UNIVERSITY OF CALIFORNIA
RIVERSIDE

Experimental Implementation of Distributed Time-Varying Optimization Algorithms Using Crazyflie Platform

A Thesis submitted in partial satisfaction
of the requirements for the degree of

Master of Science

in

Electrical Engineering

by

Yifan Zhang

June 2019

Acknowledgments

Firstly I would like to thank Dr. Wei Ren to be my advisor. He put time and efforts to help me through the process of my thesis and let me have access to the lab resources. He also gave me advice on writing this thesis paper, without whose help this thesis would not exist. Secondly I would like to thank my parents for the support they gave me, both from physical aspects and spirit aspects. It was a long and hard way to get here writing these words. And it was tough to raise a child to get a graduate degree from the very beginning without the experience of being graduate students themselves. But they always support me through all the obstacles. Thirdly, I would like to thank the co-worker in COVEN lab, especially Shan Sun. She had endorsed me so much through the process of designing experiment, coding, and writing this thesis. I would not have done this experiment without the help from her. The other members from COVEN lab also helped a lot. Pengxiang Zhu helped me turn on VICON tracking system every time I needed, Bo Wang gave me precious advice when I got bugs on my program, former lab member Yanzhi Wu helped me control the device while I had to record the experiment, Hanzhe Teng helped me about the basic control of the device used in the experiment, former lab member Qianjun Liu accompanied me with the process from the beginning of this project, and more. Everyone in COVEN lab in the past year had helped me somehow. And Finally, I would like tothank my landlord for being so nice to me during the past months, I would like to thank my neighbors for accompanying me, I would like to thank my friends for giving me advice and being nice to me, And I would like to thank my committee members, they agreed my request to be my committee members, I appreciate that.

To my parents and everyone helped me for all the support.

ABSTRACT OF THE THESIS

Experimental Implementation of Distributed Time-Varying Optimization Algorithms
Using Crazyflie Platform

by

Yifan Zhang

Master of Science, Graduate Program in Electrical Engineering
University of California, Riverside, June 2019
Dr. Wei Ren, Chairperson

Formation control and trajectory following using distributed optimization algorithms is validated in this thesis. The crazyflie platform is used for validating the algorithms. Then the implementations on different algorithms and different numbers of robots take place. Firstly, distributed optimization algorithm with general time-varying cost functions is implemented to make the crazyflies follow a line trajectory in a square formation. Then another distributed optimization algorithm with time-varying cost functions and non-identical hessian is used to get the crazeflies follow a circle trajectory in a square formation. After achieving relatively high accuracy of implementation of the two distributed optimization algorithms, a target tracking task is performed to prove the feasibility of the algorithms. In all, this thesis explores a new way for distributed optimization algorithms on multi-agent systems such as UAVs.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Distributed optimization problems are attracting attention nowadays. Distributed optimization represents an interesting aspect of a multi-agent system where all the agents work cooperatively to find the optimal solution for the whole team using only local information and local communication. This can be used in many situations such as temperature control using multiple heat generators or radiator fins [9] or a large number of robots trying to find and track something(either a landing point, a moving person, or something else depending on the needs) when they lose the GPS signal [10]. Typical usage is in the field of robotics where each robot acts as an agent. They will be assigned a local cost function individually and they also achieve the goal as a whole team, such as performing, cleaning or even measuring tasks where multiple robots could be much better than a single one. In such a situation, an algorithm to control multiple agents is essential and a distributed algorithm is better than a centralized algorithm. A centralized algorithm has a central store to store all the data and compute them in real time, which will cost too much computing power and communication between the agent and the center cannot be assured. And the optimization problem occurs when the agent only gets information about its own cost function and needs to communicate with the neighbors to calculate the optimal solution in real time. In this way, they will act as a group to find the optimal solution for the whole team.

In this thesis, distributed optimization algorithms with time-varying cost functions on small Unmanned Aircraft Vehicles (UAVs) are implemented to prove the feasibility of these algorithms on a real-world problem. Compared with running the algorithms in simulators, the implementation into the real-world can be significantly different. Each UAV is assigned with a local function serving as a trajectory for itself. The goal is to achieve a team optimal solution represented as a formation moving at the same speed following the same trajectory. The UAVs we used are crazyflies, which are small and easy to operate.

1.2 Literature Review

There are existing works on finding the theoretical minimum time-varying cost function between many agents[1]. In addition, there are some other related works on various platforms with different algorithms, such as [2], [3], and [5]. While they focus on formation control with collision avoidance or use video odometry to estimate the robot parameters or the coverage problem in an unknown environment, the focus in this study is mainly on the optimization demonstrated in [1]. However, the algorithm demonstrated in [1] has too many added assumptions. Our work focuses on how to remove or relax some of the assumptions while still maintaining the feasibility of the whole theory. Our experiment focused on the implementation of those algorithms.

1.3 Contribution

My contribution in this thesis is mainly in designing and building distributed time-varying optimization controller on Crazyflie. I designed and implemented the controller on the Crazyflie platform. This includes the following small contributions:

- Usage of Bugbot Simulation
- Usage of the Crazyswarm package with VICON
- Online Tracking Algorithm Designing
- Data Logging and Processing

1.4 Organization

Chapter 1 serves as an introduction to the point of the whole thesis with a discussion concerning the motivation and real-world possible usage of this thesis project and contribution.

Chapter 2 presents the algorithm from the mathematical basis to the meaning of the whole algorithm and also offers a guide through the simulation on Bugbot to demonstrate the theoretical feasibility of this thesis project.

Chapter 3 gives a guide through the hardware and software used in the implementation.

Chapter 4 Chapter 4 presents a discussion on the test results and error analysis of different algorithms on different numbers and formations of Crazyflies.

Chapter 5 offers conclusions and ideas for future work.

Chapter 2

Distributed Optimization with Time-Varying Cost Function (DOTVCF) Coverage

In this part, the mathematical foundation of the algorithms are discussed.

2.1 Problem Formulation

Consider a network with n agents. In a time-varying distributed optimization problem, all the agents work together to get the optimal trajectory. More formally,

$$\min \quad F(r) = \sum_{i=1}^{n} f_i(r_i, t) \quad s.t. \quad r_i = r_j \forall i, j \in (1, ..., n), \tag{2.1}$$

where $r_i \in \mathcal{R}^m$ denotes the states of agent i, $r = [r_1, \cdots, r_n] \in \mathcal{R}^{nm}$ is the aggregated states of all the agents, and $f_i(r_i, t)$ are convex functions known only to agent i.

A distributed time-varying optimization algorithm was proposed in [1]:

$$\begin{aligned} u_i &= -\sum_{j \in \mathcal{N}_i} \beta_{ij} \text{sgn}(r_i - r_j) + \phi_i \\ \dot{\beta}_{ij} &= ||r_i - r_j|| \\ \phi_i &= -H_i^{-1}(r_i, t)(\nabla f_i(r_i, t) + \frac{\partial}{\partial t} \nabla f_i(r_i, t)), \end{aligned} \tag{2.2}$$

where u_i denotes the control input of agent i; $\nabla f_i(r_i, t)$ and $H_i(r_i, t)$ are, respectively, the Gradient and Hessian of the local cost function $f_i(r_i, t)$ with respect to r_i; and sgn() is a sign function which can be used to restrict the range of input, as described in Figure 4.1.For the Crazyflies platform which are crazyflies, u_i stands for the speed command sent to the Crazyflies.

