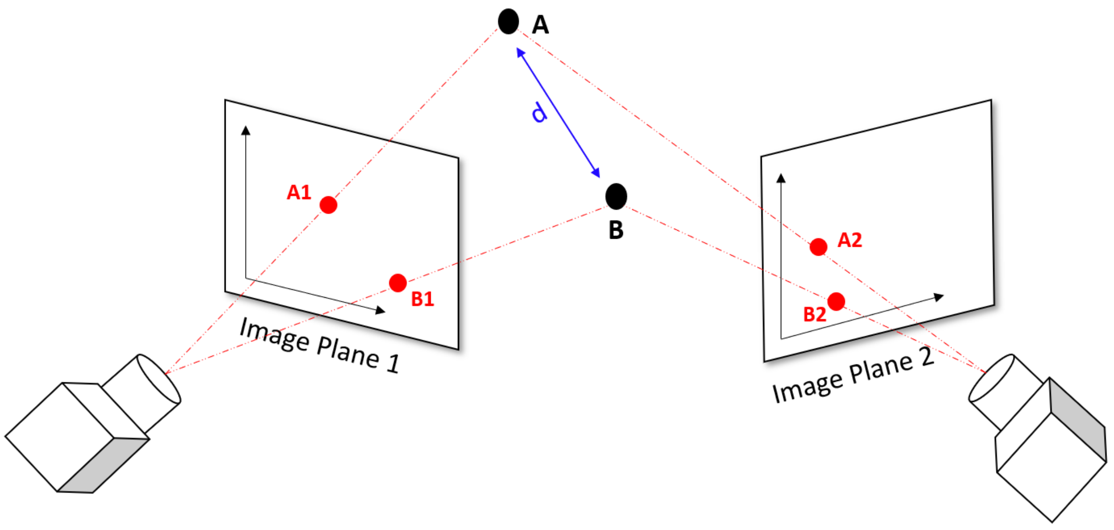


Figure 3.7: Sample 2D data capture

Calibration: In the figure 3.7 A and B are two different points in 3D plane with known distance d between them. Both A and B are projected on to the image plane of the cameras as A_i and B_i for the i^{th} camera. With known $\overline{AB}$ distance and $\overline{A_iB_i}$ distance along with camera distance from illumination origin to image plane, the distance of each camera from the sample points is calculated by the software using similarity principle. This helps in determining position and orientation of the cameras. Gathering such 2D data from each camera at same instant, 3D coordinates are determined by triangulation process.This is the basis for the calibration. Initially, all the cameras are arranged at different locations and their orientation is adjusted to cover and estimate of maximum area by looking in the camera output in motive software in grey-scale mode. Once the arrangement is done, the calibration is done with the help of calibration wand and ground plane [18] as shown in the fig 3.8. Initially the calibration wand, which has 3 markers

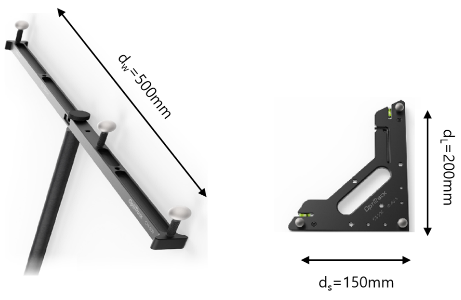


Figure 3.8: Calibration wand and Ground plane

with the spacing between end markers is of length 250mm (or 500mm adjustable and is pre-fed in software) is waved around. With known end spacing, when the two markers of the wand as seen by the cameras in Fig 3.8, using the 2D coordinates of markers data from each camera, the orientation and arrangement of cameras is known by gathering at least 3000 samples of data. Once the ground plane is set we now have a local map in which a new sample is 3D reconstructed with the help of calibrated parameters. Fig 3.9 shows various levels of tracking precision.

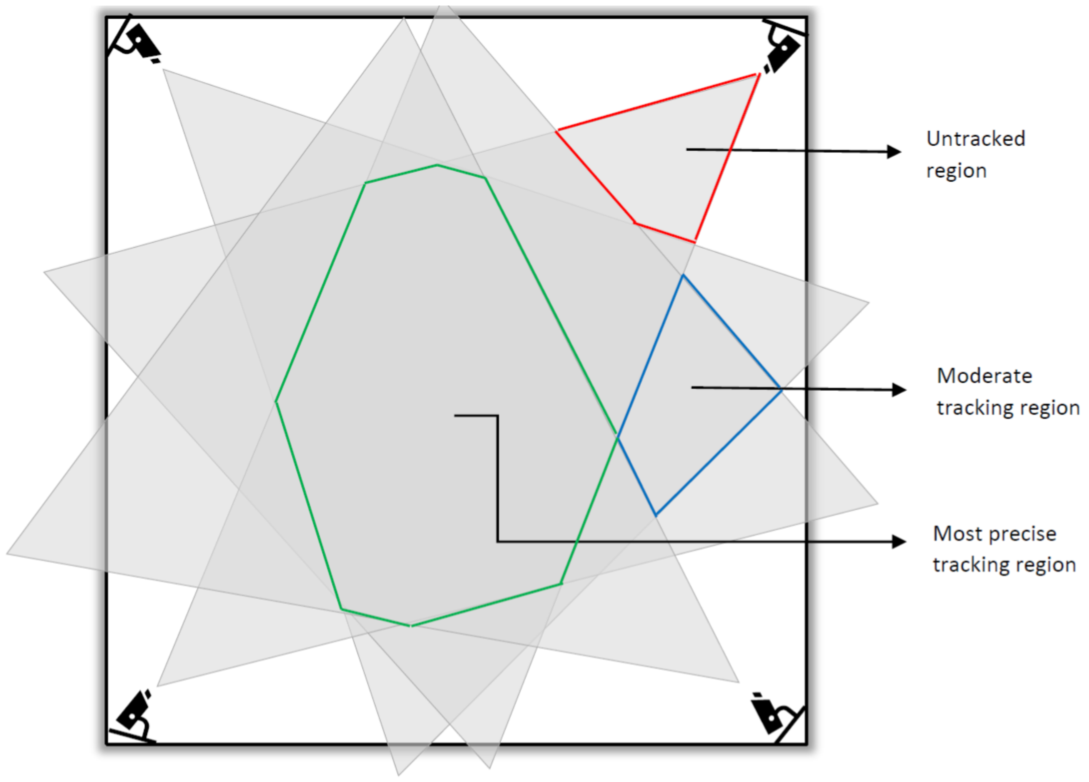


Figure 3.9: Accuracy of tracking regions

3.2.2 Rigid Body Data Capture

Once the calibration has been done as mentioned in the previous section, we now have a local map in x, y, z planes. In order to obtain movement data which constitutes both position and orientation of the object in interest, at least three markers arranged in non-collinear arrangement are required. This is because with less than 3 markers, orientation in at least one axis cannot be defined. Once the markers are placed on the object, they can be seen in the motive software. All the markers are then selected, and a rigid body is formed in the software. The orientation of the object at the time of rigid body formation is taken as zero orientation by the software. Now all the movements of the objects are tracked till all the markers are in the visible region of at least two cameras. This capture can be seen in the Fig 3.10

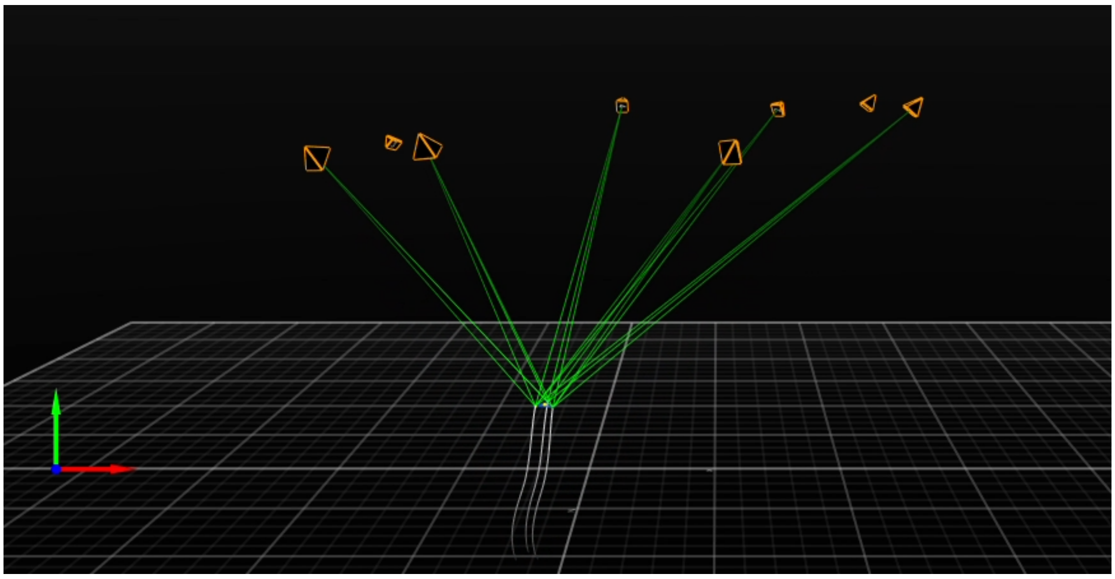


Figure 3.10: Tracking of rigid body

Now the captured data needs to be streamed in real time in order for it to be used by other applications. This process is facilitated by two methods. One is transmitting through VRPN server and the other through NatNet streaming. In the VRPN streaming, IP address of the network to which data is to be streamed is fed in the motive software and in the client machines data is extracted from that network. NatNet streaming is out of scope for this project. For storing data and post processing it, the object movements are recorded and then exported as .csv files. Demo of the data streaming is described in [20],[21]. The entire process involved in the motion capture is summarised in the block diagram Fig 3.11

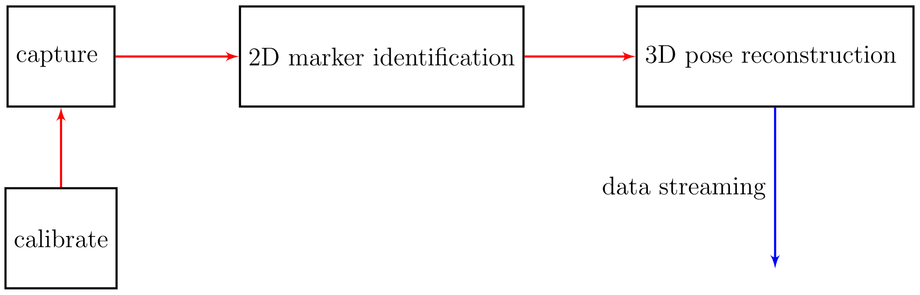


Figure 3.11: Motion capture process block diagram

3.3 Robot Operating System

The Robot Operating System (ROS) is an open source flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms. It provides the service like operating system, including low-level device control, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers [22]. In ROS terminology, source code is organized into 'packages' where each package typically consists of one or more targets when built. These targets may be executable programs, generated scripts or anything else that is not just a static code. ROS framework consists of key components such as nodes, topics, services, launch files, packages etc.,

Node:

ROS typically consists of program files called nodes which execute a set of commands. With the help of topics nodes communicate between each other. This communication is enabled by ROS master which allows nodes to publish messages to a topic and subscribe messages from another topic.

