UNIVERSITY OF CALIFORNIA
RIVERSIDE

Experimental Implementation of A Distributed Time-Varying Constrained Optimization Algorithm by Using UAVs

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

Master of Science

in

Mechanical Engineering

by

Jie Xu

September 2020

Thesis Committee:

Dr. Wei Ren, Chairperson
Dr. Jun Sheng
Dr. Marko Princevac

The Thesis of Jie Xu is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

I am grateful to my advisor, without whose help, I would not have been here. I want to thank my lab fellows, especially Shan Sun, for all the help they offer during the past year.

To my parents for all the support.

ABSTRACT OF THE Thesis

Experimental Implementation of A Distributed Time-Varying Constrained Optimization Algorithm by Using UAVs

by

Jie Xu

Master of Science, Graduate Program in Mechanical Engineering
University of California, Riverside, September 2020
Dr. Wei Ren, Chairperson

In this thesis, a distributed time-varying constrained optimization algorithm is applied to a multi-robot multi-target navigation problem with experimental demonstration on a multi-Crazyflie platform. To illustrate the distributed time-varying constrained optimization framework proposed in [13], a multi-robot multi-target navigation problem is considered, where there are multiple robots and multiple moving targets in an unknown space in presence of obstacles. The objective is to have the robots stay close while simultaneously ensuring that each independent moving target stays in the detection range of at least one robot. Since there are unknown obstacles in the environment, collision avoidance should be guaranteed during the tracking process. The multi-robot multi-target problem can be formulated as the distributed time-varying constrained optimization problem by taking the distance between the targets and robots as objective functions and the distance between the robots and obstacles in the environment as constraints. The experiment on a multi-Crazyflie platform is implemented. The experimental results demonstrate the effectiveness of the optimization algorithm in [13].

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Distributed optimization algorithms allow for decomposing certain optimization problems into smaller, more manageable sub-problems that can be solved in parallel. Therefore, they are widely used to solve large-scale optimization problems such as optimization of network flows [8], big-data analysis [14], design of sensor networks [10], multi-robot teams [12], and resource allocation [7]. There has been significant attention on distributed convex optimization problems, where the goal is to cooperatively seek the optimal solution that minimizes the sum of private convex objective functions available to each individual agent.

Continuous-time algorithms play an important role in distributed optimization algorithms in the literature. The distributed continuous-time optimization algorithms have applications in coordinated control of multi-agent systems [11]. For example, multiple physical robots modeled by continuous-time dynamics might need to track a team optimal trajectory. Note that most studies in the literature focus on stationary optimization

problems in which both the objective functions and constraints do not explicitly depend on time. However, in many applications, the local performance objectives or engineering constraints may evolve in time, such as the multi-robot multi-target navigation problem. The multi-robot multi-target navigation problem involves multiple robots and multiple moving targets in an unknown space in the presence of obstacles. The objective is to have the robots stay close while simultaneously ensuring that each independent moving target stays in the detection range of at least one robot. Since there are unknown obstacles in the environment, collision avoidance should be guaranteed during the tracking process. The multi-robot multi-target problem can be formulated as a distributed constrained optimization problem with both time-varying objective functions and time-varying constraints by taking the distance between the targets and robots as objective functions and the distance between the robots and obstacles in the environment as constraints. In [13], a distributed continuous-time algorithm for time-varying constrained optimization problems is proposed. The goal of this thesis is to illustrate the effectiveness of the algorithm in [13] by implementing it in the multi-robot multi-target navigation problem using multiple unmanned aerial vehicles (UAVs).

Crazyswarm is a robot operating system (ROS) package that is developed to conduct experiments on the multi-robot systems [9]. The crazyswarm ROS package is well designed in the sense of ROS architecture and communication setup. It combines the use of the crazyflie drones and the vicon system to do precise trajectory planning and drone navigation. The package makes it possible to conduct experiments to verify the algorithm mentioned in this thesis.

1.2 Contribution

The contribution of this thesis is designing and validating the distributed time-varying constrained optimization algorithm with the crazyflies. This includes

1) the assemble of the crazyflie drones and the setup of the drones with the vicon system by using the crazyswarm ROS package, which includes the assignment of radio addresses, the selection of control and estimation strategies, and the flight test with manual control;

2) the experimental implementation of the distributed time-varying constrained optimization algorithm for the multi-robot multi-target navigation problem, which involves the setup of the robot communication topology, calculation of the targets and obstacles' positions, implementation of the distributed constrained optimization algorithm and generation of the velocity commands;

3) the collection of the experimental data and the analysis of the result.

1.3 Organization

Chapter 1 serves as the introduction of the thesis including the motivation and potential real world applications of this thesis. Main contributions are also mentioned.

Chapter 2 shows the problem definition. It also introduces the algorithm to be used.

Chapter 3 gives information of all the main facilities used to conduct the experiment.

Chapter 4 discusses the robot navigation problem and the experimental process

and results.

Chapter 5 offers conclusions of this thesis and future work.

Chapter 2

Distributed Time-Varying Constrained Optimization Algorithm

The details of the distributed time-varying constrained optimization algorithm proposed in [13] are presented in this section.

2.1 Preliminaries

2.1.1 Notation

Let $\mathbb{R}$, $\mathbb{R}^n$ and $\mathbb{R}^{m\times n}$ denote the set of real numbers, the set of n by 1 real vectors, and the set of m by n real matrices. Let $\mathbb{R}_{>0}$ represent the set of positive real numbers. Let $\mathbf{0}_n$ denote the vector of n zeros, and I_n denote the $n \times n$ identity ma-

trix. For a square matrix $A \in \mathbb{R}^{n\times n}$, A^{-1} denotes the inverse of A. For vector $x \in \mathbb{R}^n$, $\text{sgn}(x) = [\text{sgn}(x_1), \cdots, \text{sgn}(x_n)]^T$, where $x_i, i \in [1, n]$, denotes the ith element of x, and $\text{sgn}(x_i)$ denotes the signum function which is defined as:

$$\text{sgn}(x_i) = \begin{cases} 1 & \text{if } x_i > 0, \\ -1 & \text{if } x_i < 0, \\ 0 & \text{otherwise.} \end{cases}$$

Let $\nabla f(x,t)$ and $\nabla^2 f(x,t)$ denote, respectively, gradient and Hessian of function $f(x,t)$ with respect to the vector x.

2.1.2 Graph Theory

An undirected graph, is denoted by $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{A})$, where $\mathcal{V} = \{1, ..., n\}$ is the node set, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the edge set, and $\mathcal{A} = [a_{ij}] \in \mathbb{R}^{n\times n}$ is the weighted adjacency matrix with entries $a_{ij},\ i, j \in \mathcal{V}$. For an undirected graph, an edge (j, i) implies that node i and node j are able to share data with each other, and $a_{ij} = 1$ if $(j, i) \in \mathcal{E}$ and $a_{ij} = 0$ otherwise. Here $a_{ij} = a_{ji}$. Let $\mathcal{N}_i = \{j \in \mathcal{V} : (j, i) \in \mathcal{E}\}$ denote the set of neighbors of node i. A path is a sequence of nodes connected by edges. An undirected graph is connected if for every pair of nodes there is a path connecting them.

2.2 Problem Formulation

Consider a network consisting of n agents. Each agent is regarded as a node in an undirected graph, and each agent can only interact with its local neighbors in the network.