A graph is connected when there is a connected path for any two agents in this network. In a connected graph, there are no unreachable vertices.[14] The topology defines the relationship among the agents; note that it can be different from the final team formation and can have various shapes. Moreover, the topology only represents the communication relationship, has nothing to do with the team formation, and is a kind of connected graph. In the real world, because each Crazyflie has a diameter, they cannot be treated as a dot, so in this case, we need to add an offset to each algorithm. This can be carried out simply by substituting r_i with $\hat{r_i} = r_i + h_i$ in the algorithm. h_i is a constant that varies for each different agent. In this case, the original algorithm changes into the following one:

$$\begin{aligned}
u_i &= -\sum_{j\in\mathcal{N}_i} \beta_{ij}\mathrm{sgn}(\hat{r_i} - \hat{r_j}) + \phi_i \\
\dot{\beta}_{ij} &= ||\hat{r_i} - \hat{r_j}|| \\
\phi_i &= -H_i^{-1}(\hat{r_i}, t)(\nabla f_i(\hat{r_i}, t) + \frac{\partial}{\partial t}\nabla f_i(\hat{r_i}, t)),
\end{aligned} \tag{2.3}$$

In the experiments,the system dynamics were chosen as $\dot{r_i} = u_i$,where in this case, u_i is the speed. This dynamic model was used for all of the experiments in this study.

2.2 Algorithm Design

In this section,a brief discussion on how to arrive at the functions implemented on the Crazyflies is presented. Although we can obtain the basic algorithm from [1], the algorithm (2.2) presented in [1] only works in situations where all the ϕs are bounded, i.e., there exists a positive constant $\bar{\phi}$ such that $\|\phi_i\|_2 \leq \bar{\phi}$. Furthermore, algorithm (2.2) only works under identical Hessian assumptions.Hence, these assumptions are relaxed to make them fit real-world applications.

2.2.1 DOTVCF with Convex Cost Function

In (2.2), the ϕ_i needs to be bounded as the result might not converge for an unbounded ϕ_i. When ϕ increases, it dominates the whole equation, which may cause a failure of the consensus. Motivated by [15], we propose the following algorithm which can achieve distributed optimization under an unbounded ϕ_i:

$$\begin{aligned} u_i &= -\sum[||\phi_i|| + ||\phi_j|| + \gamma_i + \gamma_j]\text{sgn}(r_i - r_j) + \phi_i \\ \phi_i &= -H_i^{-1}(r_i,t)[\nabla f_i(r_i,t) + \frac{\partial}{\partial t}\nabla f_i(r_i,t)], \end{aligned} \tag{2.4}$$

where the term $||\phi_i|| + ||\phi_j|| + \gamma_i + \gamma_j$ is introduced to address the possibly unbounded ϕ, the term $\text{sgn}(r_i - r_j)$ is introduced to achieve consensus, and ϕ_i is introduced to achieve optimization. As long as the topology is connected and undirected, the Hessians of the local cost functions are identical,thus algorithm (2.4) can achieve distributed optimization.

To verify the effectiveness of algorithm (2.4), the following cost functions are chosen as an example. Note that this function is not the only choice for this algorithm; it is just an example used in the experiments.

$$f_i = (x_i - it)^2 + (y_i - it)^2 \tag{2.5}$$

where $x_i \in \mathcal{R}, y_i \in \mathcal{R}$ denotes the X-Y coordinates in r_i. The reason to choose cost functions (2.5) is that their corresponding ϕ_is are unbounded since while time approaches infinity, the ϕ_i do as well.

When placed into our algorithm(2.4), ϕ_i changes to this:

$$\phi_i = -[x_i - it - i, y_i - it - i]^T, \tag{2.6}$$

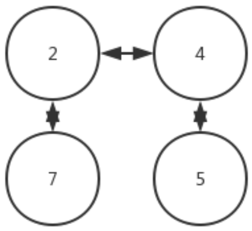


Figure 2.1: Topology for 4 agents

and the Hessian changes into this:

$$H_i = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

, Thus, algorithm (2.4) can be re-written as

$$\begin{aligned} u_i &= -\sum [||\phi_i|| + ||\phi_j|| + \gamma_i + \gamma_j]\mathrm{sgn}([x_i, y_i]^T - [x_j, y_j]^T) + \phi_i \\ \phi_i &= -[x - it - i, y - it - i]^T, \end{aligned} \tag{2.7}$$

In this form, u_i equals to the speed command that we send to each Crazyflie,while i represents the number of Crazyflies. u_i varies for each Crazyflie depending on the topology.

For example, Crazyflie 4 in the topology is set to be able to communicate with Crazyflie 2 and Crazyflie 5, so when the algorithm is implemented, the command sent to crazyflie 4 is as followa:

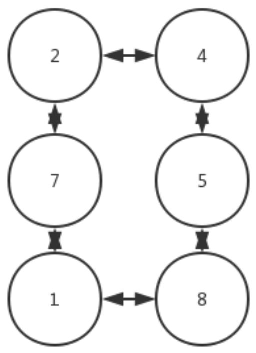


Figure 2.2: Topology for 6 agents

$$
\begin{aligned}
&u_4 = (\mathrm{sgn}(r_4 - r_5) + \mathrm{sgn}(r_4 - r_2)) * \\
&(norm([-it - i, -it - i]^T) + norm([-jt - j, -jt - j]^T) + \gamma_i + \gamma_j) \\
&- (r_4) + [-it - i, -it - i]^T.
\end{aligned} \tag{2.8}
$$

2.2.2 DOTVCF with Nonidentical Hessian

In the last part, the Hessian matrix is assumed to be identical.However, in reality, the Hessian miht be different for a different agent and this difference can be very large, which can affect the performance. In this section,this restriction is relaxed and nonidentical Hessians are assumed.

The original function is (2.2), where Hessians $H_i(r_i, t)$ are assumed to be identical. In this part, we propose the following algorithm is proposed to remove the identical Hessian assumption.

$$\begin{aligned} u_i &= -\beta H_i^{-1}(r_i,t)\sum_{j\in N_i}\operatorname{sgn}(r_i-r_j)+\phi_i \\ \phi_i &= -H_i^{-1}(r_i,t)[\nabla f_i(r_i,t)+\frac{\partial}{\partial t}\nabla f_i(r_i,t)], \end{aligned} \tag{2.9}$$

where β is the control gain to be designed and the inverse of the Hessian term is added in front of the sign function while the other parts remain unchanged. This can help because the sign function is used to combine all the agents together. Adding another term can counter the difference in the Hessians. In algorithm (2.9), the term $\beta H_i^{-1}(r_i,t)\sum_{j\in N_i}\operatorname{sgn}(r_i-r_j)$ is used to achieve consensus under the situation of nonidentical Hessians, and the term ϕ_i is introduced to achieve optimization. As long as the topology is connected and undirected, and the control β is big enough, the algorithm (2.9) can achieve distributed optimization.

