DISTRIBUTED NONLINEAR MODEL PREDICTIVE FORMATION CONTROL OF QUADROTOR TYPE UAVS IN CLUTTERED AND DYNAMIC ENVIRONMENTS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

MUHLİS SAMİ SATIR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

MAY 2023

Approval of the thesis:

DISTRIBUTED NONLINEAR MODEL PREDICTIVE FORMATION CONTROL OF QUADROTOR TYPE UAVS IN CLUTTERED AND DYNAMIC ENVIRONMENTS

submitted by **MUHLİS SAMİ SATIR** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Date: 10.05.2023

ABSTRACT

DISTRIBUTED NONLINEAR MODEL PREDICTIVE FORMATION CONTROL OF QUADROTOR TYPE UAVS IN CLUTTERED AND DYNAMIC ENVIRONMENTS

Satır, Muhlis Sami
M.S., Department of Electrical and Electronics Engineering
Supervisor: Assoc. Prof. Dr. Mustafa Mert Ankaralı
Co-Supervisor: Assoc. Prof. Dr. Erol Şahin

MAY 2023, 73 pages

Unmanned aerial vehicles (UAVs) have gained widespread use in various applications, including surveillance, inspection, and delivery. One important aspect in the operation of UAVs is the ability to fly in formation, where a group of UAVs maintains a desired geometric configuration while performing a given task. In this thesis, we use model predictive control (MPC) to address the problem of formation control for a group of UAVs. We first review the literature on formation control for UAVs, highlighting the advantages and limitations of different control approaches. We then describe the MPC formulation for formation control of multi-UAV systems, taking into account the dynamics and constraints of the UAVs. We also discuss the challenges and trade-offs in the design of the MPC controller, including the choice of the prediction horizon and the control input constraints. We present a novel approach that combines MPC formation control with graph generation algorithms to allow for the formation of UAVs to start from arbitrary points providing greater flexibility and adaptability to different scenarios.

Finally, we demonstrate the proposed formation control method on a swarm of mini UAV both in simulation and on physical platforms.

Keywords: Model Predictive Control, Swarm, Formation Control, Unmanned Aerial Vehicles

ÖZ

YOĞUN VE DİNAMİK ENGELLİ ORTAMDA DÖNERKANAT İHA'LARIN DAĞITIK DOĞRUSAL OLMAYAN MODEL ÖNGÖRÜLÜ FORMASYON KONTROLÜ

Satır, Muhlis Sami
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. Mustafa Mert Ankaralı
Ortak Tez Yöneticisi: Doç. Dr. Erol Şahin

Mayıs 2023 , 73 sayfa

İnsansız hava araçları (İHA) gözetleme, denetleme ve teslimat dahil olmak üzere çeşitli uygulamalarda yaygın bir kullanıma sahiptir. İHA'ların önemli bir operasyonel kabiliyeti olan formasyon uçuşu, bir grup İHA'nın belirli bir görevi yerine getirirken istenen bir geometrik konfigürasyonu koruduğu düzende uçma yeteneğidir. Bu tezde, bir grup İHA için formasyon kontrolü problemini çözmek için model öngörülü kontrol (MPC) kullanılmıştır. Öncelikle, İHA'lar için formasyon kontrolüne ilişkin literatürü farklı kontrol yaklaşımlarının avantajlarını ve sınırlamalarını vurgulayarak sunulmuştur. Daha sonra, İHA'ların dinamiklerini ve kısıtlamalarını dikkate alarak çoklu İHA sistemlerinin formasyon kontrolü için MPC formülasyonunu açıklanmıştır. Ayrıca tahmin ufkunun seçimi ve kontrol girişi kısıtlamaları da dahil olmak üzere model öngörülü kontrolcünün tasarımındaki zorlukları ve ikilemleri tartışılmıştır. MPC formasyon kontrolünü çizge üretme algoritmalarıyla birleştirip, İHA'ların formasyona rastgele noktalardan başlayarak ulaşmasını sağlayan yenilikçi bir yaklaşım sunulmuştur. Bu yöntem, farklı ortamlara ve problemlere karşı daha fazla esneklik ve uyum sağlamıştır.

Son olarak, önerilen formasyon kontrol yöntemini gerçek dünyadaki bir uygulamaya uygulayarak mümkünlüğünü ve kullanışlılığını gösterilmiştir. Önerilen yaklaşımın potansiyel uzantıları ve uygulamalarının yanı sıra bu alanda gelecekteki araştırma yönlerini tartışılmıştır.

Anahtar Kelimeler: Model Öngörülü Kontrol, Sürü, Formasyon Kontrolü, İnsansız Hava Araçları

To my beloved daughter, wife, and my parents

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my supervisor Assoc. Prof. Dr. Mustafa Mert Ankaralı for his guidance, and encouragements. I also thank my co-supervisor, Assoc. Prof. Dr. Erol Şahin, for introducing me to the world of swarm robotics. In this study, the METU ROMER infrastructure was used. I would like to thank Assoc. Prof. Dr. Ali Emre Turgut and all friends in the ROMER for their support.

I am grateful to my colleagues at Aselsan Inc., starting from Yasin Furkan Aktaş and Simay Atasoy, who have provided a great work environment and helped me through invaluable discussions. I would like to acknowledge Aselsan Inc., which provides physical platform.

I would also like to give special thanks to all members of "Autonymous" team. They helped to increase my engineering strength and gain perspectives on how to look at a problem to resolve. I would like to open an extra bracket to Eyüp Halit Yılmaz for his criticism to my works.

And above all, I would like to thank my mother Tuba, my father Murat and my sister Çağla for being there for me throughout this journey. I am grateful to my wife Reyyan, for her endless love and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

MRS	Multi-Robot System
UAV	Unmanned Aerial Vehicle
ROS	Robot Operating System
MPC	Model Predictive Control
DMPFC	Distributed Model Predictive Formation Control
UWB	Ultra Wide Band
BVC	Buffered Voronoi Cell

CHAPTER 1

INTRODUCTION

1.1 Motivation

Flocks of birds, schools of fish, and herds of goats exhibit some of the fascinating behaviors of swarm intelligence. Individuals in a flock are less likely to get hunted and are able to follow a more precise route with less energy consumption [4]. These advantages of collective motion inspire engineers to mimic swarm behaviors.

In order to obtain a computer model that behaves like flocking birds, Reynolds [5] defined simple rules namely separation, alignment, and cohesion. With Reynolds' flocking model, swarm behavior was modeled mathematically for the first time.

Another interesting biological swarm is ant colonies. The common needs of survival, such as gathering food, seeking or building shelter, and reproduction, led to formation of colonies. This collective behavior facilitates defense against enemies, ensuring safety of offsprings, a primitive division of labor, and building complex structures. The colony functions properly without explicit instructions from a supervisor. Every individual of the colony accomplishes tasks based on their decision-making, which depends on signals from other worker ants and environmental features. Such interactions occur via pheromones, detected by antennae sensitive to chemical stimulation. Behavior of such a biological swarm is decentralized since each biological "robot" remains independent from global knowledge or supervision but uses its own local sensing, decision, and control mechanisms.

Recent advances in the computation capability of embedded systems, sensor technology, and communication systems have allowed robots to collaborate with other robots. With these developments, a swarm of robots could be deployed in search, surveillance, reconnaissance, and mapping missions. Design and development of a multi-robot system (MRS) unlocks a number of features that are infeasible with a single robot. These features include multi-tasking, fault-tolerance, cost-effectiveness, scalability, versatility, adaptability, and flexibility [2], [6]

With in the multi-robot research, the coordination and cooperation problems are examined under a large collection of topics: aggregation, consensus, agreement, rendezvous, synchronization, social foraging, flocking, coverage, scheduling, and formation [7–14].

The objective of formation is to maintain a certain shape with constant relative distances between robots during missions [2]. Including those mentioned above, deploying a MRS with a predefined shape increases the efficiency of the task such as the creation an aerial image of a large area with high spatial resolution using a UAV swarm. Each individual UAV needs to be properly positioned relative to nearby UAVs such that captured images can be stitched together, with no gaps, to obtain a complete map of the area [14]. Another application increases the collective sensing capability of interconnected sensors by ensuring optimal placement of individual robots for target tracking problem [15]. Lastly, by intelligent planning of formation for a group of UAVs deployed in the field, it becomes possible to approximate the behavior of an antenna that is orders of magnitude larger than the size of each individual robot. As a result, sensitivity of the overall system is improved [16].

1.2 Problem Definition

Distributed formation control of MRS depends on relative information observed by each individual robot through sensors that measure distance, angular displacement, angular velocity, and relative displacement to other individuals. Since these relational measurements play a critical role in determining the behaviors of robots, formation control fundamentally differs from centralized control algorithms and conventional distributed control systems in that formation control does not depend on internal states with global coordinate values [14].

Formation control requires the simultaneous execution of a number of tasks such as navigating to goal, formation keeping, obstacle and reciprocal avoidance. Without any priority of one task to the other, the system must; move from point A to point B as a whole (measured by either the center of mass or center of volume), preserve the relative positions and overall shape of the formation during the operation, perform obstacle avoidance to assure successful determination of the motion, etc. In an inherently decentralized system such as formation control, the solution to these tasks should not depend on the existence of a supervisor robot or master control unit. The resultant behavior of the MRS should naturally follow from the designed decision-making process of individual robots which is the core problem addressed in the scientific research on formation control.

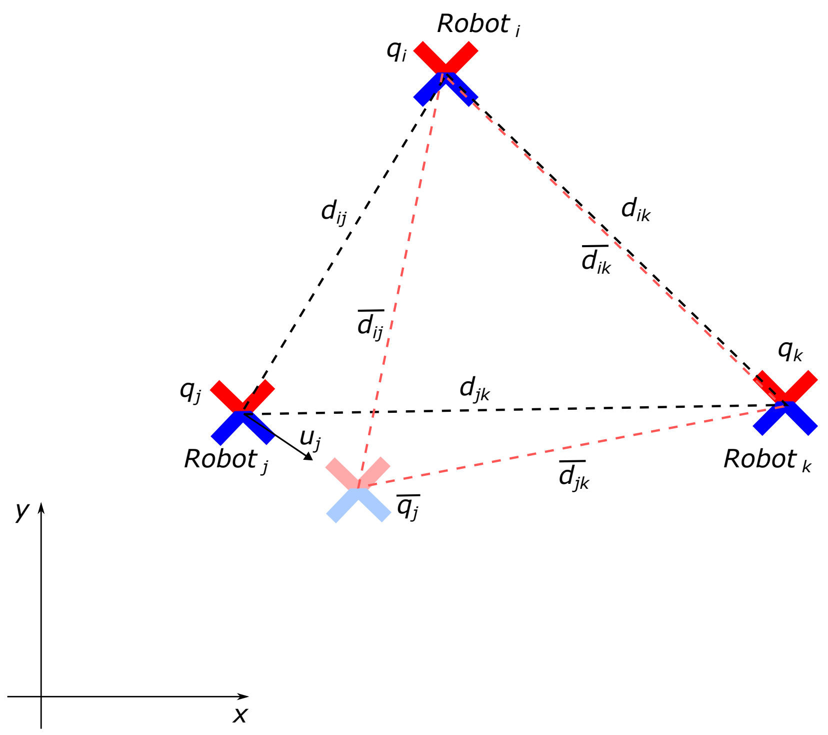

Figure 1.1: Formation of equilateral triangle. Euclidean distance between the robot i and robot j denoted as d_{ij}. The desired distance between robot i and j is given by the constant $\overline{d_{ij}}$. u_j is the control input for robot j. Note that the robot is symbolized by a half-blue and half-red cross. The red side represents the front of the robot.

Consider a system of M mobile robots where q_i is the position of the i^{th} robot relative to a world coordinate frame, and u_i is the corresponding control input. $||q_i(k)-q_j(k)||$ represents the Euclidean distance between the robot i and robot j at time instant k and it can be denoted as d_{ij}. The desired distance between robot i and j is given by the constant $\overline{d_{ij}}$. Fig. 1.1 shows the defined positions, control inputs, measured and desired distances for equilateral triangle formation.

The goal of formation acquisition is the formation and maintenance of a predefined geometric shape by a group of robots in space. The control objective for formation acquisition can be mathematically described as to design u_i such that

$$d_{ij} = \| q_i(t) - q_j(t) \| \longrightarrow \overline{d_{ij}} \text{ as } k \longrightarrow \infty \qquad \forall i, j, i \neq j \tag{1.1}$$

1.3 The Outline of the Thesis

The structure of this thesis is organized as follows. We provide a comprehensive survey of related literature in Chapter 2. Background information is provided on Graph Theory, Model Predictive Control (MPC), Deleanuey Triangulation, and details of experimental setup and software in Chapter 3. The proposed methodology regarding the robot model, single robot navigation with MPC, obstacle avoidance, reciprocal avoidance, MPC cost functions for formation, and Distributed Model Predictive Formation Control (DMPFC) are presented in Chapter 4. Chapter 5 contains results obtained from simulations and experiments, a discussion on performance, and comparisons. Finally, Chapter 6 concludes the thesis.

CHAPTER 2

LITERATURE SURVEY

2.1 Classification of Control Strategies

Formation control strategies of MRS can be classified into three: centralized, decentralized, and distributed control [3].

The centralized control strategy is built around a central controller that receives all required information (such as states and sensor data etc.) from the robots and send the control commands to them as depicted in Fig. 2.1 (a).

The centralized control strategy is simple and easy to implement [3]. But, it requires high-performance processors and is vulnerable against to single point of failure. Moreover, since the central computer must be located on the ground station or on a single robot, it will need to communicate with other robots in order to receive information and control them. While controlling a robot, it is important to get the state information with little delay and to send the control command for the stability of the robot. Communication speed becomes a problem for centralized control while increasing the number of robots.

The decentralized control strategy splits of the formation control problem into independent subproblems. In a decentralized control system, the entire system is no longer controlled by one controller, but by numerous independent controllers comprised of decentralized controllers deployed on each module. A general definition of the decentralized system in discrete time is provided by Tsitsiklis in [17]. The definition states that the decentralized system consists of a few interconnected modules. According to the states of the current module, each module has a corresponding controller. The formation control method is decentralized in a multi-robot system if each robot employs its own controller and navigates in accordance with its own measurement (detection or sensing) [3].

The distributed control strategy refers to a control system in which the control functions are distributed throughout the system rather than being centralized in a single device [18]. In a distributed control, individual control loops are each controlled by a local controller, which communicates with other controllers and field devices such as sensors and actuators through a network. In the distributed control strategy, information is exchanged between the controllers while the decentralized control scheme does not exchange information.

Fig. 2.1 (b) and (c) shows the decentralized and distributed control strategies and their differences [3]. The dashed lines represent the communication links of robots. In the perspective of multi-robot systems, Olfati-Saber [19] defines the distributed control strategy if the communication occurs only between the neighbors. This means that if each robot communicates with all the other robots, the control strategy is not distributed. The distributed control system is essential for the system's scalability.

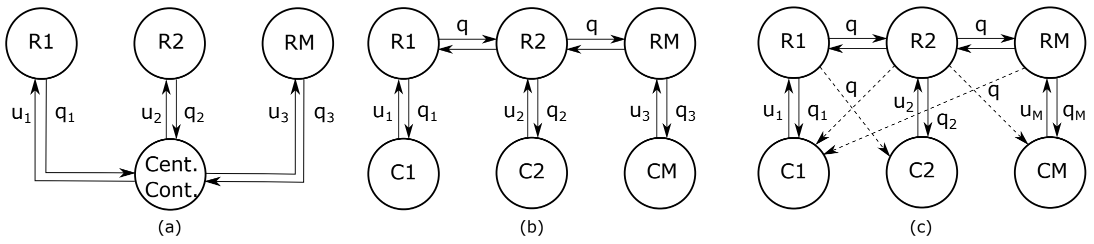

Figure 2.1: Decentralized and Distributed Control Strategy. Adapted from [3]

2.2 Formation Control Strategies

Formation is a form of coordination in MRS, in which each robot must maintain relationship with respect to neighboring robots. The interconnections between robots are modeled as edges in a directed acyclic graph, labeled by a given relationship [2]. The aim of formation control is to generate appropriate control commands to drive multiple robots to achieve the prescribed constraints on their own states, and a large body of the research has focused on consensus-based formation control, which utilises the inter-robot distance information to allow the formation to retain a certain shape while navigating [20].

2.2.1 Leader - Follower Method

In this method, one of the robots is assigned as the leader,while others are assigned as followers. In this video the leader's trajectory specifies the entire formation trajectory. On the other hand, the main disadvantage of this approach is that there is no explicit feedback from the followers to the leader [2]. In addition, the failure of the leader may fail the entire formation [3].

Desai, Ostrowski and Kumar [21] proposed two types of decentralized leader-follower control strategies named as Separation-separation and separation-bearing. In separation-bearing, denoted as $l - \phi$, control, the aim is to maintain the desired distance, l, and desired angle ϕ between the leader and the follower as depicted in Fig. 2.2(b)

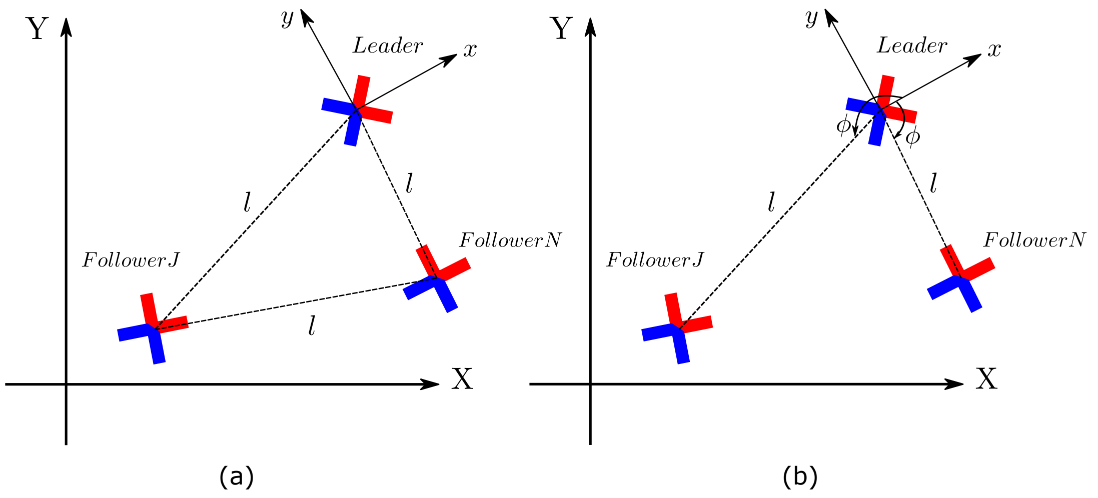

Figure 2.2: Leader-Follower Methodologies (a) Separation-Separation (b) Separation-Bearing

The objective of the separation-separation, denoted as $l - l$, control is to maintain the desired distance between the follower and its two neighbors as shown in Fig. 2.2(a).

Some works based on a standard leader-follower structure can be found in [22], [23], [24]. In [25], a *virtual leader* is defined in order to replace the actual vehicle since the system robustness is critical.

2.2.2 Virtual Structure Method

In the virtual structure method, the desired motion of a predefined virtual structure is converted to the desired motion of each robot via rotation and translation matrices [26]. Each robot tracks its desired trajectories via its individual controllers. Since it is easy to define the relationships of the robots, it is easy to obtain stable formations, especially during maneuvers. However, this approach limits the ability of formation reconfiguration and this approach also leads to a single point of failure [27]. Do and Pan [28] propose a combination of virtual structure and path tracking method to allow the change of formation shape.

2.2.3 Behavior Based Method

Balch and Arkin proposed a behavior-based method in [29]. The behavior-based method defines weights for each behavior and the formation problem is solved by the summation of these weighted behavior vectors. Generally, the behaviors defined in articles for the formation control are *move-to-goal, avoid-obstacle, avoid-robot, maintain-formation* behaviors. The final controller output was determined as the weighted combination of each behavior. The advantage of the behavior-based method is decentralization since it does not requires communication. However, no guarantee on the stability of the formation is available.

Antonelli et al. [30] present a behavioral-based method that handles static and dynamic obstacles. In [31], several behaviors such as moving to the goal, avoiding obstacles, wall-following, avoiding robot, and formation keeping are defined for formation generation and formation keeping in cluttered environments. Lee and Chwa [32] propose a decentralized algorithm using only defining behaviors based on relative positions between neighbor robots and obstacles.

