School of Computer Science
College of Engineering and Physical Sciences

MSc. Project

Multi-robot Object Delivery in Formation Based on Convex Optimization

A thesis submitted to the University of Birmingham
for the degree of MSc. Robotics
School of Computer Science
University of Birmingham

Student Name: Weijian Zhang
Student ID: 2332563
Supervisor: Dr. Masoumeh Mansouri

September 2022

Abstract

Multi-robot formations can be used in various applications, such as in automated factories, where multiple robots cooperate to operate an object for transport and work alongside humans and other robots. The above scenario includes subproblems such as task allocation, formation generation, formation maintenance, and routing problem. This project aims to develop a relatively complete multi-robot cooperative control system, ignoring dynamical properties (mass, inertia, etc.), in the context of warehouse automation management problems. The system enables multiple robots to generate and maintain specific formations based on the object's shape, cooperate in transporting the object, and solve routing and task allocation problems.
This project focuses on the following tasks: First, we propose a distributed multi-robot motion planning method to solve the problem of multi-robot task allocation and formation generation in a dynamic environment. We combine the artificial potential field method with the leader-follower-based consensus control to solve the common problem of unreachable goals and motion conflicts in the multi-robot field. Secondly, we propose a heuristic graph search method based on convex optimization computed for obstacle-free regions to generate globally optimal smooth paths for robot formations efficiently. In addition, we solve the motion planning problem for robot formations on a given trajectory. In this problem, we require robot formations to track the established trajectory, avoid conflicts with other robots or obstacles, and achieve cooperative work. Finally, we implemented comprehensive simulation experiments on MATLAB to verify the superiority and robustness of the system we designed.

Keywords— multi-robot collaboration, formation control, convex optimization, graph search, distributed control, task allocation, formation alignment, formation tracking control, sequential trajectory planning.

Acknowledgements

I would like to thank my supervisor, Dr. M. Mansouri, for her guidance and assistance, especially for her advice and recommended papers that were extremely inspiring throughout the project. I would also like to thank my friends and family for the moral support and encouragement they have always provided me, as well as the faculty and staff at the University of Birmingham for their companionship throughout the academic year.

Contents

1 Introduction

1.1 Problem Statement

Multi-robot systems are becoming more and more common in practical application scenarios. Autonomous multi-robot systems can be used in automated factories, vigilance patrols, area exploration, and other areas to perform various tasks. As shown in Fig. 1, in a complex three-dimensional environment, a group of mobile robots needs to form specific formations to collaborate in moving large objects. Multi-robot collaboration makes it possible to perform tasks much more efficiently and fault-tolerantly. One of the most important problems of multi-robot formations is motion planning. For the multi-robot trajectory planning problem, we consider assigning starting and ending points to the robot formations and the relative position relationships between the robots. We also need to take the geometric constraints of the formations into account. The motion planning method for robot formations depends on specific tasks, such as the assignment of formation generation, formation composition, the method of maintaining formation shape, and formation transformation. All of the above formation control problems can be considered as covering control problems within a certain area.

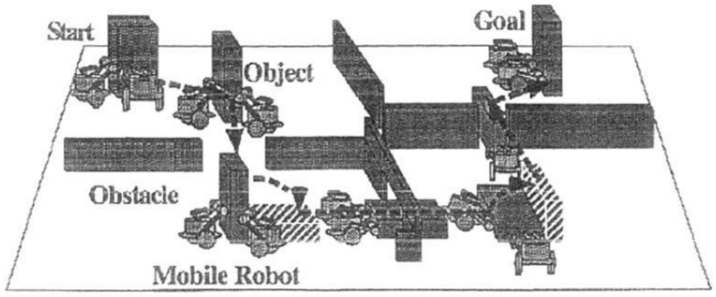


Figure 1: Multi-robot cooperative handling in complex environments (Yamashita et al. 2003)

1.2 Motivation

In a practical multi-robot formation motion planning problem, the robots can only perceive local information about their surroundings, which requires each robot to plan itself in real time. Existing multi-robot motion planning methods can handle planning problems in dynamic environments, but most require the complete trajectory of known moving obstacles. Behavior-based and group robot approaches robot relations in a group without specific formation goals. A more general approach is the sensor-based approach to motion planning. The robots make judgments based on local information provided by the sensors, and these robots coordinate and wallow according to predetermined rules. However, such methods rely too much on sensor accuracy and established strategies and are difficult to apply to formation tasks. Therefore, we propose a path planner that can quickly respond and continuously update the path in real-time according to the changing environment during robot motion.

Second, the robots must coordinate with each other to generate target formations. A common approach to coordinated multi-robot obstacle avoidance is to assign priority to each robot and then introduce distributed control to coordinate the robot's velocities based on the set priority. However, most existing prioritization strategies are mostly static and based on conditions where global environmental information is known (He et al. 2003). Robots that reach the formation position first as they approach their formation target position may block the paths of other robots and trigger motion rushes in, ultimately resulting in the inability to generate a specific formation. This project aims to introduce consistency control based on distributed control to solve the motion conflict problem.

In addition, the problem of multi-robot formations moving on a set path is also investigated in this paper. The problem is important in scenarios with frequent tasks and limited resources, such as handling large objects. Considering the global path planning of the formation, most path planners based on random sampling only consider the point robot model to generate paths, and the generated trajectories are not smooth. Our approach considers the robot's kinematic constraints during path generation to make the generated paths smoother. It can be embedded in other improved algorithms of sampling-based path planners. For trajectory tracking control, Sun et al. (Sun et al. 2009) pro-

posed a synchronous control approach to solve a similar problem. However, this method only solves the problem of how the robots follow their respective known optimal trajectories. Peng et al. (Peng & Akella 2005) modeled multi-robot motion planning as an integer mixed nonlinear programming problem. The optimization objectives of existing trajectory tracking methods rarely consider the need for formation. Therefore, we propose a motion planning method that considers formation keeping.
The contributions of this project are as follows: In addition, the contributions of this project are as follows:

- Propose an efficient heuristic graph search global path planner to compute the optimal global trajectory in the convex region considering the formation constraints, and introduce kinematic constraints to ensure the
- Propose a distributed multi-robot motion control strategy for generating formation and implementing consistency control when the robots approach the target formation position, thus avoiding the problem of unproducible formation due to motion conflict.
- Introduce leader-follower control to achieve formation retention under set paths, and the proposed online local path planning enables robot formations to avoid unknown obstacles.
- Introduce time-configuration space to achieve multi-robot collaboration by setting static priority to apply sequential trajectory planning.

1.3 Outline

This paper is organized as follows: Section 2 describes some of the work related to global path planning for robot formations and multi-robot motion planning and control; Section 3 talks about the background of the algorithms and methods used in this project, which is necessary to understand the improvements and enhancements made in this project; Section 4 describes the methods used for multi-robot global path planning in this project; Section 5 describes the methods used for multi-robot motion planning and control in this project; Section 6 details the experiments conducted and analyzes the experimental results; Section 7 discusses an overall discussion of the achievements and limitations of this project; Section 8 gives an overall description of the project; Appendix A includes additional information for more details on the project structure and running code.