Suppose that each agent satisfies the following continuous-time dynamics

$$\dot{x}_i(t) = u_i(t), \tag{2.1}$$

where $x_i(t) \in \mathbb{R}^m$ is the state of agent i, and $u_i(t) \in \mathbb{R}^m$ is the control input of agent i. The goal is to design $u_i(t)$ using only local information and interaction, such that all the agents work together to find the optimal trajectory $x^*(t) \in \mathbb{R}^{m*n}$ which is defined as

$$\begin{aligned} x^*(t) = \quad & \sum_{i=1}^{n} f_i[x_i(t), t], \\ \text{s.t.} \quad & g_i[x_i(t), t] \leq \mathbf{0}_{q_i}, \quad x_i(t) = x_j(t), \quad \forall i, j \in \mathcal{V}, \end{aligned} \tag{2.2}$$

where $x(t) \in \mathbb{R}^{m*n}$ is the stack of all the agents' states, $f_i[x_i(t), t] : \mathbb{R}^m \times \mathbb{R}_{>0} \to \mathbb{R}$ are the local objective functions, and $g_i[x_i(t), t] : \mathbb{R}^m \times \mathbb{R}_{>0} \to \mathbb{R}^{q_i}$ are the local inequality constraint functions. It is assumed that $f_i[x_i(t), t]$ and $g_i[x_i(t), t]$ are known only to agent i. Note that, the optimal solution $x^*(t)$ is time varying and forms a trajectory.

The following assumptions are required.

Assumption 1 *The graph $\mathcal{G}$ is fixed, undirected and connected.*

Assumption 2 *All the objective functions $f_i[x_i(t), t]$ are uniformly strongly convex in x_i, for all $t \geq 0$.*

Assumption 3 *All the constraints $g_i[x_i(t), t]$ are uniformly convex in x_i, for all $t \geq 0$.*

Assumption 4 *For all $t \geq 0$, there exists at least one $y \in \mathbb{R}^m$ such that $g_i[y(t), t] < \mathbf{0}_{q_i}$ for all $i \in \mathcal{V}$. Therefore, the Slater's condition holds for all time.*

Assumption 5 *If all local states $x_i(t)$ are bounded, then there exists a constant $\bar{\alpha}$ such that $\sup_{t\in[0,\infty)} \|\frac{\partial}{\partial t}\nabla f_i[x_i(t),t]\|_2 \leq \bar{\alpha}$ for all $i \in \mathcal{V}$ and $t \geq 0$.*

Assumption 6 *If all local states $x_i(t)$ are bounded, then there exist constants $\bar{\beta}$ and $\bar{\gamma}$ such that $\sup_{t\in[0,\infty)} \|\frac{\partial}{\partial t}\nabla g_{ij}[x_i(t),t]\|_2 \leq \bar{\beta}$ and $\sup_{t\in[0,\infty)} \|\frac{\partial}{\partial t} g_{ij}[x_i(t),t]\|_2 \leq \bar{\gamma}$, for all $i \in \mathcal{V}$, $j \in [1,\cdots,q_i]$ and $t \geq 0$.*

The uniform strong convexity of the objective function implies that the optimal trajectory $x^*(t)$ is unique for all $t \geq 0$.

2.3 Algorithm Design

In order to find the optimal trajectory $x^*(t)$ defined in (2.2), we apply the following controller, which is proposed in [13], for agent i:

$$\begin{aligned} u_i(t) &= -\beta\{\nabla^2 \tilde{L}_i[x_i(t),t]\}^{-1} \sum_{j\in\mathcal{N}_i} \operatorname{sgn}[x_i(t)-x_j(t)] + \phi_i(t), \\ \phi_i(t) &= -\{\nabla^2 \tilde{L}_i[x_i(t),t]\}^{-1} \left\{ \nabla \tilde{L}_i[x_i(t),t] + \frac{\partial}{\partial t}\nabla \tilde{L}_i[x_i(t),t] \right\}, \end{aligned} \tag{2.3}$$

where $\beta \in \mathbb{R}_{>0}$ is a positive control gain, and $\tilde{L}_i[x_i(t),t]$ is a penalized objective function of agent i, defined as,

$$\tilde{L}_i[x_i(t),t] = f_i[x_i(t),t] - \frac{1}{\rho(t)} \sum_{j=1}^{q_i} \log\{1-\rho(t) g_{ij}[x_i(t),t]\}, \tag{2.4}$$

where $g_{ij}[x_i(t),t]:\mathbb{R}^m\times\mathbb{R}_{>0}\to\mathbb{R}$ denotes the $j-$th component of function $g_i[x_i(t),t]$, and $\rho(t)\in\mathbb{R}_{>0}$ is a time-varying barrier parameter satisfying

$$\rho(t)=a_1e^{a_2t},\ a_1,a_2\in\mathbb{R}_{>0}. \tag{2.5}$$

Here, the role of term $-\beta\{\nabla^2\tilde{L}_i[x_i(t),t]\}^{-1}\sum_{j\in\mathcal{N}_i}\mathrm{sgn}[x_i(t)-x_j(t)]$ in (2.3) is to drive all the agents to reach a consensus on states ($\lim_{t\to\infty}\|x_i(t)-\sum_{j=1}^n x_j(t)\|_2=0$). The Hessian-dependent gain $\beta\{\nabla^2\tilde{L}_i[x_i(t),t]\}^{-1}$ is introduced to guarantee the convergence of the algorithm under nonidentical $\nabla^2\tilde{L}_i[x_i(t),t]$. While the second term, $\phi_i(t)\in\mathbb{R}^m$, is an auxiliary variable playing a role in minimizing the penalized objective function $\tilde{L}_i[x_i(t),t]$ given by (2.4). Note that the log-barrier penalty functions (see the second term in (2.4)) are used to incorporate the inequality constraints into the penalized objective function.

To make the algorithm (2.3) work, the initial states $x_i(0)$ need satisfy

$$g_i[x_i(0),0]<\mathbf{0}_{q_i},\qquad\forall i\in\mathcal{V}. \tag{2.6}$$

Note that in algorithm (2.3), each agent just needs its own information and the relative states between itself and its neighbors.

Chapter 3

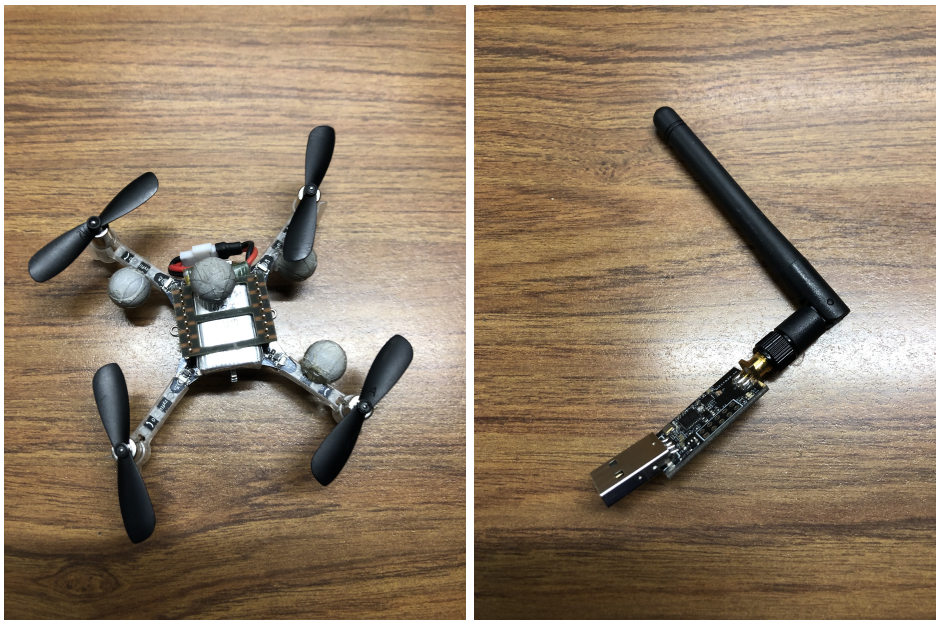
Experiment Environments

We use the crazyflies 2.0 as well as the vicon tracking system for experimental implementation. We give information of the crazyflies and the vicon system in this section.