2.2.4 Artificial Potential Field Method

The artificial potential field (APF) method proposed by Khatib [33] for a single robot in a cluttered environment, has the advantage that it can be extended to use in multi-robot systems by introducing inter-robot forces. Since the method requires less communication and fewer calculations, it is preferable for real-time applications. On the other hand, the local minimum problem can be considered as the main drawback. Local minimum problem for APF occurs when all artificial forces are balanced out, such as when an obstacle is directly in the path of the robot or when there are many obstacles close together.

In [34], collision-free, distributed and bounded potential functions are presented for each robot. The method presented by Masoud [35] allows the robots to leave or join the formation. Wu et al. [36] proposed a collision avoidance based on obstacle envelope modeling. In [37], several local attractive and repulsive potentials are defined for obstacles and robots. Moreover, the robustness is increased with the global attractive potential field.

2.2.5 Comparison

Table 2.1 compares three of the main formation control strategies with. The most commonly adopted strategy is the leader-follower control scheme from deployment perspective. The wide range of its applications cover most unmanned vehicle platforms and are not limited to just mobile robot platforms. Its relative simplicity in terms of implementation is the deciding factor in such a diverse range of applications. The leader-follower approach is similar to the commonly observed group management structures, an individual is selected as the supervisor and all the other members of the group behave according to the guidance of the leader. As a result of this centralized structure, the formation relationship can be more explicitly observed. The leader-follower approach also utilizes a centralized communication structure and therefore requires stable connections to all other vehicles in the group. There is no need for extra connections between the members of the group which decreases the number of connections and overhead of communication compared to decentralized approaches.

The disadvantage is, the overall structure is heavily dependent on the performance of the leader vehicle which increases the risk of failure. If a member of the group loses connection to the leader or there is a malfunction in the leader vehicle, the formation becomes hard to control.

Virtual structure approach leads to a more stable formation because the vehicles are designed to follow the rigid body of the virtual structure. The downside is that the overall structure becomes harder to modify and adapt. In order to incorporate changes in formation to the system, the virtual structure design needs to be reworked which comes at a cost in terms of computational resources. The inflexibility of virtual structure particularly imposes a limitation on critical features of a multi-robot system such as obstacle avoidance. This means that formation shape regeneration is infeasible for virtual structure as given in Table 2.1.

Three main types of formation problem is summarized. The first problem type is formation generation and maintenance. This involves creating a formation shape from a starting point where unmanned vehicles are situated at random locations and orientations. Once the shape is achieved, it must be sustained to carry out the mission. Formation maintenance during trajectory tracking is the second problem type. This category requires maintaining the formation shape with precision while the formation is in operation, following a predefined trajectory.Finally, formation shape variation and re-generation is the last formation control problem type. Similar to Type 2, this category necessitates maintaining the formation shape; however, shape modification and re-generation are also required while avoiding an obstacle [1].

In the behavior-based control strategy, a single control command is able to carry out multiple sub-tasks that are required in a mission, which makes it the most adoptable methodology. However, the difficulties in obtaining a systematic mathematical representation and absence of a stability analysis makes it infeasible for wide-spread application. A one-size-fits-all approach is hard to come up with and therefore, trends of future development hint that hybrid approaches are adopted for different situations. For instance, in the open space with a high priority of system stabilization, the leader-follower strategy or the virtual structure methodology can be used. If the application necessitates navigation in a complex environment, the behavior-based approach be-

Methods	Advantages	Disadvantages	Formation maintenance types
Leader-follower	1. Easy to be designed and implemented 2. Efficient communication within the system	1. Highly dependent on the leader vehicle 2. Lack of the feedback from the follower to the leader	Formation generation Formation maintenance during trajectory tracking Shape regeneration
Virtual structure	1. Good performance in shape keeping 2. Good representation of the relationship and the coordination between each vehicle in the formation	1. Not flexible for shape deformation 2. Not easy for collision avoidance	Formation generation Formation maintenance during trajectory tracking
Behavior-based	1. Capable of dealing with multi-task mission	1. Not easy to mathematically express the system behavior 2. No guarantee on stability	Formation generation Formation maintenance during trajectory tracking Shape regeneration

Table 2.1: Comparison of formation control strategies. Adapted from [1]

comes preferable.

This subsection compares the main control strategies of formation control challenges: i) formation shape generation; ii) formation course tracking; iii) formation reconfiguration and iv) task allocation. Table 2.2 lists the features of formation control methods an their capacity to overcome the main formation problems. Also, table 2.2 compares the control strategies according to the following features: i) centralization vs. distribution; ii) stability of formation; and iii) real-time implementation.

2.3 Contributions and Novelties

This thesis develops of a novel approach for the formation control of multiple robots that combines MPC with graph generation algorithms. It enables robots to form a formation starting from arbitrary points, increasing the flexibility and adaptability of the control strategy to different environments and problems.

Morover, this thesis develops formulations of the MPC problem and analyzes the trade-offs between computational complexity and performance. Specifically, the effect of sharing prediction information among robots on performance has been evaluated. Sharing prediction information by sampling has been suggested as an innovative method. In this way, while the communication load and calculation time of the system is reduced, the distance and control effort are reduced. The use of MPC techniques to deal with uncertainty and disturbances in the system is also presented. One of the main advantages of the proposed method is that the system is scalable thanks to the distributed development of the MPC.

Simulation and experimental results demonstrate the effectiveness of the proposed control strategy in maintaining the desired formation of multiple UAVs under various scenarios. Overall, this thesis presents a comprehensive study on the use of MPC for the formation control of multiple UAVs, and the combination of MPC with graph generation algorithms is highlighted as a key innovation.

	Virtual structure	Behavior-based	Leader-follower	Graph-based	Potential Field
Formation shape generation	No	Yes	No	Yes	Yes
Formation trajectory tracking	Yes	Yes	Yes	Yes	Yes
Formation reconfiguration	No	No	No	Yes	Yes
Task assignment	No	Yes	No	Yes	No
Distribution vs centralization	Low	High	Medium	Low	High
Formation stability	High	Low	Medium	High	Medium
Real-time implementation	Low	High	High	Low	High

Table 2.2: Comparison between formation control strategies and their ability to cover the main formation problems and challenges. Adapted from [2]

CHAPTER 3

BACKROUND INFORMATION

3.1 Introduction to Graph Theory

We model a formation composed of N robots labeled by $i \in \mathcal{V} = 1, ..., N$ with a undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where the vertex set $\mathcal{V}$ represents the robots, and the edge set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ contains the pairs of robots (i, j) for which robot i and robot j are neighbors. The state of the i^{th} robot is represented by q_i. Euclidean distance is denoted with the $\| \ . \ \|$, and $d_{ij} = \| \ q_i - q_j \ \|$ is the Euclidean distance between the robot i and robot j. The desired distance between the robot i and robot j is denoted as $\overline{d_{ij}}$. The formation is defined as $\mathcal{F} = (\mathcal{G}, \mathcal{D}_{\mathcal{F}})$ where the $\mathcal{D}_{\mathcal{F}} = \left(\ \overline{d_{ij}} \ \mid \ (i, j) \in \mathcal{E} \ \right)$ is the objective distance set.

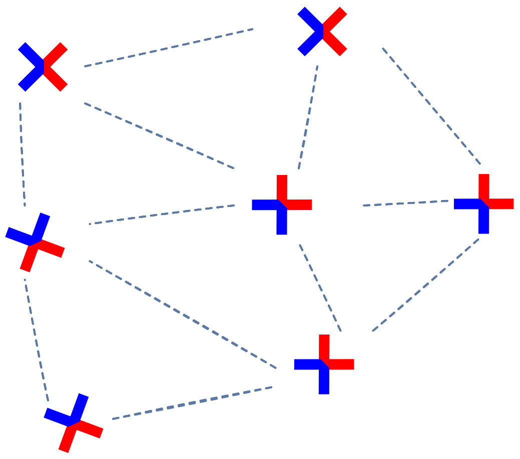

Figure 3.1: A graph network that represents a formation of robots

As seen in Figure 3.1, mesh networks can be constructed from a set of triangle in form of graph. Swarm systems can sustain the triangulated formation by controlling of distance as defined edges.

3.2 Formation Graph Generation using Delaunay Triangulation

The process of generating a graph using triangulation involves utilizing mathematical algorithms that partition a given set of points into non-overlapping triangles that are mutually connected. Triangulation algorithms are a set of mathematical techniques employed to accomplish this. In the context of graph generation, the triangles created by these algorithms can be utilized to establish edges between the points, resulting in a planar graph. The vertices of the graph correspond to the points used in the triangulation, and the edges correspond to the triangles.

To construct a graph using the triangulation method, the process begins with an empty set of triangulation edges. Subsequently, for each pair of robots, an edge is drawn. If this edge does not intersect with any of the edges in the triangulation set, it is added to the triangulation edge set.

The computational complexity of this algorithm is $O(n^3)$ because for each pair $O(n^2)$ we compare intersection $O(n)$. This is the simplest method for triangulation but it is not efficient and gives a nondeterministic triangulation results.

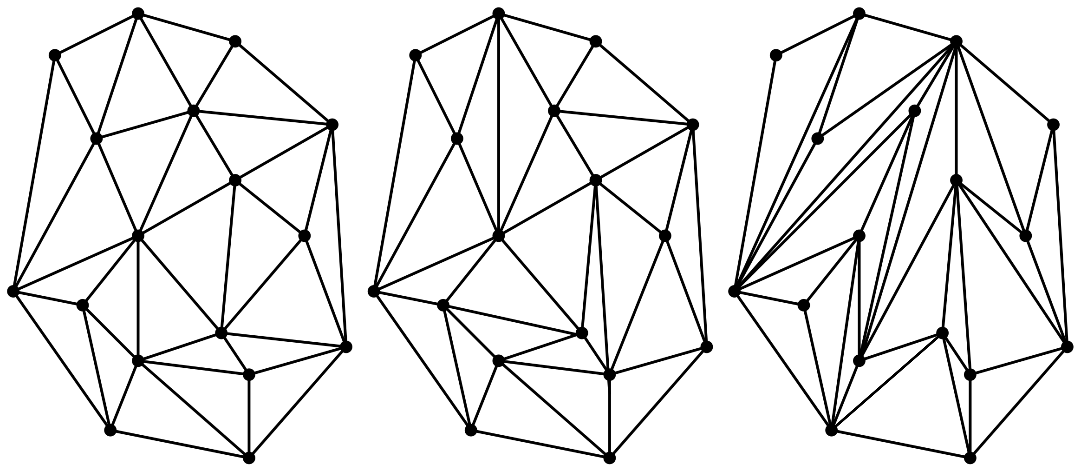

Figure 3.2: Graph Triangulation Instances

The resulting triangles and their corresponding edges may differ depending on the algorithm used. These variations can ultimately lead to differences in the order and structure of the resulting graph. Therefore, the choice of triangulation algorithm can have a significant impact on the resulting graph order. Fig. 3.2 demonstrates the effectiveness of different triangulation techniques in achieving improved and evenly-distributed results, with superior results appearing on the left-hand side of the image. The triangulation approach utilized on the left-hand side of the figure is obtained through the use of Delaunay triangulation.

Delaunay triangulation is a type of triangulation algorithm that creates a triangulation of a set of points in a way that satisfies a specific criterion named as Delaunay criterion. The Delaunay criterion states that a triangulation of a set of points is a Delaunay triangulation if and only if the circumcircle of each triangle in the triangulation contains no other points of the set. In other words, the Delaunay triangulation is the unique triangulation that maximizes the minimum angle of all triangles in the triangulation and ensures that no point is inside the circumcircle of any triangle. This property makes the Delaunay triangulation useful in many applications where the quality of the triangulation is important, such as in finite element analysis, mesh generation, and computer graphics.

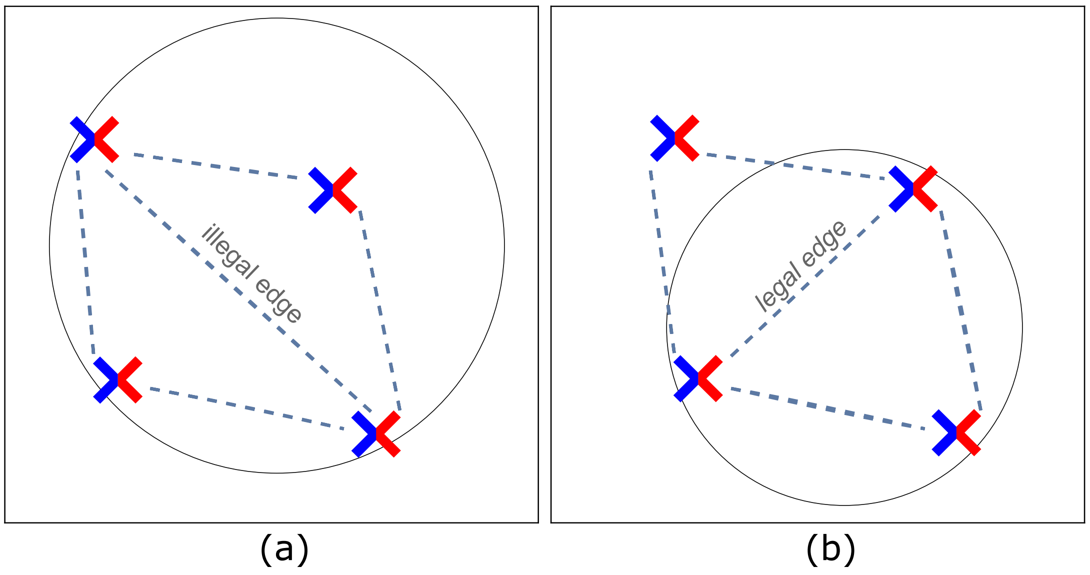

Figure 3.3: Illegal-Legal Triangulation Examples

In a Delaunay triangulation, an edge is considered illegal if the circumcircle of the triangle it forms with its neighboring triangles contains another point within it. If an edge is illegal, it violates the Delaunay criterion, which states that the circumcircle of any triangle in the triangulation should not contain any other points. On the other hand, a legal edge in a Delaunay triangulation is an edge that satisfies the Delaunay criterion, meaning that the circumcircle of the triangle it forms with its neighboring triangles does not contain any other points. A legal edge can be added to the triangulation without violating the Delaunay criterion. Legal triangulation is actually to tend to choose shorter edge. To explain more formally, there are 4 points to be triangulated $P = p_i, p_j, p_k, p_l$. If the point p_i lies in in interior of circumcircle of p_j, p_k, p_l, $p_k - p_l$ is illegal edge, but if it is in interior circumcircle, $p_k - p_l$ is a legal edge. As seen in Fig. 3.3 on the left side p_i is interior of circumcirle of p_j, p_k, p_l and $p_k - p_l$ edge is illegeal but on the right side p_k is not in interior of circumcircle of p_i, p_k, p_j so $p_i - p_j$ is a legal edge and this triangulation is legal.

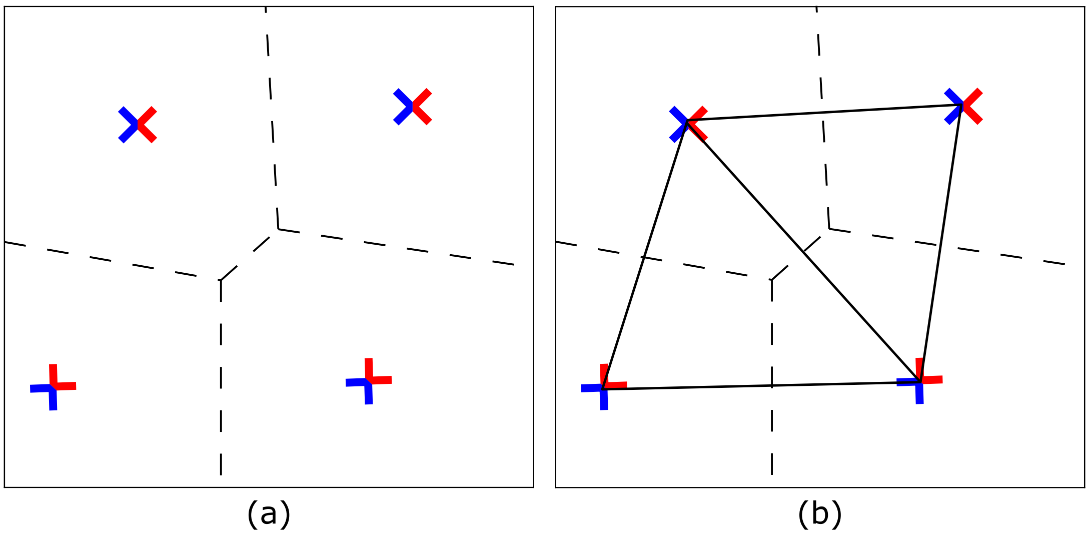

Figure 3.4: (a) Voronoi Diagram (b) Delaunay Triangulation using Voronoi Diagram of 4 UAVs

Delaunay triangulation can be obtained by several algorithms such as sweep plane algorithm [38], randomized incremental construction [39] and conversion from Voronoi diagram. One of the most known and simplest method is conversion Voronoi diagram to triangulation by retrieving edges between neighbour Voronoi cells. This method has a $O(n \cdot logn)$ time complexity in worst case. Voronoi diagram is simply defines the regions which has a closest points set. Given points $P = p_0, p_1, .., p_n$, the Voronoi region of point p_i, $V(p_i)$ is the set of points at least as close to p_i as to any other point in P, as in the Eqn. 3.1.

$$V(P_i) = \{ \quad q \quad | \quad ||p_i q|| \ < \ ||p_j q|| \quad , \forall j \neq i \quad \} \tag{3.1}$$

Where $||pq|| = \sqrt{\sum_{j=1}^{d}(p_j - q_j)^2}$. Voronoi diagram can be computed using sweep line Fortune's algorithm [40] which has $O(n{\cdot}logn)$ time and $O(n)$ storage complexity. As seen in Fig. 3.4, when edges are drawn between neighbour sites of each site, Delaunay triangulation can be simply acquired.

To construct the Delaunay triangulation using the Voronoi diagram, the following steps should be followed in a methodical manner. Firstly, compute the Voronoi diagram of the given point set. The Voronoi diagram comprises of polygons, each representing the region of space that is closer to that point than any other point in the set. Next, draw a line segment for each edge that connects two adjacent polygons in the Voronoi diagram. Each line segment represents an edge in the Delaunay triangulation. Subsequently, remove any edges that intersect with other edges in the Delaunay triangulation. Such intersecting edges are not part of the Delaunay triangulation and should be eliminated. Repeat steps 2 and 3 until no more edges can be added or removed, to ensure the construction of the complete Delaunay triangulation.

3.3 Model Predictive Control

Model Predictive Control (MPC) is a form of control that looks ahead in time to predict the future behavior of a system, and uses this information to optimize the control actions that are applied to the system. In contrast to traditional control methods, which only consider the current state of the system, MPC takes into account the dynamic behavior of the system over time, and uses optimization techniques to find the best control actions to achieve a desired objective. This makes MPC well-suited for applications where there are constraints or uncertainty, and where it is important to control the system in a precise and efficient manner. MPC is used in a wide range of applications, including robotics, aerospace, chemical processing, and power systems.

In MPC, an optimization problem should be defined with a cost function and constraints. For a robot, the cost function of MPC can be selected as the positional error or energy consumption and the constraints can be defined as the obstacles, velocity limits or dynamic limitations of robot. The optimization problem is solved iteratively over a series of time steps. At each time step, the MPC controller predicts the future behavior of the system over a fixed time horizon, and then computes a sequence of control actions that minimizes the cost function while satisfying the constraints on the system's state and control inputs. Once MPC calculates the control input sequences, only the first element of the sequence is applied to the system. In the next time step, the system's state is updated using the system's dynamics model, and the optimization problem is re-solved at the next time step using the updated state. This process is repeated at each time step until the end of the time horizon is reached. By solving the optimization problem iteratively over a series of time steps, the MPC controller is able to adapt to changes in the system's behavior and continue to find the optimal control actions. This allows the MPC controller to maintain good performance even in complex and dynamic systems.

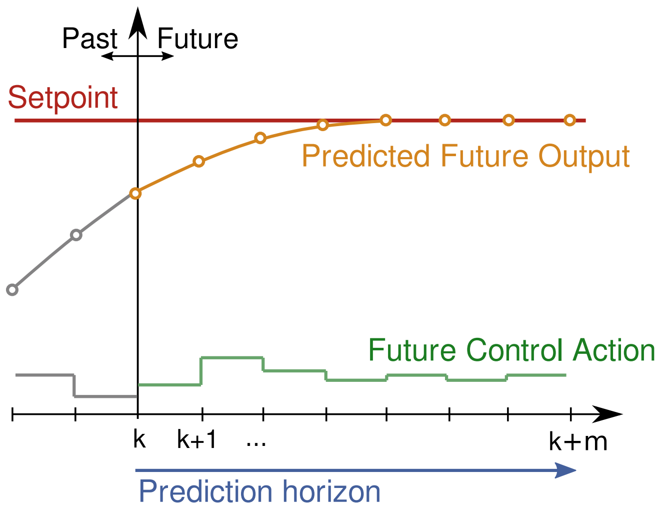

Figure 3.5: Model Predictive Control

To design an MPC controller, a mathematical model of the system's dynamics is first developed. This model describes how the system's state (such as position, velocity, and orientation) evolves over time in response to control inputs. The model can be derived from first principles, or it can be estimated from experimental data.