2 Related Work

2.1 Multi-robot task allocation

Multi-Robot Task Assignment (MRTA) aims to find the best task allocation solution for a robot considering the robot's nature, usage, and task load when multiple robots perform multiple tasks so that the robot gets the maximum benefit with the minimum cost. There are three main types of task assignment methods: centralized control, distributed control, and hierarchically distributed control (Khamis et al. 2015). Currently, the main models for distributed task assignment are the market bidding model (Zhu et al. 2014), the auction algorithm model (Lee et al. 2014), and the reinforcement learning model (Omidshafiei et al. 2017). Multi-robot task assignment is a typical NP-Hard problem, and the optimization solution algorithms mainly include heuristic algorithms and mathematical planning algorithms. Although mathematical planning algorithms can find the optimal solution, the time and space complexity grows exponentially as the number of robots increases. The heuristic algorithm cannot guarantee the solution's optimality but is computationally efficient. The heuristic algorithm is better for solving MRTA problems in complex environments and is the best strategy for practical task allocation problems.

2.2 Global Path Planning

In complex environments, deadlock-free navigation of multiple robots without considering dynamic obstacles can be achieved by computing global paths from initial to target configurations and intermediate collision-free configurations for a group of robot formations (Alonso-Mora et al. 2017). Kushleyev et al. (Agarwal et al. 2013) used a mixed integer quadratic programming approach for UAV formations in a known 3D environment for obstacle coordination. Saha et al. (Saha et al. 2014) used pre-computed motion primitives using an SMT solver to synthesize individual robot trajectories. The above approach can guarantee the planning of global paths for each robot but does not consider the arbitrary nature of the formation definitions, which depend on a particular formation. Another typical approach is based on random sampling in map space, where the geometric constraints of the formation are taken into account in the overall trajectory of the formation to compute a safe configuration of a set of robot formations in a feasible path. Barfoot and Clark et al. (Barfoot & Clark 2004) consider a circle enclosing this formation, treat a formation in the same way as a single robot, and then based on a probabilistic roadmap (PRM), compute a global path for that formation. Krontiris et al. (Krontiris et al. 2012) proposed a geometric motion framework that directly computes a probabilistic roadmap for the formation based on the geometry of the formation. Alonso et al. (Alonso-Mora et al. 2016) computed a set of feasible configurations and traversable safety regions in free space, using a sampling of unexplored areas of the workspace from these sets. The above sampling-based methods have probabilistic completeness and can guarantee a feasible global path after a certain number of iterations. Still, the randomness of sampling makes it difficult to guarantee computational efficiency and does not guarantee to find of the globally optimal path. We introduce queueing constraints in the sampling process to generate a set of feasible configurations and thus are applicable to any formation shape. In addition, we introduce a heuristic search and compute the optimal configuration within a rasterized convex safety region, allowing us to find feasible globally optimal paths efficiently.

2.3 Computing Convex Regions

The idea of computing convex regions in the workspace originated from early work with cell decomposition (Barraquand et al. 1992, LaValle 2006). Recently, many algorithms have been proposed for the approximate convex decomposition of workspaces. For example, lien et al. (Lien & Amato 2004) decompose polygons containing zero or more holes into "approximately convex" blocks within the allowed concavity range. Similarly, Liu et al. (Liu et al. 2010) defined the convex decomposition problem as an integer linear programming problem and obtained an approximate optimal solution by minimizing the total decomposition cost under certain concave constraints. However, these convex optimization methods select the approximate convex blocks that may yield regions that intersect with obstacles, contrary to our original intention of obtaining safe convex regions. Another decomposition approach grids the workspace and triangulates it. Ayanian et al. (Ayanian et al. 2011) combine environmental triangulation with navigation functions for multi-robot control. Demyen et

al. (Demyen & Buro 2006) proposed a real-time path search based on triangulation for non-point objects, which significantly reduces the search space and can find the first path, or find the optimal path, very quickly. Sarmiento et al. (Sarmientoy et al. 2005) propose a sample-based convex cover algorithm that records the intersection of overlapping convex regions, thus capturing the connectivity of the workspace. Fischer et al. (Fischer 1993) find a maximal region of a convex polygon in a discrete environment using a set of points labeled positive or negative, whose vertices, boundaries, and interior contain no negative points. Deits and Tedrake et al. (Akin et al. 2015) directly create a single large convex region in a local region in the workspace that will be separated from all obstacles without covering it with samples. Our approach builds on the work of Deits and Tedrake et al. (Akin et al. 2015). It exploits the nature of overlapping convex regions to achieve collision-free navigation between feasible configurations of robot formations. We combine heuristic sampling and formation constraints to construct the space of feasible configurations of robot formations.

2.4 Trajectory Optimization

Simba et al. (Simba et al. 2016) proposed criteria for a good trajectory: trajectory that is at least second-order differentiable and continuous at each point; local controllability, where any change in the trajectory is locally controllable and affects only a limited area; and the ability to set arbitrary first- and second-order derivatives at the starting point. Sampling-based trajectory generation methods often do not consider the criteria mentioned above. The obtained trajectories are usually a linear set of trajectory slices with sharp parts at the inflection points (Ho & Liu 2009). Frequent robot rotations are required in motion planning, consuming time and causing unnecessary wear and tear on the robot's hardware. Therefore trajectory optimization is necessary. Wang et al. (Wang et al. 2012) modeled the trajectory optimization problem as a segmented quadratic or cubic Bézier curve. However, low-order Bézier curves do not guarantee the continuity of curvature and the standard. Neto et al. (Neto et al. 2010) proposed seventh-order Bézier curves, but they require high computational cost and may oscillate. Mellinger et al. (Mellinger & Kumar 2011) introduced velocity, acceleration, and input constraints to generate optimal trajectories in real-time by solving a quadratic programming problem, which can satisfy those mentioned above. However, the optimized trajectory is not guaranteed safe and will deviate from the original trajectory. We introduce inequality constraints and trajectory error terms for convex safety regions based on the Minimum Snap method to generate smooth and safe trajectories.