3.1 Crazyflie 2.0

The crazyflie 2.0 as shown in Figure 3.1a is a small quad-rotor drone. It is developed by Bitcraze [1]. It is equiped with four $7 * 16$ mm coreless DC motors and four 45 mm plastic propellers. The motors are powered by a 240 mAh LiPo battery. The battery can provide energy up to 7 minutes of flight. We choose to use crazyflie 2.0 because it can be used indoors, thus reducing unexpected influence such as wind.

The drone has two micro-controllers onboard. The main micro-controller is a 168MHz 32-bit *STM32F405 ARM Cortex-M4* processor which is used for communication with the ground computer through a USB dongle. The USB dongle is shown in Figure 3.1b. The other micro-controller is a 32 MHz *nRF51822 ARM Cortex-M0* processor for energy



(a) One Crazyflie with four markers. (b) One USB dongle.

Figure 3.1: Crazyflie and the communication radio

and communication management. The communication between the drone and the USB dongle is encoded with the 2.4 GHz *Crazyradio PA*. The drone also has a built in inertial measurement unit (IMU). The IMU is *MPU 9250* with three axes: gyro, accelerometer and magnetometer, with an additional high precision pressure sensor (*LPS25H*) [4].

The control of the dynamic motions of the drone can be achieved by controlling the rotation speeds of the four propellers. If we assume that the space where we conduct the experiment is modeled with x, y, z axes where z represents the height, an object in a 3-dimensional (3-D) space, such as the drones used in our experiment, has 6 degrees of freedom: the positions in the x, y, z axes and the orientations with respect to the x, y, z axes. Theoretically, we cannot control all the 6 degrees of freedom with four propellers. When designing a controller, two degrees of freedom of the drone must be set free [6]. In

this thesis, the compromise is that we will not control the orientations of the drone with respect to the x, y axes because our focus is on the positions.

The radio communication frequency is designed to be 100Hz for the crazyflie and will experience some losses at some time instances. The communication frequency needs to be high enough to ensure smooth trajectory. If no command is provided in 0.01 seconds, the radio will re-send the previous command. In our experiment, this case will always happen because we use a script to calculate the commands. It takes the ground computer about 0.05 seconds to generate one command. As a result, the actual communication frequency is around 20 Hz. From our testing, this frequency is still high enough to make smooth flights.

The crazyflie has its official crazyflie ROS package to set up communications and conduct manual flights. The ROS package provides a user client which is the essential part of the experiment. It can be used to assign communication addresses for many crazyflies, which will allow us to control a group of crazyflies at the same time. The client can also show the data of the IMU and verify its accuracy.

When connected to a crazyflie using USB, the interface of the client has a button saying "connected on usb://0". The index number of the USB will change as the USB port that we use changes. By clicking on the button and connecting to the crazyflie, more functions will appear as shown in Figure 3.2. Here all the IMU data can be accessed. Press the connect button and in the showing up menu, continue to press configure crazyflie 2.0; there will be a window showing up to configure the bandwidth, radio communication address and radio communication channel of this crazyflie. The communication address needs to be carefully assigned here to comply with the crazyswarm setup which will be mentioned

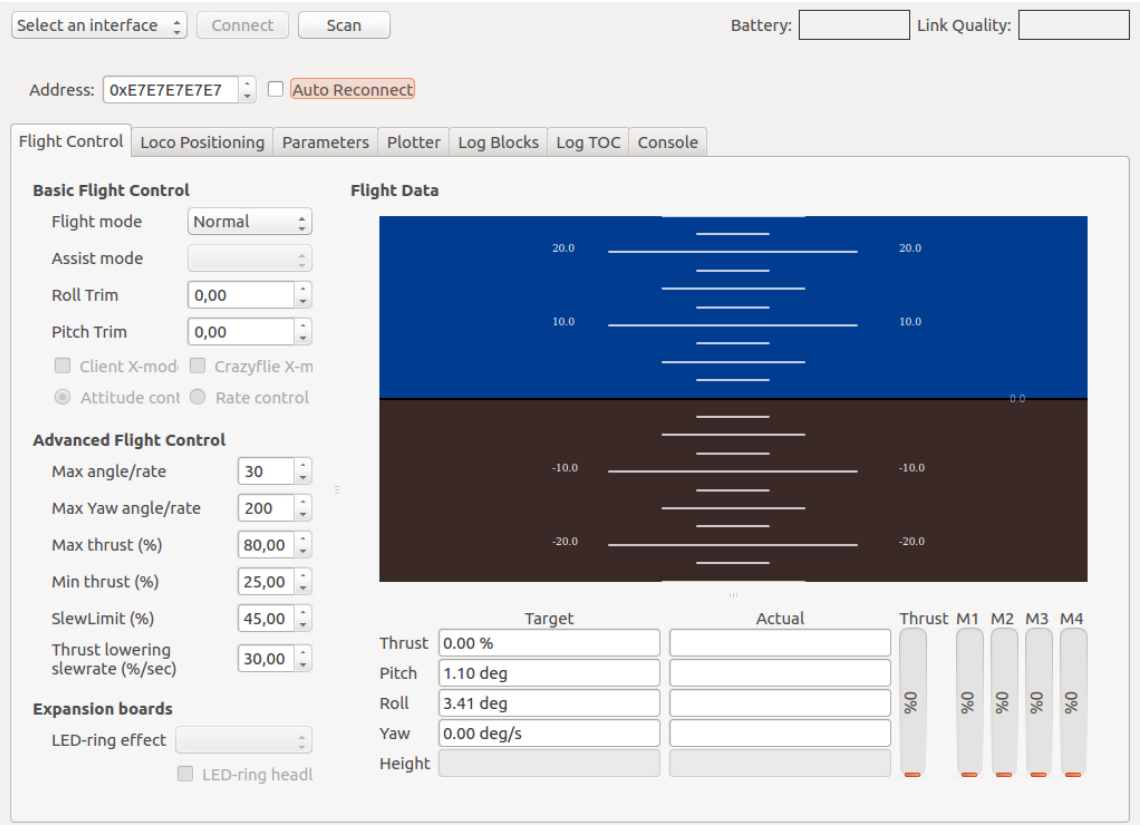


Figure 3.2: Client interface.

later in this thesis. The bandwidth is also important when we are trying to fly multiple crazyflies because the radio bandwidth may not be sufficient to send the commands. The radio communication channel can be set a value between 0-100. One value of a radio channel can handle up to 15 crazyflies' communication in our experiment according to [5]. Each crazyflie should have a unique radio address but the radio channel could be the same.