Next, the objective of the MPC controller is identified. This could be to track a desired trajectory, minimize energy consumption, or maximize performance. The MPC control problem is then formulated as an optimization problem, where the objective is to find a sequence of control actions that minimizes a cost function while satisfying constraints on the system's state and control inputs.

An optimization algorithm is used to solve the MPC control problem. This typically involves discretizing the time horizon over which the control actions will be applied, and then iteratively solving a sequence of optimization problems to find the optimal control sequence.

Once the MPC controller has been designed, it is implemented on the system and used to control its behavior. Feedback from sensors on the system is used to update the model and re-solve the optimization problem at each time step. This allows the controller to adapt to changes in the system's behavior and to continue to find the optimal control actions.

The performance of the MPC controller can be validated by testing it on the system in a variety of scenarios, and making any necessary adjustments to the controller design or optimization algorithm. Overall, MPC is a powerful control method that can be used to control complex, dynamic systems in a precise and efficient manner.

3.4 Crazyflie

The Crazyflie 2.1 (Bitcraze, Sweden) [41] is a mini autonomous flying robot with four coreless brushed motors, as shown in 3.6. An electronic circuit board with a CPU and several communication and sensing components is housed in its body. A 350 mAh single-cell lithium polymer battery powers the motors and electronic parts of the device. The Crazyflie weighs 27 grams in its most basic version, but when equipped with the essential positioning hardware, it weighs 37 grams. This enables it to fly for about four minutes while carrying additional equipment. The Crazyflie, however, is unable to fly autonomously in its most basic configuration since the IMU it carries is insufficiently accurate and has drifting problems.

Due to its appealing design, the software architecture developed for physics-based simulations may be simply altered to operate with actual robots. With a few minor adjustments, the high-level controller node employed in the simulations can also be

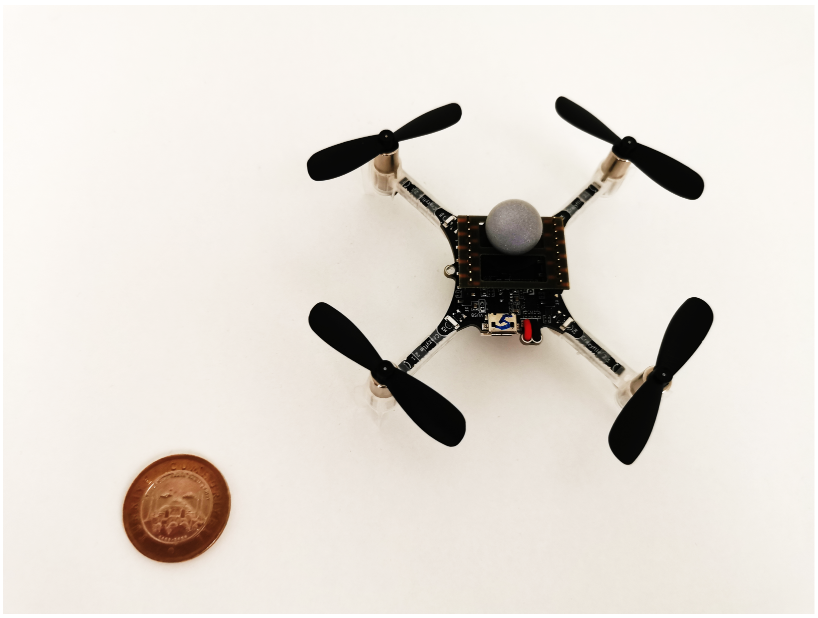

Figure 3.6: Crazyflie 2.1 used in experiments

used to actual robots. The Crazyflie firmware is utilized in physics-based simulations with the SITL (software in the loop) technique, and the communication ports are located on the station computer. These ports are used by the high-level controller node to transmit and receive flight orders and state data. These ports are adjusted to correspond to the serial radio connectivity for actual robots.

3.5 Robot Operating System (ROS)

The Robot Operating System (ROS) is a set of software libraries and tools that help build robot applications [42]. ROS creates a communication network and operating system that runs on Linux. This network and operating system handle synchronization, routing, and communication tasks for a given communication architecture, and outputs the results to designated routes for specific applications. When used with a real robot, a ROS framework on a computer can accept sensory input, such as images from a camera, signals from a serial communication system, or information from the internet. ROS is a software framework that allows users to program various outputs, such as motor control signals, environmental estimates, or signals for the robot to act on. It also provides a way for developers to create ROS-specific libraries, programs, and algorithms that can be shared within the community. This has helped to accelerate the use of ROS, particularly in the field of robotics. Many hardware products, such as LIDARs, motors, cameras, and aerial vehicles, now come with ROS libraries, drivers, and tutorials, which makes them more appealing to ROS users. These collections of libraries, drivers, and applications are known as ROS packages.

In this thesis, We used ROS to control the Crazyflie drone. The high-level controller codes that generate reference velocity commands, as well as commands to start, stop, take off, or land are written in Python. The low-level controller programs, such as the position controller, velocity controller, attitude controller, and motor drivers, are located in the Crazyflie firmware and maintained by its developers. The high-level controller only specifies what action to take, such as what velocity to maintain or what position to reach.

The ROS programs that are used for physics-based simulations can be easily adapted for use with real robots by modifying the destination of the commands produced by the high-level controller, and the source of feedback about the platform.

3.6 Experimental Setup

In the experiment setup, the most common and open tools are preferred for fast integration and testing. For that purpose infrastructure has built up with following hardware and software configuration.

Hardware Configuration

1. Crazyflie2.1
   (a) FreeRTOS
   (b) Onboard Sensors: Imu, barometer
   (c) Extra Sensors: Optic flow, LH Deck, UWB Deck, MoCap Markers
2. Base Station
   (a) Intel i7-10700H:

    (b) Nvidia Quadro RTX3000

3. Positionning Infrastructure:
    (a) Lighthouse Base Stations,
    (b) UWB Anchors
    (c) Motion Capture

Software Configuration:

1. Ubuntu-18.04
2. ROS Melodic:
3. C++/Python:

Supported Features by Software Architecture:

1. Central and distributed simulation
2. Central and distributed real flight
3. Hybrid usage can be used as well.
4. Different positioning systems are supported like lighthouse, UWB and motion capture.

The crazyswarm [43] is developed at University of Southern California and used in the most of projects about swarm UAVs. Crazyswarm is developed using ROS/C++ and provides well documented Python API to control swarm of UAVs, but it supports only motion capture localization system and central management of UAVs by base project.

In our test environment, crazyswarm project is used as base and some modifications and extensions are carried out to support multiple localization system and distributed UAV control. For distributed control, each UAV starts as separate and simultaneous ROS node and for controlling dedicated UAV crazyflie class which is part of crazyswarm API is used. Each UAV controller node is publish control commands

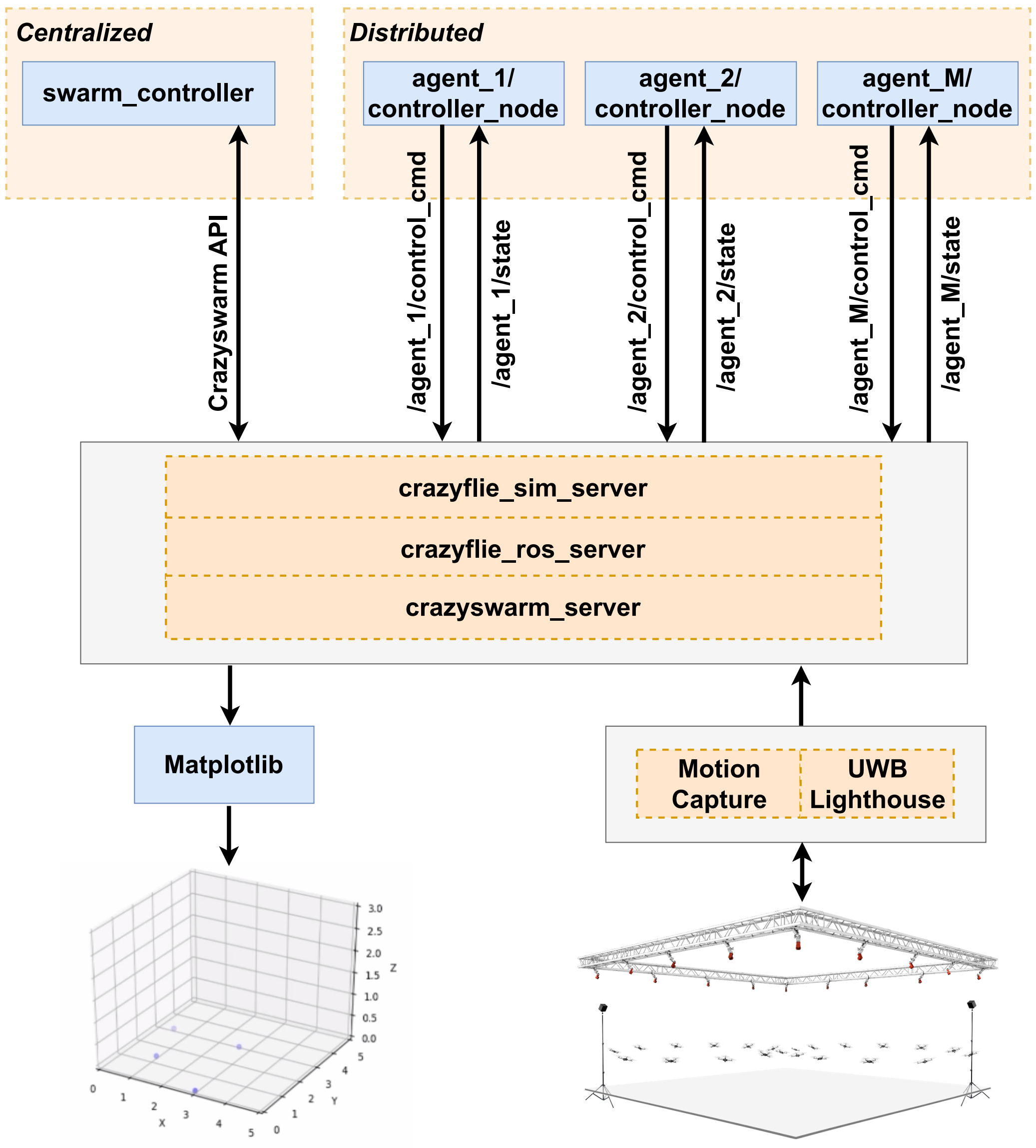

Figure 3.7: System Architecture of Test Environment

like takeoff, landing, cmd_vel, stop etc.. to crazyswarm_server and listen to releated drone state topics sent by crazyswarm server such as position, battery.

Crazyflie supports several positioning systems which are classified as onboard and external positioning system. In onboard positioning system UWB deck and lighthouse deck are used on crazyflie as positioning hardware. Otherwise, in external positioning system, motion capture systems are widely used [44]. In our test environment, experiments can be realized via these three positioning systems. Having different type of localization system brings some advantages for our test environment. Each positioning system has different sensitivity of position such that error range of calculation can be ranked as UWB($10cm$) $>$ Lighthouse($10^{-1}cm$) $>$ Motion Capture($10^{-2}cm$) and each of them has different calculation frequency. This variability offers the opportunity to experiment with algorithms under different positioning sensitivity scenarios. Furthermore, each laboratory could have different positioning system and our software infrastructure allows algorithms to be integrated in each of them. crazyswarm is already capable of an external motion capture positioning system. To extend with onboard positioning system, crazyflie_ros project is integrated into our infrastructure to handle onboard positioning system. Although the project is deprecated, onboard functionality of the project works well and ROS interface is synchronized with crazyswarm API for fast switching of positioning mode.

Crazyswarm simulation provides useful features such that it has crazyflie flight dynamics, option to add noise and same API for real flight. However, it only provides central swarm controller interface such that it takes control commands, updates drone positions and visualize in a same process and doesn't use ROS. For distributed simulation crazyflie_sim_server node is implemented. crazyflie_sim_server listens to control commands' topics, execute them via ROS and visualize UAVs in a separate process from robot controller nodes simultaneously.

CHAPTER 4

METHODOLOGY

4.1 Introduction

An optimizatıon problem is said to be non-convex if its objective function or constraints violate the properties of convexity. Convexity is a mathematical property that is characterized by the fact that any line segment connecting two points on a convex set lies entirely within the set. Convex optimization problems are generally easier to solve than non-convex optimization problems because they have a unique global minimum that can be efficiently found using optimization algorithms.

In the case of MPC-based formation control, the model is often non-convex due to the presence of nonlinear constraints or objective functions. Nonlinear functions are generally non-convex because their curvature changes along the function, causing the line segments connecting two points on the set to fall outside the set. Additionally, the constraints or objective functions may be non-convex due to the presence of products or ratios of decision variables, which can lead to complex and nonlinear shapes.

The proposed method involves a nonlinear system model and a nonconvex optimization problem that needs to be solved at each time step to generate control inputs. The nonlinear model captures the dynamics of the agents while the non-convex optimization problem ensures that the agents move in a coordinated manner while maintaining their formation. However, the nonlinear model and non-convex optimization can make the problem computationally challenging, requiring the use of advanced optimization solvers and numerical techniques. Despite these challenges, MPC-based formation control has shown promising results in various MRS applications.

4.2 Simplified Kinematic Model of Robot

We uses the single-integrator model widely used in multi-robot coordination control problems such as consensus and formation control [14, 45–48]. For a general two-dimensional case, consider a group of M mobile robots, where all robots are in accordance with single integrator dynamics. First order dynamics for robot i defined as

$$\dot{q}_i = u_i \text{ for } i = 1, 2, \ldots, M \tag{4.1}$$

where $q_i = [x_i \;\; y_i]^T$ is the position and $u_i = [u_{ix} \;\; u_{iy}]^T$ is the velocity-level control input of the i^{th} robot with respect to the world coordinate frame.

The single integrator model, can arbitrarily assign a velocity vector input, beyond the physical limits of the robot. Crazyflies and many other drones can be controlled with velocity command unless there is a drastic change in the velocity vector. In order to avoid this situation, control effort cost function is implemented to restrict a drastic change in the velocity vector. The internal controllers available in Crazyflie and the costs we determined for the input u, make it possible for us to command high-level velocity inputs.

In order to obtain a suitable kinematic model for the Crazyflie moving in a 2D plane, heading information should be added to the state. In addition, angular velocity should be taken into account for a more realistic kinematic model. Therefore, new states are defined as follows

$$q_i = \begin{bmatrix} x_i \\ y_i \\ \theta_i \end{bmatrix} \tag{4.2}$$

where x_i and y_i is the position and θ_i is the heading of the robot i. Then, the control is defined as

$$u_i = \begin{bmatrix} v_i \\ \phi_i \\ \omega_i \end{bmatrix} \tag{4.3}$$

where v_i is velocity magnitude, ϕ_i is the orientation of the velocity vector from the the heading, and ω_i denotes the angular velocity. Fig. 4.1 illustrates the states and control inputs of the Crazyflie kinematic model.

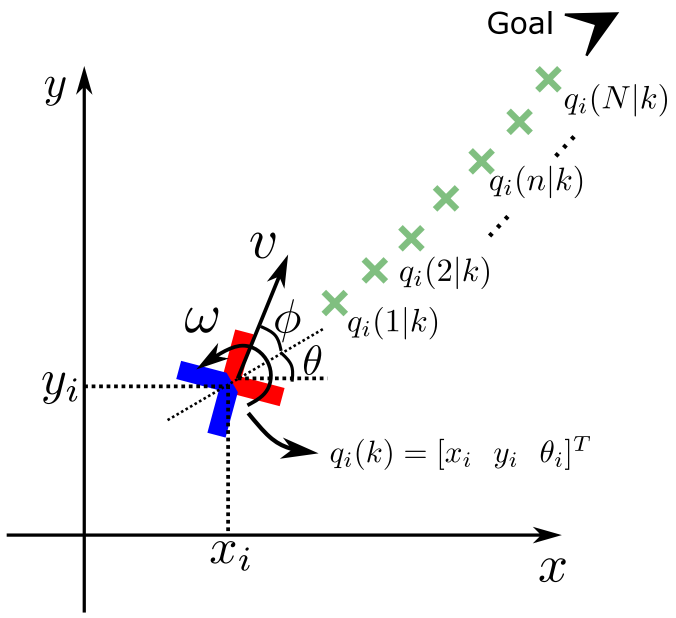

Figure 4.1: Simplified Kinematic Model. The predicted trajectory is shown with small green crosses. Note that the drone is symbolized by a half-blue and half-red cross. The red side represents the front of the drone.The black arrow shows the goal location and the final heading.

Consider a group of M robots with a discrete-time single-integrator model. Discrete-time nonlinear kinematic model is formalized as follows,

$$q_i(k+1) = f(q_i(k), u_i(k)) \;\; \text{for} \;\; i = 1, 2, \ldots, M \tag{4.4}$$

$$\begin{bmatrix} x_i(k+1) \\ y_i(k+1) \\ \theta_i(k+1) \end{bmatrix} = \begin{bmatrix} x_i(k) \\ y_i(k) \\ \theta_i(k) \end{bmatrix} + \begin{bmatrix} v_i(k)\cos(\theta_i(k) + \phi_i(k)) \\ v_i(k)\sin(\theta_i(k) + \phi_i(k)) \\ \omega_i(k) \end{bmatrix} \Delta T \tag{4.5}$$

4.3 Navigation of single UAV with Model Predictive Control

At the sampling instant k, the state vector $q_i(k)$ is available through measurement for the robot i. With the given state $q_i(k)$ and the prediction horizon N, the predicted future states are denoted as $q_i(1|k), q_i(2|k), ..., q_i(n|k), ..., q_i(N|k)$, where $q_i(n|k)$ is the predicted state for sampling instant $k+n$. The cost function for a single robot from the start to the goal point is given in Eqn. 4.6. Note that, the notation $\| z(.) \|_A^2$ is used for $z(.)^T A z(.)$ for any given vector $z(.)$ and matrice A.

$$J_{i,nav} = \sum_{n=0}^{N-1} [\ \| q_i(n|k) - q_i^{goal} \|_Q^2 + \| u_i(n|k) \|_R^2\ ] + \| q_i(N|k) - q_i^{goal} \|_{Q_f}^2 \quad (4.6)$$

Where, N is the prediction horizon, Q is the state cost matrix, R is the input cost matrix and Q_f is the final state cost matrix. q_i^{goal} is the position and heading of the given goal point for robot i, $q_i(n|k)$ and $u_i(n|k)$ are the predicted state and predicted control input at the sampling instant $k+n$ respectively.

For a single robot and an environment with no obstacles, $J_{i,nav}$ is sufficient for navigating the robot to the goal point. For this simplistic scenario, total cost function $J_{i,total}$ is considered to be equal to $J_{i,nav}$ and the optimization problem is formalized as follows,

$$\min_{q,u} J_{i,total} \quad (4.7)$$

subject to

$$q_i(0) = q_i^0 \quad (4.8a)$$

$$q_i(n+1|k) = f(q_i(n|k), u_i(n|k)), \qquad n = 0, 1, ..., N-1 \quad (4.8b)$$

$$q_{min} \leq q_i(n|k) \leq q_{max}, \qquad n = 0, 1, ..., N \quad (4.8c)$$

$$u_{min} \leq u_i(n|k) \leq u_{max}, \qquad n = 0, 1, ..., N \quad (4.8d)$$

Equation (4.8a) defines the initial condition. Equations (4.8b), (4.8c) and (4.8d) indicate the system dynamics, map margins and input constraints, respectively.

4.3.1 Control Effort Cost Function

The control effort aims to reduce the energy consumption of the control commands and hence minimizes the input effort. To achieve this, it calculates the rate of change of the input commands and penalizes abrupt acceleration and deceleration. It also increases the method's usability as a high-level controller since it prevents drastic changes in the reference control inputs. The control effort is defined as the difference equation of consecutive control inputs $\Delta u_i(n|k) = u_i(n|k) - u_i(n-1|k)$. The control effort cost penalizes the changes in the control commands.