Topic:

Topics are interconnecting channels over which nodes exchange messages. Topics may have anonymous publish/subscribe semantics, which decouples the production of information from its consumption. Nodes that require data from a topic subscribe to it and node that generate data publish to the relevant topic. There can be multiple publishers and subscribers to a topic.

Services:

Request, reply is done through a Service, which is defined by the pair of request and reply messages. ROS node offers a service under and a client calls the service by sending the request message and waits for the reply. Thus, Services are an another way for communication between the nodes. As mentioned above service calls are bi-directional i.e. the information could flow in both the directions. Services are defined using .srv files, which are compiled into source code by a ROS client library. These three entities constitute basic functionalities of ROS. The communication interconnection between them is represented in Fig 3.12

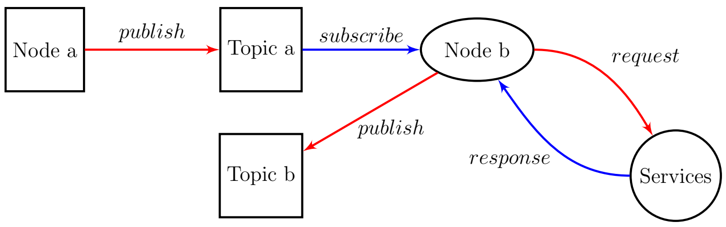


Figure 3.12: ROS framework

Launch files:

Launch files are of the format .launch and use a specific XML format. It consists of the node names and launch file names and parameters to be set, written in a defined manner. All the mentioned nodes and launch files are spawned at the same instance when launched.

Packages:

Software in ROS is organized in packages. A package might contain nodes, services, datasets and others that make a useful module. The necessity of the package is to provide code re usability. Some of the common directories and files found in packages are

i. *Include/package-name:* C++ include headers
ii. *msg:* Folder containing Message (msg) types both standard and custom types are allowed
iii. *src/package-name/:* Source files, particularly Python source.
iv. *srv:* Folder containing Service (srv) types
v. *scripts:* executable scripts
vi. *CMakeLists:* CMake build file etc.

To build and use packages that are interdependent on each other, catkin workspace is used which a folder in which we can modify, install and build packages.

3.4 Crazyflie ROS Package

In this project, instead of building from scratch, the **Crazyflie-ROS** [23] package developed by Mr. Wolfgang Hoenig which was elaborated in the book [24] and **VRPN-ROS-clinet** [21] maintained by Paul Bovbel, for converting Motion capture data into ROS understandable message format are used. The VRPN-ROS-client packages has subscribers that subscribe to the raw motion captue data of the rigid body and publish the same data in the format of standard ROS messages. The Crazyflie-ROS package has several nodes that are Publishers, Subscribers, Services as well as topics, launch files and parameters. One of the key implementation was the position and yaw (tracking) controller which was written in C++ programming language. The output of this controller is sent to the crazyflie 2.0 quadrotor via telemetry link, thus combining the off-board and on-board control architecture as shown the Fig 3.13.

The error in **x** direction is mapped to Pitch reference(θ_{ref}), error in **y** direction to (ϕ_{ref}), error in **z** direction as reference thrust (Ωref) to achieve and

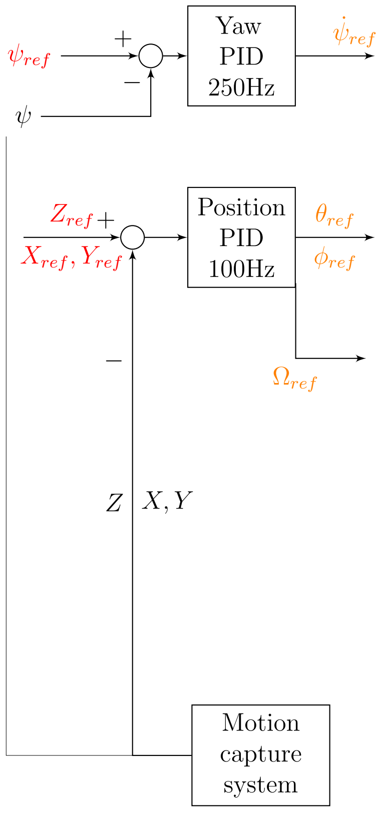


Figure 3.13: Offboard PID Position and Yaw controller architecture

maintain the altitude required by acting against its weight. Detailing of the x, y position PID controller is seen in Fig 3.14. The reason for such mapping in **x**, **y** axes is because as seen in fig 3.3 change in pitch commands produces movement local **x** axis, and change in roll commands produces movement in local **y** axis. But when seen from the global perspective, the same may not be replicated because the quadcopter local axes may not be aligned with the global map axes. Although rotation and

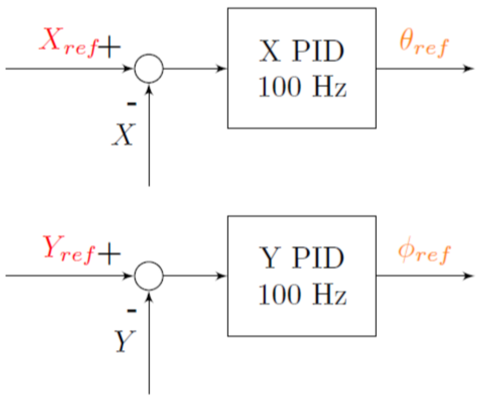


Figure 3.14: X, Y axes position PID controller

transformation can be applied, but it is in general a good idea to try and keep both the axes frames aligned so that quadcopter orientation can always be determined easily and hence the reference yaw (ψ_{ref}) is always set to 0 thus, the nose of the quadcopter is always made to point along the x-axis such that the body axes of the quadcopter is aligned with the reference axes of the map and by maintaining that yaw orientation, the moment along the x-axis,y-axis are decoupled to pitch and roll. Below equations describe the decoupling.

$$\begin{aligned} X_{global-error} &= X_{global-reference} - X_{global-position} \\ Y_{global-error} &= Y_{global-reference} - Y_{global-position} \end{aligned} \tag{3.2}$$

- Without Alignment of local axes to global axes

$$\begin{aligned} X_{body-error} &= X_{global-error} * \cos\psi - Y_{global-error} * \sin\psi \\ Y_{body-error} &= X_{global-error} * \sin\psi + Y_{global-error} * \cos\psi \end{aligned} \tag{3.3}$$

- With Alignment of local axes to global axes i.e.., $\psi = 0$

$$\begin{aligned} X_{body-error} &= X_{global-error} \\ Y_{body-error} &= Y_{global-error} \end{aligned} \tag{3.4}$$

For this architecture, the following important ROS nodes, topics, parameters are used in Crazyflie-ROS package.

i. **crazyflie/goal** The reference **x**, **y**, **z** values are published to this topic
ii. **/tf:** The pose data of the agents are published to this topic in the transform message format
iii. **crazyflie/controller:** This is the node which has position controller written in it.
iv. **crazyflie/cmd-vel:** The reference roll, pitch, yawrate, thrust values are published to this topic
v. **vrpn-client-node:** This node receives the data from Mocap system and published to **/tf** topic

vi. **URI:** It means Unique Resource Identifier that is used to connect to the crazyflie quadcopter

vii. **crazyflie-server:** This node subscribes to the topic **/crazyflie/cmd-vel** and packs it into CRTP protocol format for sending to the quadcopter

3.5 Software in the Loop

Software-in-the-Loop (SITL) is the testing of a software or organised pack of code, which mimics the actual hardware modelling, dynamics, response and also several other environments. With this the need of actual hardware whenever required is eliminated during the testing. This helps in foreseeing any errors, eliminate damage of hardware to possible extent by debugging after testing under various environments. One such an SITL called dronekit-SITL is used in this project to simulate the environment and hardware of the ardupilot autopilot. Below sections describe the Autopilot and dronekit SITL in detail

3.5.1 Ardupilot UAV Flight stack

Ardupilot [25] is an open source software designed particularly for the control and operation of Unmanned vehicles, robots. It was developed by few enthusiasts initially for controlling small robotic models of aircrafts, rover, sailboat. It consists of navigation, orientation and several other low and high level controls of software running in autopilot used on the vehicles. It is basically a firmware, which is flashed on to the autopilot hardware. Once it is installed, sensory data is being utilised to control the vehicle as desired. In this project ardupilot version 4.0.0 is used for the control of quadcrotor UAV although the UAV isn't a hardware but an SITL which is discussed in next section. Fig 3.15 shows a simplified version of the ardupilot flight stack. It has blocks of cascaded code for navigation control, altitude control, attitude and rate control mechanisms. The communication between all the code blocks takes place using the MAVLink protocol, which is a standard protocol used particularly in UAVs for data transmission. Detailed review on MAVLink protocol has been discussed in [26]. Few key flight modes related to this project are mentioned below.