The crazyflie user client also provides the functionality that the drone can be manually controlled with a game pad. However, this is not the focus in this thesis. It is only used to verify the proper functionality of the crazyflies.

3.2 Vicon System

The vicon system is an optical passive motion capture system. The system is designed to track and analyze subjects moving in a workspace [2]. It uses retro-reflective markers that are placed on the "objects" tracked by infrared cameras to capture the position and orientation information of the "objects". We use the system to track our drones in this thesis.

The vicon system in the Cooperative Vehicle Networks (COVEN) lab consists of 10 cameras. They form a shape of a rectangle and all of them point towards the experiment area in the lab. The cameras used in the vicon system are high speed cameras. The speed varies depending on the camera types but is generally above 100 Hz. It is a reliable data source for the detection of drone movements in our lab. One camera is shown in Figure 3.3.

Together with the cameras, the retro-reflective markers are used in our experiment. The markers can reflect infrared so it is easy to be detected with the vicon cameras. We use

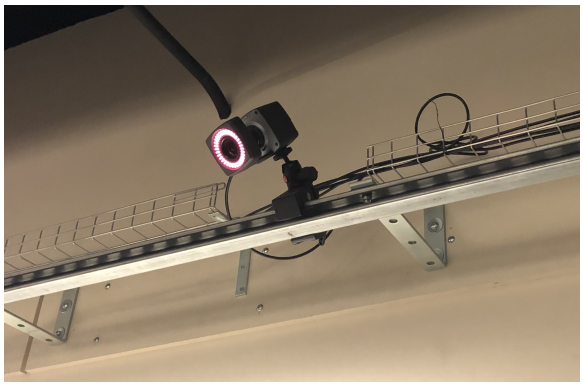


Figure 3.3: Vicon camera.

four markers to represent an "object". In Figure 3.1a, the crazyflie is attached with four markers. However, we only need three markers to define the position and orientation of an "object" in a 3-D space if the relative positions of the markers do not change. The fourth marker is used in case the cameras fail to detect all the markers at some time instances. The "object" defined in our experiment can be a drone or an obstacle. The data is not always reliable. If the markers can be seen by more cameras, the data given by the vicon system will be more accurate. This system is very easily affected by other light resources or reflecting objects like sunshine or shining steel in the area. During each time of experiment, all the aforementioned aspects need to be carefully dealt with.

The vicon system comes with its own server which has a user interface to define objects and calibrate cameras. As shown in Figure 3.4, the ten cameras are all green, which means that they all work normally. On the left of the figure, there is an "objects" menu where the obstacles and the crazyflies are defined in our experiment. To define an object, first, three or four markers need to be attached to the object in the real world. The reflective positions of the markers on one "object" will not change during one experiment.

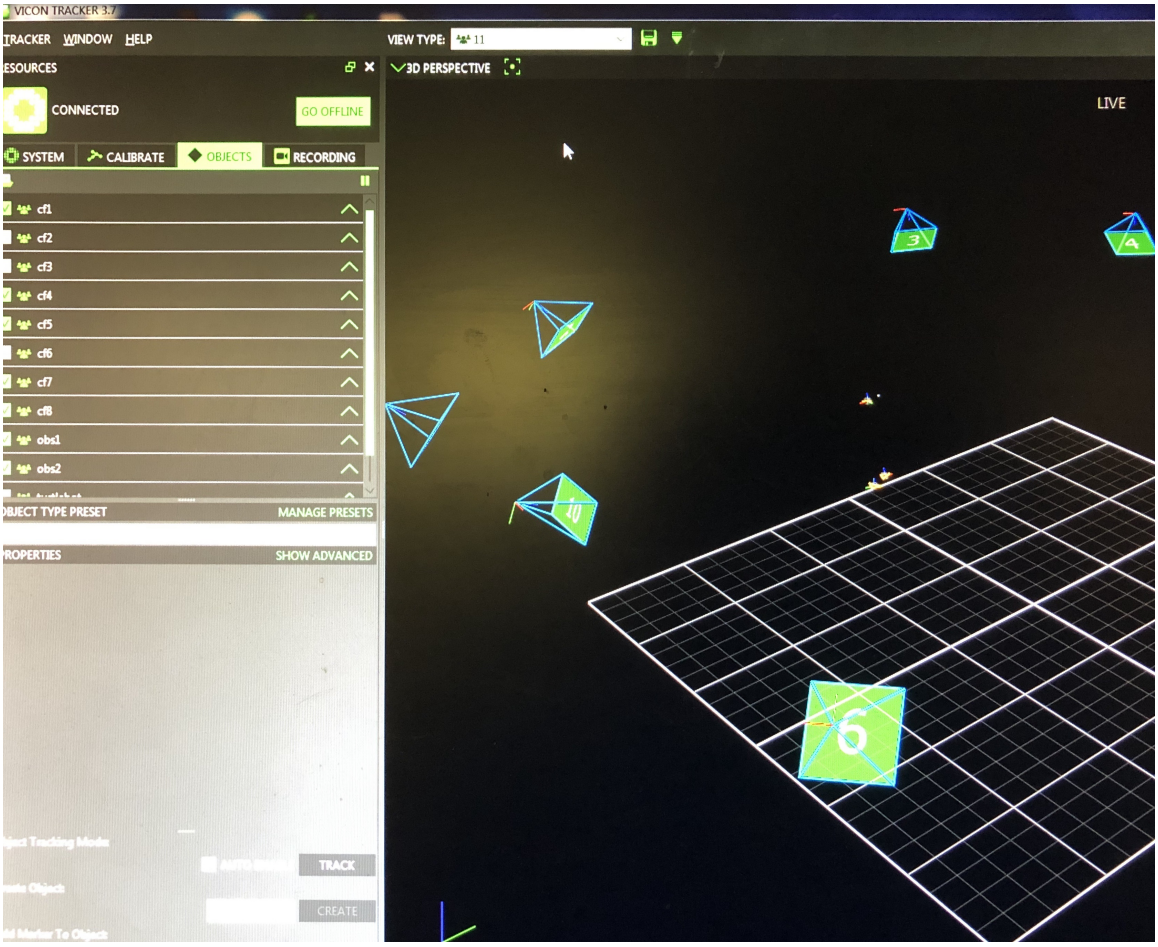


Figure 3.4: Vicon interface.

This is called a marker configuration of the object. The configurations of any two objects should be different for the system to distinguish between them. Second, put the object in the space where cameras can easily capture it. Usually the object is put in the center of the workspace. Third, on the host computer, select the corresponding markers, create an object and give it a name. The names for the crazyflies here need to be identical to the names defined in the crazyswarm package. In our experiment, the names are given as "cf1", "cf2" and so on.