In this part, function $f_i(r_i,t)$is chosen to be as follows. Again, this function can be very different, and this is just an example in the experiment, as there exists infinite numbers of choices for the cost function :

$$f_i = (ix_i-\sin(t))^2+(iy_i-\cos(t))^2 \tag{2.10}$$

which represents a circle in a real implementation.

By substituting $f_i(r_i,t)$ with the function provided above, our ϕ_i changes to become:

$$\phi_i = -H_i^{-1}[2ix_i-2\sin(t)-2\cos(t), 2iy_i-\cos(t)+\sin(t)]^T, \tag{2.11}$$

and Hessian H_i changes to become:

$$H_i = \begin{bmatrix} 2i & 0 \\ 0 & 2i \end{bmatrix}, \tag{2.12}$$

In this part, the topology is designed as in Figure 2.2.

For example, under the topology in Figure 2.2, Crazyflie number 4 should be able to communicate with Crazyflie number 2 and Crazyflie number 5.Thus, we have

$$u_4 = -\beta \text{sgn}(r_4 - r_5) - \beta \text{sgn}(r_4 - r_2) - [x_i - \frac{1}{i}\sin(t) - \frac{1}{i}\cos(t), y_i - \frac{1}{i}\cos(t) + \frac{1}{i}\sin(t)]^T \tag{2.13}$$

2.2.3 Moving Target Tracking

In the moving target tracking problem, the main purpose is to minimize the distance between target and agents(Crazyflies). Now $f_i(x_i, t)$, turns into :

$$f_i = (x_i - position^i_{target})^2, \tag{2.14}$$

where x_i denotes the position of agent i and $position^i_{target}$ denotes the position of the target sensed by agent i. By using this cost function, the agent can communicate with its neighbors so as to track a moving the target by minimizing the summation of all of the local cost functions.

In this experiment, the same algorithm as in section 2.2.1 is used, because it is an unbounded function, the target position cannot be guaranteed to be bounded, So ϕ_i is changed:

$$\phi_i = -(position_i^{target} + velocity_i^{target}) \tag{2.15}$$

$$\phi_i = -H_i^{-1}(r_i, t)[\nabla f_i(r_i, t) + \frac{\partial}{\partial t}\nabla f_i(r_i, t)]. \tag{2.16}$$

In reality, because the Crazyflies cannot crash and they may have different measurements of the same target because of error, each Crazyflie has different targets.

The topology and numbers of Crazyflies are the same as those in Section 2.2.1. All the Crazyflies will be assigned an identical target. The goal is to maintain the formation (as a representation of the topology) while tracking each target independently. This algorithm can deal with unbounded ϕ cases but not with nonidentical Hessian just like the first proposed algorithm.

Chapter 3

Hardware and Software

There are many UAVs serving in our daily life and research nowadays, such as in[10].In addition, there are other considerations about implementing the algorithm on other platforms such as P3-AT mobile robots[5]. However, Crazyflies are small and easy to operate [6] [4], thus they were used in the implementation of the algorithm.

3.1 Crazyflie Hardware

Crazyflies are small unmanned aircraft weighing 27 grams with only basic flight-control functions assembled. However, they can be used for more complex jobs such as motion capturing [6], [4], [13]. The basic components are a single Crazyflie 2.0 controller board, 2 clockwise propellers, 2 counter-clockwise propellers, 4 motor-mounts, a 240-mAh LiPo battery which can provide a flight time of up to 13 minutes[6], [4]. 4 coreless DC motors and a battery holder expansion board. The Crazyflies needed to be assembled after being purchased, as shown in Figure 3.1 with a slight modification with VICON markers.

The markers are used for motion capture in the VICON system.

The Crazyflie also had a radio named Crazyradio PA attached to an external computer's USB port. It is used to send commands to the Crazyflies from the computer, as shown in the Figure 3.2,

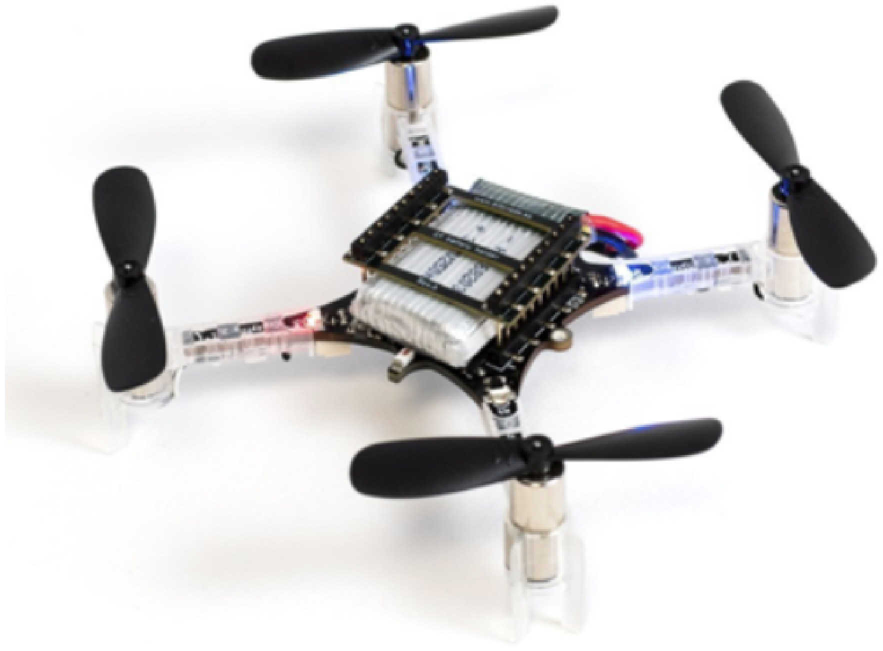


Figure 3.1: A Crazyflie

3.2 Official Client User Interface [11]

The official client window was enough to test the flight ability and enables the function to manually control a single Crazyflie. However, this client lacked the ability to execute external scripts, and for this reason, its use was limited to only flight test. Although its function was relatively limited, it still served as an essential part of the whole experiment.