2.5 Trajectory Tracking

Pure tracking control is the most commonly used path tracking strategy. Coulter et al. (Coulter 1992) first proposed a pure-pursuit algorithm for explicit path tracking. Ollero et al. (Ollero et al. 1994) designed a real-time fuzzy controller to adjust the control parameters of pure pursuit and dynamically adjust the forward-looking distance according to the path point, robot speed, and tracking error. Urmson et al. (Urmson et al. 2006) introduced an integral correction to the pure tracking tracker to reduce the tracking error due to the error between the desired and actual steering angles. Stentz et al. (Stentz et al. 2002) introduced a scaling term with the heading angle error with a scaling term as feedback control. Wit et al. (Wit et al. 2004) considered both the position of a point ahead and the desired direction to calculate the current turn radius, maintaining the geometric meaning of the trajectory. Shan et al. (Ollero & Heredia 1995) proposed the CF-pursuit method based on pure-pursuit. Like pure-pursuit control, they use a fuzzy system to directly calculate the path curvature and replace the circle with a clothing curve to reduce the tracking error. In addition to geometric tracking methods, Pan Zhao (Zhao et al. 2012) proposed using an adaptive PID controller to track the predetermined path. Although the PID controller is more accurate, the parameters are more difficult to determine. Awais et al. (Abbas 2011) designed an online model predictive control (MPC) controller to track the planned path, but it is computationally intensive. Considering the tracking error, computational efficiency, and the limitation of low speed in this project's robot handling task, we chose to use pure tracking control with dynamically adjusted forward-looking distance as the trajectory tracking strategy.

2.6 Multi-robot Motion Coordination

Distributed multi-robot systems have been expanded in several fields, and one of the most popular research areas is the motion coordination problem. This includes multi-robot path planning, formation generation, and formation maintenance.

2.6.1 Multi-robot Motion Planning

Multi-robot motion planning can be divided into two categories: coupled and decoupled methods (Arai & Ota 1992). Coupled planning methods integrate the configuration space of each robot into a composite state space to search for feasible paths. Using a search-based algorithm such as the A-star algorithm (Hart et al. 1968) ensures that the solution obtained has resolution integrity. Coupled methods can rely on a central planner to handle the state information of all robots and have a simple structure. Still, the main limitation of this class of methods is that the complexity of the spatial search grows exponentially with the number of robots. Hopcrofi et al. (Hopcroft et al. 1984) demonstrated that the multi-robot motion planning problem is PSPACE difficult. To solve the complexity problem, Toumassoud et al. (Tournassoud 1986) proposed the artificial potential field method, but the artificial potential field method suffers from the problem of local extremum points.
The decoupled motion planning method greatly reduces the computational complexity of multi-robot motion planning. The decoupled method first designs the path for each robot and then solves its possible conflicting problems. Erdmann and Lozano (Erdmann & Lozano-Perez 1987) constructed a "configuration time-space" for the robot, which adds temporal information to the robot state. The theory based on the time-configuration space has led to the method of prioritized planning (Van Den Berg & Overmars 2005). Such methods first assign priority to robots and then plan the path of each robot in the time-configuration space in priority order. For the problem of how to assign priorities, Ferrari et al. (Ferrari et al. 1998) used a fixed priority. They used a random detour to allow low-priority robots to avoid high-priority robots. Buckley et al. (Buckley 1988) used a heuristic to assign high priority to robots that can take a straight line. Another decoupled planning method is the path coordination method (O'Donnell & Lozano-Pérez 1989). This method pre-plans the path of each robot and then avoids collisions by coordinating the robots' movements (forward, stop, or backward) on their set paths (Bien & Lee 1992).
In addition, the parallel computing nature of distributed path planning results in reduced computational complexity and can be combined with existing single-robot motion planning methods. However, the lack of global information about the robots makes it difficult to coordinate operations between robots (Herrero-Perez & Matinez-Barbera 2008). Distributed planning methods are generally based on sensor information (Harinarayan & Lumelsky 1994), i.e., they rely exclusively on sensor information for motion planning, and each robot works independently in a coordinated manner. This approach is usually implemented using the artificial potential field method (Ge & Cui 2002); however, the motion of the robots in such methods is not restricted to accomplish the formation task. Ge and Cui et al. (Broggi et al. 2000) restrict the motion of the robots to collision-idle paths. Furthermore, Broggi et al. (Broggi et al. 2000) restrict the motion of the robots to the path network (roadmap). In this project, we use a fixed priority, apply an artificial potential field method to search for paths in the time-configuration space, and introduce consistency control to maintain the formation.

2.6.2 Multi-robot Formation

The formation of multiple mobile robots is a typical problem in multi-robot systems. To achieve multi-robot formation control, each robot needs to know its geometric position relative to the other robots. Currently, the main methods for geometric position determination are the Leader-follower method (Cowan et al. 2003), the behavior-based method (Scharf et al. 2004) and the virtual structure method (Ren & Beard 2004*a*). Many robot formation control methods use the Leader-follower method framework (Desai et al. 2001, Vidal et al. 2003). In this case, the follower computes its control signal based on its leader's state information to maintain a certain distance and direction. Most Leader-follower methods model robot formations as graphs, imposing kinematic constraints on each node of the graph (each robot) and the edges of the graph (relative position relationships between robots) as formation constraints (Desai et al. 2001, Vidal et al. 2003). Feddema et al. (Feddema et al. 2002) analyzed such methods' stability through the theory of interacting correlation systems. Tan and Lewis et al. (Tan & Lewis 1997) were the first to propose virtual structures. In this approach, the

robots maintain a specific geometry among themselves, thus forming a virtual rigid structure (Olfati-Saber & Murray 2002). Leonard et al. (Leonard & Fiorelli 2001) used artificial potential fields to replace the rigid formation structure. They controlled the group behavior of the robots by assuming a virtual robot as the pilot robot of the entire formation. Lawton et al. (Lawton et al. 2003) combined Leader-follower and virtual structure methods to use formation position error and tracking error as a criterion for formation stability. Ren and Beard et al. (Ren & Beard 2004*b*) used the virtual structure method for UAV formation control and feedback controllers to reduce formation error. Behavior-based control methods produce overall robot formation behavior by designing corresponding control rules for basic robot behaviors (obstacle avoidance, heading to target, maintaining formation, etc.). Balch and Arkin et al. (Tan & Lewis 1997) first proposed a behavior-based control method by applying a pilot-based formation to design controllers that maintain formation behavior. Ge and Sam et al. (Ge et al. 2004) introduced artificial potential field functions to coordinate mechanisms for behavior, where gravitational and repulsive forces mutually constrain robots to keep their distance from other robots. In general, multi-robots usually do not assume the existence of a centralized controller (Chen & Wang 2005), and robot controllers are designed based on local information. Without a pre-designated leader, it is difficult for robots to coordinate with each other to achieve global consensus. In addition, roboticists have mainly focused their work on control strategies for robot formations. More limited work has been done to study the motion planning problem for multi-robot formations.

3 Background

This section aims to overview and discuss the methods and algorithms used in this project.