There is also a "calibrate" menu where we can calibrate the cameras to a localization error which is usually less than 0.5 mm. The calibration procedure is as follows: select the "calibrate" menu; wave the calibration wand with pre-installed markers constantly in the workspace; after the calibration menu indicates it has enough measurements, wait until the computer automatically calculate the result; put the wand in the origin and align the axes; click "set origin" on the calibration menu. The calibration wand is a predefined wand that has five markers attached on it. When the wand is waved in the workspace, the cameras will capture the motion of the wand. One camera can capture a useful frame when it can see all the five markers on the wand. After each camera captures enough useful frames (usually larger than 3000 frames), a filter is applied to minimize the measurement error of the wand. The localization errors with respect to the x, y, z axes are usually less than 0.5 mm.

As the defined crazyflie with markers appear in the area where cameras are pointing, the positions and the orientations can be given by the vicon system. It is recommended that the cameras are calibrated before each time of experiment.

The crazyflies and the vicon system are used together in our experiment. We attach the markers on the crazyflie to make the vicon system detect it. We also make several paper cylinders and attach the markers on it to represent obstacles in our experiment. One thing to note is that the paper cylinders only represent the center positions of the obstacles. The actual shape is defined virtually in our script.

The information flow of the whole process is organized as follows: the vicon system captures the positions and orientations of the crazyflie and sends them to a computer; the computer handles all the information and uses them to generate velocity control commands; the radio sends these commands to the crazyflies.

3.3 Crazyswarm ROS Package

The Crazyswarm package is a ROS stack created by USC-ACT lab and is well documented in [5]. This ROS package designs an architecture for crazyflie 2.0 and vicon. It improves the tracking by introducing an alternative frame-to-frame tracking method, improves the state estimation with applying an extended kalman filter on the IMU, designs several trajectory planning methods, sets up communication for large amount of crazyflies with the ground computer, and presents an effective controller for the crazyflies [9]. We make use of the Crazyswarm ROS package to set up communications between crazyflies and the host computer and to make changes on some control parameters.

As mentioned before, the crazyflie ROS package is an essential operating tool. The crazyswarm ROS package also makes use of the crazyflie package to realize control commands. The crazyflie ROS package uses low level control commands. This kind of

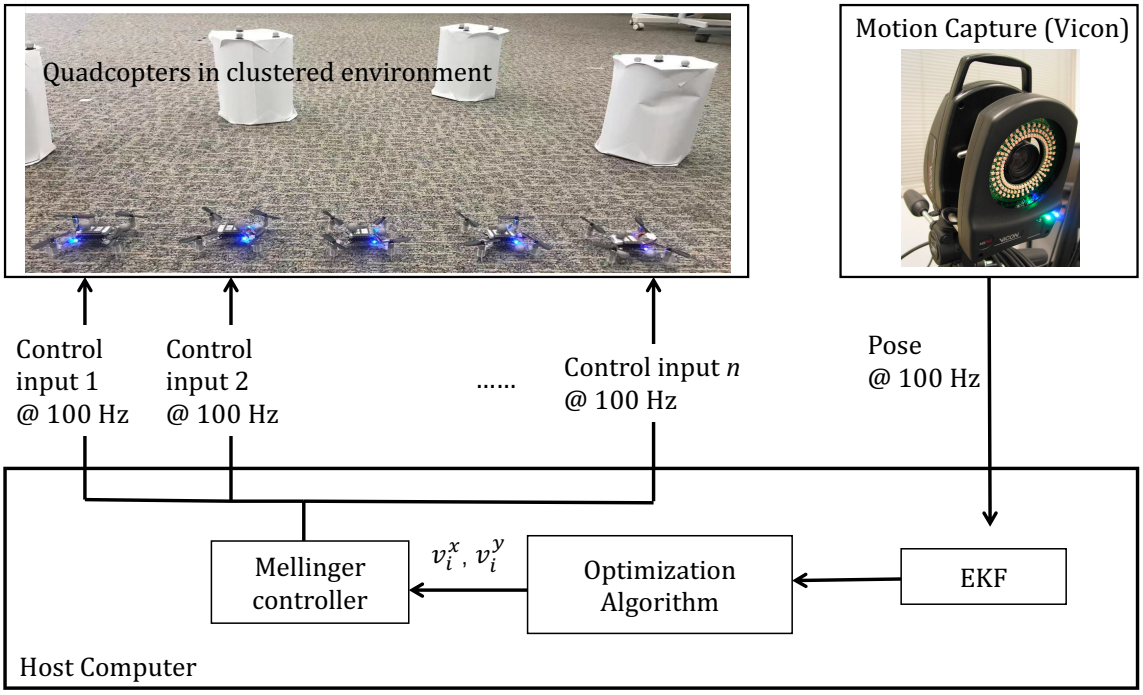


Figure 3.5: The crazyswarm architecture for information flow

commands are designed to be sent with radio frequency at 100 Hz. The input parameters are basic commands like velocities and accelerations. However, the use of the crazyflie ROS package is very complex. Many steps are needed for simple movements. For multi-agent experiments such as the experiment in this thesis, the crazyflie ROS package even needs to get down to communication port allocations. As a result, the ROS services and topics defined by the crazyflie ROS package are not directly used in this thesis. Nevertheless, this package provides the basic structure for communications and lowest level of command inputs.

The crazyswarm ROS package provides a complete information flow architecture for the crazyflies and the vicon system. The design of the architecture is shown in Figure 3.5. To use the information flow architecture, special attention needs to be paid here on

API name	Drone response
Takeoff	The drone will take off
Land	The drone will land
goTo	The drone will go to a point in space with fixed time
cmdFullState	The drone will try to reach the given state during the next 0.01 second
cmdPosition	The drone will try to go to the given position during the next 0.01 second
cmdVel	The drone will try to reach the given 3-D velocity during the next 0.01 second
Position	The script will return the current position of the drone

Table 3.1: Python APIs of Crazyswarm

the assignment of the crazyflies' radio communication address. The radio address for the crazyflie needs to be "0xE7E7E7E70X" where "X" represents a crazyflie number. This number should be corresponding to the crazyflies' names in the "objects" menu in the vicon system's user interface. For example, the object "cf2" in the vicon system should have the radio address of "0xE7E7E7E702". For the details of the architecture, please refer to [9]. The low level server of the crazyswarm ROS package is written in C++ programming language. The server registers the all the ROS services and the ROS topics. The high level scripts use these ROS services and ROS topics to send control sequences to the crazyflies. The high level scripts are written in python programming language.

The usage of Crazyswarm is straightforward. The high level python APIs that we use are provided in Table 1. As long as we can get the control inputs from the algorithm (2.3) constantly, these control inputs can be sent to the crazyflies and the micro-controller on the crazyflies will transfer the the signals to motor speeds.

The crazyswarm ROS package has a simulation tool which is used in this thesis. When the crazyswarm ROS package is activated, the simulation command can be used the same way as normal commands by adding "–sim" to the last of the command. For instance,

if we want to run the "individual hover" script, we can type "python individual hover" in the command window and the crazyflie will take off and hover in the air. If we want to do that in simulation, we need to type "python individual hover –sim". This simulation is a numerical simulation. It takes into consideration of the services and the topics in the ROS package but does not consider the dynamic model of the drone.

Chapter 4

Tests and Results

The details of the implementation and the experimental results are presented in this section.