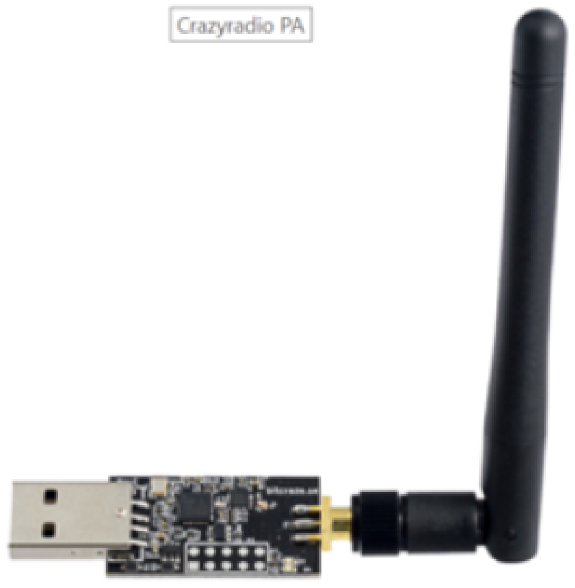
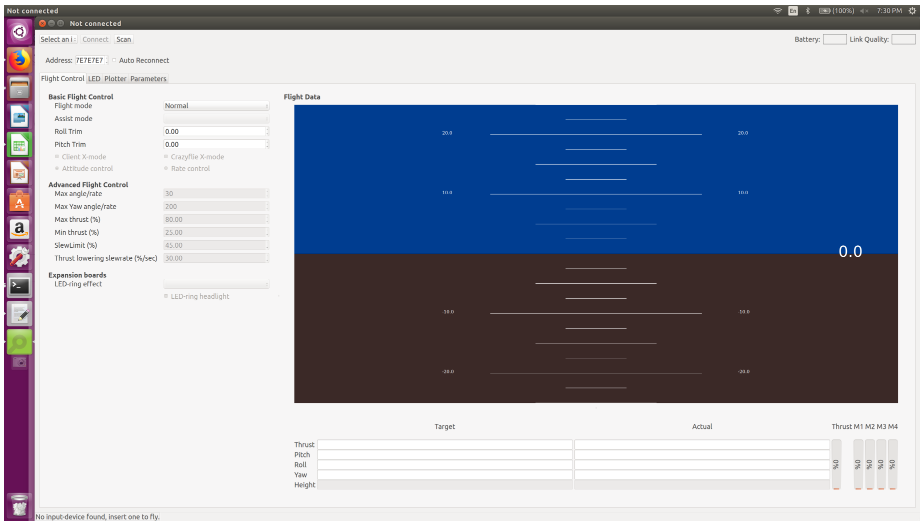


Figure 3.2: A Crazyradio

It enabled changing the communication address which made it possible to control a large number of Crazyflies at the same time by assigning a different address each time to each one. Figure 3.3 shows the user interface when it not connected to any Crazyflies.

When connected to a Crazyflie, the interface offers a choice says "connected on usb://0", then Moves cursor to "connected on usb://0", and some buttons then appear(the interface should be in full screen mode otherwise nothing show up). Press the connect button showing up menu, and press configure2.0; this window should show up as shown in Figure 3.4. In this window, the ability to change the address is offered. The radio channel parameter should be changed to another value between 0-100 when the number of Crazyflies exceeds 15 according to [4]. The radio bandwidth is the bandwidth the Crazyflies occupy when radio communication happens.



Address: 7E7E7E7 Auto Reconnect

Basic Flight Control

Flight mode: Normal

Assist mode:

Roll Trim: 0.00

Pitch Trim: 0.00

Client X-mode — Crazyflie X-mode

Attitude control — Rate control

Advanced Flight Control

Max angle/rate: 30

Max Yaw angle/rate: 200

Max thrust (%): 80.00

Min thrust (%): 25.00

SlewLimit (%): 45.00

Thrust lowering slewrate (%/sec): 30.00

Expansion boards

LED-ring effect:

LED-ring headlight

No input-device found, insert one to fly.

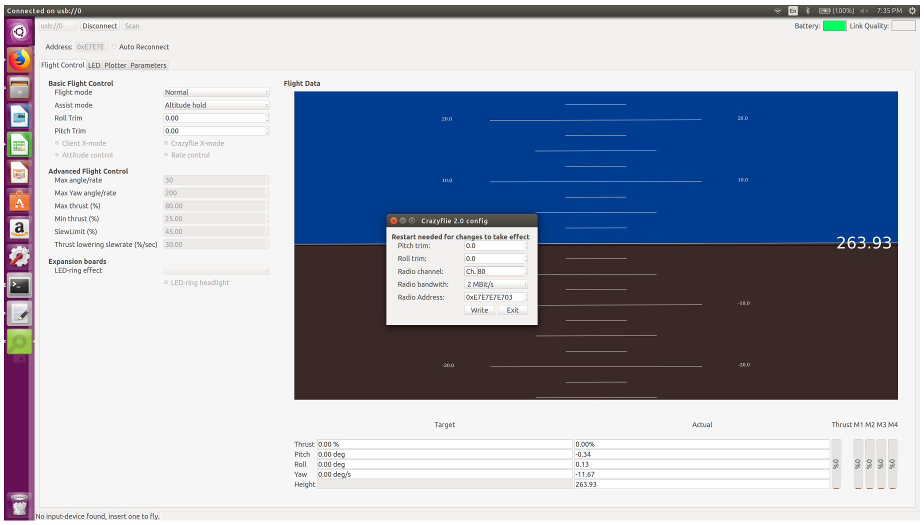
Figure 3.3: Manual controls and settings interface

Each Crazyflie should have a different address, but everything else can remain the same, which is true for 4 Crazyflies using Crazyswarm or the Crazyflie ROS platform (which were used in the experiment). For using Crazyswarm, changing the address to 0xE7E7E7E70X is necessary, where "X" represents a Crazyflie number. For example, for Crazyflie 3, this address should be "0xE7E7E7E703".

If a gaming pad is plugged into a computer and open the client window opened, the crazyflie Crazyflies can be controlled manually. This is not recommended because it highly depends on the gaming pad, and so can only be used it for entertainment.

3.3 The Crazyswarm Package

The Crazyswarm package was created by USC-ACT lab [6]and is well documented [7] on the website. It is relatively easy to control the Crazyflies to do complicated movements using this package, and according to the high-level commands available, it is easy to use for



Connected on usb://0

usb://0 Disconnect Scan

Address: 0xE7E7E7E ☐ Auto Reconnect

Battery: Link Quality:

Flight Control | LED | Plotter | Parameters

Basic Flight Control

Flight mode	Normal
Assist mode	Altitude hold
Roll Trim	0.00
Pitch Trim	0.00
Client X-mode	Crazyflie X-mode
Attitude control	Rate control

Advanced Flight Control

Max angle/rate	30
Max Yaw angle/rate	200
Max thrust (%)	80.00
Min thrust (%)	25.00
SlewLimit (%)	45.00
Thrust lowering slewrate (%/sec)	30.00

Expansion boards

LED-ring effect

LED-ring headlight

Crazyflie 2.0 config

Restart needed for changes to take effect

Pitch trim:	0.0
Roll trim:	0.0
Radio channel:	Ch. 80
Radio bandwith:	2 MBit/s
Radio Address:	0xE7E7E7E703

Write Exit

Flight Data

	Target	Actual
Thrust	0.00 %	0.00%
Pitch	0.00 deg	-0.34
Roll	0.00 deg	0.13
Yaw	0.00 deg/s	-11.67
Height		263.93

Thrust	M1	M2	M3	M4
0%	0%	0%	0%	0%

No input-device found, insert one to fly.