3.1 Hungarian Algorithm

The Hungarian algorithm is a combinatorial optimization algorithm for solving task assignment problems in polynomial time (Yao et al. 2022). Consider the multi-robot task assignment problem, where given n tasks are assigned to N robots (the number of tasks is the same as the number of robots) to maximize the completion efficiency. Assume that the cost of the ith robot to complete the jth task is c_{ij} non-negative, each task can be completed by only one robot, and each robot can only complete one task. The problem can be modeled as an objective minimization function as follows:

$$\begin{aligned} &\min \sum_i \sum_j c_{ij} a_{ij} \\ &s.t. \sum_i a_{ij} = \sum_j a_{ij} = 1 \; i = 1, ..., N, \; j = 1, ..., N \\ &where \; a_{ij} = \begin{cases} 1 & \text{assign the } i\text{th robot to complete the } j\text{th task} \\ 0 & \text{the } i\text{th robot is not assigned to the } j\text{th task} \end{cases} \end{aligned} \tag{1}$$

The objective of the Hungarian algorithm is to find the optimal solution to the allocation problem by finding n zero elements in different rows and columns of the transformation coefficient matrix. The algorithm first subtracts the minimum from each row and column of the coefficient matrix so that there are zero entries in each row and column and tries to find the optimal solution to the allocation. If, after the above steps, there is still a row without zero elements, the assignment is made by adding zero elements to the matrix. The above steps are repeated until n zero elements are found in different rows and columns, and the optimal solution is obtained.

3.2 Iterative Regional Inflation by Semidefinite Programming

IRIS (Akin et al. 2015) is a recently developed greedy convex segmentation technique for computing convex obstacle-free regions. The IRIS algorithm calculates both an ellipse and a set of hyperplanes that separate it from all obstacles in the workspace. It defines an ellipse as a linear transformation of the unit circle in a two-dimensional plane:

$$E(C, d) = \{\tilde{x} = Cx + d, \; ||x|| \leq 1\} \tag{2}$$

and define the set of hyperplanes with linear inequality constraints:

$$P = \{x \,|\, Ax \leq b\} \tag{3}$$

Thus, the problem of computing the largest ellipse and separating the ellipse from all obstacles in the hyperplane can be transformed into solving the following nonconvex optimization problem:

$$\begin{aligned} A^*, b^*, C^*, d^* &= \arg \max_{A,b,C,d} \log\det C \\ s.t. \; A_i v &\geq b_i, \; v \in S_j, \; j = 1, ..., N \\ \sup_{||x|| \leq 1} A_i \; (Cx + d) &\leq b_i, \; i = 1, ..., N \end{aligned} \tag{4}$$

where A_i,b_i are the hyperparameters of the current hyperplane, S_j represents the set of points in the convex obstacle, V represents the vertices of j_{th} obstacles, and N is the number of obstacles. The first inequality constraint guarantees that the vertices of all obstacles are outside the separating hyperplane. The second inequality constraint ensures that the ellipse is contained within the hyperplane. The above nonconvex optimization problem is decomposed into two steps: the first step computes the hyperparameters A and b that define the hyperplane, and the second step finds the maximum ellipse that satisfies the linear inequality constraint.

3.3 Minimum Snap

The trajectory is generally represented by a polynomial of order n with time t, as follows:

$$p(t) = [1, t, t^2, ..., t^n] \cdot p \tag{5}$$

At any given moment, by deriving the trajectory with respect to time, velocity, acceleration, jerk and snap can be calculated:

$$\begin{aligned} v(t) = p^{'}(t) &= [0, 1, 2t, 3t^2, 4t^3, ..., nt^{n-1}] \cdot p \\ a(t) = p^{''}(t) &= [0, 0, 2, 6t, 12t^2, ..., n(n-1)t^{n-2}] \cdot p \\ jerk(t) = p^3(t) &= [0, 0, 0, 6, 24t, ..., \frac{n!}{n-3!}t^{n-3}] \cdot p \\ snap(t) = p^4(t) &= [0, 0, 0, 0, 24, ..., \frac{n!}{n-4!}t^{n-4}] \cdot p \end{aligned} \tag{6}$$

Minimum Snap (Mellinger & Kumar 2011) requires the trajectory to satisfy a series of constraints, such as setting the position, velocity, or acceleration of the starting and ending points, the connection of the position, velocity, and acceleration of adjacent trajectories, the intermediate point through which the trajectory is desired, and setting the limits of maximum velocity and maximum acceleration. To solve the optimal concrete trajectory that satisfies the constraints, we need to construct an optimal function Minimum Snap. Minimizing the objective function as Snap, the above problem is modeled as a constrained optimization.
Setting the position, velocity, acceleration, or higher derivative of a point to a specific value produces the following equality constraint:

$$\begin{aligned} position\ constraint &: [1, t_0, {t_0}^2, ..., {t_0}^n, 0...0] \cdot p = p_0 \\ velocity\ constraint &: [0, 1, 2t_0, ..., n{t_0}^{n-1}, 0...0] \cdot p = v_0 \\ acceleration\ constraint &: [0, 0, 2, ..., n(n-1){t_0}^{n-2}, 0...0] \cdot p = a_0 \end{aligned} \tag{7}$$

For the location of the intermediate passage points, the corresponding equality constraints should also be constructed in the same way as above.
The equality constraints on the successive positions, velocities, and accelerations between adjacent segments i and $i+1$ are as follows:

$$[0...0, 1, t_i, {t_i}^2, ..., {t_i}^n, -1, -t_i, -{t_i}^2, ..., -{t_i}^n, 0, ...0] \cdot p = 0 \tag{8}$$

A trajectory consisting of k points contains three equality constraints at the starting point, three equality constraints at the ending point, k-1 constraints through intermediate points, and 3(k-1) continuity constraints at intermediate points. In total, there are 4k+2 constraints.
The above problem can be modeled as a quadratic programming (QP) problem with constraints, and the optimization function is Snap:

$$\begin{aligned} \min f(p) = \min \int_0^T (p^{(4)}(t))^2\, dt = \min \sum_{i=1}^{k} \int_{t_{i-1}}^{t_i} (p^{(4)}(t))^2\, dt &= \min \sum^{k} p^T Q_i p \\ s.t.\ A_{eq} p &= b_{eq} \end{aligned} \tag{9}$$

3.4 Artificial Potential Field Method

The artificial potential field method is commonly used in local path planning. It introduces the concept of "field" from traditional mechanics into motion planning. The robot moves under this virtual force field. It defines the external environment of the robot as a function of the potential field. It controls the robot to avoid obstacles and drive to the target point on local information. The basic idea is based on an artificial potential field formed by the gravitational field generated by the target point and the repulsive field generated by the obstacle, and the direction of gradient descent of the potential field function is considered as the direction of the collision-free path, as shown in Fig. 2. The direction of the gravitational field F_{att} generated by the target point to the mobile robot is the

direction of the robot to the target point, and the other part is the repulsive force field F_{rep} generated by the obstacles to the mobile robot. The direction is the pointing of the obstacle to the robot, and the resultant force F of both is the direction of the robot's motion. Equations (12) to (13) define the typical gravitational and repulsive force fields U_{att} and U_{rep}, and the corresponding gravitational and repulsive forces F_{att} and F_{rep}. The artificial potential field method allows the robot to better adapt to changes in the surrounding environment with good real-time performance.