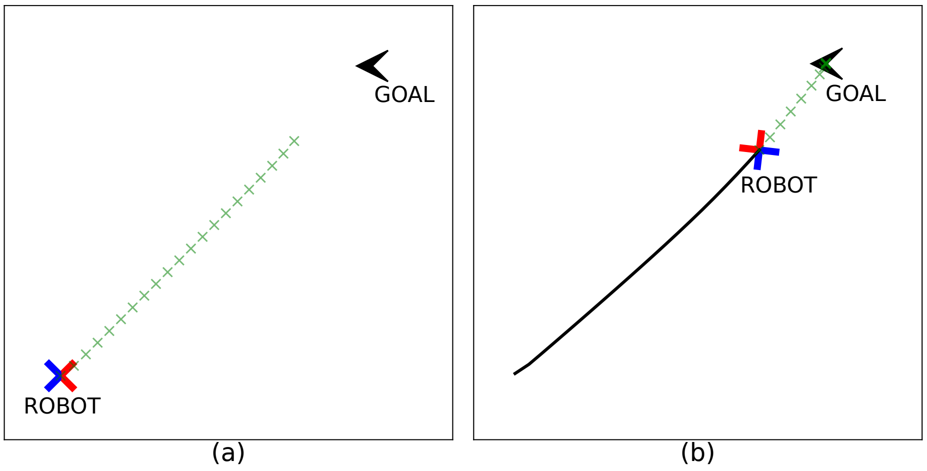

Figure 4.2: Robot is approaching the given goal point by making use of both navigation cost and control effort. (a) At the start position, robot generates its predicted trajectory as indicated with a green crosses. (b) The trajectory followed by the robot is shown with a solid black line.

Cost function for control effort can be defined as follows:

$$J_{i,eff} = \sum_{n=1}^{N-1} \parallel \Delta u_i(n|k) \parallel^2_{R_{eff}} \tag{4.9}$$

$J_{i,total}$ in equation 4.7 can be defined as $J_{i,total} = J_{i,nav} + J_{i,eff}$. Fig. 4.2 shows the drone navigating to its goal with the cost function which includes navigation and control effort.

4.4 Constraints for Obstacle Avoidance

In order to ensure safe navigation, robots need to keep a distance from the obstacles in the environment. To avoid a potential collision with an obstacle, constraints should be added to the optimization problem. In this thesis, both the drones and obstacles are considered as circular objects. The actual radius of the drone is denoted as r_{robot}. Additional distance d_{safety} is added to robot in order to increase safety. For the constraint, r_{safety} is defined as the summation of the drone radius r_{robot} and safety distance d_{safety}. $r_{obstacle}$ is radius of an obstacle and $q_{obstacle}$ is position of the obstacle.

The Euclidean distance between the center of the obstacle $q_{obstacle}$ and the center of the drone (and its predictions) can be denoted as $||q_i(n|k) - q_{obstacle}||$ at time instant k. For all the predictions and current state of each robot i, the Euclidian distance should be larger than the summation of r_{safety} and $r_{obstacle}$. Fig. 4.3 shows an environment with an obstacle and the defined radii.

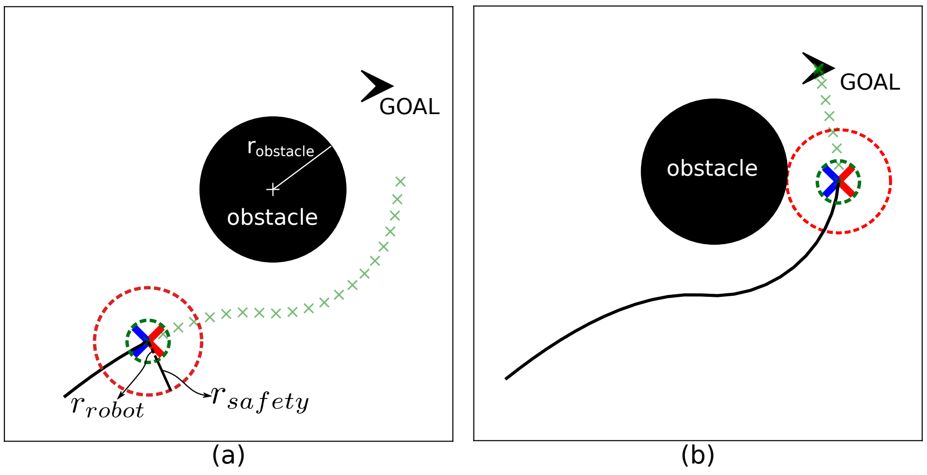

Figure 4.3: Drone is approaching to the given goal point by making use of obstacle avoidance constraint. (a) At the start position, drone generates its state predictions as indicated with a green curve. r_{robot}, indicated with a black dashed line, is the radius of the dashed green circle, r_{safety} is the distance between dashed green and red circles, $r_{obstacle}$ is the radius of the obstacle. (b) The trajectory followed by the robot is given with a solid black line.

The optimization problem given in equation 4.7 can be rewritten with updated obstacle constraint 4.11a as follows:

$$\min_{q,u} J_{i,total} \tag{4.10}$$

subject to

$$q_i(0) = q_i^0$$

$$q_i(n+1|k) = f(q_i(n|k), u_i(n|k)), \qquad n = 0, 1, ..., N-1$$

$$q_{min} \leq q_i(n|k) \leq q_{max}, \qquad n = 0, 1, ..., N$$

$$u_{min} \leq u_i(n|k) \leq u_{max}, \qquad n = 0, 1, ..., N$$

$$(r_{safety} + r_{obstacle}) - ||q_i(n|k) - q_{obstacle}|| \leq 0, \qquad n = 0, 1, ..., N \tag{4.11a}$$

4.4.1 Dynamic Obstacle

For a dynamic obstacle the center of obstacle becomes a function of time instant k, and labeled as $q_{obstacle}(k)$. Since the MPC is an online optimization process, dynamic obstacles can be considered as static obstacles for each small time instant k. By updating the $q_{obstacle}$ at the constraint in Eqn. 4.11a for each time instant k dynamic obstacle problem can be solved with the same cost function as Eqn. 4.10.

Fig. 4.4 illustrates a dynamic obstacle and a robot moving towards each other. The robot continuously calculates its own trajectory prediction based on the current position of the obstacle at each time step.

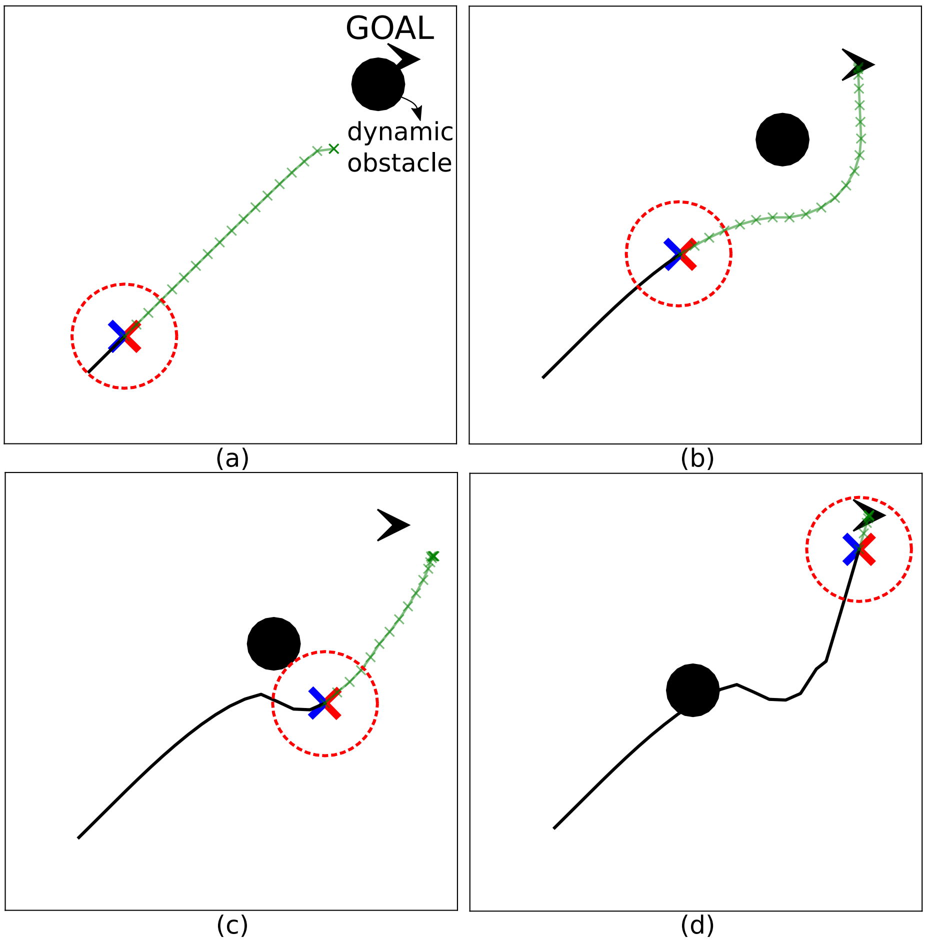

Figure 4.4: Drone is approaching to the given goal point by avoiding dynamic obstacle coming towards it (a) at time t = 1.0 s (b) t = 2.6 s (c) t = 4.0 s (d) t=6.0 s

4.5 Robots' Reciprocal Avoidance with Position Sharing Method

In this method, each robot has only the position information of other robots in the environment. As in the case of the dynamic obstacle, each robot predicts its own trajectory by adding the current positions of the other robots to its optimization problem as a constraint. Fig. 4.5 demonstrates two robots moving towards each other, where each robot plans its trajectory prediction solely based on the instantaneous position of the other robot.

In Eqn. 4.12, reciprocal avoidance constraint is defined for robot i. Given constraint should be applied for each robot j in the system. Note that, at instant k, the state vector $q_j(k)$ is available for each robot j with its own measurement and published to other robots.

$$(2r_{safety}) - ||q_i(n|k) - q_j(k)|| \leq 0, \quad n = 0, 1, ..., N \tag{4.12}$$

4.6 Robots' Reciprocal Avoidance with Trajectory Sharing Method

Introducing a system where a robot is capable of distinguishing between other robots and dynamic obstacles, while also being aware that these robots employ their own predictive models, brings about notable improvements in its path-planning capabilities. However, this enhanced performance comes at the expense of increased information sharing.

In the previous approach, each robot only required positional information in the form of three floating-point values (representing x, y coordinates and θ heading angle) for all other robots ($M-1$ in total), necessitating the sharing of $3 \cdot (M-1)$ floating-point values. However, the current method takes a step further by requiring the sharing of the complete predicted positional information of other robots. This mandates the sharing of $3 \cdot (M-1) \cdot (N+1)$ floating-point values, accounting for the predictions horizon (N).

To address the trade-off between computational and communication costs while still

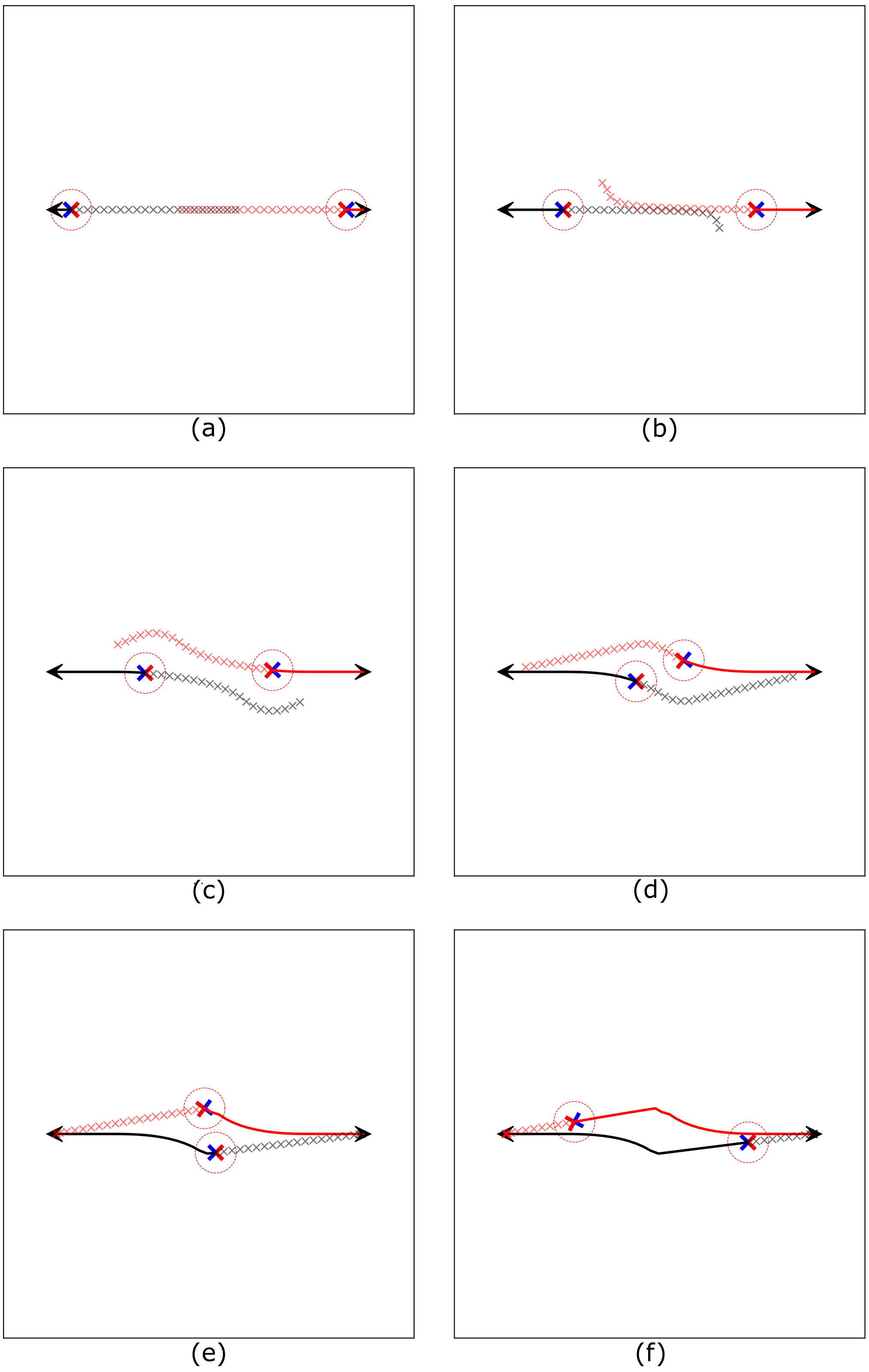

Figure 4.5: Reciprocal avoidance of two robots with position sharing method in position swap scenario (a) at time t = 0.8 s (b) t = 1.8 s (c) t = 2.6 s (d) t = 3.6 s (e) t = 4.4 s (f) t = 6.4 s

achieving performance improvements, an alternative sampling-based method is introduced in the subsequent sections. This approach effectively reduces both the computational load and the communication load. By selectively sampling points from the predicted trajectories, the method maintains a reasonable level of accuracy while mitigating the burden of transmitting and processing extensive trajectory information.

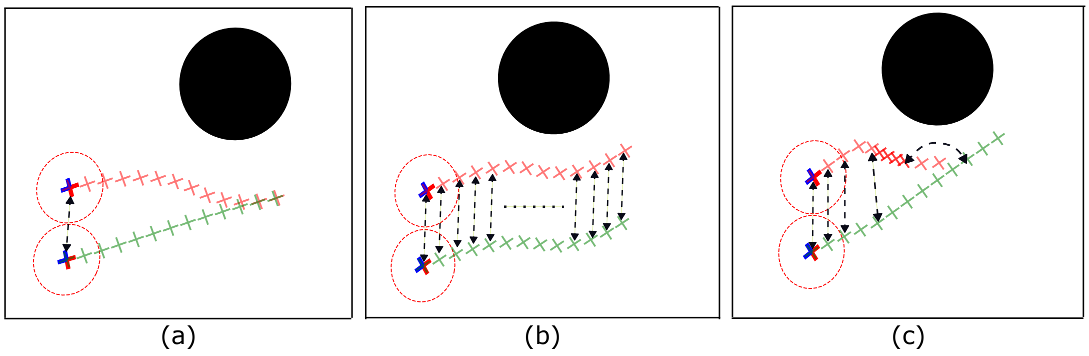

Figure 4.6: Reciprocal avoidance of two robots with (a) position sharing (b) trajectory sharing (c) sampled trajectory sharing

Fig. 4.6 illustrates the proposed methods in a simple environment involving a single obstacle and two robots. The shared information between the robots is represented by black arrows. The impact of the shared information on the individual path plans of the robots is evident from the figure.

4.6.1 Complete Trajectory Sharing

In this method, robots share their predicted state sequences, $\tau_i(k) = \{q_i(n|k)\}_{n:0,N}$, with their neighbors to solve the next optimization problem simultaneously. Although sharing all predictions increases the communication load of the system and the total calculation time, it has shortened the traveled path and reduced the control effort. The reciprocal avoidance constraint should be modified as

$$(2r_{\text{safety}}) - ||q_i(n|k) - q_j(n|k)|| \leq 0, \quad n = 0, 1, ..., N \tag{4.13}$$

In Fig. 4.6(b), it can be observed that by sharing the entire predicted trajectory, the trajectory of the lower robot adjusted based on the received information.

In Fig. 4.5, it can be observed that the robots plan collision-free trajectory, $\tau_i(k)$, based only on the instantaneous position of the other robot, $q_j(k)$, while in Fig. 4.7, each robot plans its own trajectory, $\tau_i(k)$, based on the trajectory information shared with it, $\tau_j(k)$, ensuring that its own trajectory does not collide with the trajectory of the other robots.

Although sharing all predictions increases the communication load of the system and the total calculation time, it has shortened the traveled path and reduced the control effort.

4.6.2 Sampled Trajectory Sharing

Sharing the robot's predicted trajectory through sampling can significantly reduce both communication and computation loads. While reducing the loads, it is important to perform the sampling in a way that minimizes the impact on collision avoidance and trajectory smoothness.

Due to the fact that the closer predictions of the robot contain more critical information regarding collision avoidance and trajectory smoothness, the sampling method should be chosen accordingly. Hence, the sampling was performed by defining an exponential function so that samples are more frequent from close predictions and sparser from distant ones.

The sampled trajectory is defined as $\hat{\tau}_i(k) = \{q_i(n|k)\}_{n:0,1,R,R^2,\ldots,R^a}$ for the robot i which can be shared with other robots. An exponential sampling rate R can be selected to suit for application. The choice of sampling rate influences the density of samples along the predicted trajectory. The last term that will be shared is the R^a which is the largest possible power of R less than the N.

In the following chapter, a comprehensive performance comparison is conducted among four approaches: complete trajectory sharing, uniform sampling, and two different exponential sampling rates. This analysis aims to assess the effectiveness of each method in terms of collision avoidance, trajectory smoothness, and the associated communication and computation loads.