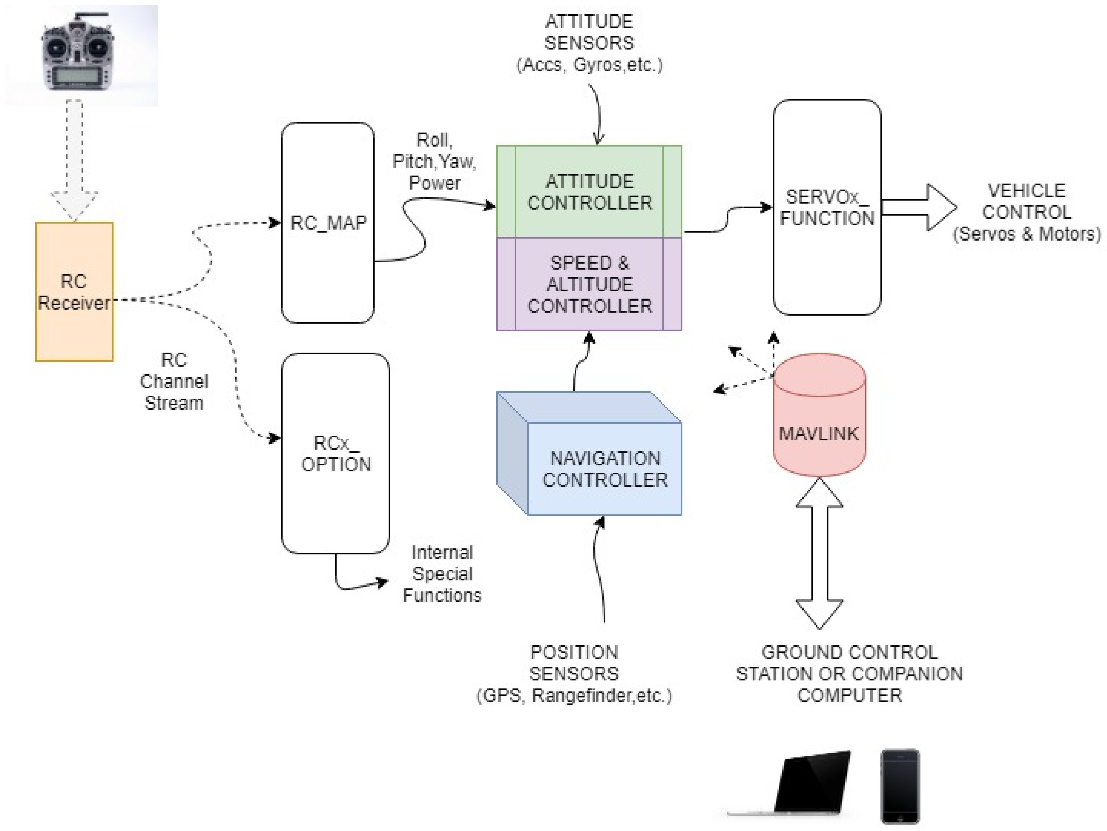

ATTITUDE SENSORS (Accs, Gyros,etc.)
RC Receiver
RC_MAP
Roll, Pitch,Yaw, Power
ATTITUDE CONTROLLER
SPEED & ALTITUDE CONTROLLER
SERVOx_ FUNCTION
VEHICLE CONTROL (Servos & Motors)
RC Channel Stream
RCx_ OPTION
NAVIGATION CONTROLLER
MAVLINK
Internal Special Functions
POSITION SENSORS (GPS, Rangefinder,etc.)
GROUND CONTROL STATION OR COMPANION COMPUTER

Figure 3.15: Ardupilot control architecture

Guided Mode: In this mode, the vehicle accepts navigation, altitude, attitude commands from the user or from software. This method is best suited while testing the hardware as well as the software.

Auto Mode: In auto mode, it is required that vehicle is loaded with predefined set of commands, and that once vehicle is set in auto mode, it starts following the commands and the user cannot bypass those commands unless mode is changed to guided or stabilize. This mode is suitable for general operation but not for testing or during unfavourable conditions.

Loiter Mode: This mode is similar to the position hold. At the instance mode is switched to the loiter, quadrotors hovers at the same location while planes circle around the location specified.

There also several other modes like altitude hold, stabilize etc., which also play a key role when needed.

3.5.2 Dronekit SITL

Dronekit [27] is a popular developer tool for UAVs. It offers API in python for customising the UAV flight. Using this several functions can be written on top of ardupilot as per user requirement and helps in executing it onboard itself using companion computers. Dronekit also offers dronekit-SITL for simulating the UAV with ardupilot flight stack onboard. Thus combination of dronekit-SITL and dronekit python API helps in testing out customised functions on the UAV without actual hardware . When the dronekit-SITL is started, it open up a tcp connection of the aircraft software model and user needs to connect to this tcp connection. Once the connection is established, user can send the Mavlink commands using the dronekit python API.

CHAPTER 4

Implementation and Results

4.1 Virtual Structure control

With reference to the virtual structure approach described in section 2.1.1, uni-directional formation control is implemented in this project. With crazyflie 2.0 as agent, controller is developed in ROS framework and results are mentioned in this division.

4.1.1 Controller design

As stated in section 2.1.1, the key steps in the control design is structure framing, goal generation, tracking controller design.

Structure Framing

Taking into account the number of agents available, suitable and desired structure has been framed by selecting an origin point and realising the structure nodes i.e., nodes' position with respect to the origin selected. This generates a three dimensional vector of **x**, **y**, **z** position for each node. These vectors act as reference for each agent in the formation acquisition. Each time the structure needs to be changed, following the similar process as above, vectors are generated.

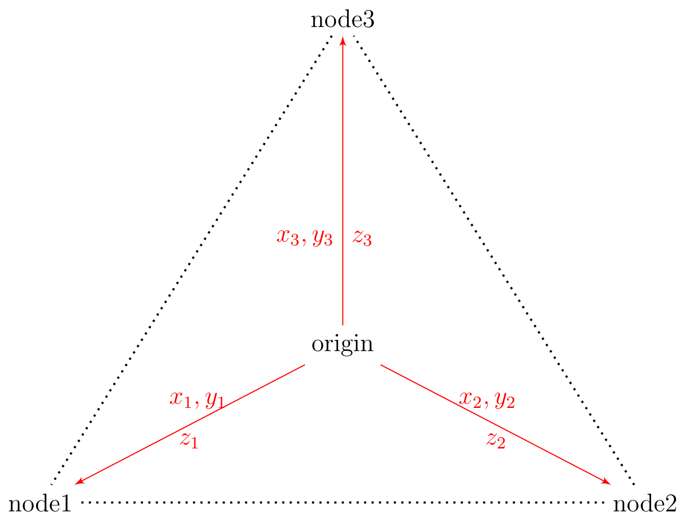


Figure 4.1: Virtual structure with 3 nodes

Goal generation

Once the node vectors have been generated, they act as targets for the formation acquisition. But formation maneuvering consists of translation, rotation, scaling properties.

Translation **T**(δx, δy, δz) include shifting the entire Virtual structure which is achieved by the shifting of the structure origin shown in equation 4.1

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_{new-origin} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{old-origin} + \begin{bmatrix} \delta x \\ \delta y \\ \delta z \end{bmatrix}_{translate} \tag{4.1}$$

Rotation $\mathbf{R}(\psi, \theta, \psi)$include rotation of structure by rotating the node vectors about the origin as depicted in eq 4.2 R_i is the rotation vector about the i_{th} axis

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_{new-node} = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}}_{R_z(\phi)} \underbrace{\begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}}_{R_y(\theta)} \underbrace{\begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{R_x(\psi)} \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{old-node} \tag{4.2}$$

Scaling $\mathbf{S}(\alpha)$include expanding or compressing the structure by a desired factor. Here α is the scaling factor

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_{new-node} = \alpha \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{old-node} \tag{4.3}$$

Performing the above operations generate the desired targets for each agent. In other words,

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_{Target_{i^{th}agent}} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{origin} + \mathbf{T}(\delta x, \delta y, \delta z) + \mathbf{S}(\alpha)\mathbf{R}(\psi, \theta, \psi) \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{node_i} \tag{4.4}$$

Tracking controller

In order for agents to track the desired node positions, The off-board PID position and yaw controller described in Fig 3.13 is utilised. It is embedded with the on-board architecture of the crazyflie 2.0 which is described in Fig 3.4. Together the combined control architecture looks as in Fig 4.2.

The tuned PID Values for 20% maximum overshoot are presented in table 4.1. The crazyflie quadcopter has right hand coordinate system of body axes. The local map formed by calibrating the motion capture system is left hand coordinate system. Hence when the crazyflie nose is aligned along the **x** axis of the local map, the **y** axis of the local map and the crazyflie are in opposite direction. Hence alter the movement of the crazyflie in **y** axis due to error between body frame and local frame, sign of the p, i, d values are altered.