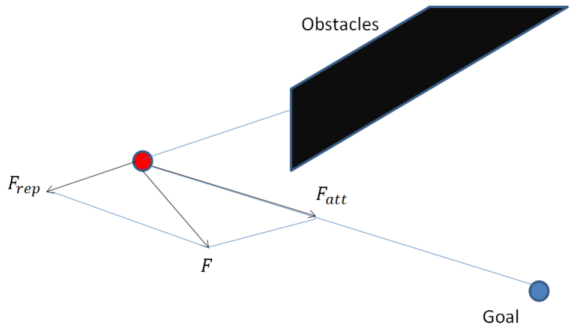


Figure 2: Artificial Potential Field Method

$$U_{att}(q) = \frac{1}{2}\epsilon dis(q, q_{goal})^2 \tag{10}$$

$$U_{rep}(q) = \begin{cases} \frac{1}{2}\eta(\frac{1}{dis(q, q_{obs})} - \frac{1}{p_0}^2) & dis(q, q_{obs}) \leq d_0 \\ 0 & dis(q, q_{obs}) > d_0 \end{cases} \tag{11}$$

$$F_{att}(q) = -\nabla U_{att}(q) = \epsilon dis(q, q_{goal}) \tag{12}$$

$$F_{rep}(q) = -\nabla U_{rep}(q) = \begin{cases} \frac{1}{2}\eta(\frac{1}{dis(q, q_{obs})} - \frac{1}{d_0})\frac{1}{dis^2(q, q_{obs})}\eta dis(q, q_{obs}) & dis(q, q_{obs}) \leq d_0 \\ 0 & dis(q, q_{obs}) > d_0 \end{cases} \tag{13}$$

The general artificial potential field method suffers from the following problems:

- Too much gravity: goal points farther away from the robot can exert a greater gravitational force on it, leading to a direct hit on the obstacle.
- The resultant force is 0: when the sum of the gravitational and repulsive forces of the robot at exactly a certain point is 0, it will be trapped in a local minimum and stop moving.
- Unreachable target: when the target is not far from an obstacle, the repulsive force on the robot will be greater than the gravitational force, and it will not reach the target point.

3.5 Pure Pursuit Control

Pure Pursuit is a geometry-based lateral control algorithm for path tracking. It controls the robot by calculating the desired angular velocity to move from the current position to a pre-sighting point in front of the robot. The algorithm continuously updates the pre-sighting points moved on the path based on the robot's current position and velocity until it reaches the end of the path. The pre-sight distance directly affects the selection of the pre-sight point.

As shown in Fig. 3, the robot at the current position points to the next path point to be followed, and L is the distance from the robot's current position to the target path point. Given the current position, speed, and the set of path points the robot has currently followed, the control information such as target speed, acceleration, corner angle, and corner angular speed of the vehicle is output. Assume that the robot can travel to this pre-target point with a turning radius of R. The orientation angle of the pre-target point is 2α. Taking the time slice into account, the front wheel turning angle δ at the next moment can be estimated given the angle $\alpha(t)$ of the path target point at moment t and the forward-looking distance L from the target way point. Based on the geometric relationship, we have:

$$\frac{l}{\sin 2\alpha} = \frac{R}{\sin \frac{\pi}{2} - \alpha} \tag{14}$$

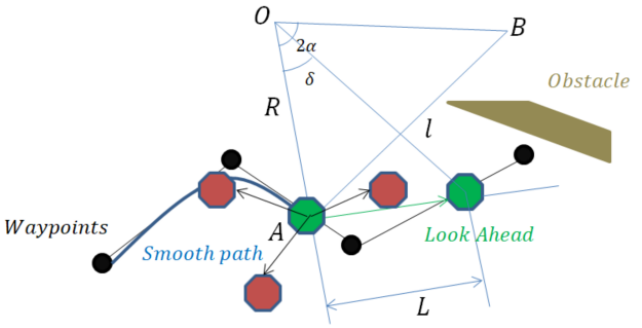


Figure 3: Pure Pursuit Control

$$\tan\delta = \frac{L}{R} \tag{15}$$

Combining the two equations above, the steering angle δ at the next moment can be obtained as follows:

$$\delta(t) = \arctan(\frac{2L\sin\alpha(t)}{l}) \tag{16}$$

Thus, the robot can follow a planned global path whose curve approximates a circular arc.
The forward-looking distance L is a linear function of the robot's longitudinal velocity, which will be dynamically adjusted according to the robot's motion:

$$l = k_v v + l_0 \tag{17}$$

The larger the forward-looking distance, the smoother the tracked trajectory, and conversely, the smaller the tracking accuracy will be reduced.

3.6 Consensus Control

In multi-robot cooperative control, the control protocol of each robot is distributed according to a formation-specific graph topology. The topological graph of a multi-robot formation can be represented as $G = (V, E)$, where its vertices V denote the set of robot states and E denotes the state of communication connections between robots. The robots control the motion of each robot on the adjacent edges. In graph theory, the graph is represented by an adjacency matrix A, where the element corresponds to when two robots can communicate with each other $A_{ij} = 1$, otherwise $A_{ij} = 0$. The sum of the input-output streams of each robot is described by a degree matrix D. D is a diagonal matrix describing each robot's importance for the formation's control. As shown in Fig. 4, the formation consists of six robots, where the starting point of the arrow indicates the issuer and the pointing one is the receiver, and the corresponding adjacency matrix, degree matrix, and Laplace matrix are calculated in Equation (18). It can be seen from the figure that $robot_2$ corresponds to the largest value in the degree matrix and has a dominant role in the formation control, so it is chosen as the leader.