Figure 3.4: Channel Address Selection

controlling a large quantity of Crazyflies as testers only need to write a python script and run it in a separate terminal.

Crazyswarm has a complex structure. Basically, the low-level script, which is Crazyswarm server.cpp, registers some ROS services and ROS topics while launching radio communication. The high-level scripts can use these topics and services to send commands to the Crazyflies. In the experiment, the high-level programs were all written in python and used the API provided by the low-level Crazyswarm server.cpp.

3.3.1 Crazyswarm Simulation

Crazyswarm simulation is pretty easy to use; for example, the individual hover.py is under the path /Crazyswarm/ros ws/src/Crazyswarm/scripts. If we open a terminal under the same folder, type “python individual hover.py –sim”, the script mentioned can be tested via a simulation. However, the high-level API used to control the Crazyflies has not yet been implemented in a simulation and other ways to control the speed are not stable. This makes it difficult to simulate any 3D trajectory planning, and another offline simulation platform is also needed. As stated before, the Bugbot python library was used to test the code and the algorithm before implementing on real machines.

3.3.2 Crazyflie ROS Package [12]

Crazyflie ros package was developed by USC-ACT lab[6], and at the time writing, is still being maintained by them. It is a package which can serve as a low-level part to control the Crazyflies. In fact, Crazyswarm is based on this package. It contains its own method for controlling speed, which, however, is too complex; too many commands are needed for very simple movements. Nevertheless it provides a way to build the basis ROS structure for communicating between the Crazyflie and the host computer.

3.3.3 Configuration

To communicate with the VICON tracking system and using a Kalman Filter which is built in to Crazyswarm, it is necessary to be careful with the configuration. First, adding markers to the Crazyflies is needed, which should be assigned differently on each Crazyflie so the tracking system can keep track of them. The configuration used in this experiment for 8 Crazyflies is as in Figure 3.5. Next, to use Crazyswarm, all of the Crazyflies need the input of their initial positions for the initialization of the built-in Kalman Filter. According to [7], the initial position can have several centimeters of inaccuracy but the more accurate, the better. The initial Crazyflie positions for each experiment were stated for each experimental session. Following this, flashing the pre-built firmware into each Crazyflie used in the experiment was needed. This includes the stm32 and nrf firmware and the radio module. After launching Crazyswarm on a separate terminal, the Crazyswarm package was ready for use.

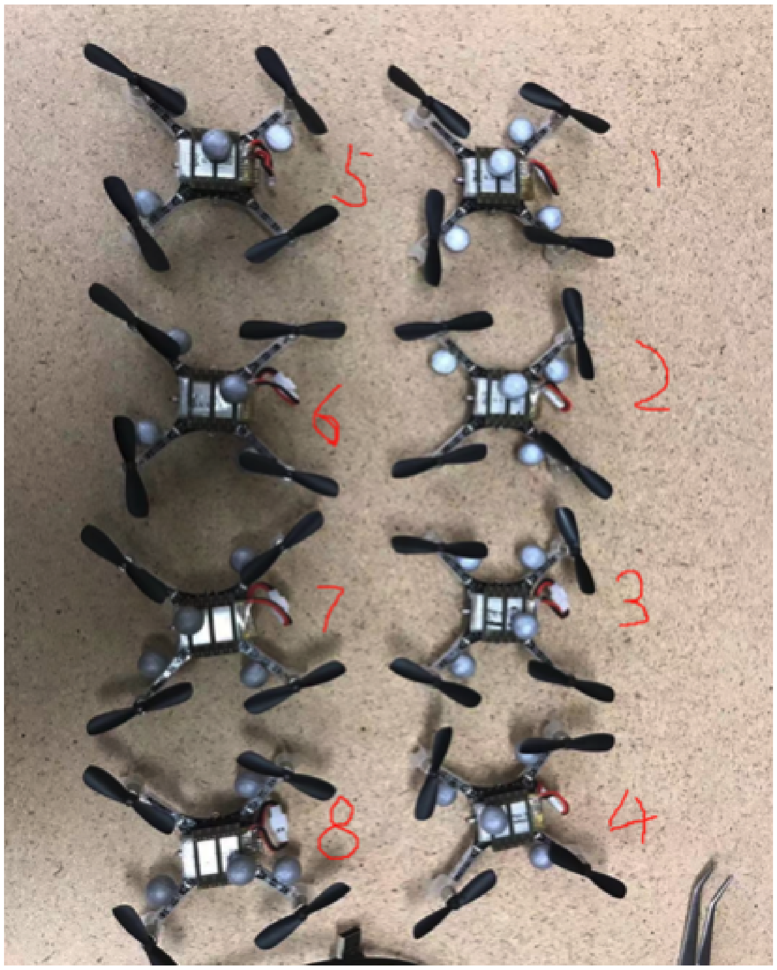


Figure 3.5: Crazyflies with a marker configuration

3.3.4 Crazyswarm Python API

There are many Python APIs provided in the Crazyswarm class, this class is defined in folder pycrazyswarm/crazyflie.py

The high-level APIs can be found in the appendix.

3.3.5 Added New Function

During the testing of the Distributed Optimization of Time-Varying function, a class object was created that used high-level commands to form a formation between the drones and to follow the trajectory dynamically using cmdFullState(). This class sets the distance between the different Crazyflies to maintain their formation while following the trajectory. It represents a dynamic online path planning algorithm which can be written as

$$u_i = \alpha * \sum_{j \in \mathcal{N}_i} \operatorname{sgn}(r_i - r_j - h_j) + \beta * (desire_{position} - r_i - h_i), \tag{3.1}$$

where u_i is the velocity and h_i represents the offset between each Crazyflie to prevent them from crashing and to maintain the formation. The desire-position parameter is given from outside the method and when provided, is added to the offset; h_i represents the offset to the target position of each Crazyflie. h_i sometimes is the same as h_j, but sometimes not because the offset depends on how the topology is defined.

Because this algorithm is similar to most consensus algorithm, it is believed that it has the potential to realize another consensus algorithm.

The added new functions can be found in appendix.

3.4 VICON Configuration

VICON system tracking calibration was a part of the experiment that caused many problems and is discussed briefly in this part. Figure 3.6 shows the VICON system user interface run on another computer in the lab.