Control	X	Y	Z	Yaw
P	40	-40	4950	200
I	2.1	-2.3	2200	20
D	20	-20	5300	0

Table 4.1: PID values for off-board position and yaw controller

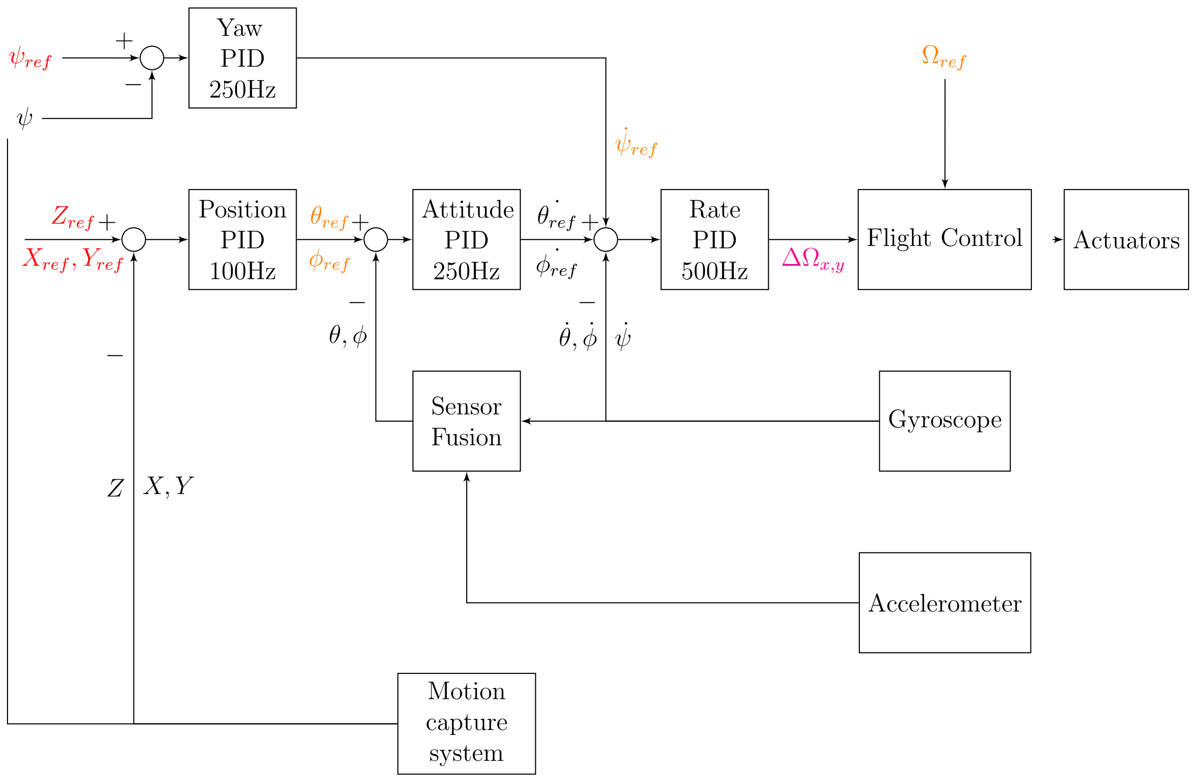


Figure 4.2: Offboard control embedded to Onboard controller

4.1.2 Results

The Structure realisation for 3 agents is mentioned in below table (realised coordinates are in meter)

	X	Y	Z
Origin	0	0	0
Node1	0.25	0	0.5
Node2	-0.25	1	0.5
Node3	-0.25	-1	0.5

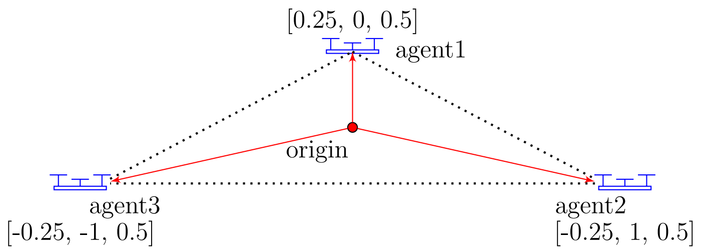


Figure 4.3: Virtual structure realisation

Formation Acquisition

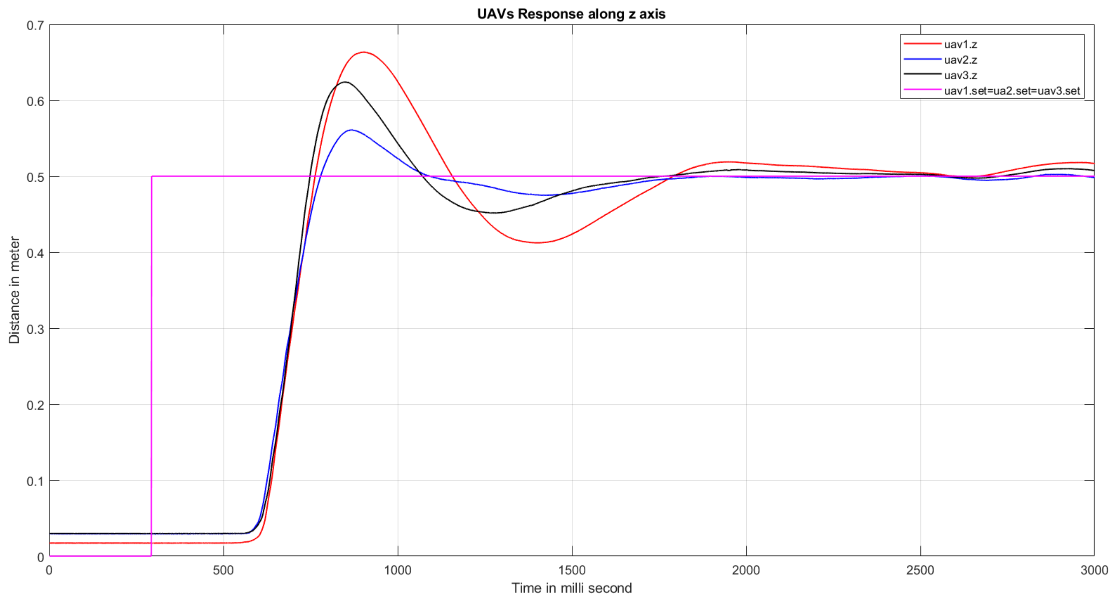


Figure 4.4: 3 UAVs response along Z-axis-Virtual Structure

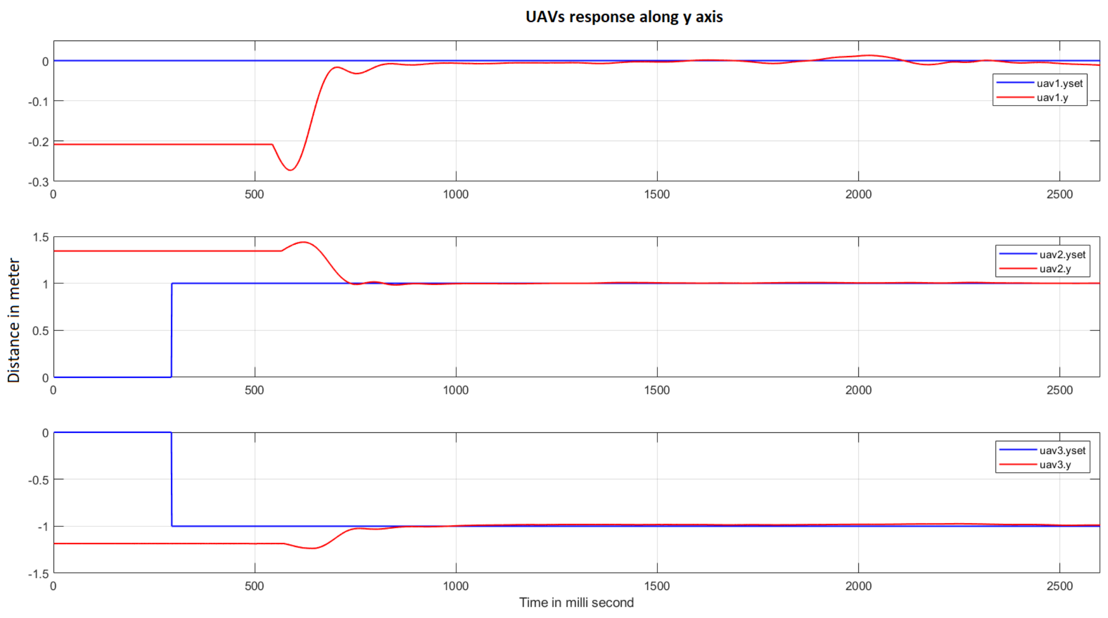


Figure 4.5: 3 UAVs response along Y-axis-Virtual Structure

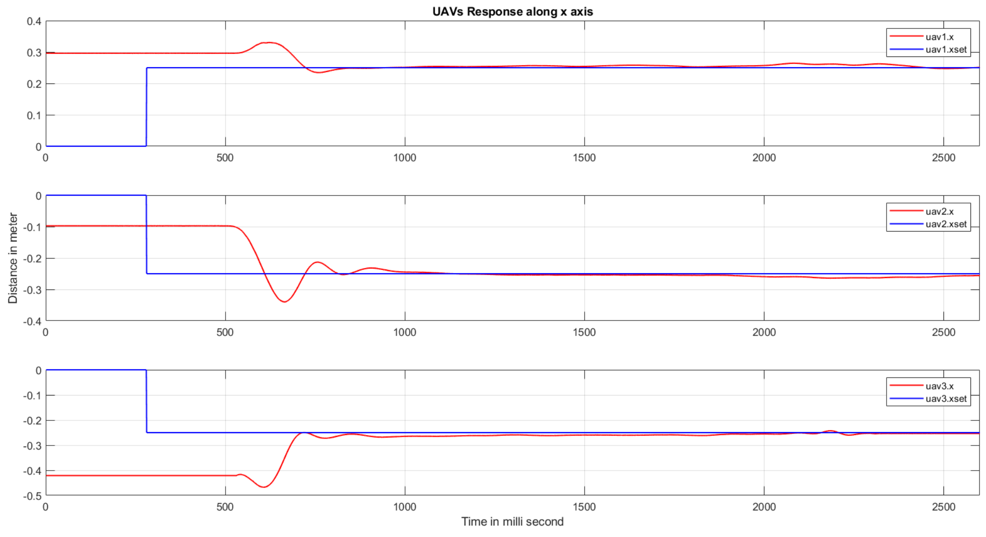


Figure 4.6: 3 UAVs response along X-axis-Virtual Structure

Formation Maneuvering

Translation $\mathbf{T}(\delta\text{x}, \ \delta\text{y}, \ \delta\text{z}) = \left[0.15m; 0.3m; 0m\right]$ applied to initial formation