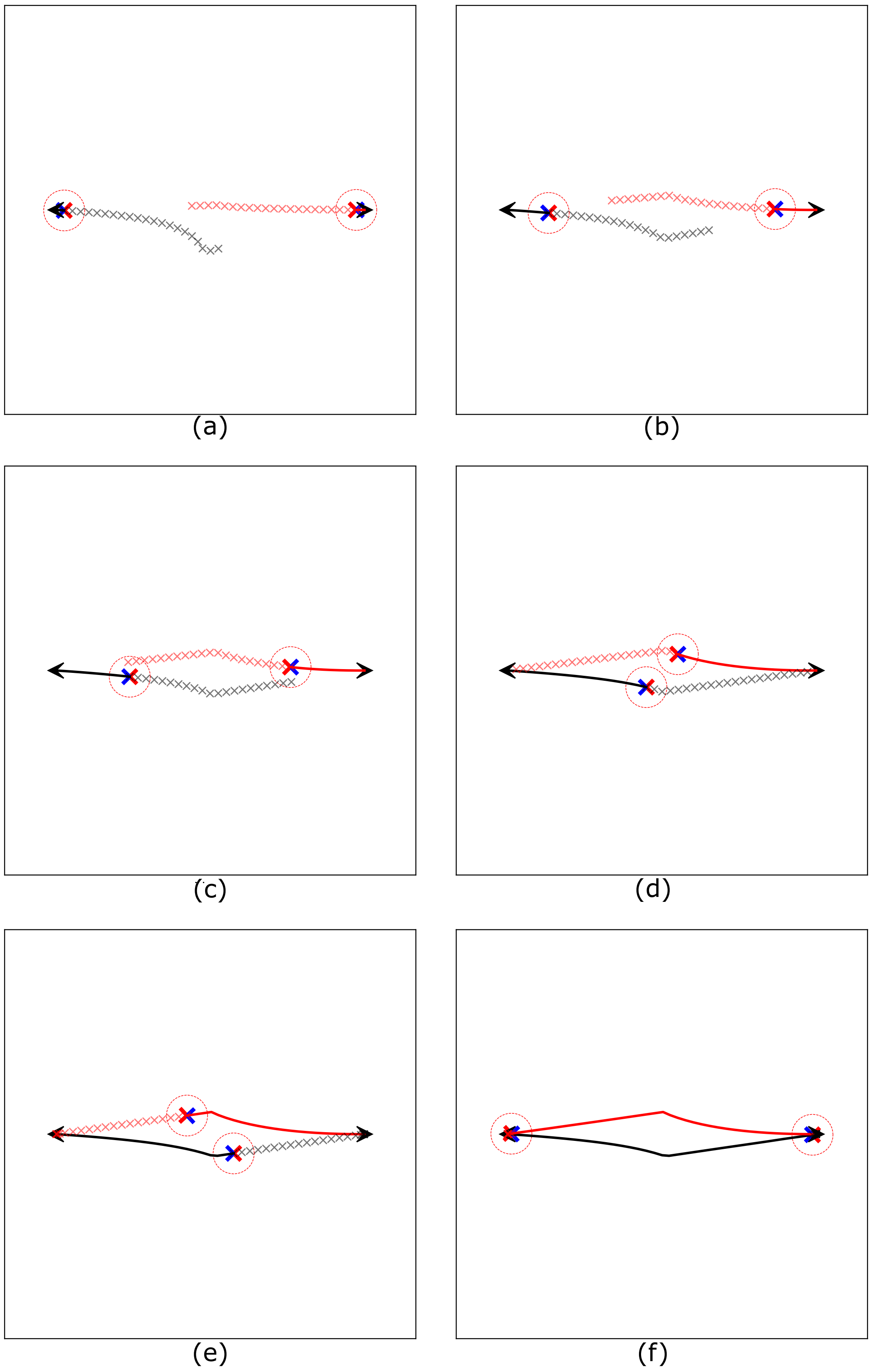

Figure 4.7: Reciprocal avoidance of two robots with prediction sharing method in position swap scenario (a) at time t = 0.6 s (b) t = 1.4 s (c) t = 2.2 s (d) t = 3.8 s (e) t = 4.4 s (f) t = 6.8 s

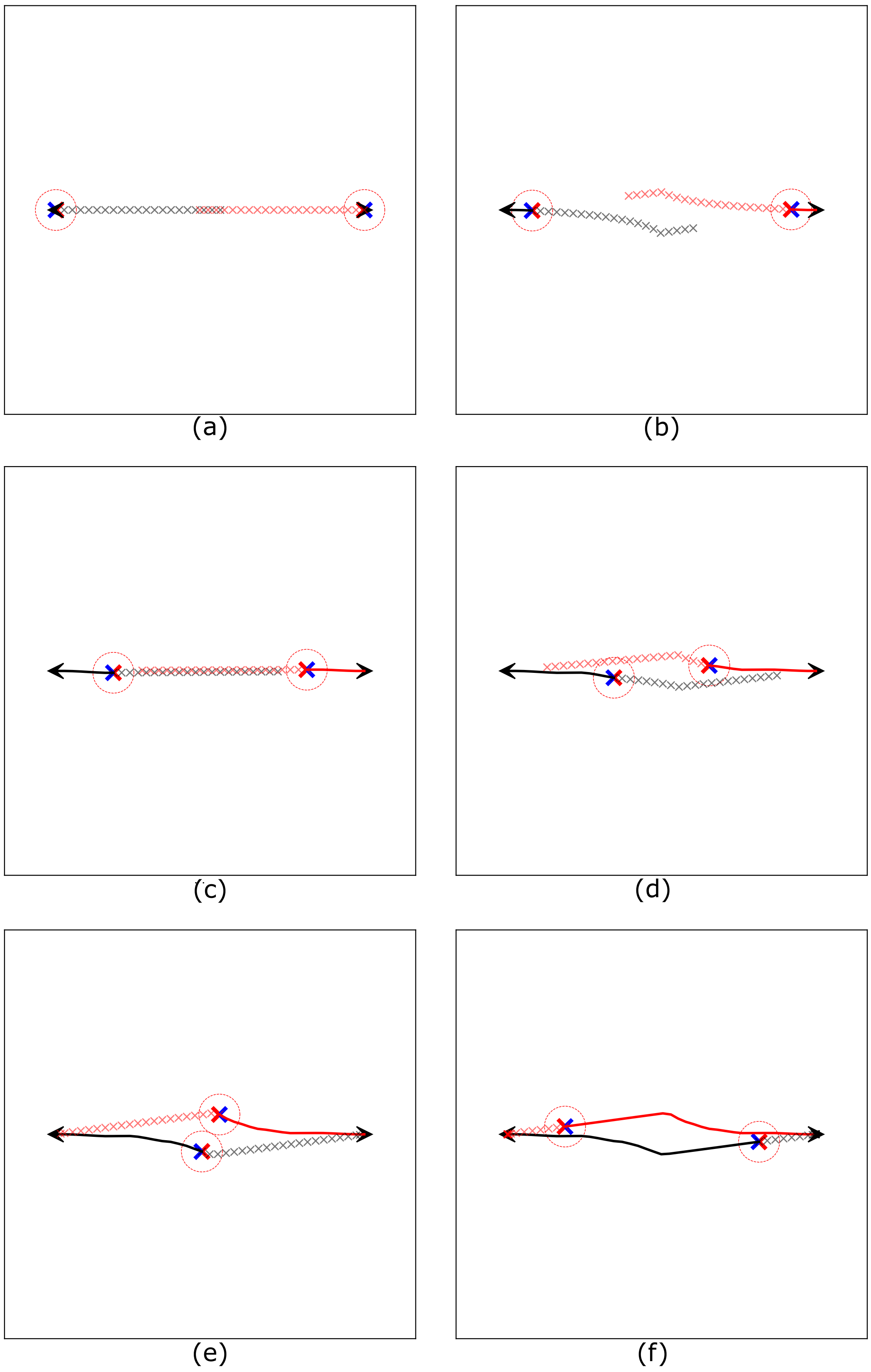

Figure 4.8: Reciprocal avoidance of two robots with sampled number of prediction sharing method in position swap scenario (a) at time t = 0.4 s (b) t = 1.0 s (c) t = 1.8 s (d) t = 3.0 s (e) t = 4.0 s (f) t = 6.6 s

4.7 Cost Function for Formation Control

To incorporate the formation behavior into the system, it is necessary to add a new cost function to the MPC. This function should ensure that the distance between two neighboring robots namely i and j is equal to the desired formation distance $\bar{d}(i,j)$.

$$\begin{aligned} J_{i,\text{form}} &= \sum_{n=0}^{N-1} \sum_{j=0}^{M_i} \Big(S_0(\| \, q_i(n|k) - q_j(n|k) \, \| - \bar{d}(i,j))^2 \\ &+ S_1(\theta_i - \theta_j)^2 \Big) \end{aligned} \tag{4.14}$$

M_i describes the number of neighbors of the i^{th} robot. $\bar{d}(i,j)$ means the given desired distance between the robots i and j. A quadratic cost is described as the difference between the measured distance and desired distance. In addition, if the heading alignment is required, a quadratic cost is added for the heading difference of robots. S_0 and S_1 are the weights for the formation cost and heading alignment cost respectively.

The optimization problem given in Eqn. 4.10 is still valid with the following update i the total cost function

$$J_{i,total} = J_{i,nav} + J_{i,eff} + J_{i,form}$$

Treating the formation as a cost rather than a constraint in MPC-based formation control offers several advantages. Firstly, it provides flexibility in adapting and modifying the formation during runtime, allowing for dynamic prioritization of formation maintenance or reconfiguration based on mission requirements or environmental conditions. Secondly, it enables smooth transitions between different formations by gradually adjusting the cost weights associated with each configuration, ensuring seamless and continuous control. Lastly, by incorporating the formation as a cost, the controller can optimize multiple objectives simultaneously, such as obstacle avoidance or energy efficiency, resulting in a versatile and adaptable control behavior. Additionally, this approach allows the controller to handle infeasible formations by relaxing the formation cost and finding the best compromise or approximation, ensuring stable and feasible control even in challenging conditions.

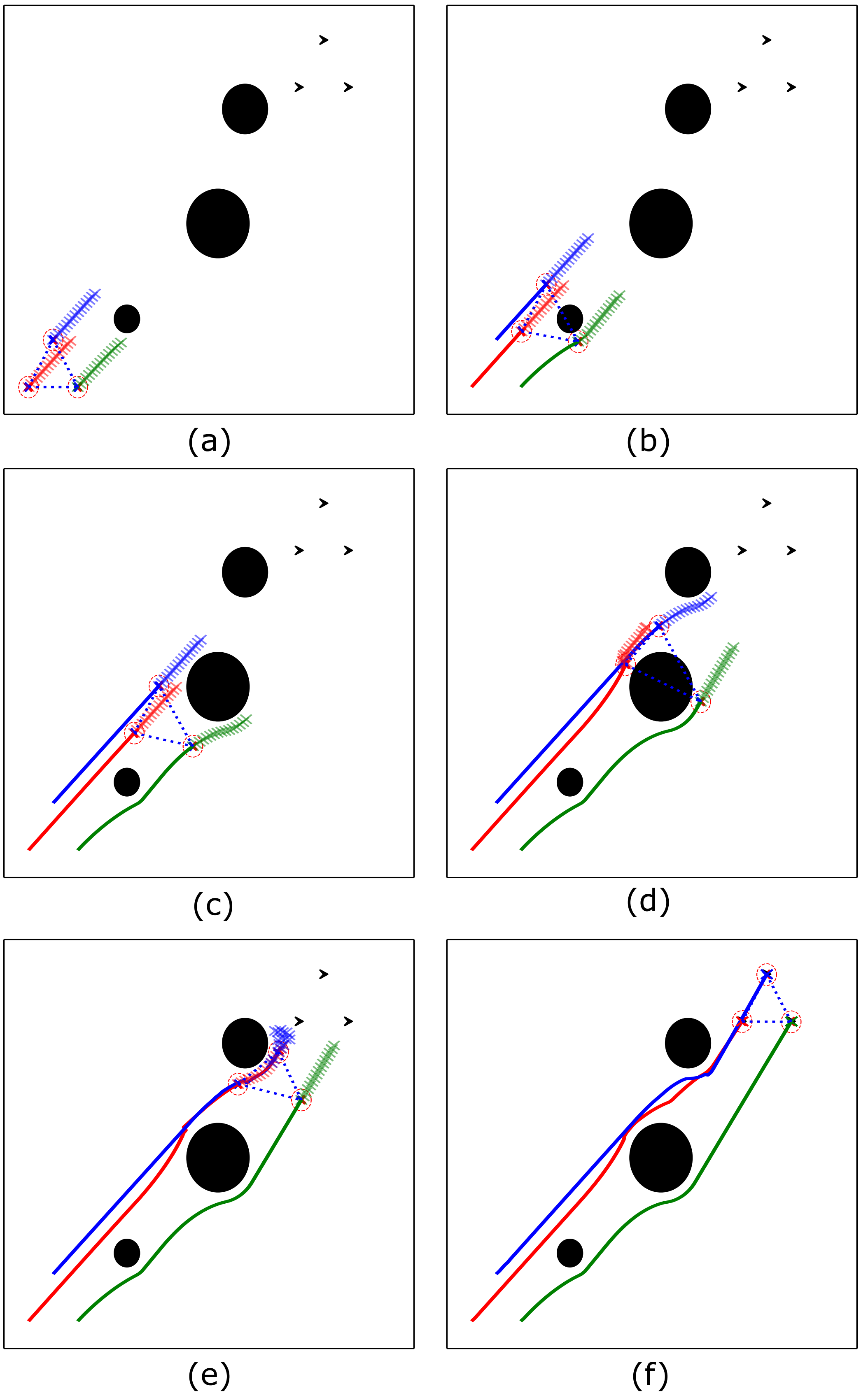

Figure 4.9: Three robot moving to goal without formation (a) at time t = 0.4 s (b) t = 4.0 s (c) t = 8.0 s (d) t = 12.0 s (e) t = 16.0 s (f) t = 20.4 s

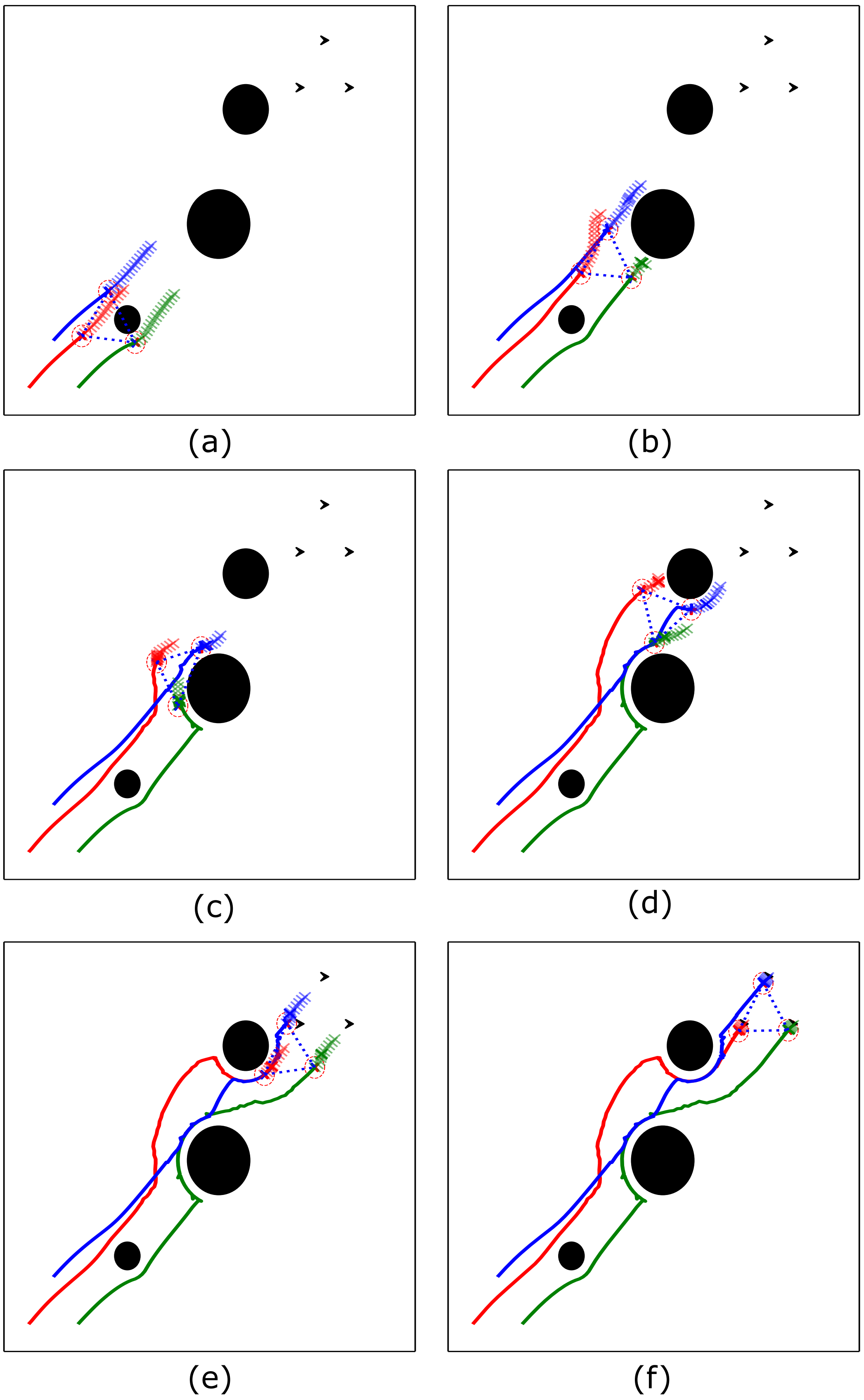

Figure 4.10: Three robot moving to goal with formation (a) at time t = 4.0 s (b) t = 8.0 s (c) t = 12.0 s (d) t = 16.0 s (e) t = 22.0 s (f) t = 28.0 s

4.8 Discussions

In the discussion section, we will provide a detailed analysis of the experimental results obtained from our study on the MPC-based formation control problem in 2D. We will start by introducing the algorithm of proposed method. Then, we will compare the performance of different optimization solvers and methods for converting nonconvex problems into convex ones, based on metrics such as computation time and solution quality.

Furthermore, we will provide an in-depth discussion on the rationale behind our decision to limit our study to 2D formation control and the implications of our findings for real-world applications. Specifically, we will examine the advantages and disadvantages of 2D formation control compared to 3D formation control, and the trade-offs involved in choosing between the two.

Finally, we will then discuss how the results of our experiments relate to the theoretical expectations and highlight the limitations and opportunities for future research in this area.

4.8.1 Solver

Model Predictive Control (MPC) is an advanced control strategy that uses mathematical models to predict future system behavior and optimize control actions. To solve the optimization problem in MPC, a variety of solvers can be employed. These include:

Interior-point solvers: These are iterative solvers that operate by finding the solution to a sequence of linear equations, and then using a Newton-based method to search for the optimal solution. Interior-point solvers are widely used in MPC due to their ability to handle large-scale problems efficiently.

Active-set solvers: These are optimization algorithms that search for the optimal solution by iteratively adding or removing constraints. They are efficient for small to medium-sized optimization problems.

Gradient-based solvers: These solvers employ gradient information to iteratively search for the optimal solution. They are efficient for problems that have smooth and convex objective functions.

Quadratic programming (QP) solvers: QP solvers are used when the MPC optimization problem can be formulated as a quadratic program. These solvers can handle large-scale problems efficiently and are widely used in MPC applications.

Sequential quadratic programming (SQP) solvers: These solvers employ a similar approach as QP solvers but use a sequence of quadratic approximations of the objective function to iteratively search for the optimal solution. They are efficient for problems that have nonlinear constraints.

The choice of solver depends on the specific requirements of the MPC problem, such as the size of the problem, the complexity of the model, and the desired level of accuracy and computational speed.

Interior-point solvers have several advantages over other optimization solvers when it comes to Model Predictive Control (MPC) applications. They can handle a wide range of nonlinear and non-convex optimization problems, making them suitable for many different MPC applications. They are highly efficient for large-scale problems with hundreds or thousands of decision variables, making them well-suited for complex MPC models. Interior-point solvers have a high degree of numerical stability, which means they can produce accurate solutions even when faced with numerical issues such as ill-conditioned matrices. They can handle a wide range of constraints, including inequality and equality constraints, making them a versatile choice for MPC optimization problems. Interior-point solvers are generally faster than other optimization solvers, such as active-set solvers, when solving large-scale MPC problems. Overall, the advantages of interior-point solvers make them a popular choice for MPC applications where large-scale, nonlinear optimization problems are common.

4.8.2 Pseudocode of Proposed Method

Algorithm 1 describes how the DMPC code operates independently on each robot. According to the selected method, each robot receives state or prediction information from its neighbors, updates the constraints of its own Optimal Control Problem (OCP), applies the first step of the optimal control command obtained from the solved OCP, and simultaneously shares its own position or prediction information with its neighbors. This iteration continues for each robot until it reaches its target.

Algorithm 1: Distributed Model Predictive Control

```
    Input: $q_i^0, q_i^{\text{goal}}, M_i, \bar{d}(i,j)$
    Output: $q_i(k), q_i(n|k)$
 1: Initialize MPC Parameters: $Q, Q_f, R, R_{\text{eff}}, N, T$
 2: Define the OCP in Eqn. 4.10 with Eqns. 4.6,4.9,4.14
 3: while $q_i^{\text{goal}}$ not reached do
 4:     for each robot $j$ in $M_i$ do
 5:         $constraint \leftarrow q_j(k)$ (position sharing) or
 6:         $constraint \leftarrow \tau_j(k)$ (predicted trajectory) or
 7:         $constraint \leftarrow \hat{\tau}_i(k))$ (sampled predicted traj.)
 8:     end for                                ▷ Communicate with all neighbors
 9:     Update the constraints as given in Eqns. 4.12 or 4.13
10:     $u_i^* \leftarrow$ Solve OCP with given constraints
11:     Apply the first step of calculated $u^*$
12:     Publish your state $q_i(k)$ or predicted trajectory $\tau_i(k)$ or its sampled
            version $\hat{\tau}_i(k))$
13: end while
```

4.8.3 Converting a Non-convex Problem to a Convex Problem

Converting a non-convex optimization problem to a convex optimization model can be challenging and is not always possible. However, in some cases, it is possible to transform a non-convex optimization problem into a convex optimization problem by making certain assumptions or approximations.