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}, \quad D = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix}, \quad L = \begin{bmatrix} 2 & -1 & -1 & 0 & 0 & 0 \\ -1 & 4 & -1 & -1 & 0 & -1 \\ -1 & -1 & 3 & -1 & 0 & 0 \\ 0 & -1 & -1 & 3 & -1 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & -1 & 0 & 0 & -1 & 2 \end{bmatrix} \tag{18}$$

First-order consensus control aims to make the position and velocity between leader and follower converge through control inputs. Convergence of the formation is achieved when the relational constraint network reaches a certain steady state. The first-order consensus system model of a formation system can be expressed as

$$\dot{x_i} = u_i \tag{19}$$

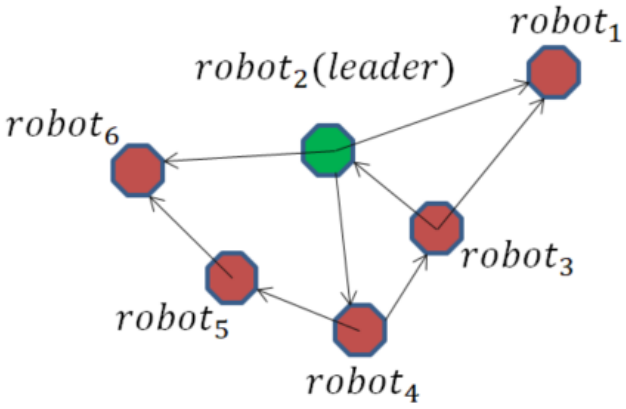


Figure 4: Formation control based on graph theory

where x_i denotes the state of the intelligence i at a given moment and u_i denotes the control input of the intelligence i at the current moment. The control input in the ideal case can be expressed as follows:

$$u_i = \Sigma a_{ij}(x_j - x_i), j \in N_i \tag{20}$$

where a_{ij} are the elements forming the adjacency matrix, and N_i is the set of agents adjacent to the agent i. By introducing an n-dimensional state vector $x = [x_1, ..., x_n]^T \in R^n$, the following dynamic equation is obtained:

$$\dot{x} = -Dx + Ax = -Lx \tag{21}$$

The above equation shows that the dynamic properties depend on the Laplace matrix L. Guattery et al. (Guattery & Miller 2000) have shown theoretically that the system can achieve consistency, and the final state value is the initial state when and only when the formation topology has a spanning tree and the eigenvalues of L lie on the left half of the complex plane:

$$c = \sum_{i=1}^{N} p_i x_i(0) \tag{22}$$

The state transition equation in discrete time can be expressed as follows:

$$x_i(k+1) = x_i(k) + u_i(k) \tag{23}$$

4 Multi-robot Formation Global Path Planning

4.1 Overview

This section covers global path planning for robot formations. Section 4.2 presents an IRIS-based method for computing convex obstacle-idle regions. In Section 4.3, we propose an efficient heuristic graph search algorithm for finding a globally optimal path by introducing a formation constraint. Finally, in Section 4.4, we propose an improved minimum snap method with the introduction of convex region constraints and trajectory errors to plan smooth and safe trajectories for the formation.

4.2 Generating Convex Regions of Obstacle-free Space

In this section, the computation of convex obstacle regions is inspired by IRIS (Iterative Regional Inflation by Semi-definite programming) (Akin et al. 2015), and we use the set of separated hyperplanes returned by the algorithm as our set of convex obstacle-idle spaces. The overall computational procedure of the convex obstacle-idle region is given in Algorithm 1. Given a collection of sampling points and a set of obstacles Obs, we find a hyperplane defined by $A \leq b$ and a maximum ellipse $E(C,d) = \{x = Cx + d, ||x|| \leq 1\}$ such that the hyperplane separates the ellipse from all obstacles.

Algorithm 1: Input a starting point p_0 and a set of obstacles Obs, output convex obstacle-free regions P represented by $Ax \leq b$ and corresponding maximal ellipse $\epsilon = \{Cx + d \,|\, \|x\| \leq 1\}$ which satisfies $\epsilon \subseteq P$.

```
Initialization:
C_0 ← εI_{2×2}
d_0 ← p_0
i ← 0
while detC_i − detC_{i−1}/detC_{i−1} < threshold do
    (A_{i+1}, b_{i+1}) ← SeparatingHyperplanes(C_i, d_i, Obs)
    (C_{i+1}, d_{i+1}) ← MaximalEllipse(A_{i+1}, b_{i+1}, Obs)
    i ← i + 1
    return (A_i, b_i, C_i, d_i)
end
```

We take the point selected from the two-dimensional space as the ellipse parameter d and initialize a small circle with this point as the center. If unfortunately, the chosen d does not fall in the accessible space, this can be solved by reversing the direction of several separated hyperplanes to choose new sampling points. After the initialization, we try to find the separated hyperplane to ensure that the generated ellipse does not collide with any obstacle.

Each iteration uses a hyperplane for each obstacle, separating it from the generated ellipse until the nearest point to the ellipsoid is found on the obstacle. Finding the separated hyperplane is reduced to a quadratic programming problem of solving the minimum distance by linear mapping by first finding the closest obstacle vertex to the ellipse. After finding the closest obstacle vertex to the circle, linear mapping is applied to find the closest point to the ellipse. Here the motivation for choosing the nearest point is the distance between the nearest obstacle, and the ellipse separating the hyperplane can generally separate the obstacles that will be farther away. By defining the inverse of the linear mapping in equation (24), the problem of finding the optimal point is transformed into a quadratic semi-definite optimization problem with linear constraints. After solving the problem, the optimal solution x^* applies linear mapping to the point that is the closest obstacle vertex to the ellipse. To compute the x^* and the tangent plane, the ellipse is expressed as the following inequality:

$$\varepsilon = \{x \,|\, (x-d)^T C^{-1} C^{-T} (x-d) \leq 1\} \tag{24}$$

The hyperparameter A_i is obtained by calculating the vector at the surface perpendicular to the ellipse can be obtained by calculating the gradient of the elliptic function:

$$\begin{aligned} A_i &= \nabla[(x-d)^T C^{-1} C^{-T} (x-d)]|_{x^*} \\ &= 2C^{-1} C^{-T} (x^* - d) \end{aligned} \tag{25}$$

Since this hyperplane crosses x^*, the other hyperparameter b_i of the hyperplane can be found by the hyperparameter A_i:

$$b_i = A_i x^* \tag{26}$$

Subsequently, it is checked if the vertices of all other obstacles are in the hyperplane, i.e., only each vertex is checked if it satisfies the inequality defined in (3). If the inequality holds, then the vertex is separated by the already existing hyperplane, so the calculation of the separated hyperplane for this vertex is ignored. Until all obstacles are separated from the existing ellipse by the hyperplane. The flow of this part is shown in Algorithm 2.

Algorithm 2: Input an ellipse ϵ represented by matrix C and d and a set of obstacles Obs, output a set of hyperplanes defined by A and b that are tangent to *epsilon* and satisfy $\{x \in \mathbb{R}_2 — Ax \leq b\} \bigcap Obs = \varnothing$. Sub-functions $ClosestObstacle, ClosestVertice, TangentPlane$ are described in Sect.

initialization
$Obs_{outside} \leftarrow \varnothing$
$Obs_{rest} \leftarrow Obs$
$i \leftarrow 1$
while $Obs_{rest} \neq \varnothing$ **do**
  $Obs^* \leftarrow ClosestObstacle(C, d, Obs_{rest})$
  $x^* \leftarrow ClosestVertice(C, d, Obs^*)$
  $(a_i, b_i) \leftarrow TangentPlane(C, d, x^*)$
  foreach $Obs^*{}_i \in Obs_{rest}$ **do**
    if ${a_i}^T x_j \geq b_j, \forall x_j \in Obs^*{}_i$ **then**
      $Obs_{rest} \leftarrow Obs_{rest} \backslash Obs_i^*$
      $Obs_{outside} \leftarrow Obs_{outside}\ Obs^* \bigcup Obs_i^*$
    end
  end
  $i \leftarrow i + 1$
end
$A \leftarrow \forall a_i^T, b \leftarrow \forall b_i$
return (A, b)

4.3 Heuristic Graph Search

In this section, we propose a heuristic graph search-based global path planning algorithm based on the decomposition of the global map into convex obstacle-free regions in the previous section, which can quickly find a feasible trajectory or search a large enough region to find a globally optimal path according to a specific need. In addition, we consider queueing constraints to ensure that the generated queueing configuration is safe and efficient. In addition, using a priori information from the map, the planner can explore new convex obstacle-idle regions heuristically and more efficiently. Finally, we define a nonlinear quadratic programming problem based on convex regions and queueing constraints, which solves the optimal configuration within the convex regions and efficiently finds paths to the goal points.