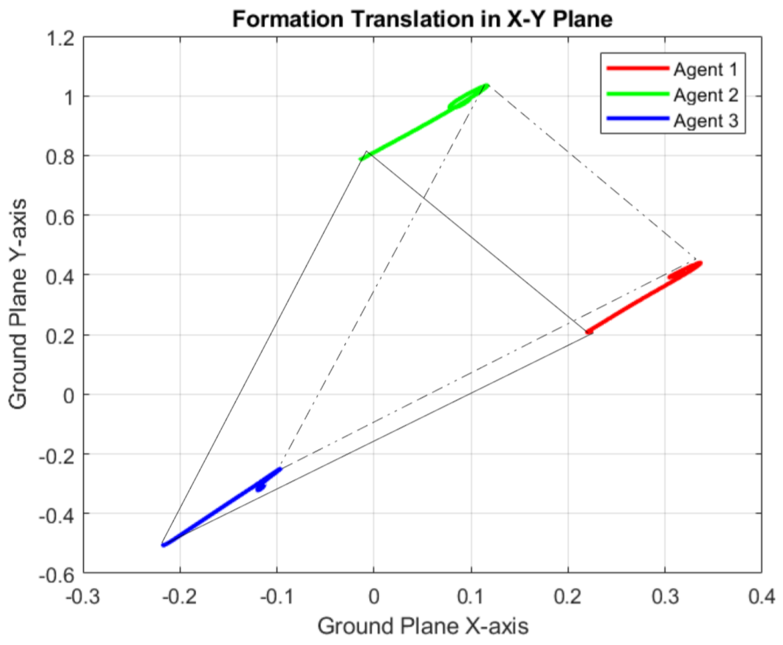


Figure 4.7: Formation Translation in X-Y plane-Virtual Structure

Scaling $\mathbf{S}(\alpha)$ where $\alpha = 1.5$ applied to initial formation structure

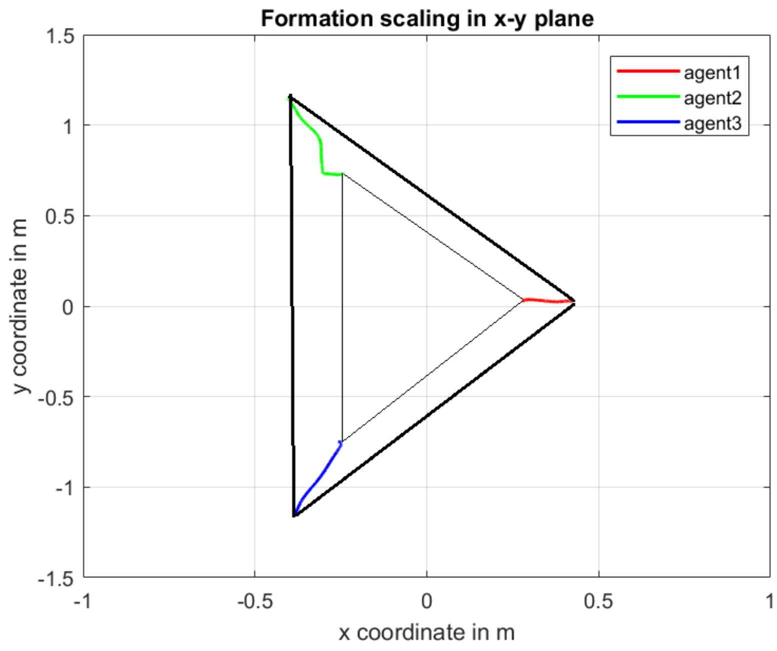


Figure 4.8: Formation Scaling-Virtual Structure

Rotation $\mathbf{R}(\phi, \theta, \psi) = R_x(0)R_y(0)R_z(90^o)$ applied to initial formation

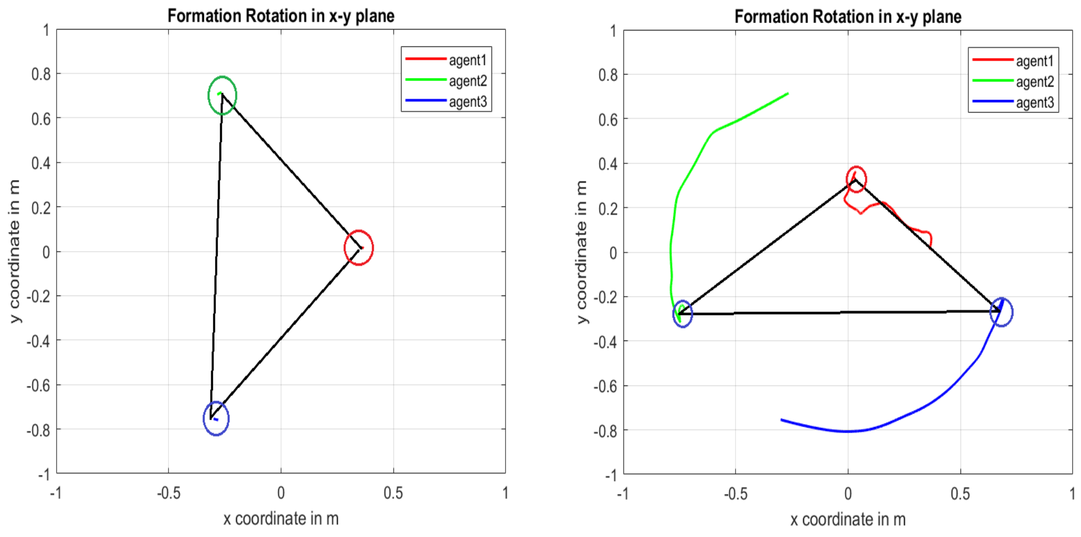


Figure 4.9: Formation Rotation about Z axis-Virtual Structure

The Fig 4.10 is graph of topics and nodes interconnections used to implement Virtual Structure formation.

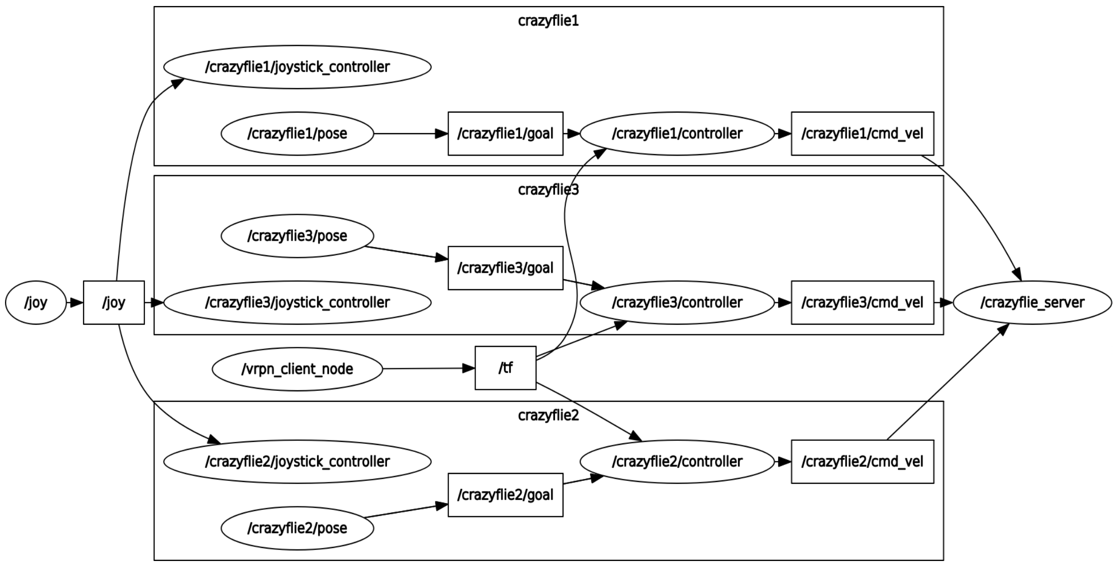


Figure 4.10: ROS Graph of the Algorithm Implementation-Virtual Structure

Source code stack for the implemented algorithm is avilable at `https://gitlab.com/iit-delhi/formation-control.git`

4.2 Leader Follower control

With reference to the leader-follower approach described in section 2.1.2, distance-bearing constraint formation control is implemented in this section. With crazyflie 2.0 as agent, controller is developed in ROS framework and results are mentioned in this section.

4.2.1 Algorithm implementation

As stated, distance-Bearing constraint is also known as position based approach which implies that the distance- bearing constraint is modified mathematically in terms of 3D positional arguments. As mentioned in section 2.1.2, the key steps in the control design is defining distance-bearing constraint, computing targets for the follower agents w.r.t to leader agent, tracking controller design for leader and follower.

Distance-Bearing constraint formation

In this implementation, altitude of both leader and follower are designed as the same and the bearing constraint is restricted to 2D plane.

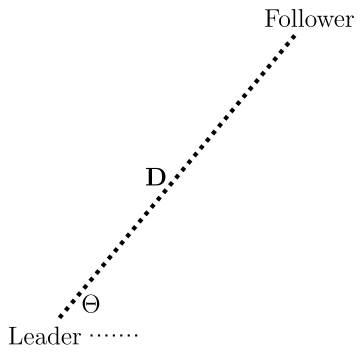


Figure 4.11: Distance bearing constrained leader follower formation

Let the **distance constraint** be **D** and **bearing constraint be** Θ. Leader agent target and current position are represented below.

$$\textit{Leader agent target position}: \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{leader_{target}}$$

$$\textit{Leader agent current position}: \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{leader_{curent}}$$

Follower target computation

Now that the leader agent target and its current position are known along with the distance-bearing constraint, follower target is computed.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_{follower_{target}} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{Leader-current} + \begin{bmatrix} D\cos\Theta \\ D\sin\Theta \\ 0 \end{bmatrix} \tag{4.5}$$

Tracking controller

In order for agents to track the desired target positions, The PID position and yaw controller combined with the on-board architecture as depicted in fig 4.2 is utilised in this implementation.