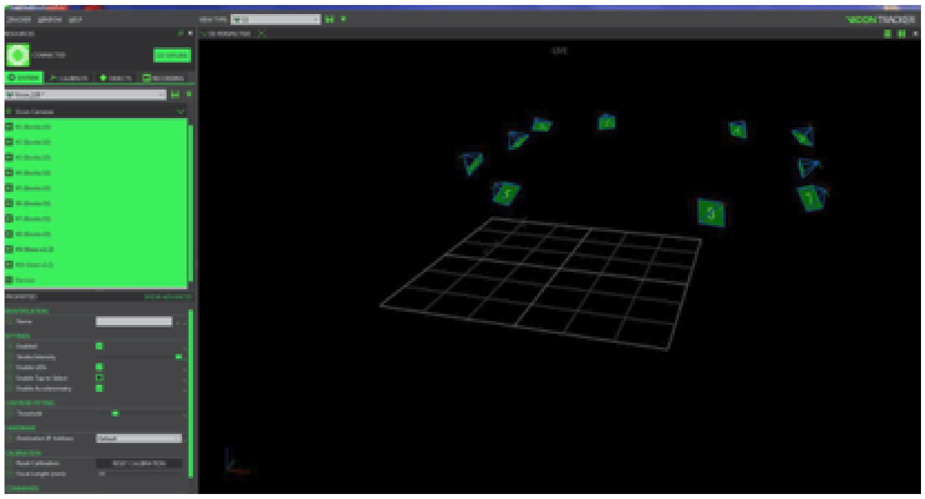


Figure 3.6: The VICON user interface

This interface also indicates whether all cameras used in the lab were functioning normally. To carry out the experiment, a new object needed to be added to this interface under the (object) tag.

After all of the cameras had been calibrated and all Crazyflies used in the experiment had been added to this interface, the origin needed to be set properly for each experiment. Because the lab is relatively small, the available space for the experiment was 4x4x2 m3. To make the Crazyflies move for at least half a minute in the lab, the origin was set to the corner in the experiment with DOTVCF and Convex Cost Function, for which an algorithm was implemented to make the Crazyflies follow a line. In another experiment with DOTVCF and Changing Hessian, we set the origin in the center of the lab because the Crazyflies were expected to follow a circle which requires as much spare space as could be provided.

Chapter 4

Tests and Results

In this part, we will talk about the experiment results regarding different algorithms presented in the chapters before. In each part, we will talk about initial set up positions, initial topology graph, initial parameters, and expected results.

4.1 Lab Environment Setup

Our lab has the empty space of 4 meters$\times$4 meters$\times$2 meters, the speed of crazyflie could not exceed the limitation of 1 m/s. And because the limitation of the platform stability, the speed in our algorithms should be somewhere near 0.5 m/s. All crazyflies start at randomly initialized places and take off to the height of 0.5 meter. The all experiment process will be at this height. We have 10 VICON cameras facing the center of the lab. 8 of them are Bonita cameras, 2 of them are Vero cameras. All crazyflies under experiment have different vicon markers assigned to them. These setup look different under all kinds of transformation including rotation and transition. At the end of the experiment when there is an emergency or the crazyflies fly out of the experiment region, an emergency stop signal is sent through the host computer and all crazyflies stop immediately.

4.2 DOTVCF with Convex Cost Function Results on Four Crazyflies

For DOTVCF with convex cost function algorithm we have such mathematical equation as in Equation 2.4.

This function represents a line starts from (0,0,0.5) to infinity in real-world, and it could be computed before experiment as we know the initial positions. All crazyflies should catch up with this trajectory by having an average trajectory as close as possible to it. However, because the length of the lab is not infinity in reality, we have to stop the program when crazyflies flying out of the research area.

In this equation, u is the speed. β and α are constants and could be changed during each test. They have big effects on performance. Also, the communication is limited. Because we setup the communication mode to only communicate to the neighboring crazyflie, the sum between i and j got down to only neighbors and the crazyflie itself, for example, if the crazyflie 2 only has a neighbor crazyflie 4, the sum would get down to $sgn(x2 - x4)$.

And the sgn() function is changed to the function shown below to have a smoother performance:

In our python scripts, we are sending speed commands in every loop, the execution time for each loop is relatively fast. To fix this, we only change the speed commands every

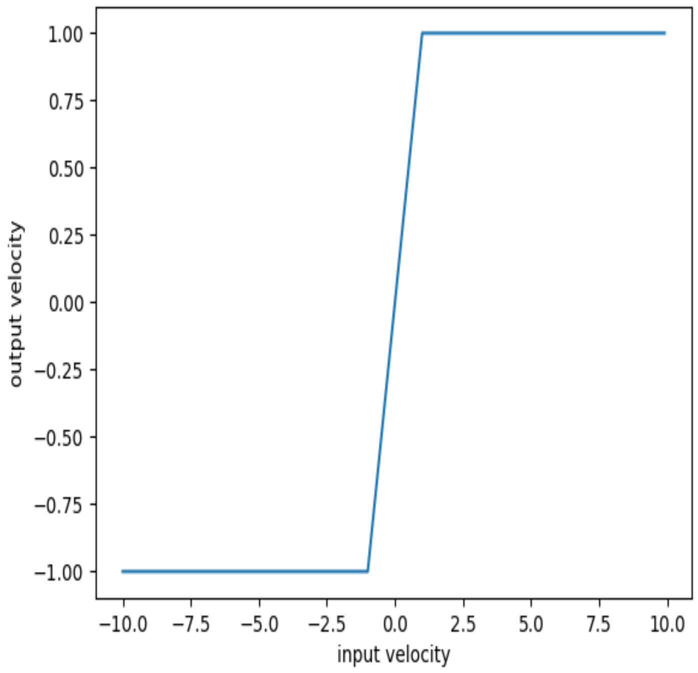


Figure 4.1: An approximation of sign function used in the experiment

few milliseconds. In this experiment, this was set to 0.01s which is 10 milliseconds.

At first, we were afraid that the speed of crazyflie would be too fast, so we set the beta to be 0.6 and assigned a parameter of 0.01 to ϕ. This made the crazyflies to converge to a formation fast. But this made the team moved too slowly to be observed.

Then we realized that set a parameter other than 1 to u_i is inappropriate. Then the parameter was set back to 1, and the parameter in front of β was kept as 0.6. Because it could not be larger than 1 in a randomly placed formation. Compulsively assigning parameters larger than 1 would cause crashing among the crazyflies. Also, the stability of crazyflies should be considered when choosing those parameters.

When we changed the parameter of β, a stable flight is achieved at the end of the flight. But shaking of crazyflies becomes a problem too obvious to be ignored.

Consequently, β should be kept as small as possible to achieve stable flights in a relatively long time. We chose β of 0.6, which is smaller than the ideally parameter but is proved in the later experiments that the crazyflies could have stable flight in a long time.

The initial position of 4 crazyflies are $r_{20} = [0.0, 0.5]^T$, $r_{40} = [0.0, 1.0]^T$, $r_{50} = [1.0, 0.0]^T$, and $r_{70} = [0.5, 0.0]^T$.

And the topology is presented as the 2.1. This means the crazyflie 2 could communicate with number 4 and number 7, the crazyflie 4 could communicate with crazyflie 2 and 5, the crazyflie 5 could communicate with number 4, the crazyflie 7 could communicate with the number 2. By communicating with each other, the crazyflie could get target's real-time position and offset it needs.