One common approach is to use convex relaxation, which involves replacing the original non-convex constraints or objective function with a convex approximation that has the same or tighter bounds. This can be done by applying a series of mathematical techniques, such as linearization, quadratic relaxation, or semidefinite programming, to transform the non-convex constraints or objective function into a convex form. The resulting convex approximation may not always yield the optimal solution to the original non-convex problem, but it can provide a feasible solution that is close to the global minimum.

Another approach is to use reformulation techniques, which involve reformulating the original problem into an equivalent convex optimization problem. This can be done by introducing additional decision variables or constraints that transform the original problem into a convex form. For example, a non-convex optimization problem with product terms can be reformulated as a convex optimization problem using a logarithmic transformation.

It is important to note that converting a non-convex optimization problem to a convex optimization model can be computationally expensive and may not always be feasible, particularly for complex models or in real-time applications. Therefore, it is often necessary to use specialized optimization algorithms and solvers that can efficiently handle non-convex optimization problems.

4.8.4 Obstacle Avoidance and Formation Control with Soft Constraint

Hard constraints are requirements that must be satisfied exactly, while soft constraints are requirements that can be violated, but at a cost. In the context of obstacle avoidance for mobile robots, soft constraints can be used to encourage the robot to avoid obstacles, while still allowing it to navigate through tight spaces if necessary.

For example, a soft constraint can be added to the cost function that penalizes the robot for getting too close to obstacles. The penalty could be proportional to the distance to the obstacle, so that the closer the robot gets, the higher the cost. This would encourage the robot to avoid obstacles, but still allow it to navigate through narrow passages if the cost of violating the constraint is lower than the cost of taking a longer path.

In the context of MPC, hard constraints can be used to enforce the desired formation shape and inter-robot distances. These constraints would ensure that the robots maintain a specific formation, and any violation of the constraint would result in a very high cost in the optimization problem. This type of constraint enforces the desired formation strictly, which may not always be feasible or optimal.

On the other hand, soft constraints are requirements that can be violated at a cost, allowing for flexibility and adaptability in the formation control. Soft constraints can be used to encourage the robots to maintain a desired formation, while still allowing for some variation in the formation shape and movement. Soft constraints can be defined based on the desired inter-robot distances or angles and can be included in the MPC optimization problem as a penalty term. This type of constraint allows the robots to move and adapt to the environment more naturally, while still keeping some formation constraints.

4.8.5 2D Quadcopter Formations and Down-wash Effect

Quadcopter formations are commonly worked in 2D due to several practical reasons. Firstly, quadcopters are often used in applications that occur in 2D environments, such as surveillance, inspection, and mapping. These applications typically do not require quadcopters to move vertically or operate in complex 3D environments, so working in 2D simplifies the control and planning algorithms. Secondly, working in 2D allows for simpler and more cost-effective sensing and communication systems. Most quadcopters use basic sensors like accelerometers, gyroscopes, and magnetometers to estimate their position and orientation, which are usually more accurate in 2D than in 3D. Similarly, communication systems used to transmit data and commands between quadcopters and a ground station are generally simpler and more reliable in 2D.

Practical applications of quadcopter formations in 2D include swarm robotics, target tracking, and environmental monitoring. In swarm robotics, multiple quadcopters can be controlled to form a coordinated group and carry out tasks, such as search and rescue or object transport. Target tracking involves using multiple quadcopters to track a moving object or person, such as a suspect in a police chase or a wildlife animal. Environmental monitoring can involve using multiple quadcopters to collect data on air or water quality, weather patterns, or other environmental factors.

The down-wash effect is a well-known phenomenon in the field of quadcopter control, wherein the airflow generated by the rotors of a quadcopter flows downward and creates a disturbance on the ground or nearby objects. This disturbance can result in destabilization of other nearby quadcopters or objects, making it challenging to maintain a desired formation. In micro UAV swarms, 2D formations are often preferred over 3D formations due to their ability to reduce the impact of the down-wash on the neighboring quadcopters. By aligning the quadcopters along a single plane, the down-wash effect is largely confined to that plane, allowing for greater control and stability. Additionally, 2D formations are often simpler to implement and control compared to 3D formations. This is because the motion planning and control algorithms can be designed using simpler mathematical models, allowing for faster and more efficient computation. The use of 2D formations is a practical choice for small UAV swarms, which often operate in confined spaces and require rapid response times.

CHAPTER 5

SIMULATION RESULTS AND EXPERIMENTAL WORK

In order to evaluate the performance of the proposed model predictive control approach for formation flight, we implemented the proposed control system on mini quadrotors Crazyflie 2.1 and tested the formation control performance in a variety of scenarios, including static and dynamic obstacles, as well as different numbers of quadcopters.

The implementation of the algorithm was carried out in Python, utilizing the Robot Operating System (ROS) for swarm management. The casADi [49] library, a nonlinear optimizer, was employed for the optimization tasks involved in the MPC framework. The simulations and experiments were performed on a laptop equipped with an Intel i7 2.4 GHz processor, running Ubuntu 20.04 operating system.

By conducting these simulations and experiments, we aimed to evaluate the performance of the proposed MPC approach under diverse conditions, allowing for a comprehensive analysis of its capabilities, robustness, and scalability in real-world scenarios.

All the parameters utilized in the algorithm for simulations and experiments were carefully chosen and documented in Table 5.1. These parameters include critical values, weightings, thresholds, and any other relevant variables necessary for the successful execution of the MPC approach. The specific values for each parameter were selected based on prior research, practical considerations, and the desired objectives of the formation flight experiments. By specifying and adhering to these parameter values, we aimed to ensure consistency and reproducibility in the evaluation and analysis of the MPC algorithm's performance.

Table 5.1: Control parameters and constraints used in experiments

Parameter	**Symbol**	**Value**	**Unit**
Time Step	k	0.2	s
Prediction Horizon	N	20	-
Robot's Actual Radius	r_{robot}	8	cm
Safety Distance	d_{safety}	32	cm
Robot Safety Distance	r_{safety}	40	cm
State Cost Matrix	Q	diag(10,10,0.1)	-
Final State Cost Matrix	Q_f	diag(10,10,0.1)	-
Input Cost Matrix	R	diag(0.5,0.001,0.05)	-
Control Effort Cost Matrix	R_{eff}	diag(0.5,20,0.05)	-
Velocity Constraint	v_{max}	0.8	m/s
Angular Velocity Constraint	w_{max}	0.6	rad/s

5.1 Monte-Carlo Experiments

Fig. 4.5, 4.7 and 4.8 illustrate the paths followed by robots in position swap scenarios with the proposed methods. Figure 5.1 presents the confidence plots derived from Monte Carlo trials performed under the different process noises (SNR $= 0$, SNR $= 1$, SNR $= 2$). It can be observed that the sampled trajectory sharing method decreases the average path length taken and yields more stable results. It also reduces the average computation time. Two important observations can be made: First, the sampled trajectory method generates paths that are almost as short as the complete trajectory method. This indicates that sampling does not move us away from optimality. Second, the paths generated with sampled trajectory sharing created less variance than the paths generated with complete trajectory sharing. One possible explanation is that exponential sampling implicitly moves the problem into the exponential decay of rewards in time within the reinforcement learning paradigm.

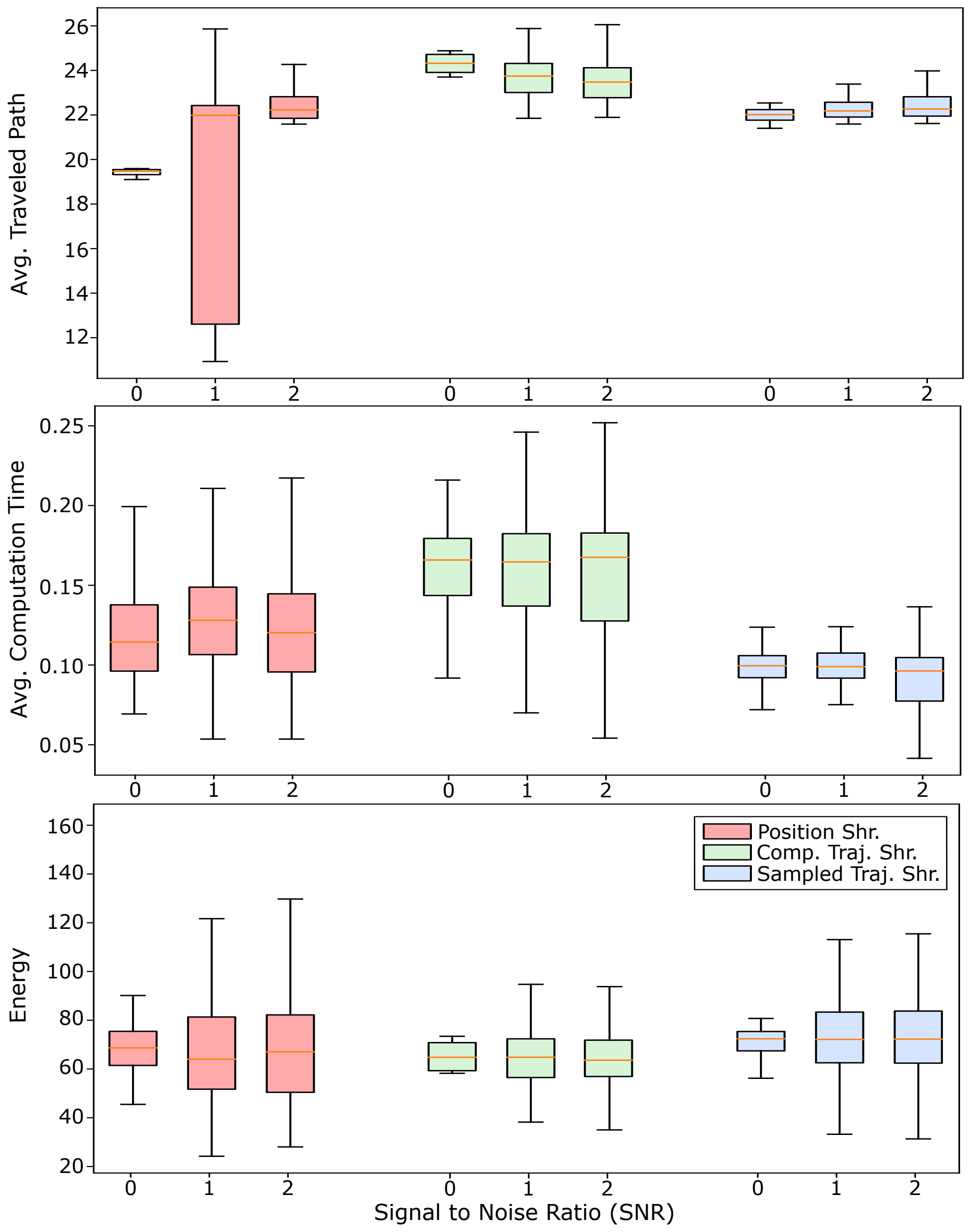

Figure 5.1: The average traveled path, the average computation time and energy ($u^T Ru$) for each sharing method under the different noise levels. The box extends from the lower to upper quartile values of the data, with a line at the median. The whiskers extend from the box to show the range of the data. Only the position sharing method cannot guarantee reaching the goal in a both noisy and noise-free environment. Sampled Trajectory Sharing method reduces the average traveled path and the average computation time. The average computation time and the average traveled path have less variance for the proposed method.

The term $u^T Ru$, commonly referred to as energy, represents the squared sum of the commanded linear and angular velocities in the system. It is used in the MPC framework and is aimed to be minimized during the optimization process. By minimizing this term, the MPC aims to find control inputs that result in smoother and more efficient motion. When plotting the energy term for different methods, we expect to observe variations in the energy values, reflecting the efficiency and smoothness of the control strategies employed. Lower energy values indicate more efficient control methods with reduced commanded velocities and potentially smoother trajectories.

Fig. 5.2 illustrates the results of the systematic Monte Carlo trials where we analyzed the effects of prediction horizon length on the system performance. A short prediction horizon can make it difficult to find the optimal path as it cannot see the entire trajectory. On the other hand, when the prediction horizon is too long, several factors can impede finding the optimal path:

Computation Complexity: Increasing the prediction horizon requires evaluating a larger number of future states, which can significantly increase the computational complexity. This can lead to longer computation times and may not be feasible in real-time applications.

Sensitivity to Uncertainties: A longer prediction horizon amplifies the impact of uncertainties in the system, such as measurement errors or dynamic changes in the environment. This sensitivity can result in deviations from the optimal path if the predicted states do not accurately reflect the actual future states.

Environmental Changes: In dynamic environments, a longer prediction horizon may encounter unexpected changes, such as the sudden appearance of obstacles or alterations in the terrain. These unforeseen changes can render the initially planned optimal path ineffective or even infeasible.

Therefore, selecting an intermediate prediction horizon can yield better results. An intermediate value can provide sufficient foresight into the future while maintaining a reasonable computation time and complexity. It is also more tolerant to uncertainties and can adapt better to environmental changes.

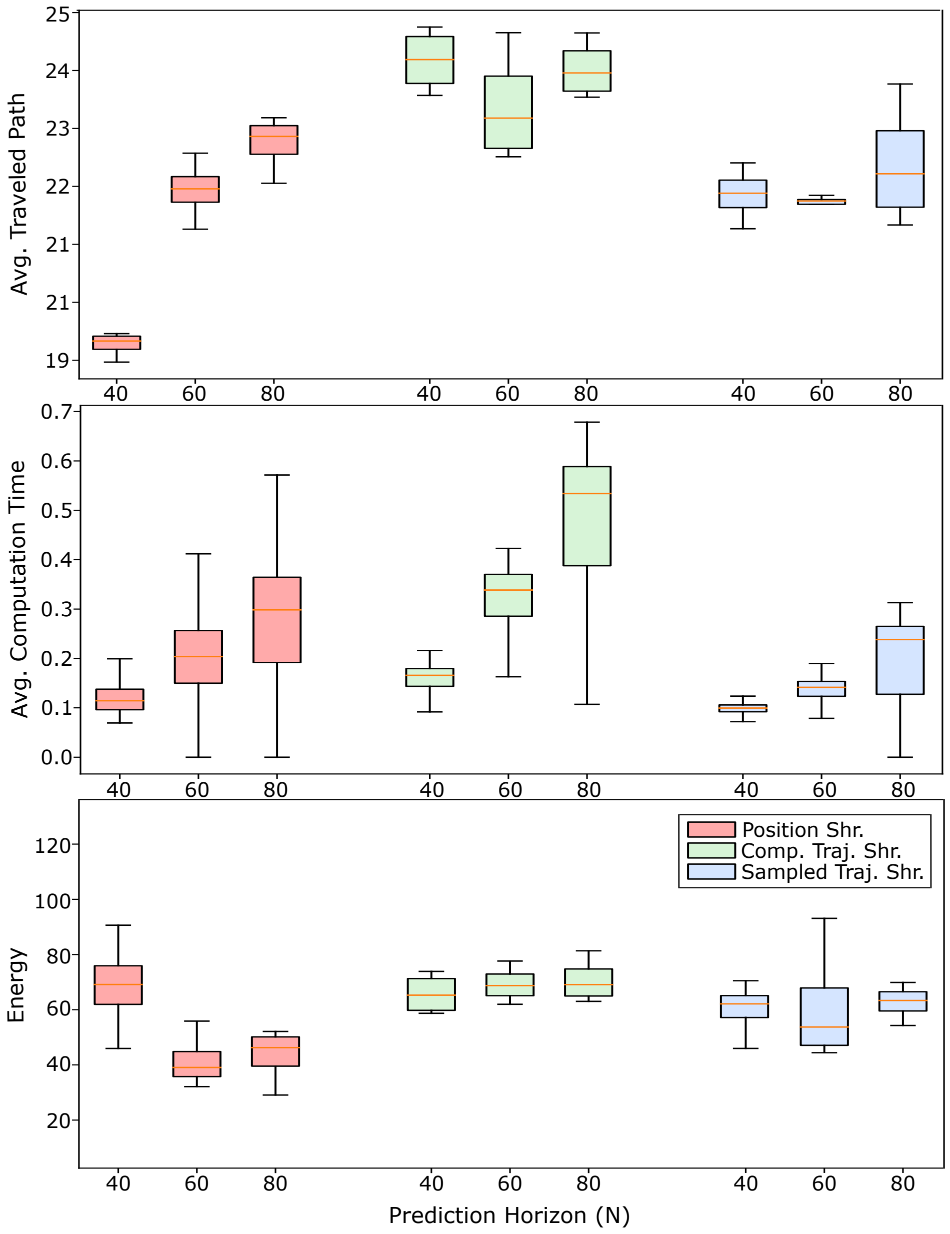

Figure 5.2: The average traveled path, the average computation time and energy for each sharing method under the different prediction horizons. An excessively short or long prediction horizon may not be able to find the optimal path. Instead, an intermediate value often yields better results. It balances the ability to foresee sufficient future states while keeping computation time and complexity manageable. Additionally, an intermediate prediction horizon is more resilient to uncertainties, model inaccuracies, and environmental changes. As expected, increasing the prediction horizon for each method results in an increase in the average computation time. However, in the proposed method, the change in computation time is minimized relative to the variation in the prediction horizon.

Fig. 5.3(a),(b) illustrate the formation motion under the process noise (SNR $=$ 1). Fig. 5.3(c) presents the variation of distance between the robots over time. The desired distance for each edge of the equilateral triangle formation is defined as 2 meters. The maximum and minimum values obtained from Monte Carlo trials are shown with shaded regions in Fig. 5.3(c).

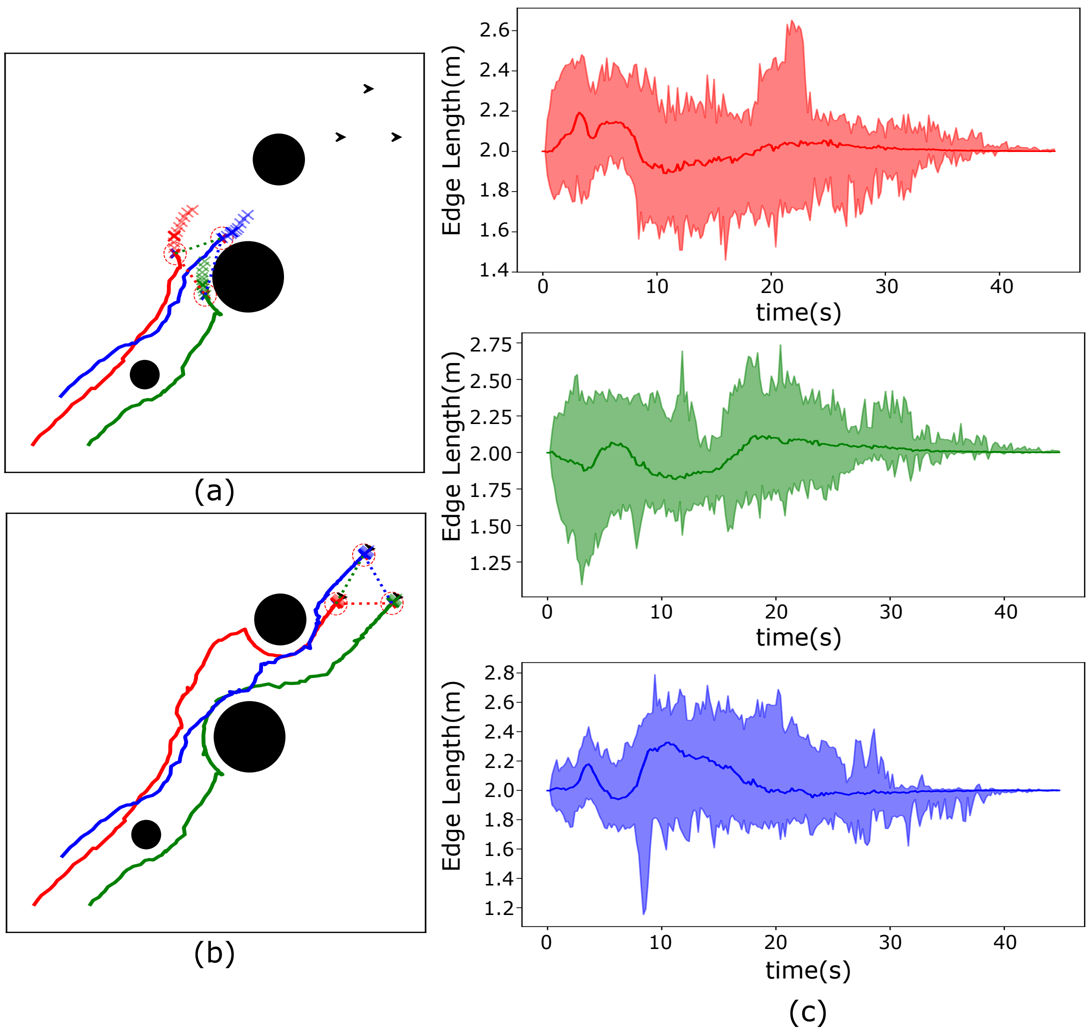

Figure 5.3: Three robots navigating to goal locations while keeping the formation under process noise at (a) t = 11.0 s (b) t = 30 s. (c) The plots indicate the inter-agent distance averages (solid lines) and the ranges (shaded region). The colors correspond to the edges indicated with dashed lines.