4.2.2 Results

The structure realisation for 3 agents is mentioned in below table (realised coordinates are in meter) with $D = 1.2m$ $\Theta = 153^o$

	X	Y	Z
Leader target	0.25	0	0.5
Follower offset	-0.5	-1	0
Follower target	-0.25	-1	0.5

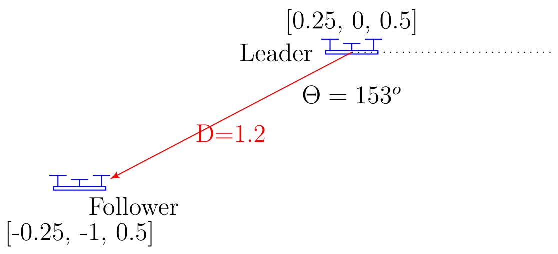


Figure 4.12: Leader follower formation realisation

Formation Acquisition

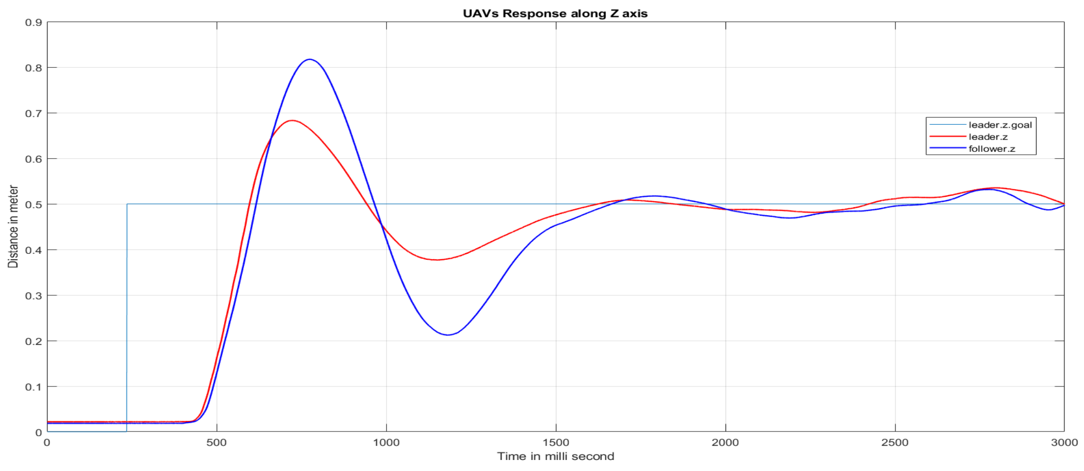


Figure 4.13: Leader and follower UAVs response along Z-axis

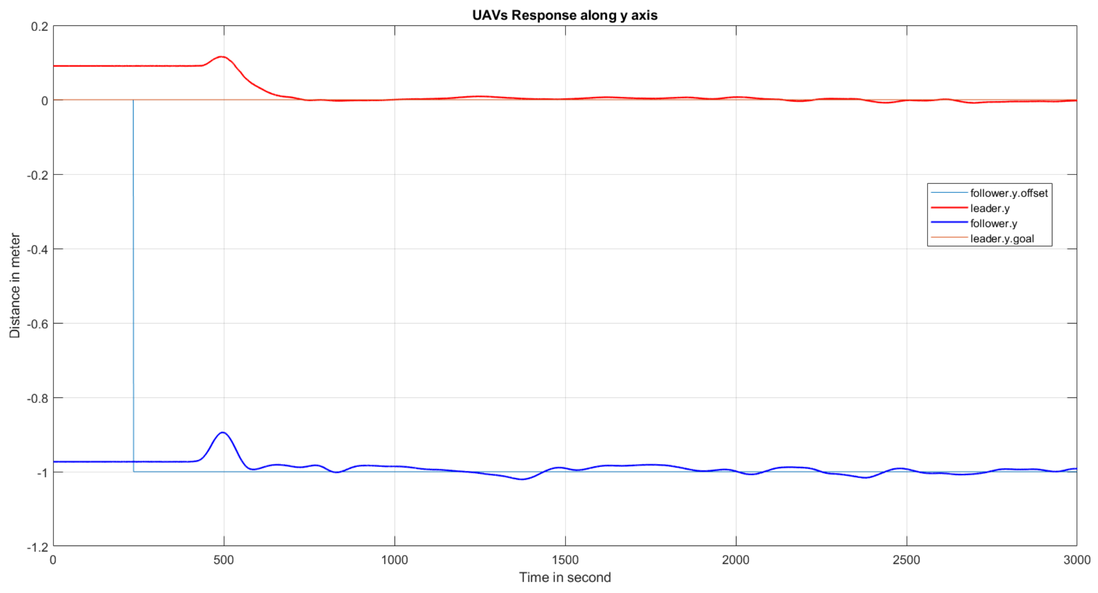


Figure 4.14: Leader and follower UAVs response along Y-axis-

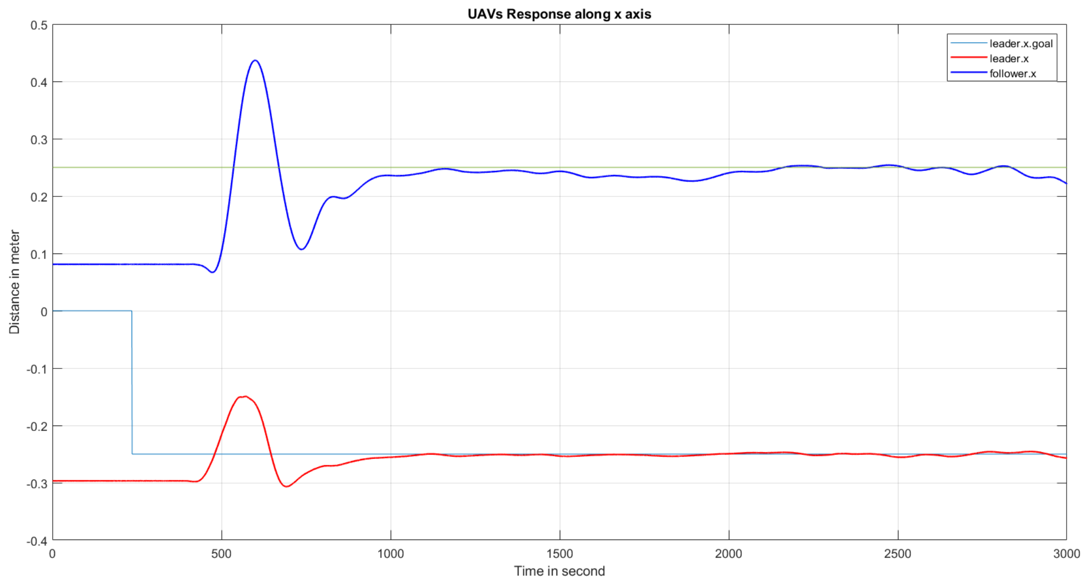


Figure 4.15: Leader and follower UAVs response along X-axis-

Below is the ROS graph of the topics and nodes used in this implementation. Source code stack for the implemented algorithm is avilable at `https://gitlab.com/iit-delhi/formation-control.git`

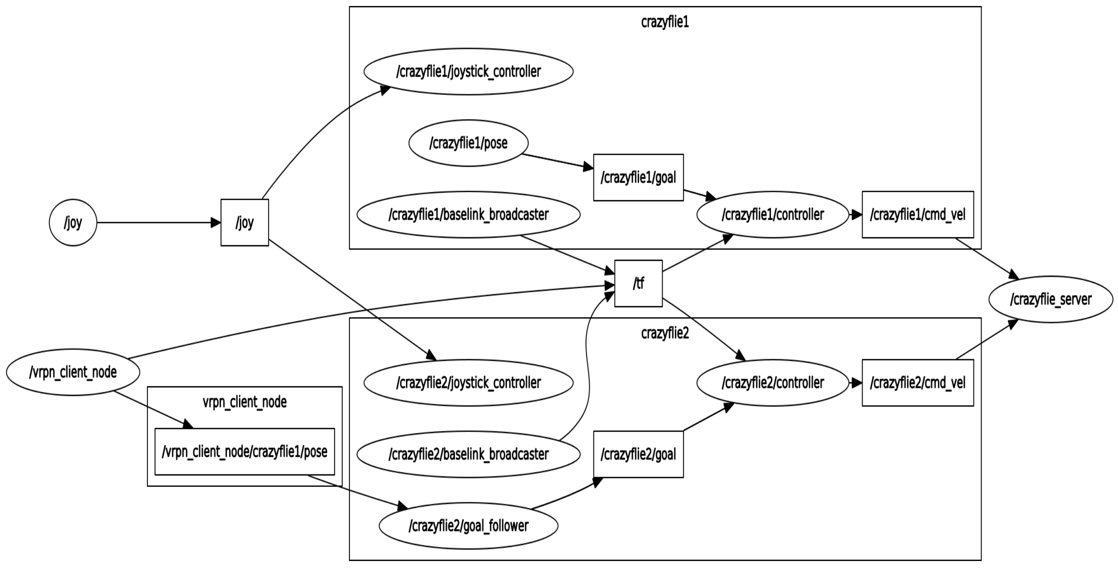


Figure 4.16: ROS graph of the leader follower Implementation

4.3 Graph Rigidity based control

The modified consensus control described in section 2.1.3 at Eq 2.6,2.7 is taken as the basis for this controller design. Ardupilot and Dronekit-SITL are used instead of Crazyflie 2.0 for this controller implementation. Simulations are done and the results attached in this section.

4.3.1 Controller design

The first step in the controller design is to realise the graph of the formation. A desired formation is realised by the 2D vectors $q_1 \ldots q_n$ in the X-Y plane. Once the formation has been realised, using the realisation a rigid graph is formed and the weighted adjacency Matrix A is derived. Eq 2.6 describes the velocity input for the single integrator model of the agent. But the velocity is with reference to the agent's body frame, which keeps changing with conditions like winds, unbalance etc., To compensate for this modification has been done

$$\dot{p}_i(t) = v_i \,, \qquad v_i = k \sum_{j \in N_i} \omega_{ij} \left(p_j(t) - p_i(t)\right) \tag{4.6}$$

$$p_i(t+1) = p_i(t) + v_i \cdot dt \tag{4.7}$$

where $p_i(t)$ is the current position of the agent i. In Eq 4.7 $p_i(t+1)$ is the updated position of the agent if at time t v_i has been applied. But since the ardupilot flight model cannot be taken as single integrator model due to yaw feature. Hence the modified consensus law in Eq 2.6 is altered as described in Eq 4.8

$$p_{i\,target} = p_i(t) + k \sum_{j \in N_i} \omega_{ij} \left(p_j(t) - p_i(t)\right) \cdot dt \tag{4.8}$$