The result is shown in 4.2: The red line is average trajectory which should match

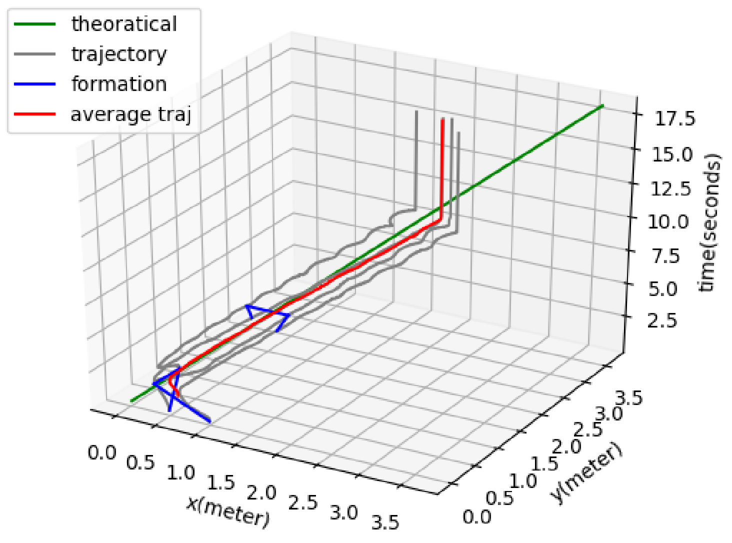


Figure 4.2: Results of 4 Crazflies

theoretical trajectory in theory. The green line is the theoretical trajectory. The blue lines are the connections between crazyflies. Notice that the z axis is time. There is a small gap between red line and green line about 10mm, which matches the error of 1 crazyflie flight control error. The crazyflies kept their formation during the test whiling following the designed trajectory. In conclusion, the result is as we expected.

Because the crazyflies are flying out of the lab, we manually stopped the crazyflies, so there is a sharp line at the end of the graph. It should not influence the result of this experiment.

4.3 DOTVCF with Convex Cost Function Results on Six Crazyflies

For DOTVCF with convex cost function we have equation as Equation 2.9: This is the same with the case of 4 crazyflies, the only difference is how the crazyflies are connected and the initial position. The sgn() function is kept the same as in Figure 4.1 The topology is as 2.2 ,where each of the crazyflie could communicate with its two neighbors but cannot know the whole topology. The reason for changing the diagram is to make the topology more complex and more convincing. Because when we only have 4 crazyflies, the topology is too simple that may bring out doubts that some of our agents could get almost all information in a topology.

The initial position for this experiment is randomly selected. It is presented in the Figure 4.3 by the first blue rectangle. During the flight, the crazyflies formation will change back to the one as in 2.2. The result is presented in Figure 4.3.

The result plot shows that the average trajectory (red line) is close to the theoretical trajectory (green line) and at the end of the flight, the distance between these two lines is about 0.1m, which is as we expected.

4.4 DOTVCF with Changing Hessian Results on Six Crazyflies

Everything else is not changed except the topology and the β parameter in our algorithm, comparing to the 4 crazyflies case. The topology used is shown as the 2.1. The

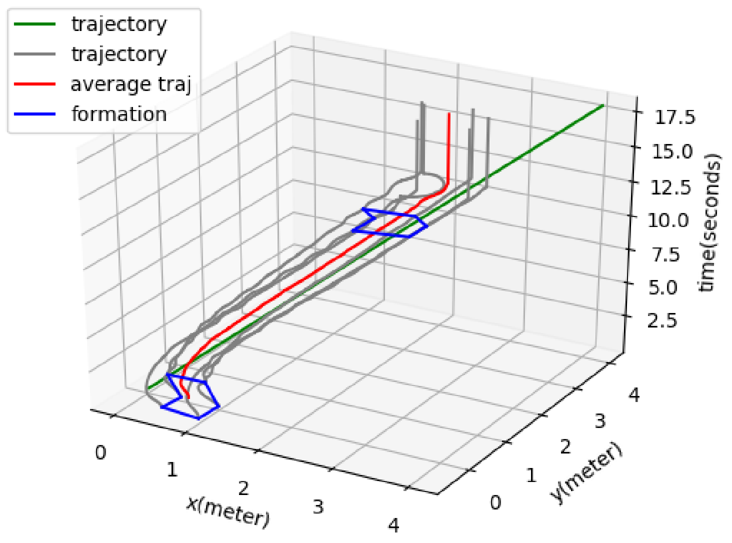


Figure 4.3: Results of 6 Crazyflies

initial position of crazyflies is randomly selected.

The initial position of 6 crazyflies are $r_{20} = [0.0, 1.0]^T$, $r_{40} = [0.5, 1.0]^T$, $r_{50} = [0.5, 0.5]^T$, $r_{70} = [0.0, 0.5]^T$, $r_{10} = [-0.5, 0.0]^T$, and $r_{80} = [0.0, 0.0]^T$.

The result is shown in Figure 4.4 The average line (green line) is close to the theoretical line(red line), the result is as we expected.

4.5 Experiment of Target Following With DOTVCF

In this experiment, we use 6 real targets instead of 6 virtual targets represented as a virtual functions f in the last algorithms. We use a board with 6 marked area noted on it and initialized its position randomly. And we tracked both the target and the crazyflie with VICON system. The position data of corresponding target will be sent to the correct

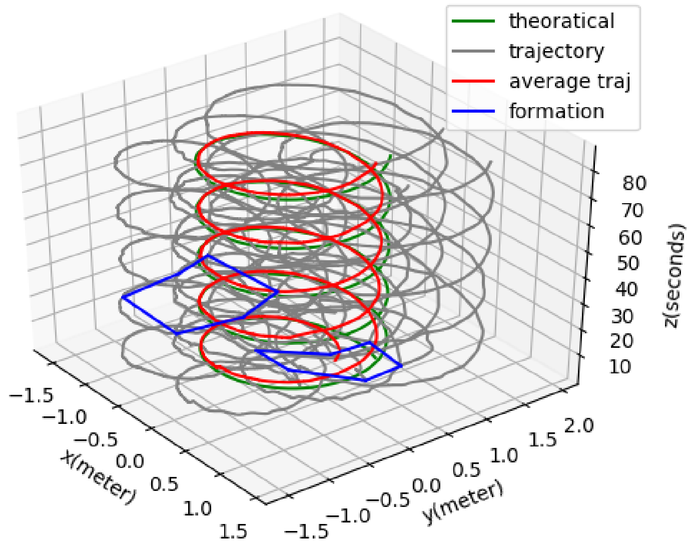


Figure 4.4: Result of DOTVCF on Six Crazyflies

crazyflie. For example, the position data of target 2 will be sent to crazyflie number 2 and crazyflie number 4 and crazyflie number 5 in our experiment, and then we will calculate the target velocity based on the position data we got when we communicate with VICON system.

The initial position of target is randomly selected as stated before, the initial position of crazyflies are the same as the last experiment as shown in Figure 4.5.

The process of experiment is as follows: the crazyflies will first take off to a specific height, and we set this height to be 0.5 meter out of the security concern. Then the algorithm takes control. The crazyflies will begin following the target meanwhile maintaining the formation. The reason we use a fixed height is because the height control is not one of our focus in this experiment and it is not accurate in the experiment.

The initial position is the same as in section 4.4, representing as in Figure 4.4.

The six targets are as shown in Figure 4.5. The markers on the whiteboard represents the targets, and there is a small cart which could be dragged by hand under the whiteboard. How it looks like is shown in Figure 4.5.