5.1.1 Computation Time and Shared Data Package Analysis

The state matrix q_i is composed of three floating-point values. A data package is defined as a single state matrix. With the position sharing method, each robot transmits a single data package containing its current state. In contrast, the complete trajectory sharing method involves the sharing of $N + 1$ data packages, as each robot provides its current state and predicted states. In the sampled trajectory sharing method, $\lceil \log_R(N) \rceil + 1$ data packages are exchanged among the robots, where R is the sampling rate.

Table 5.2: Shared data package and computation time of sharing methods under different prediction horizons for 2 robot position swap scenario

	Position Sharing	Complete Traj. Sharing	Sampled Traj. Sharing
Shared Data Package (N=20)	1	21	6
Avg. Comp. Time (N=20)	39.12 ms	37.11 ms	35.95 ms
Shared Data Package (N=40)	1	41	7
Avg. Comp. Time (N=40)	86.21 ms	71.92 ms	66.04 ms

The experiment was conducted by setting the sampling rate R to 2, and the results were recorded for different simulation environments in terms of the number of shared data packages and computation time. The impact of reducing the amount of shared data on the average computation time can be observed in the tables 5.2, 5.3. Additionally, the cases where the sampling rate R is set to 2 and 3 were compared with the complete trajectory sharing and uniform sampling methods, and the results are shown in Fig. 5.5.

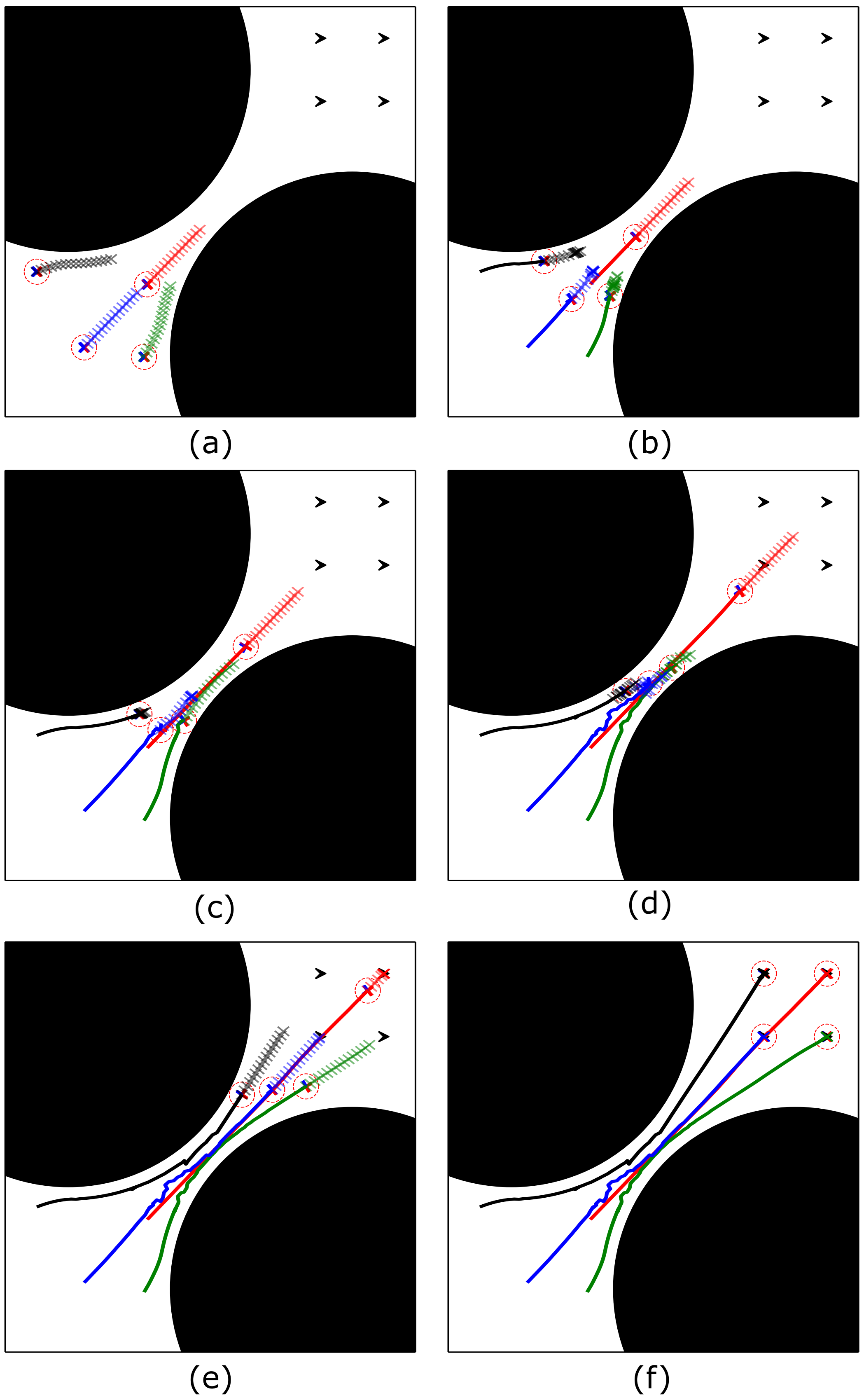

Figure 5.4: Narrow passage scenario with complete prediction sharing method (a) at time t = 0.4 s (b) t = 3.0 s (c) t = 6.0 s (d) t = 9.0 s (e) t = 13.0 s (f) t = 19.0 s

Table 5.3: Shared data package and computation time of sharing methods under different prediction horizons for narrow passage scenario

	Position Sharing	Complete Traj. Sharing	Sampled Traj. Sharing
Shared Data Package (N=40)	1	41	7
Avg. Comp. Time (SNR=0)	135.50	172.61 ms	96.74 ms
Avg. Comp. Time (SNR=1)	127.32	109.57 ms	82.21 ms
Shared Data Package (N=60)	1	61	7
Avg. Comp. Time (SNR=0)	199.15	263.18 ms	195.83 ms
Avg. Comp. Time (SNR=1)	162.22	187.71 ms	131.13 ms
Shared Data Package (N=70)	1	71	8
Avg. Comp. Time (SNR=0)	275.58	385.72 ms	258.67 ms
Avg. Comp. Time (SNR=1)	187.78	211.18 ms	163.56 ms

Fig. 5.5 illustrates the average traveled path, average computation time, and energy values for different sampling methods under varying levels of noise. The graphs depict the sharing of the complete trajectory, $\tau_i(k) = \{q_i(n|k)\}_{n:0,N}$, uniform sampling of all predictions that are multiples of 3, $\hat{\tau}_i(k) = \{q_i(n|k)\}_{n:0,3,6,\ldots,3\cdot a<N}$, and exponential sampling with two different sampling rates $\hat{\tau}_i(k) = \{q_i(n|k)\}_{n:0,1,R,R^2,\ldots,R^a<N}$ ($R = 2$ and 3).

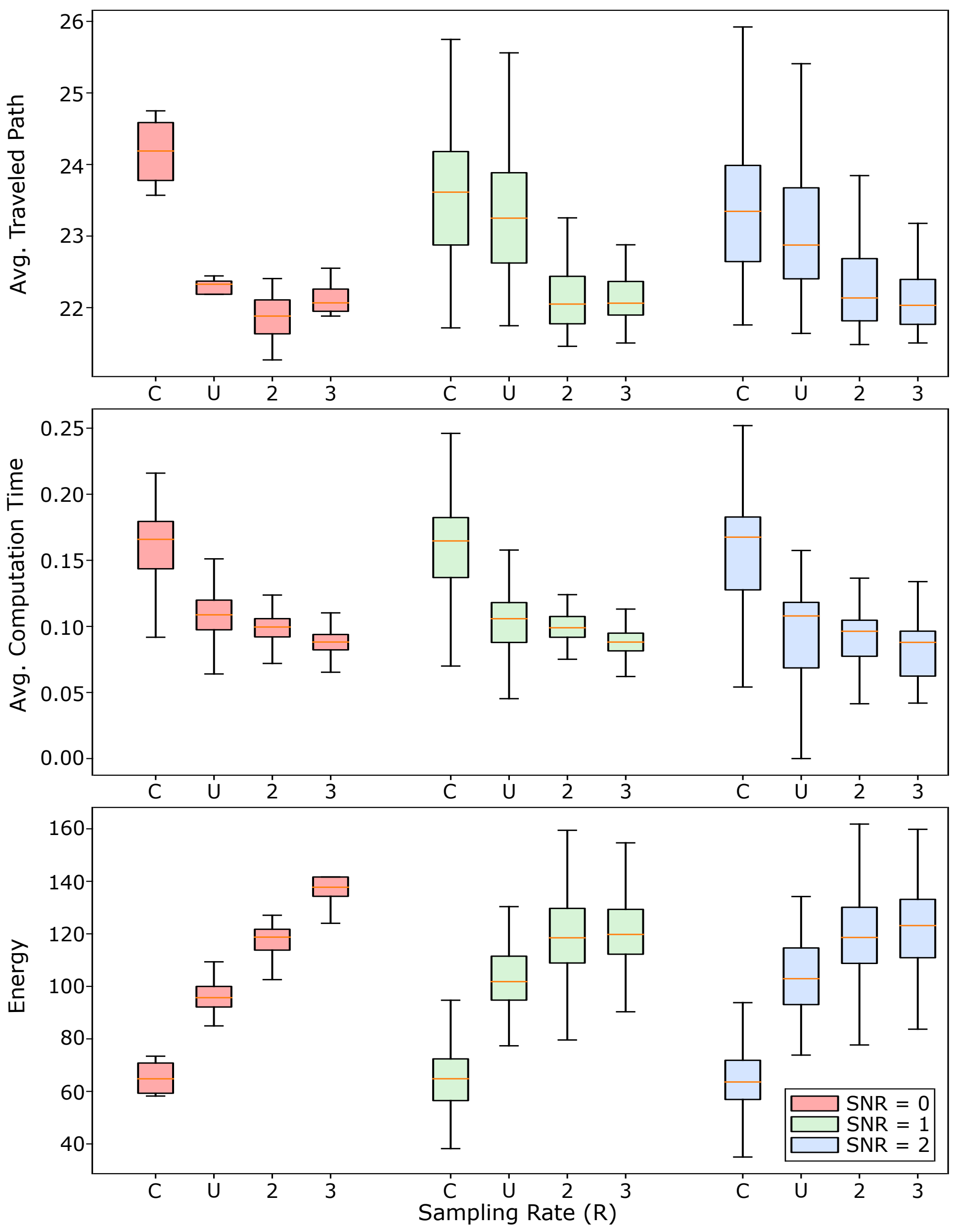

Figure 5.5: The average traveled distance, average computation time, and energy values are shown for different sampling methods, including the complete trajectory sharing method, under various noise levels and constant prediction horizon $N = 40$. As the number of shared data packages decreases, the computation time decreases for all levels of noise. Particularly in noise-free environments, sampling has a positive effect on the average traveled distance. In a noise-free environment, a sampling rate of 2 yields slightly better results compared to a sampling rate of 3. As the noise level increases, the difference between uniform sampling and complete sharing decreases, and the difference between sampling rates 2 and 3 also decreases.

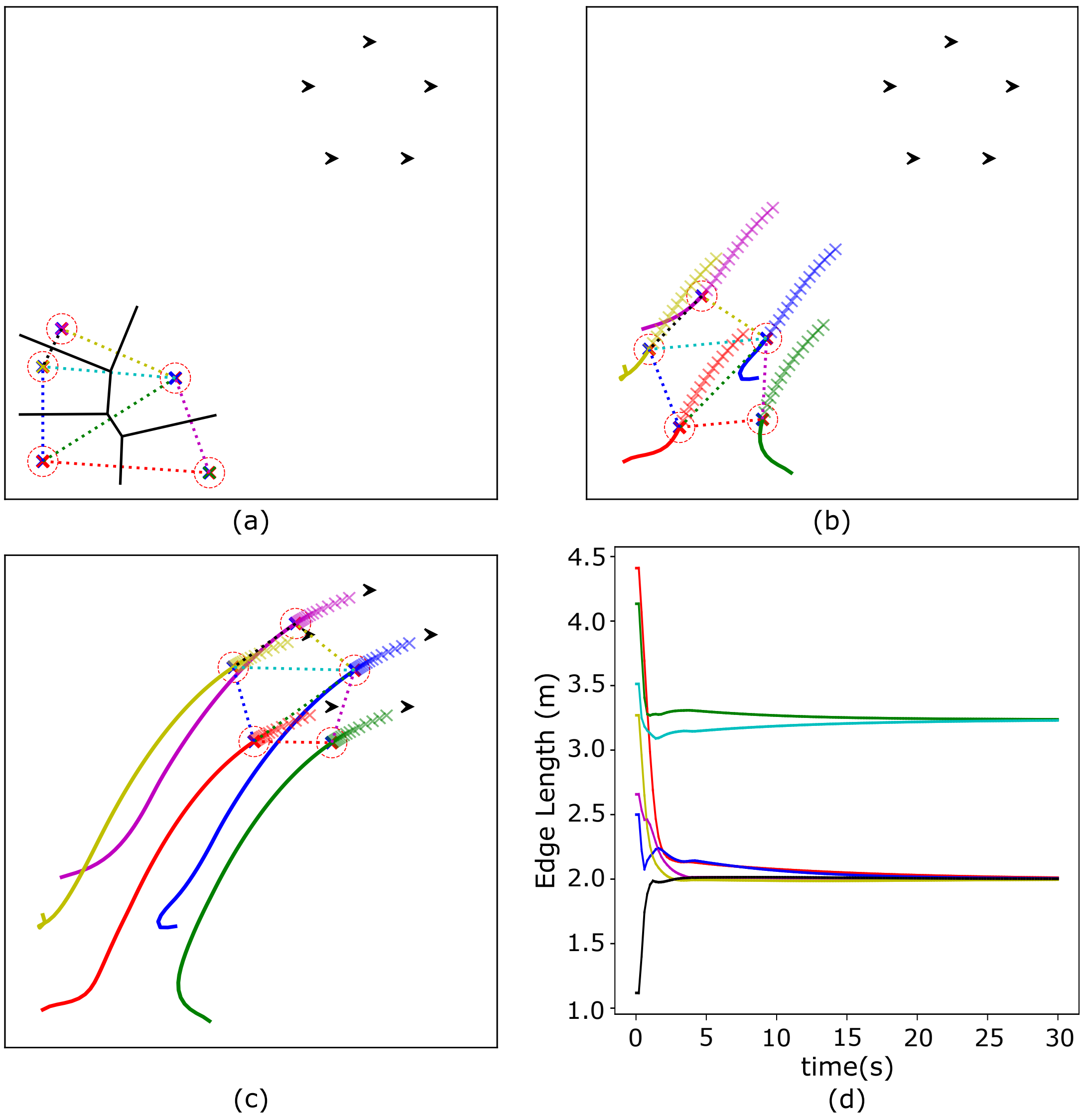

Figure 5.6: (a) By using the concept of Voronoi Diagram and Delaunay Triangulation a graph structure consisting of five quadcopters is obtained. The constructed Voronoi Diagram is indicated with solid black lines. (b) Robots are reaching the desired formation while navigating through the goal points at t = 5.0 s . (c) Robot positions at time t = 15 s. (d) The plot shows the edge lengths of the corresponding graph structure. The change in the edge lengths is represented by the indicated colors of the dashed lines.

Figure 5.6 illustrates the formation control using the sampled trajectory sharing method. As the robots navigate towards their respective targets, they also share their sampled trajectories with their neighbors to establish the desired formation shape. Table 5.4

provides information on the amount of shared data and the average computation time when attempting formation control with different methods.

Table 5.4: Shared data package and computation time of sharing methods under different prediction horizons for Regular Pentagon Formation

	Position Sharing	Complete Traj. Sharing	Sampled Traj. Sharing
Shared Data Package (N=20)	1	21	6
Avg. Comp. Time (N=20)	NA	48.83 ms	31.19 ms
Shared Data Package (N=40)	1	41	7
Avg. Comp. Time (N=40)	NA	93.62 ms	56.30 ms

5.2 Comparison of the Results

The Buffered Voronoi Cell (BVC) [50] is a motion planning algorithm designed to compute a path for mobile robots that ensures they can move safely and collision-free between their starting and goal positions. The algorithm utilizes a Voronoi diagram, which divides the space around obstacles into regions of influence. In addition, BVC incorporates a buffer zone around the mobile robots that prevent collisions with obstacles and other robots while balancing safety and efficiency. The buffer zone's size is adjustable to suit the task's specific requirements.

BVC generates a motion plan that the mobile robots execute using a feedback control strategy. The feedback control strategy is responsible for adjusting the robots' movement in real-time based on sensor data such as odometry and laser scans. It guarantees that the robots follow the planned path and avoid collisions.

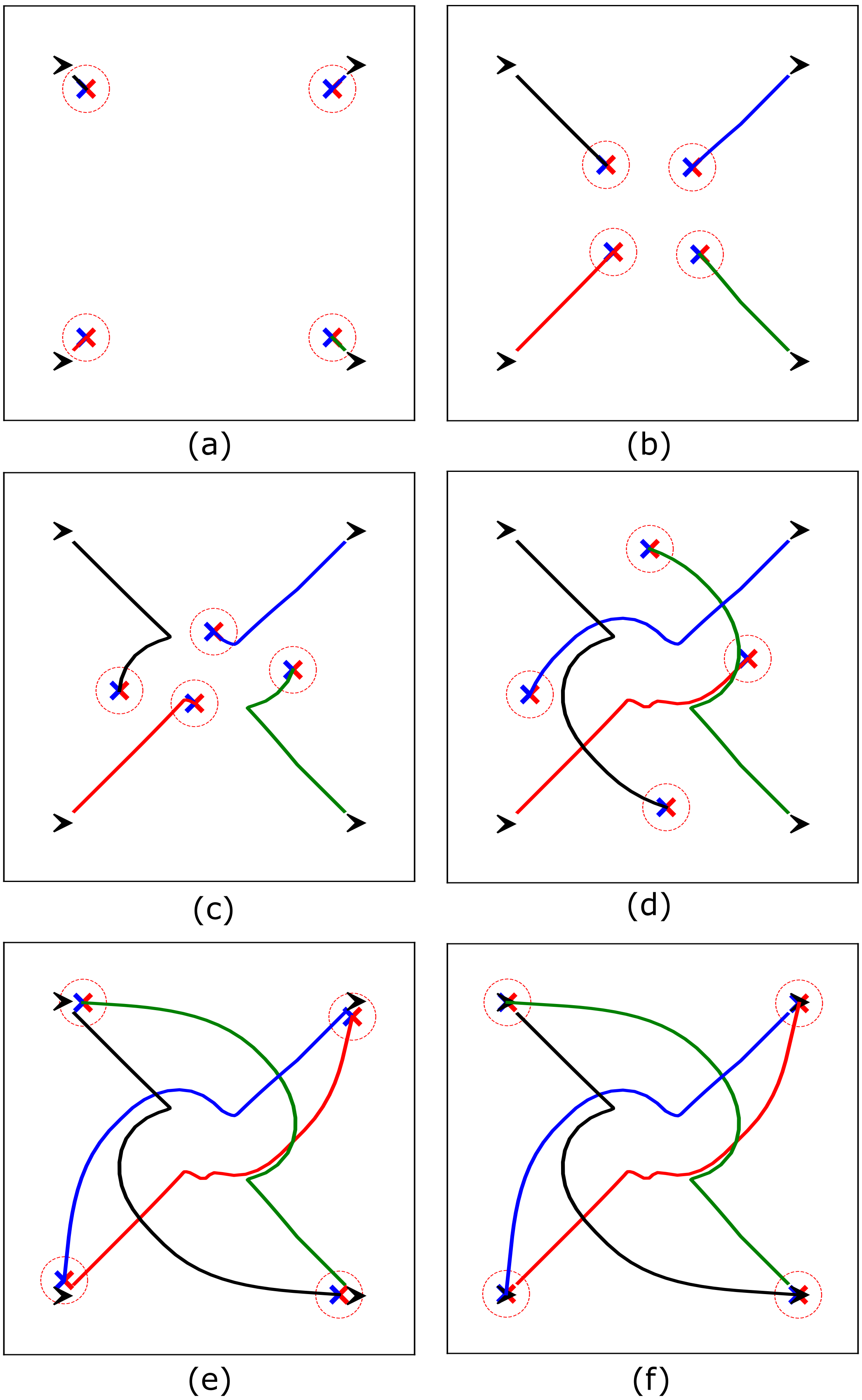

Figure 5.7: Four robot position swap scenario with BVC method(a) at time t = 1.0 s (b) t = 3.5 s (c) t = 6.0 s (d) t = 9.0 s (e) t = 12.0 s (f) t = 16.0 s

One significant difference between BVC and MPC is their optimization approach. BVC utilizes a static optimization method, which involves precomputing the Voronoi diagram to generate collision-free paths. In contrast, MPC utilizes a dynamic optimization approach that involves real-time problem-solving based on current sensor readings and predictions of future robot motion. Despite the differences, both BVC and MPC offer unique benefits. MPC's advantages include predictive control, real-time optimization, constraints handling, and optimality. Predictive control allows MPC to predict future robot motion and optimize the path accordingly, making it more suitable for complex tasks than BVC. Real-time optimization enables MPC to adjust the path according to changes in the environment. Constraints handling allows MPC to handle complex constraints such as nonholonomic constraints and velocity and acceleration limits. Optimality ensures that MPC generates globally optimal paths that minimize a cost function, leading to more efficient paths.