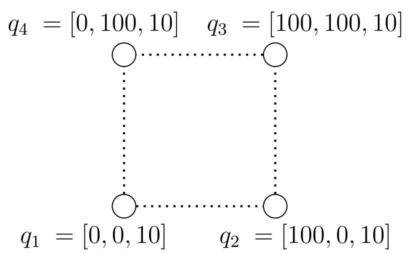
Now that the position targets for each agent are derived, using the navigation/position controller of the ardupilot and UTM library, the x, y targets are modified into the latitude, longitude targets and formation acquisition can be achieved. To embed the translation and scaling options,(rotation seems trivial here because this formation doesn't have any reference to rotate about) following changes are done to the control law. Translation $T(\delta x, \delta x, \delta z)$ and Scaling $S(\alpha)$ defined in 4.1.1 are used. The final control law is,

$$p_{i\,target} = p_i(t) + T(\delta x, \delta x, \delta z) + k \sum_{j \in N_i} \omega^*_{ij} \left(p_j(t) - p_i(t)\right) \cdot dt \tag{4.9}$$

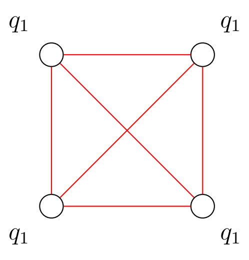
$$\omega^*_{ij} = (S(\alpha)\|q_i - qj\| - \|p_i - p_j\|) \tag{4.10}$$

4.3.2 Simulation results

A 2D square formation of 4 agents is desired and is realised with the following vectors(altitude is kept constant at 10m)



(a) Formation realisation



(b) Rigid graph realisation of the formation

Figure 4.17: Desired formation graphical realisation

$$\text{Weighted adjacency matrix A} = \begin{bmatrix} 0 & 100 & 141.4 & 100 \\ 100 & 0 & 100 & 141.4 \\ 141.4 & 100 & 0 & 100 \\ 100 & 141.4 & 100 & 0 \end{bmatrix} \tag{4.11}$$

Formation Acquisition

The quadrotor agents are spawned by the dronekit-SITL at different home locations and are made to take off to an altitude of 10m at their respective locations. Once all the agents are in the air, control law is applied

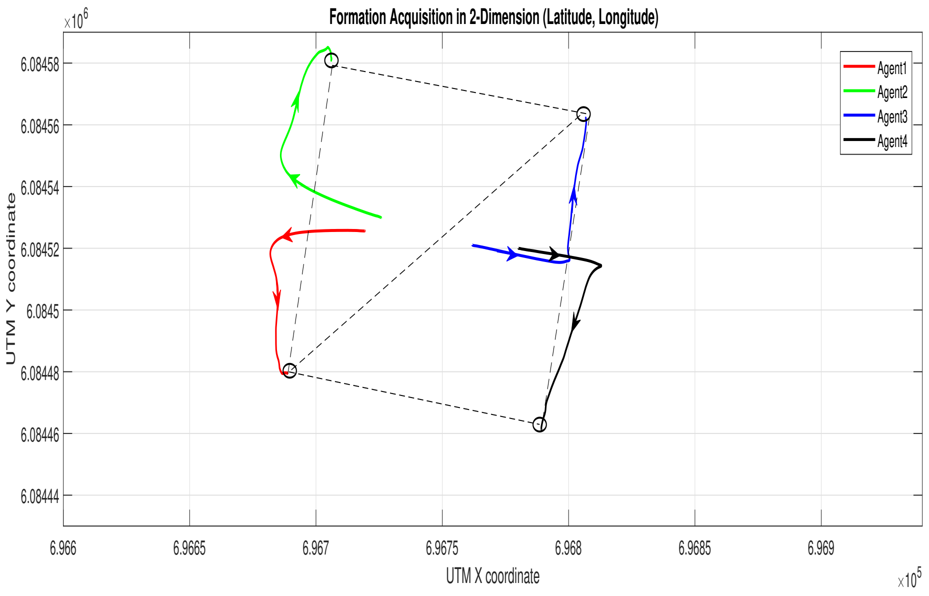


Figure 4.18: Formation acquisition of 4 UAVs-graph theoretic control

Formation Maneuvering

Translation $\mathbf{T}(\delta\text{x}, \ \delta\text{y}, \ \delta\text{z}) = \begin{bmatrix}100m; 0m; 0m\end{bmatrix}$ applied to initial formation Scaling $\mathbf{S}(\alpha)$ where $\alpha = 2$ is applied to initial formation structure

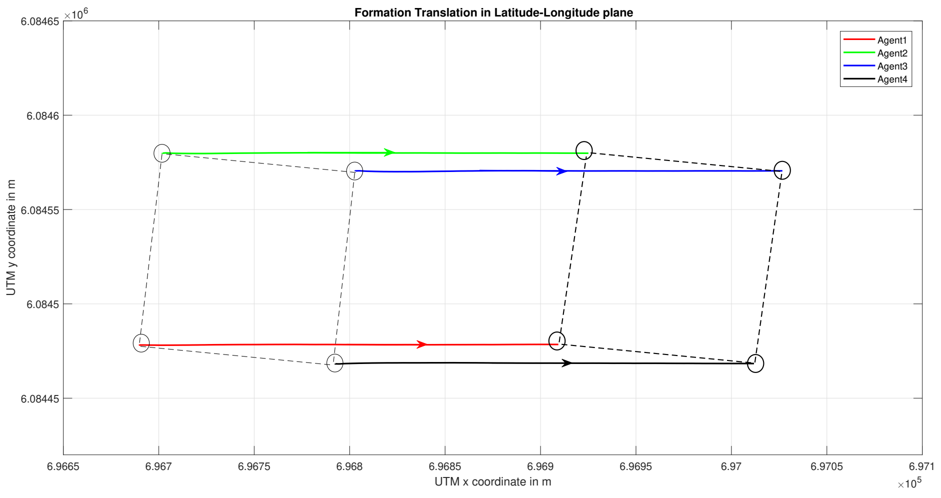


Figure 4.19: Formation translation-graph theoretic control

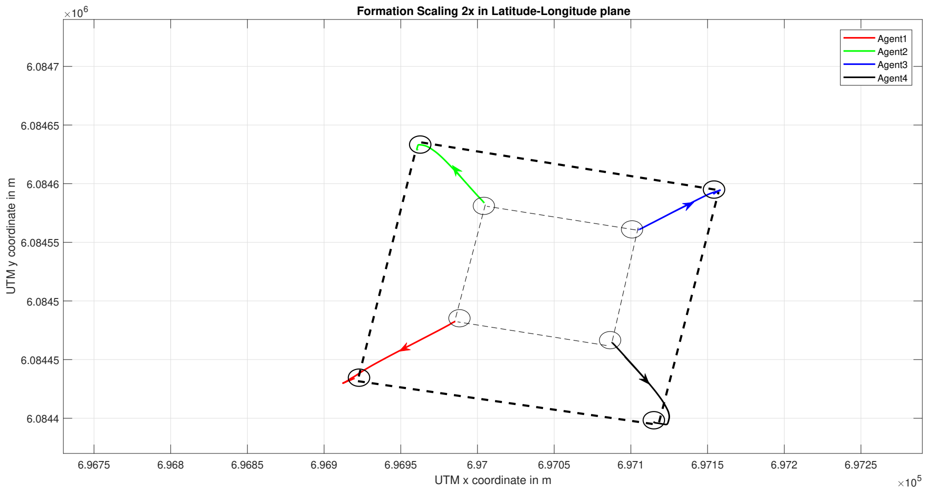


Figure 4.20: Formation scaling-graph theoretic control

4.4 Strategies comparison

In this section the comparison between above implemented formation strategies have been discussed with respect to several aspects.

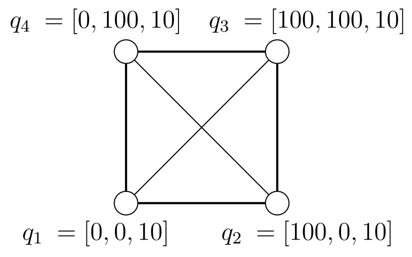
Aspect	Comparison
Formation Acquisition	While the agents start from random position, they should avoid collision with other agents while trying to reach their desired position in formation. In the case of uni-directional virtual structure there is no interaction between the agents while in Leader Follower approach too, there is no interaction between the follower agents. Hence chances of collision occurrence are high unless the agents are placed near their desired formation positions. In case of graph theoretic approach, if a rigid graph is chosen, chances of collision among agents are very less since there is agent to agent interaction existing i.e., each agents target is achieved by sensing the neighbouring agents distance from it.
Formation Maneuvering	Among the implemented, virtual structure algorithm has simple implementation of the maneuvering since it's realisation has a reference about which transformation can be applied. Next in level is the leader-follower approach also has a reference that is the leader. But rotation is complex than virtual structure since rotation can only be performed by follower agents around leader agent. For rotation of both leader and follower about each other, a reference point needs to be chosen which is against the basic nature of the algorithm. Hence complex trajectories are to be generated for their rotation. While in graph theoretic method, rotation is trivial and translation and scaling implementation have different approaches as seen in 4.9,4.10 which are mathematically tougher than the previous ones
Formation Preservation	As mentioned, since there is no interaction in the implemented uni- directional virtual structure method, effect on any agent is not transferred to the other and due to central goal generation, it has single point of failure hence poor formation preservation. While in leader-follower approach, effect on leader affects the followers hence the overshoots and similarity in response in 4.2.2, but the vice versa is false. It also has central goal generation for the leader and failing of it, fails the entire formation. In the graph theoretic method, the effect on an agent is shared by all of its neighbouring agents and hence minimal disturbance on the formation. since the control of each agent is decentralised, failure of any agent doesn't affect the entire formation although changes to formation occur.

Table 4.2: Formation strategies Comaparison

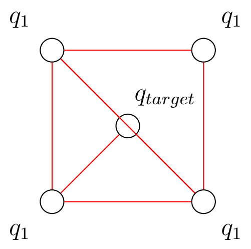
4.5 Target Interception

Among the above implemented Formation control algorithms, the best suitable for this is the graph theory based method. The reason being, as seen from the comparison in section 4.4, it has excellent formation acquisition and preserving. This method of formation control acts like a net trap, where in the agents can trap the target inside the formation without actually colliding with it with less complexity and calculations. While in the virtual structure and leader- follower methods, formation orientation needs to be adjusted dynamically to trap the target inside the formation.

The problem of pursuing a moving target occurs when it intrudes a perimeter. When the target agent intrudes a controlled zone, the agents starting from their respective positions, adjust their formation realisation and trap the target at the centroid of the formation as they move along. Below figures depict the realisation of the formation with and without a target.