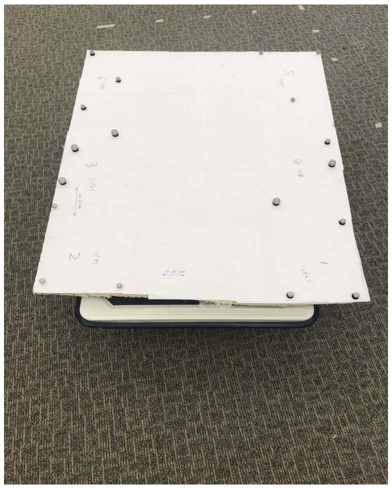


Figure 4.5: Experiment Targets

The expected result will be the crazyflies maintaining the height and formation while following the targets.

The experiment process is as shown in Figure 4.6

The result is shown in Figure 4.7

Z-axis is time, red line is target average position over time, green line is crazyflie trajectory average position over time, blue line is the formation at the beginning and a random moment during the process. The green line almost has no drift from the red line except the beginning moments when the crazyflies need time to become stable with the algorithm. In a word, the result is up to our expectation.

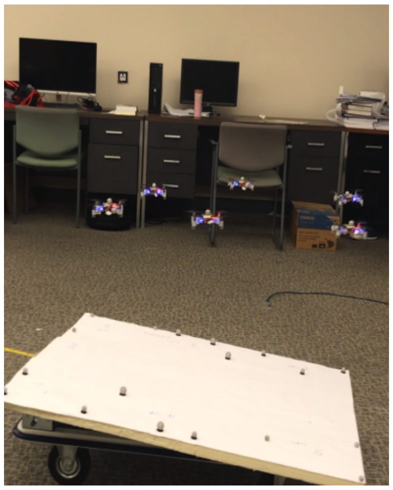


Figure 4.6: Experiment Process

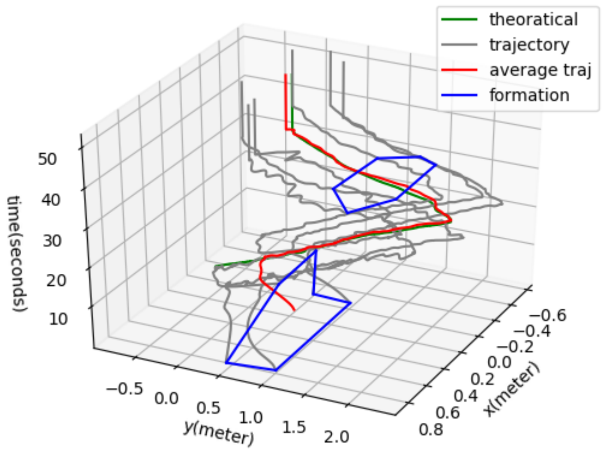


Figure 4.7: Target Tracking Results

Chapter 5

Conclusions

In our experiment, we did DOTVCF with convex cost function and changing hessian on 4 or 6 crazyflies. This has demonstrated the possibility of implementing the algorithm into the real world where there is limited information about the environment. Using our algorithm, maintaining the formation using only the local and the neighbor information (without the need to know the whole formation) is possible when there is a loss of communication to the GPS signal.

Bibliography

[1] S. Rahili ,W. Ren, Distributed Continuous-time Convex Optimization With Time-varying Cost Functions, IEEE Transactions on Automatic Control, VOL. 62, NO. 4, pp 1590-1605, 2017

[2] K. Fathian, S. Safaoui, T. H. Summers, N. R. Gans, Robust 3D Distributed Formation Control With Collision Avoidance and Application to Multirotor Aerial Vehicles, International Conference on Robotics and Automation, 2019

[3] G. K. Fourlas, Lamia, Fault detection approach for a $4-$ wheel skid steering mobile robot IEEE International Conference on Industrial Technology 2013

[4] J. A. Preiss, W. Hönig, G. S. Sukhatme, N. Ayanian. Crazyswarm: A Large Nano-Quadcopter Swarm (Extended Abstract), in Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, 2016.

[5] S. Rahili, J. Lu, W. Ren, U. M. Al-Saggaf Distributed Coverage Control of Mobile Sensor Networks in Unknown Environment Using Game Theory: Algorithms and Experiments, IEEE Transactions on Mobile Computing , VOL. 62, NO. 6, pp 1303 - 1313, 2018

[6] J. A. Preiss, W. Hönig, G. S. Sukhatme, N. Ayanian. Crazyswarm: A Large Nano-Quadcopter Swarm, in Proc. International Conference on Robotics and Automation, 2017.

[7] Crazyswarm document, https://crazyswarm.readthedocs.io/en/latest/

[8] Bitcaze, official website https://www.bitcraze.io/

[9] NASA, https://rps.nasa.gov/power-and-thermal-systems/power-systems/current/

[10] Drone project, https://support.dronedeploy.com/docs/volume-measurement-with-drones

[11] User Interface, https://github.com/bitcraze/crazyflie-clients-python

[12] W. H onig and N. Ayania, Flying Multiple UAVs Using ROS, Springer International Publishing, 2017.

[13] Light Painting With A Drone, https://www.hackster.io/krichardsson/light-paint-with-a-drone-d050af

[14] Wikipedia `https://en.wikipedia.org/wiki/Connectivity_(graph_theory)`

[15] L. Peng, W. Ren, Y.Song , J.A.Farrell, Distributed Optimization With the Consideration of Adaptivity And Finite-time Convergence, American Control Conference, 2014.

Appendix

Symbols

symbol	meaning	page
u	velocity	p 6,7,8,9,10,11
H_i	Hessian matrix	p 6,7,8,9,10,11
$position_i^{target}$	the i-th target position	p11
$velocity_i^{target}$	the i-th target velocity	p11
$sgn()$	signum function	p 6,7,8,9,10,11
∇	gradient as respect to x	p 6,7,9,10
$\frac{\partial}{\partial t}$	differential as respect to t	p 6,7,9,10
N_i	denotes the set of neighbors	p 6
$\mathcal{R}^m$	set of real domain with $m \times 1$ dimension	p 6,7
	self adaptive coefficient	p 6

Crazyswarm High-level Apis

setGroupMask()	set crazyflies as groups
takeoff()	take off
land()	land
stop()	emergency stop
goTo()	go to a stationary point
uploadTrajectory()	upload pre-defined trajectory
startTrajectory()	start pre-defined trajectory
Position()	get position coordinates
getParam()	get PID, accelerator parameters
setParam()	set PID, accelerator parameters
cmdFullState()	online path executor
cmdStop()	emergency stop
setLEDColor()	flash the LED lights

Added New Functions

`get_pos()`	return current position of the Crazyflie
`get_target_pos()`	return target neighbor position
`update_param()`	updates the position data into class private param.
`get_cf_v()`	returns and updates the velocity
`phi_restriction()`	additional constrains on *phi*
`update_cf_v()`	replacement of `get_cf_v` when using `phi_restriction()`