4.1 Multi-Robot Multi-Target Navigation Problem

We apply the optimization algorithm (2.3) to a class of the motion coordination problems: the multi-robot multi-target navigation problem. As shown in Figure 4.1, let us consider a closed and convex workspace $W \in \mathbb{R}^2$. Consider the scenario where there are n disk-shaped robots (blue quadrotors) with center positions $x_i, i \in [1, \cdots, n]$ and radius $r_i > 0, i \in [1, \cdots, n]$ and k moving targets (red triangles) in an unknown space in the presence of obstacles. The objective here is to have the robots stay close while simultaneously ensuring that each independent moving target stays in the detection range of at least one robot. Assume that the workspace is populated with Q nonintersecting spherical obstacles (black circles), where the center and radius of the ith obstacle are denoted

by $o_i \in W$ and $r_i^o > 0$, respectively. Since there are unknown obstacles in the environment, we have to guarantee collision avoidance during the tracking process. For mathematical expression, we define the so-called collision-free local workspace around x_i as [3]

$$LF(x_i) = \{p \in W : a_j(x_i)^T p - b_j(x_i) \leq 0, j = 1, \cdots, Q\}, \tag{4.1}$$

where

$$\begin{aligned} a_j(x_i) &= o_j - x_i, \quad \theta_j(x_i) = \frac{1}{2} - \frac{r_j^{o2} - r_i^2}{2\|o_j - x_i\|^2}, \\ b_j(x_i) &= (o_j - x_i)^T \left[\theta_j o_j + (1 - \theta_j)x_i + r_i \frac{x_i - o_j}{\|x_i - o_j\|}\right]. \end{aligned} \tag{4.2}$$

In order to have the robots stay close while simultaneously ensuring that each target stays in the sensing range of at least one robot, one method is to let all the robots assemble in the geometric center of all the targets with deviation vectors introduced to each robot. We tackle the navigation task by solving the following optimization problem with nonlinear inequality constraints,

$$\begin{aligned} \min \quad & \sum_i^n f_i = \|x_i - T_i(t)\|_2^2 \\ \text{s.t.} \quad & x_i = x_k \quad \forall i, k \in \mathcal{V}, \\ & a_j(x_i)^T x_i - b_j(x_i) \leq 0, \quad \forall i \in \mathcal{V},\ j \in [1, \cdots, q_i], \end{aligned} \tag{4.3}$$

where $T_i(t)$ is the geometric center of all the moving targets that robot i can sense and q_i is the number of obstacles that robot i can sense. Note that a robot might not be able to sense all the Q obstacles in the workspace, but it is safe enough to stay in the collision free area determined by the nearby obstacles. Since $a_j(x_i)$ and $b_j(x_i)$ depend on the position

of robot i, the above optimization problem has an implicit dependence on time through x_i. However, it is very hard to directly address the inequality constraints in (4.3) due to the complexity of $a_j(x_i)$ and $b_j(x_i)$ given by (4.2). Therefore here we treat $a_j(x_i)$ and $b_j(x_i)$ as $a_j(t)$ and $b_j(t)$. Based on (2.4), the corresponding penalized objective function is defined as $\tilde{L}_i = f_i(x_i, t) - \frac{1}{\rho(t)} \sum_{j=1}^{q_i} \log\{1 - \rho(t)[a_j(t)^T x_i + b_j(t)]\}$. If the communication topology between the robots is undirected and connected, problem (4.3) satisfies all the required Assumptions 1 to 6. Therefore, for robots with single-integrator dynamics defined by (2.1), the constrained optimization algorithm (2.3) can be applied to reach on agreement at the geometric center of the targets and spread the robots in a desired formation about this center. Therefore, we introduce an offset vector δ_i for each robot i and replace x_i in the algorithms (2.3) with $x_i - \delta_i$. Here, $\delta_i - \delta_j$ defines the desired relative position from robot j to robot i in the formation.

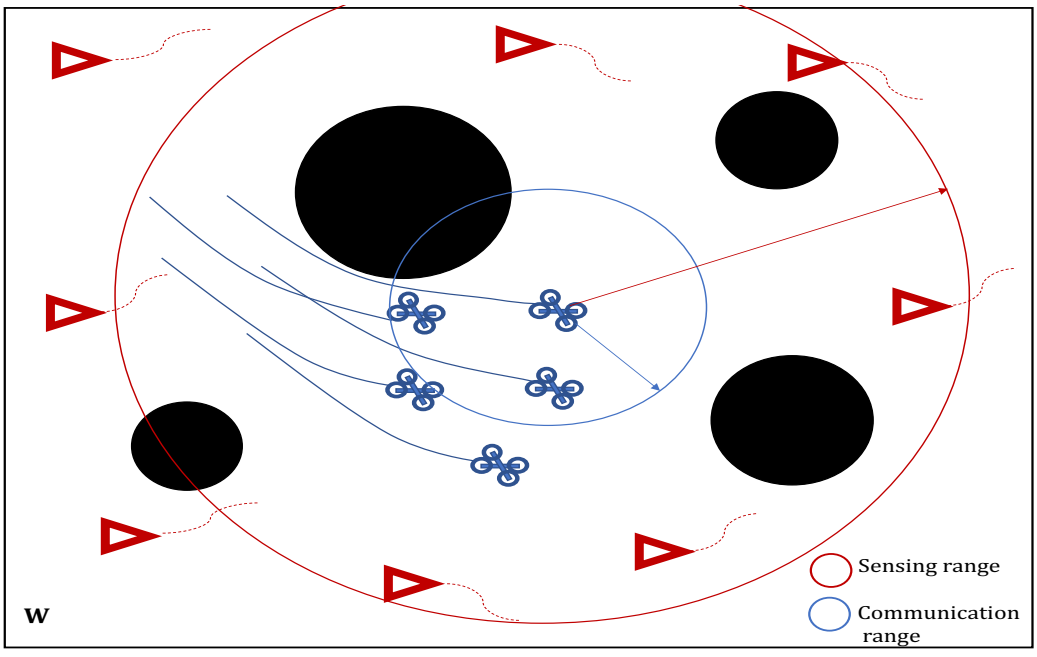
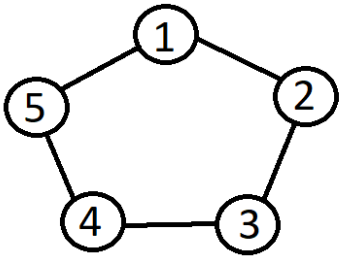
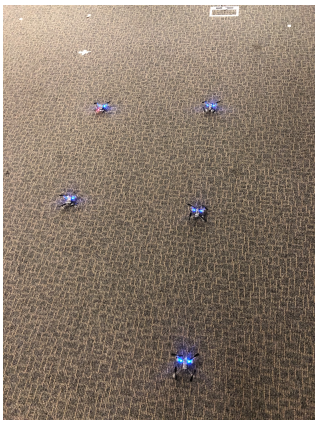


Figure 4.1: Multi-robot multi-target navigation problem



(a) Communication topology



(b) Initial shape of crazyflies

Figure 4.2: Communication topology and initial shape of crazyflies.