4.3.1 Formation Constraint

In this project, since we cannot treat robot formations as prime points, we need to introduce a formation constraint to ensure the validity of the convex region. The formation constraint depends on the geometry of the formation itself. We defined four specific formations of different shapes, a circle and a rectangle composed of four robots, a triangle composed of three robots and a straight line composed of two robots.

The formation of the different shapes of the robot is shown in Fig. 5, where the red rectangle represents the robot, and the black objects of different shapes are objects. We need to ensure that the formation of the robot does not go beyond the convex region within the convex region. We define

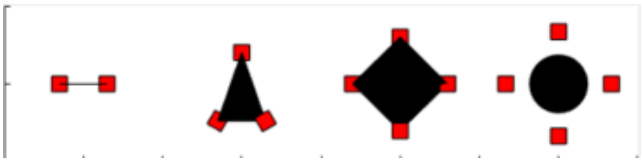


Figure 5: Different robot formations

the inequality constraint as follows:

$$A_i v_j \leq b_i, \forall i = 1, ..., N, j = 1, ..., M \tag{27}$$

where A_i and b_i are the hyper-parameters of all separated hyper-planes and v_j is each vertex of the robot formation. We require that each vertex of the robot formation lies within the convex region.

4.3.2 Heuristic Seeding

The efficiency of path planning based on graphical search depends on the selection of sampling seed points. Previously, we assumed that static obstacles are given, i.e., environmental information is valuable, so we can artificially select regions that are more likely to be free of obstacles as the locations of sampling seeds. For this purpose, a heuristic sampling method is proposed in this section. Using known global map information, suitable regions are selected as seeds to generate as many additional convex obstacle-free regions as possible as soon as possible.

First, we compute the target configuration's initial location and the convex region and then rasterize the two-dimensional space into a grid map. Next, we choose the grid cell with the maximum distance dis_{obs} between the nearest obstacle and the grid and the distance $dis_{polygon}$ from the largest existing convex region, the latter being defined as the Euclidean distance from the center of the largest ellipse in the existing convex region to the grid. Finally, we use the weighting of the two distances and $dis_{weighted}$ as our metric:

$$dis_{weighted} = w_1 * dis_{obs} + w_2 * dis_{polygon} \tag{28}$$

where w_1 is set to 0.2 and w_2 is set to 0.8. The reason is that we tend to generate new convex regions, i.e., existing convex regions, as much as possible, rather than generating convex regions away from obstacles as much as possible, which is consistent with the idea of searching the global map quickly. We count the cells within the obstacle and the cells within the existing convex region and define the distance as negative infinity. The largest distance among the remaining cells is found, and a new convex region is set at that point. The distance matrix is updated to maximize the distance to the nearest obstacle and the existing convex region. Suppose the new seed is located inside an obstacle or in the list of already existing convex regions. In that case, it is rejected from the list, and the value of the distance matrix where it is located is set to negative infinity. An overview of the above algorithm is given in Algorithm 3.

We have two ways to define the condition of iteration termination, which depends on quickly generating a feasible or globally optimal trajectory. The first is to terminate the algorithm when a certain area is explored. We define the sum of explored convex areas and obstacle areas as a percentage of the global map area or the area not currently occupied by obstacles or convex areas as a threshold. Once the threshold is exceeded, the algorithm terminates. By common sense, if we let the algorithm run long enough until convex obstacle-empty areas cover most of the 2D map, the path searched can be considered optimal. The second way is that a feasible path is found, although it is not globally optimal. We discuss the path cost and computing time due to the two iterative conditions in the experimental section of Section 6.

4.3.3 Graph Search

We define the graph as $G = \{V, E\}$ where V is the set of vertices (nodes) and E represents edges. Each vertex in the vertex list V represents a valid configuration. Once a convex region P appears that both the initial and the target configurations are contained in this convex region, the two nodes are added to the edge list E, representing the edges that connect the two nodes. Otherwise, we initialize the list of nodes to the midpoint of the starting queue and the target queue and initialize

Algorithm 3: Function: $p_{sample} = GenerateSample(O, \epsilon, P)$
Given a set of obstacles O, the set of elliptic centers for each polygon ϵ, and existing polygon P, output the optimal sampling point p_{sample}.

Raster the global map into cells $Cell$
$i \leftarrow 1$
foreach $cell_i \in Cell$ **do**
  if $cell_i$ *is inside the* O *or* P **then**
    $DistanceMatrix_i \leftarrow -Inf$
  else
    $DistanceMatrix_i \leftarrow dis(cell_i, O_{nearest} + dis(cell_i, \epsilon))$
  end
  $i \leftarrow i + 1$
end
Normalize the Evaluation Matrix
$p_{sample} \leftarrow \arg \max_{cell_i \in Cell} (DistanceMatrix)$
Generate the polygon of new node P_{sample}
$P_{sample} \leftarrow formation(P_{sample})$
foreach *existing nodes* V *and existing polygons* P **do**
  if $dis(p_{sample}, N)$ *or* $dis(P_{sample}, P) < threshold$ **then**
    $DistanceMatrix\{p_{sample}\} \leftarrow -Inf$
    return $\varnothing$;
  else
    return p_{sample}
  end
end
return p_{sample}

a list L_p of all valid configurations contained for each convex region P_i. Similarly, we initialize the list of convex regions to the convex region of the starting configuration and the convex region of the ending configuration.

If the sample is not within any existing obstacle and the generated convex region is far from the existing convex region, the convex region is added to the list. We take the existing convex region P as input and determine whether a valid configuration z exists at that sample point by introducing an inequality constraint (queueing constraint) and return the relevant parameters of the configuration point if the configuration is inside the convex region. Otherwise, we return the empty set. To speed up the graph search process, we propose a quadratic optimization problem based on the inequality constraint to find a second valid configuration that minimizes the points at a distance to the target configuration. The distance here we define as the Euclidean distance between the center of mass of the target formation to the center of mass of the formation to be solved. The inequality constraint includes the formation constraint in Section 4.3.1 of the map boundary constraint, i.e., the formation does not extend beyond the map boundary, and each vertex of the formation is in the interior of the convex region. The details of solving the effective configuration are given in Algorithm 4.