These advantages make MPC particularly useful in applications that require predictive control, real-time optimization, and constraint handling. However, the complexity of MPC makes it more challenging to implement and use than BVC.

Table 5.5: Comparison of Trajectory Sampled MPC method with BVC

	Traveled Path	Traveling Time	Computation Time
MPC	7.717 m	14.7 s	35 ms
BVC	8.512 m	16.2 s	3.7 ms

The table 5.5 highlights the trade-off between average computation time, traveling time, path length, and deadlock handling between BVC and MPC algorithms for robot path planning. The results show that while BVC has a shorter computation time, MPC outperforms BVC in terms of traveling time and path length. This is because MPC is capable of handling deadlocks thanks to its online optimization process and predictive behavior. Despite the longer computation time required for MPC, the resulting path is often shorter and faster, making it a better choice for applications that prioritize

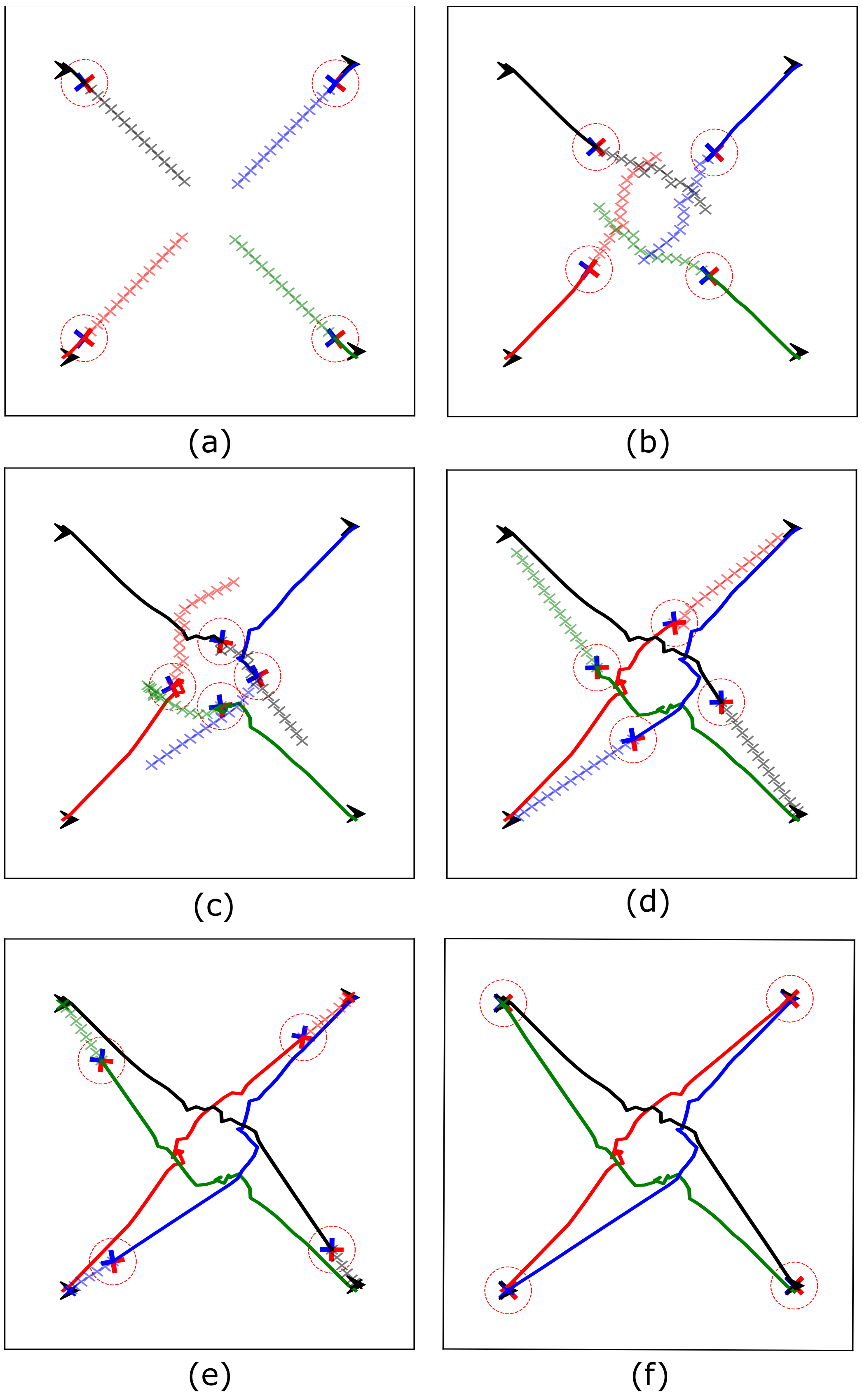

Figure 5.8: Four robot position swap scenario with MPC method(a) at time t = 1.0 s (b) t = 3.0 s (c) t = 5.0 s (d) t = 7.0 s (e) t = 9.0 s (f) t = 14.0 s

efficiency and speed. The computation time of BVC is lower due to its simplicity and solving a less complex problem. However, the simplicity of the BVC algorithm also makes it vulnerable to deadlock situations. To avoid deadlock, BVC requires a rule-based escape maneuver code, which can slow down the robots.

5.3 Crazyflie Experiments

In order to evaluate the performance of the proposed model predictive control approach for formation flight, we conducted experiments with Crazyflies. The Crazyflie UAVs are equipped with onboard sensing and control capabilities, making them well-suited for testing formation control algorithms. In our experiments, we implemented the proposed control system on the Crazyflie UAVs and tested the formation control performance in a variety of scenarios, including static and dynamic obstacles, as well as different numbers of UAVs. The results of the experiments demonstrated that the Crazyflie UAVs were able to achieve and maintain the desired formation shape and position with a high level of accuracy, even in the presence of external disturbances. These results demonstrate the feasibility and effectiveness of the proposed control approach for formation flight of multiple UAVs.

We also tested the control approach in multi-robot scenarios, in which the UAVs had to coordinate their movements to achieve a common goal. To do this, we implemented a decentralized control architecture that allowed each UAV to make its own decisions based on local information, while still being able to achieve the overall goal of the group. The results of these experiments showed that the proposed control approach was able to enable the UAVs to effectively coordinate their movements and achieve the desired multi-robot behavior. This demonstrates the potential of the proposed control approach for enabling robust multi-robot coordination in a variety of applications.

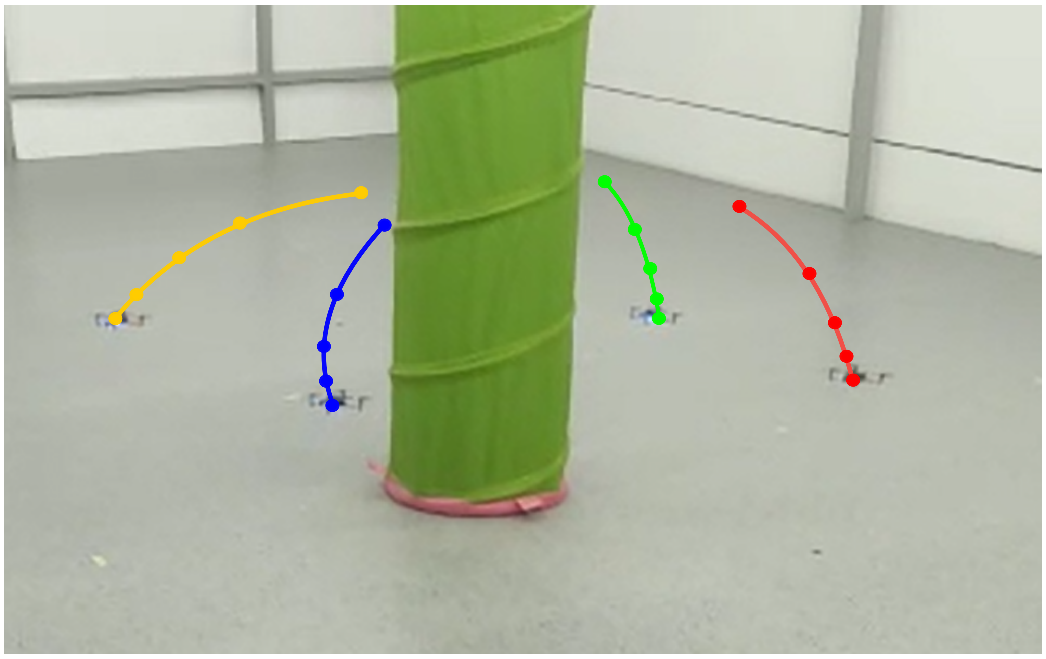

Figure 5.9: Illustration of the MPC-based distributed formation control method in which robots share their exponentially-sampled trajectories with each other that would move them among dynamic obstacles to a desired goal position. In a set of systematic experiments conducted in simulation and with mini quadcopters, we have shown that sharing of exponentially-sampled trajectories (as opposed to positions, or complete trajectories) among the robots provides near-optimal paths while decreasing the required computation cost and communication bandwidth. Surprisingly, in the presence of noise, sharing exponentially-sampled trajectories among the robots decreased the variance in the final paths.

CHAPTER 6

CONCLUSIONS

In this thesis, we propose a Model Predictive Control (MPC) based distributed formation control method for a swarm of robots that would move them among dynamic obstacles to a desired goal position. Our approach was based on the optimization of a performance index that takes into account the formation constraints and the robots' dynamics.

Specifically, after formulating the formation control, as a distributed version of MPC, we propose and evaluate three information-sharing schemes within the swarm; namely sharing (i) positions, (ii) complete predicted trajectories, and (iii) exponentially-sampled predicted trajectories.

In a set of systematic experiments conducted in simulation and with mini quadcopters, we have shown that sharing of exponentially-sampled trajectories (as opposed to positions, or complete trajectories) among the robots provides near-optimal paths while decreasing the required computation cost and communication bandwidth. Surprisingly, in the presence of noise, sharing exponentially-sampled trajectories among the robots decreased the variance in the final paths.

There are a few limitations to our study that should be considered in future work. First, we only considered a simplified model, and a more detailed model may be required for real-world implementation. Another direction is to consider more complex performance indices and constraints, such as energy efficiency or communication constraints.

Overall, our work contributes to the development of reliable and efficient formation control algorithms for UAVs.

REFERENCES

[1] Y. Liu and R. Bucknall, “A survey of formation control and motion planning of multiple unmanned vehicles,” *Robotica*, vol. 36, no. 7, p. 1019–1047, 2018.

[2] M. A. Kamel, X. Yu, and Y. Zhang, “Formation control and coordination of multiple unmanned ground vehicles in normal and faulty situations: A review,” *Annual Reviews in Control*, vol. 49, pp. 128–144, 2020.

[3] Z. Hou, W. Wang, G. Zhang, and C. Han, “A survey on the formation control of multiple quadrotors,” pp. 219–225, 2017.

[4] A. Turgut, H. Çelikkanat, F. Gökçe, and E. Sahin, “Self-organized flocking in mobile robot swarms,” *Swarm Intelligence*, vol. 2, pp. 97–120, 12 2008.

[5] C. W. Reynolds, “Flocks, herds and schools: A distributed behavioral model,” *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, 1987.

[6] V. Gazi and K. Passino, *Swarm Stability and Optimization*. 01 2011.

[7] L. Bayindir and E. Sahin, “A review of studies in swarm robotics,” *Turkish Journal of Electrical Engineering and Computer Sciences*, vol. 15, pp. 115–147, 01 2007.

[8] K. M. Passino, “Biomimicry of bacterial foraging for distributed optimization and control,” *IEEE Control Systems Magazine*, vol. 22, pp. 52–67, 2002.

[9] N. Mathew, S. L. Smith, and S. L. Waslander, “Planning paths for package delivery in heterogeneous multirobot teams,” *IEEE Transactions on Automation Science and Engineering*, vol. 12, pp. 1298–1308, 2015.

[10] H. Xiao, Z. Li, and C. L. P. Chen, “Formation control of leader–follower mobile robots’ systems using model predictive control based on neural-dynamic optimization,” *IEEE Transactions on Industrial Electronics*, vol. 63, pp. 5752–5762, 2016.

[11] A. Sion, A. Reina, M. Birattari, and E. Tuci, “Controlling robot swarm aggregation through a minority of informed robots,” in *ANTS Conference*, 2022.

[12] D. A. Amer, G. Attiya, I. Zeidan, and A. A. Nasr, “Employment of task scheduling based on water wave optimization in multi robot system,” in *2021 International Conference on Electronic Engineering (ICEEM)*, pp. 1–6, 2021.

[13] T. Zheng and J. Li, “Multi-robot task allocation and scheduling based on fish swarm algorithm,” in *2010 8th World Congress on Intelligent Control and Automation*, pp. 6681–6685, 2010.

[14] M. de Queiroz, X. Cai, and M. Feemster, *Formation Control of Multi-Agent Systems: A Graph Rigidity Approach*. 01 2019.

[15] S. Martínez and F. Bullo, “Optimal sensor placement and motion coordination for target tracking,” *Automatica*, vol. 42, pp. 661–668, 04 2006.

[16] B. Anderson, B. Fidan, C. Yu, and D. Walle, *UAV Formation Control: Theory and Application*, vol. 371, pp. 15–33. 12 2007.

[17] J. Tsitsiklis, *Problems in decentralized decision making and computation.* PhD thesis, EECS Department, MIT, 1984.

[18] F. Bullo, J. Cortés, and S. Martínez, *Distributed Control of Robotic Networks: A Mathematical Approach to Motion Coordination Algorithms*. 07 2009.

[19] R. Olfati-Saber and R. Murray, “Consensus problems in networks of agents with switching topology and time delays,” *IEEE Transactions on Automatic Control*, vol. 49, no. 9, pp. 1520–1533, 2004.

[20] K.-K. Oh, M.-C. Park, and H.-S. Ahn, “A survey of multi-agent formation control,” *Automatica*, vol. 53, pp. 424–440, 2015.

[21] J. Desai, J. Ostrowski, and V. Kumar, “Modeling and control of formations of nonholonomic mobile robots,” *IEEE Transactions on Robotics and Automation*, vol. 17, no. 6, pp. 905–908, 2001.

[22] W. Ni and D. Cheng, “Leader-following consensus of multi-agent systems under fixed and switching topologies,” *Systems & Control Letters*, vol. 59, pp. 209–217, 03 2010.

[23] Y. Hong, J. Hu, and L. Gao, "Tracking control for multi-agent consensus with an active leader and variable topology," *Automatica*, vol. 42, no. 7, pp. 1177–1182, 2006.

[24] Z. Ji, Z. Wang, H. Lin, and Z. Wang, "Interconnection topologies for multi-agent coordination under leader–follower framework," *Automatica*, vol. 45, pp. 2857–2863, 12 2009.

[25] Y. Q. Chen and Z. Wang, "Formation control: a review and a new consideration," pp. 3181–3186, 2005.

[26] K. H. Tan and M. A. Lewis, "Virtual structures for high-precision cooperative mobile robotic control," *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS '96*, vol. 1, pp. 132–139 vol.1, 1996.

[27] Y. Zhang and H. Mehrjerdi, "A survey on multiple unmanned vehicles formation control and coordination: Normal and fault situations," in *2013 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 1087–1096, 2013.

[28] K. D. Do and J. Pan, "Nonlinear formation control of unicycle-type mobile robots," *Robotics Auton. Syst.*, vol. 55, pp. 191–204, 2007.

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

[30] G. Antonelli, F. Arrichiello, and S. Chiaverini, "Experiments of formation control with multirobot systems using the null-space-based behavioral control," *Control Systems Technology, IEEE Transactions on*, vol. 17, pp. 1173 – 1182, 10 2009.

[31] D. Xu, X. Zhang, Z. Zhu, C. Chen, and P. Yang, "Behavior-based formation control of swarm robots," *Mathematical Problems in Engineering*, pp. 1–13, 2014.

[32] G. Lee and D. Chwa, "Decentralized behavior-based formation control of multi- ple robots considering obstacle avoidance," *Intelligent Service Robotics*, pp. 127–138, 2018.

[33] O. Khatib, “Real-time obstacle avoidance for manipulators and mobile robots,” vol. 2, pp. 500–505, 1985.

[34] R. Olfati-Saber and R. M. Murray, “Distributed cooperative control of multiple vehicle formations using structural potential functions,” *IFAC Proceedings Volumes*, vol. 35, no. 1, pp. 495–500, 2002. 15th IFAC World Congress.

[35] A. A. Masoud, “Decentralized self-organizing potential field-based control for individually motivated mobile agents in a cluttered environment: A vector-harmonic potential field approach,” *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 37, no. 3, pp. 372–390, 2007.

[36] Z. Wu, G. Hu, L. Feng, J. Wu, and S. Liu, “Collision avoidance for mobile robots based on artificial potential field and obstacle envelope modelling,” *Assembly Automation*, vol. 36, pp. 318–332, 08 2016.

[37] S. Ge, X. Liu, and C.-H. Goh, “Formation potential field for trajectory tracking control of multi-agents in constrained space,” *International Journal of Control*, vol. 90, pp. 1–21, 09 2016.

[38] K. Mehlhorn and S. Näher, “Implementation of a sweep line algorithm for the straight line segment intersection problem,” 11 1994.

[39] “A simple on-line randomized incremental algorithm for computing higher order voronoi diagrams,” pp. 142–151, 1991.

[40] “A sweepline algorithm for voronoi diagrams,” *Algorithmica*, vol. 2, no. 1, pp. 153–174, 1987.

[41] Tobias Antonsson, “Mocap deck.” Accessed November 8, 2022 [Online]. Available: `https://www.bitcraze.io/2018/06/mocap-deck/`.

[42] ROS, “Robot operating system.” Accessed November 8, 2022 [Online]. Available: `https://www.ros.org`.

[43] J. A. Preiss, W. Honig, G. S. Sukhatme, and N. Ayanian, “Crazyswarm: A large nano-quadcopter swarm,” pp. 3299–3304, 2017.

[44] Bitcraze, "Positioning system overview." Accessed November 8, 2022 [Online]. Available: `https://www.bitcraze.io/documentation/system/positioning/`.

[45] M. M. Asadi, A. Ajorlou, and A. Aghdam, "Distributed control of a network of single integrators with limited angular fields of view," *Automatica*, vol. 63, pp. 187–197, 01 2016.

[46] Z. Cheng, M.-C. Fan, and H.-T. Zhang, "Distributed mpc based consensus for single-integrator multi-agent systems," *ISA Transactions*, vol. 58, 04 2015.

[47] X. Xu, L. Liu, and G. Feng, "Consensus of single integrator multi-agent systems with distributed infinite transmission delays," pp. 1653–1658, 06 2018.

[48] S. Zhao and Z. Sun, "Defend the practicality of single-integrator models in multi-robot coordination control," 03 2017.

[49] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[50] D. Zhou, Z. Wang, S. Bandyopadhyay, and M. Schwager, "Fast, on-line collision avoidance for dynamic vehicles using buffered voronoi cells," *IEEE Robotics and Automation Letters*, vol. PP, pp. 1–1, 01 2017.