(a) Rigid Formation realisation without target



(b) Rigid Formation realisation with target

Figure 4.21: Formation realisation modification for target interception

The weighted adjacency matrices for 4.21a and 4.21b realisations are

$$A = \begin{bmatrix} 0 & 100 & 141.4 & 100 \\ 100 & 0 & 100 & 141.4 \\ 141.4 & 100 & 0 & 100 \\ 100 & 141.4 & 100 & 0 \end{bmatrix}, \text{A}_{modified} = \begin{bmatrix} 0 & 100 & 0 & 100 & 70.7 \\ 100 & 0 & 100 & 141.4 & 70.7 \\ 0 & 100 & 0 & 100 & 0 \\ 100 & 0 & 100 & 0 & 70.7 \end{bmatrix}$$

The sequence of steps involved in this target interception are as follows:

- Realise the desired formation for the available agents and derive the weighted adjacency matrix

- As the intrusion occurs, modify the realisation and derive the modified adjacency matrix.
- Utilising the controller design 4.9,4.10, target in motion is trapped into the formation

4.5.1 Simulation result

The simulation is done by using dronekit-SITL to spawn the quadcopters and running script of the algorithm. Here the agents and the desired realisation chosen are same as depicted in 4.21.

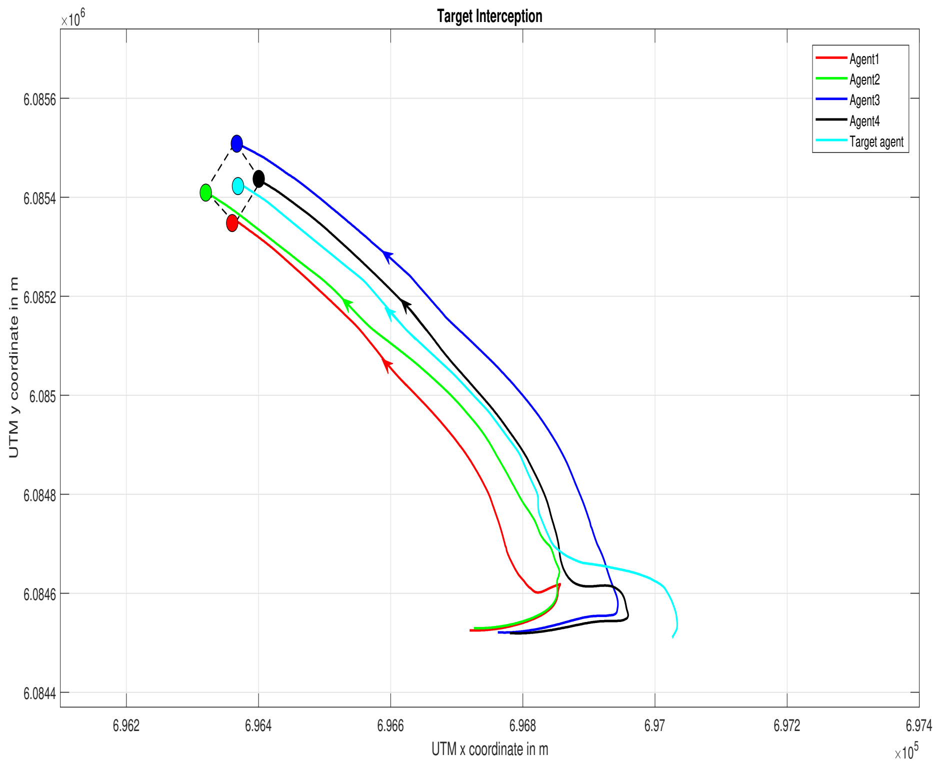


Figure 4.22: Target Interception

Source code stack for the implemented target interception algorithm is avilable at `https://gitlab.com/iit-delhi/formation-control.git`

CHAPTER 5

Conclusion

This thesis presents a platform capable of being used for researching aerial multi-agent systems control. Crazyflie quadcopters are utilised for experimentation along with the Robot operating system, and optitrack motion capture system for local positioning. Ardupilot flight stack with dronekit-SITL used for simulation.

Centralised control algorithms uni-directional virtual structure and leader-follower formation controllers were designed and implemented. Decentralised algorithm Graph theoretic controller was utilised and simulated. The algorithms performance was compared and concluded that graph theoretic approach provides superiority in formation preserving and acquisition and hence chosen for the target implementation. Finally target interception has been simulated with the help of graph theoretic approach results show the effective manner in which target is trapped inside the formation.

5.1 Future Scope

While this developed experiment platform is operational and could be used for demonstrating and research on algorithms (such as the one presented in chapter 4), there are still improvements that could be made to it, namely:

- Explore more advanced controllers for trajectory tracking such as [28] [29].
- Implement mesh-networking in the crazyflie swarm for peer to peer communication.
- Implement the off board controllers on board, thus leading to low communication delay.
- Conduct experiments with a larger number of agents (to explore network connectivity and communication issues).
- Implement path planning techniques such as [1] [30] for efficient maneuvering.

References

[1] S. Bhattacharya, M. Likhachev, and V. Kumar, "Multi-agent path planning with multiple tasks and distance constraints," in *2010 IEEE International Conference on Robotics and Automation*, 2010, pp. 953–959.

[2] K.-K. Oh, M.-C. Park, and H.-S. Ahn, "A survey of multi-agent formation control : Position-, displacement-, and distance-based approaches," 2012.

[3] Yang Quan Chen and Zhongmin Wang, "Formation control: a review and a new consideration," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005, pp. 3181–3186.

[4] Kar-Han Tan and M. A. Lewis, "Virtual structures for high-precision cooperative mobile robotic control," vol. 1, 1996, pp. 132–139 vol.1.

[5] T. Balch and R. C. Arkin, "Behavior-based formation control for multirobot teams," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 6, pp. 926–939, 1998.

[6] Wei Ren and R. W. Beard, "A decentralized scheme for spacecraft formation flying via the virtual structure approach," in *Proceedings of the 2003 American Control Conference, 2003.*, vol. 2, 2003, pp. 1746–1751.

[7] Mehran Mesbahi and Magnus Egerstedt, *Graph Theoretic Methods in Multiagent Networks*, ser. Princeton Series in Applied Mathematics, 2010.

[8] Marcio de Queiroz, Xiaoyu Cai, and Matthew Feemster, *Formation control of Multi-agent systems:A Graph Rigidity approach.* Wiley Series in Dynamics and Control of Electromechanical Systems, matlab code available at https://rb.gy/emh0cr.

[9] R. Olfati-Saber and R. M. Murray, "Graph rigidity and distributed formation stabilization of multi-vehicle systems," in *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.*, vol. 3, 2002, pp. 2965–2971 vol.3.

[10] L. Yi, S. G. Razul, Z. Lin, and C. M. See, "Target tracking in mixed los/nlos environments based on individual measurement estimation and los detection," *IEEE Transactions on Wireless Communications*, vol. 13, no. 1, pp. 99–111, 2014.

[11] Y. Lan, "Multiple mobile robot cooperative target intercept with local coordination," in *2012 24th Chinese Control and Decision Conference (CCDC)*, 2012, pp. 145–151.

[12] Bitcraze. Getting started with the Crazyflie 2.X. [Online]. Available: https://www.bitcraze.io/documentation/tutorials/getting-started-with-crazyflie-2-x/

[13] ——. Crazyflie 2.0 specifications. [Online]. Available: https://wiki.bitcraze.io/projects:crazyflie2:architecture:index

[14] Aerodynamic lift on propeller. [Online]. Available: https://en.wikipedia.org/wiki/Lift_(force)

[15] C. Luis and J. L. Ny, "Design of a trajectory tracking controller for a nanoquadcopter," *CoRR*, vol. abs/1608.05786, 2016. [Online]. Available: http://arxiv.org/abs/1608.05786

[16] H. S. Hasan, M. Hussein, S. M. Saad, and M. A. M. Dzahir, "An overview of local positioning system: Technologies, techniques and applications," *International Journal of Engineering Technology*, vol. 7, no. 3.25, pp. 1–5, 2018. [Online]. Available: https://www.sciencepubco.com/index.php/ijet/article/view/17459

[17] Motion capture system applications. [Online]. Available: https://en.wikipedia.org/wiki/Motion_capture

[18] Optitrack. Flex 13 motion capture camera. [Online]. Available: https://optitrack.com/products/flex-13/

[19] Optitrack documentation wiki. [Online]. Available: https://v110.wiki.optitrack.com/index.php?title=OptiTrack_Documentation_Wiki

[20] Vrpn streaming setup in motive. [Online]. Available: https://uclalemur.com/blog/optitrack-data-streaming

[21] Vrpn streaming to ros node. [Online]. Available: http://wiki.ros.org/vrpn_client_ros

[22] Robot operating system. [Online]. Available: http://wiki.ros.org/Documentation

[23] Crazyflie ros package. [Online]. Available: https://github.com/whoenig/crazyflie_ros

[24] A. Koubaa, *Robot Operating System (ROS)*, ser. Studies in Computational Intelligence. Springer International Publishing, vol. 3.

[25] Ardupilot flight stack. [Online]. Available: https://ardupilot.org/dev/docs/learning-the-ardupilot-codebase.html

[26] A. Koubâa, A. Allouch, M. Alajlan, Y. Javed, A. Belghith, and M. Khalgui, "Micro air vehicle link (mavlink) in a nutshell: A survey," *IEEE Access*, vol. 7, pp. 87 658–87 680, 2019.

[27] Dronekit software in the loop. [Online]. Available: https://dronekit-python.readthedocs.io/en/latest/develop/sitl_setup.html

[28] H. Yamaguchi, "Adaptive formation control for distributed autonomous mobile robot groups," in *Proceedings of International Conference on Robotics and Automation*, vol. 3, 1997, pp. 2300–2305 vol.3.

[29] W. Jasim and D. Gu, "Robust team formation control for quadrotors," *IEEE Transactions on Control Systems Technology*, vol. 26, no. 4, pp. 1516–1523, 2018.

[30] V. R. Desaraju and J. P. How, "Decentralized path planning for multi-agent teams in complex environments using rapidly-exploring random trees," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 4956–4961.