4.2 Experiment Setup

In our experiment, a 5×5 m^2 area is used to implement the experiment. We use five crazyflies in the experiment to verify the algorithm (2.3). The topology between the crazyflies is a ring, which is shown in Figure 4.2a. The communication links are undirected. To simplify the experiment, we assume that each crazyflie is only assigned one target moving in the environment. Note that the algorithm (2.3) still works for multiple targets since it only cares about the geometric center of all the targets that the crazyflie can sense. The obstacles are located at $o_1 = [-2.1 \text{ m}, -0.5 \text{ m}]$ and $o_2 = [1.8 \text{ m}, 1.6 \text{ m}]$ with radius $r_1^0 = 0.9$ m, and $r_2^0 = 0.7$ m. A picture is available in Figure 4.3 to show the paper cylinders representing the obstacles. The offset vectors which determine the crazyflies' formation

from their center are chosen as

$$\delta_1 = [0.2\sin(0.2\pi)\ \text{m}, -0.2\cos(0.2\pi)\ \text{m}]^T,$$

$$\delta_2 = [-0.2\sin(0.2\pi)\ \text{m}, -0.2\cos(0.2\pi)\ \text{m}]^T,$$

$$\delta_3 = [-0.2\cos(0.1\pi)\ \text{m}, 0.2\sin(0.1\pi)\ \text{m}]^T,$$

$$\delta_4 = [0\ \text{m}, 0.2\ \text{m}]^T,$$

$$\delta_5 = [0.2\cos(0.1\pi)\ \text{m}, 0.2\sin(0.1\pi)\ \text{m}]^T.$$

The initial positions of the five crazyflies are chosen as $x_1(0) = [-0.4\ \text{m}, 0.4\ \text{m}]$, $x_2(0) = [-1.1\ \text{m}, 0.4\ \text{m}]$, $x_3(0) = [-1.1\ \text{m}, 1.1\ \text{m}]$, $x_4(0) = [-0.4\ \text{m}, 1.1\ \text{m}]$, and $x_5(0) = [0.3\ \text{m}, 1.1\ \text{m}]$. The initial configuration of the crazyflies is shown in Figure 4.2b. We choose $\rho(t) = 125\exp(0.01t)$ and $\beta = 5$. The trajectories of the five targets are selected as

$$T_1(t) = [1.4\sin(0.06t) + \sin(0.2\pi)\ \text{m}, 1.4\cos(0.06t) - \cos(0.2\pi)\ \text{m}]^T,$$

$$T_2(t) = [1.4\sin(0.06t) - \sin(0.2\pi)\ \text{m}, 1.4\cos(0.06t) - \cos(0.2\pi)\ \text{m}]^T,$$

$$T_3(t) = [1.4\sin(0.06t) - \cos(0.1\pi)\ \text{m}, 1.4\cos(0.06t) - \sin(0.1\pi)\ \text{m}]^T,$$

$$T_4(t) = [1.4\sin(0.06t)\ \text{m}, 1.4\cos(0.06t) + 1\ \text{m}]^T$$

$$T_5(t) = [1.4\sin(0.06t) + \cos(0.1\pi)\ \text{m}, 1.4\cos(0.06t) - \sin(0.1\pi)\ \text{m}]^T.$$

It is worthwhile to mention that the initial center of the targets is different from the initial center of the crazyflies in the experiment. In fact, the initial positions of the crazyflies can be randomly selected in theory. However, the exact initial positions need to be specified by the crazyswarm package. The crazyswarm package requires a launch file to be properly

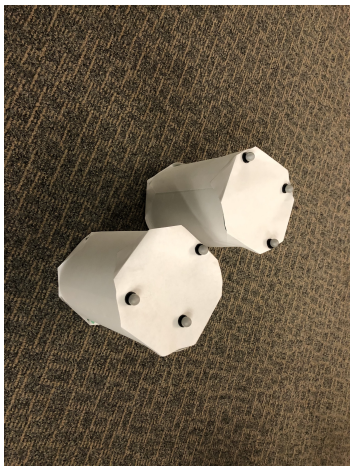


Figure 4.3: Obstacles in the environment

created which indicates the crazyflies' identification number and their initial positions. The crazyflies' identification number should correspond to the number given in the vicon system. The errors between the given initial positions and the actual initial positions should not be too large.

When the experiment begins, the crazyflie will take off to a desired height of 0.3 meters for safety concern. Then the control algorithm takes place. Every crazyflie is trying to solve the distributed optimization problem defined in equation (4.3) in a distributed way. Here, T_i is the real time position of target i. Each crazyflie has the knowledge of its own target that moves in a circle. During the experiment, each crazyflie senses the position of itself and the obstacles and then calculates $a_j(x_i)$ and $b_j(x_i)$ based on equation (4.2). The crazyflies will work cooperatively to solve the multi-robot multi-target navigation problem. The control input of each crazyflie is determined by its local information. Here the local information of each crazyflie includes not only the knowledge of its own target, but also the information of its neighbours because all the crazyflies are trying to collectively solve

the optimization problem while avoiding obstacles. Due to the introduction of the offset vectors, they will maintain their formation in the meantime. The height control is not the focus and is out of concern in the experiment.

4.3 Simulation Result

Before the experiment, the numerical simulation of the design has been conducted. A detection-avoidance algorithm is deployed to consider the sensing range of the drone. The crazyflie will not sense the obstacles unless the distance between it and any boundary of the obstacles is less than 0.3 meters. The simulation results are presented in the Figure 4.4. From the Figure 4.4, it is easy to see that the crazyflies form the shape we desire and avoid the obstacles very well throughout the time. The center position of all the crazyflies is able to track the center of all the targets with nearly zero tracking error. The numerical simulation is conducted by the simulation tool provided by the crazyswarm ROS package. The main assumptions in numerical simulations includes that we can always get accurate position data of the crazyflies and there is no saturation in the control inputs.

4.4 Experimental Result

Figure 4.5 shows the results of a real trail on the crazyflie platform. In this trail, the detection-avoidance algorithm is also deployed. As can be easily observed from the figure, the crazyflies have oscillations throughout the experiment. One reason might be we are implementing a continuous time algorithm on a discrete experimental environment setting.

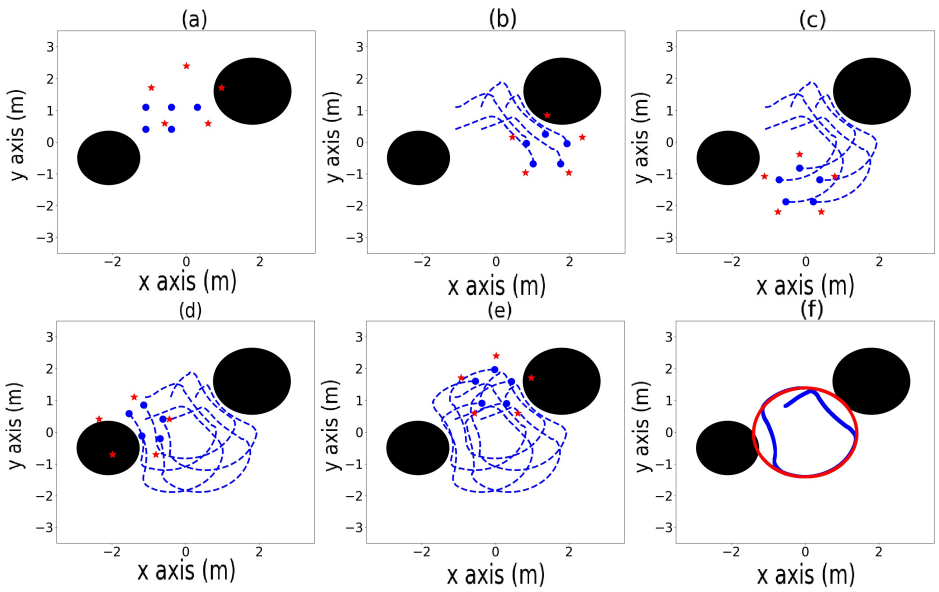


Figure 4.4: Simulation result. The blue points are the current positions of the crazyflies. The blue dash lines are the trajectories of the crazyflies. The red points are the targets that the crazyflies trying to follow. Figure (a) to (e) show the situation at time instances of 0s, 25s, 50s, 75s and 100s. In figure (f), the red line is the center trajectory of all the targets, the blue line is the center trajectory of all the crazyflies.

Although the high frequency communication can compensate for the problem, the oscillation still exists in an acceptable level. Overall, this result is very similar to the simulated one and achieves our goal, which means that the constrained distributed optimization theory works in real life.

4.5 Observations

In the simulation, the geometric center of all the crazyflies is able to track the geometric center of all the targets with zero tracking errors while avoiding the collision with the obstacles. In our experimental result, the crazyflies tremble slightly in flight and the geometric center of all crazyflies is able to track the geometric center of all the targets with

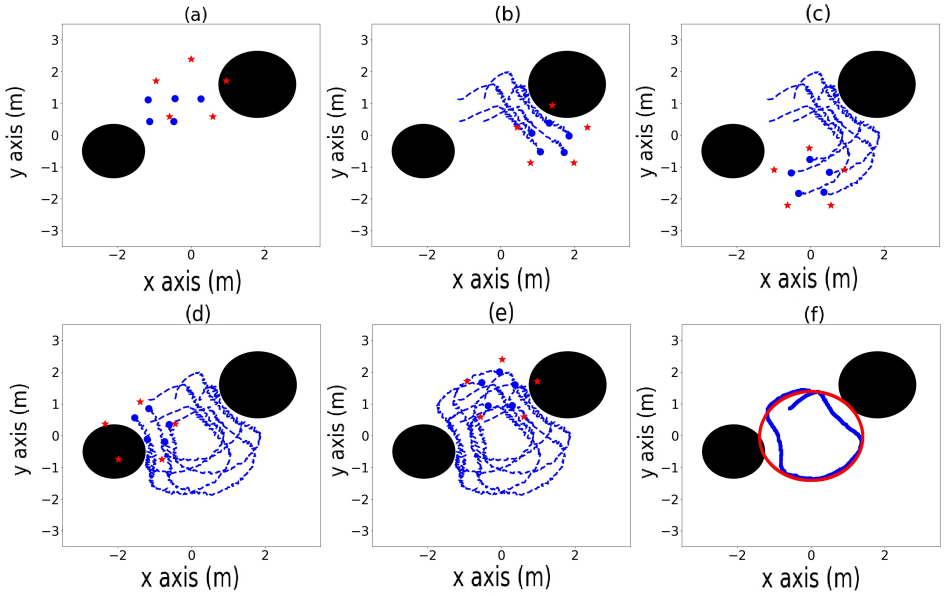


Figure 4.5: Experiment result. The blue points are the current positions of the crazyflies. The blue dash lines are the trajectories of the crazyflies. The red points are the targets that the crazyflies trying to follow. Figure (a) to (e) show the situation at time instances of 0s, 25s, 50s, 75s and 100s. In figure (f), the red line is the center trajectory of all the targets, the blue line is the center trajectory of all the crazyflies.

small tracking error. It is worthwhile to mention that the trembling phenomena and tracking error in the experiment might stem from the following aspects. First, in the simulation, there is no saturation in the control inputs to the drones. In the experiment, The control inputs would become saturated when exceeding their capacity. Second, communication delays and localization errors inevitably exist in the experiment. These aspects do not exist in the simulation. Third, from the experimental result we can see that the oscillation issue is severer when crazyflies move near to the obstacles. This is because the cost function defined in the algorithm (2.3) is changing very rapidly in that area based on the lab space that we have. Both the radio communications and the motors on the drone cannot handle the changes with that speed. If the crazyflies are tested in a larger space, the oscillation issue might be improved.

Chapter 5

Conclusions

This thesis talks about a distributed time-varying constrained optimization algorithm and shows the experimental validation of the algorithm. The effectiveness of the algorithm is validated by applying it in the multi-robot multi-target navigation problem. Using the considered distributed optimization algorithm, both the tracking process and the obstacle avoidance are achieved with only local information and local communications. The experiment succeeds in verifying that the algorithm is applicable to real life problems. Future work will seek to conduct the experiment in a larger space and will consider moving obstacles instead of static ones.

Bibliography

[1] https://www.bitcraze.io/products/old-products/crazyflie-2-0/.

[2] https://www.vicon.com/about-us/what-is-motion-capture/.

[3] Mahyar Fazlyab, Santiago Paternain, Victor M Preciado, and Alejandro Ribeiro. Prediction-correction interior-point method for time-varying convex optimization. *IEEE Transactions on Automatic Control*, 63(7):1973–1986, 2017.

[4] Wojciech Giernacki, Mateusz Skwierczyński, Wojciech Witwicki, Paweł Wroński, and Piotr Kozierski. Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering. In *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*, pages 37–42. IEEE, 2017.

[5] Wolfgang Hönig and Nora Ayanian. Flying multiple uavs using ros. In *Robot Operating System (ROS)*, pages 83–118. Springer, 2017.

[6] Amir Hussein and Rayyan Abdallah. *Autopilot Design for a Quadcopter*. PhD thesis, 10 2017.

[7] Hariharan Lakshmanan and Daniela Pucci De Farias. Decentralized resource allocation in dynamic networks of agents. *SIAM Journal on Optimization*, 19(2):911–940, 2008.

[8] Daniel K Molzahn, Florian Dörfler, Henrik Sandberg, Steven H Low, Sambuddha Chakrabarti, Ross Baldick, and Javad Lavaei. A survey of distributed optimization and control algorithms for electric power systems. *IEEE Transactions on Smart Grid*, 8(6):2941–2962, 2017.

[9] James A Preiss, Wolfgang Honig, Gaurav S Sukhatme, and Nora Ayanian. Crazyswarm: A large nano-quadcopter swarm. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3299–3304. IEEE, 2017.

[10] Michael Rabbat and Robert Nowak. Distributed optimization in sensor networks. In *Proceedings of the 3rd international symposium on Information processing in sensor networks*, pages 20–27, 2004.

[11] S. Rahili and W. Ren. Distributed continuous-time convex optimization with time-varying cost functions. *IEEE Transactions on Automatic Control*, 62(4):1590–1605, 2017.

[12] Mohamad T Shahab and Moustafa Elshafei. Distributed optimization of multi-robot motion with time-energy criterion. In *Path Planning for Autonomous Vehicles-Ensuring Reliable Driverless Navigation and Control Maneuver*. IntechOpen, 2019.

[13] Shan Sun and Wei Ren. Distributed continuous-time optimization with time-varying objective functions and inequality constraints. In *Proceedings of the IEEE Conference on Decision and Control*, Jeju Island, Republic of Korea, Dec 2020.

[14] Ruiliang Zhang and James Kwok. Asynchronous distributed admm for consensus optimization. In *International conference on machine learning*, pages 1701–1709, 2014.