For each polygon P that intersects an already existing convex region P_p, we compute such a con-

Algorithm 4: Function: $z = formation(P)$
Given the convex polygon P, output a feasible configuration z that minimize the nonlinear function under linear constraints or $\varnothing$.

if $P \in \varnothing$ **then**
  return $\varnothing$
else
  foreach $A_i, b_i \in P_i$ **do**
    $A_p \leftarrow A_i, b_p \leftarrow b_i$
  end
end
Calculate the optimal configuration;
$z^* = \arg\min_z (z - z_d)^2, s.t. V(z) \subset P, z \subset Map_{range}$;
return (z^*);

figuration z for the robot team: all its vertices are completely contained in the intersection of two convex regions, and the center of mass of the team has the smallest Euclidean distance to a valid configuration of one of the convex regions. If a configuration satisfies the above two conditions, it is added to the list of vertices of the graph. After that, our node list is then added with a valid configuration, and we iterate through all configurations of the already existing convex region $z_i \in P$, adding an edge $\{z, z_i, P_p\}$ to all convex regions containing the newly added configuration; iterate through all configurations of the newly generated convex region $z_i \in P_p$, adding an edge $\{z, z_i, P\}$ to all convex regions containing the newly added configuration. The above steps are shown in Algorithm 5.

To find the shortest path that guides the robot team from the initial configuration to the target configuration, we need to search for the shortest path based on an existing graph that consists of a set of valid configurations (vertices) and connected edges. We use a classical algorithm based on heuristic search: the A^* algorithm (Hart et al. 1968). Given a set of vertices V and a set of edges E, we compute the shortest path T. Since in Section 3.3 we present heuristic-based seeding, we can quickly find a feasible trajectory, possibly locally optimal, or run longer to find the globally optimal one.

4.4 Trajectory Optimization

The global path planning method we proposed in the previous section is based on sampling, and the trajectory consists of a series of discrete two-dimensional spatial points, which usually have abrupt changes in velocity direction at the inflection points and require large angular velocities and angular accelerations from the motion controller, which are detrimental to the robot motion. We want the optimized trajectory to roughly coincide with the original trajectory and introduce kinematic constraints to limit the velocity and acceleration in the trajectory. In this section, a new quadratic

Algorithm 5: Global path planning
Given the start point s, start configuration z_s, goal point g, and obstacle set O, returns a global trajectory T with feasible formations from s to g.

Initialize the node list and edges of the graph: V$\leftarrow z_s; E \leftarrow \varnothing$
Compute P$_{start}, P_{goal} in the obstacle - free space$
Initialize polygon list
P$\leftarrow P_{start}, P_{goal}$
Find the feasible configuration in P$_{goal}$
$z_{goal} \leftarrow$formation(P_{goal})
Add the goal configuration to the node list
$V \leftarrow z_{goal}$
Add the feasible configurations to the lists of two polygons
$L_{p_{start}} \leftarrow z_{start}, L_{p_{goal}} \leftarrow z_{goal}$
#Check for feasible formation in the intersection area of two polygons
if $\exists\ z_v \leftarrow formation(P_{start} \cap P_{goal})$ **then**
    $V \leftarrow z_v$
    $E \leftarrow \{z_{start}, z_v, P_{start}\}, \{z_{goal}, z_v, P_{goal}\}$
end
#The following heuristic graph search begins, looping until a certain area of space has been explored and then jumping out $s_{explored} > threshold$.
while $s_{explored} < threshold$ **do**
    Generate the new sampling node
    $p_{sample} \leftarrow GenerateSample(O, \epsilon, P)$
    Generate new polygon $P_{p_{sample}}$ for the new sample p_{sample}
    #Create two new feasible nodes z and z_{min}. z is the configuration of the new sample and z_{min} is the configuration that minimizes the distance to the goal.
    if $\exists\ z \leftarrow formation(P_{p_{sample}})$, $z_{min} \leftarrow formation(P_{p_{sample}})$ **then**
        $L_{P_{p_{sample}}} \leftarrow z$
        $L_{P_{p_{sample}}} \leftarrow z_{min}$
        #Iterate over existing polygons, generate edges and add feasible nodes to the list
        foreach $P_{exit} \in P$ **do**
            if $\exists\ z_{inter} \leftarrow formation(P_{p_{sample}})$ **then**
                foreach $z_i \in L_P$ **do**
                    $E \leftarrow z_{inter}, z_i, P_{exit}$
                end
                foreach $z_j \in L_{P_{p_{sample}}}$ **do**
                    $E \leftarrow z_{inter}, z_j, P_{p_{sample}}$
                end
                $V \leftarrow z_{inter}$, $E \leftarrow z_{inter}$
            end
        end
        $P \leftarrow P_{p_{sample}}$
        return $T_{shortest} = ShortestPath(G(V, E))$
    else
        return $T_{shortest} = \varnothing$
    end
end

optimization equation is proposed to solve the safe optimized trajectory by considering the trajectory error and convex region constraint for the disadvantage that the smooth trajectory generated by Minimum Snap deviates more from the original trajectory and may encounter obstacles.

4.4.1 Time Distribution

Given an average velocity, we calculate the total time T based on the total length of the trajectory, assuming that the velocity in each polynomial is constant and the time is uniformly distributed to each trajectory, so that the time allocated to each trajectory is proportional to the length of the trajectory. This gives us the time vector of the trajectory segments. By the time allocation method, the trajectory parameters can be obtained by solving the optimization function. If the maximum values of velocity and acceleration do not exceed the limit values of each trajectory segment, a feasible smooth trajectory can be obtained. Otherwise, it is necessary to adjust the time of the unsatisfied trajectory segments or insert more intermediate points.

4.4.2 Bounding Box Constraints

To make the optimized trajectory still confined to the convex region, we introduce the bounding box as an inequality constraint into the quadratic programming equation according to the equality constraint of the Minimum Snap method so that the newly computed trajectory falls within the convex region. In the time allocation step, some intermediate points are inserted on each segment of the global path. In principle, inequality constraints should be imposed on both the original trajectory points and the interpolated points, but since the fixed points themselves are feasible configurations, and in order to avoid knotting in the optimized trajectory, we only consider the interpolated points. In addition, to reduce the computational effort, instead of the direct method of imposing inequality constraints on all interpolated points at intermediate points, we first find the point that does not meet the constraint by traversing all new trajectory points after generating the trajectory under the equality constraint, and then add a bounding box to press it into the convex region according to the convex region defined by the separated hyperplane at that point. Our goal is to find a smooth curve in which all trajectory points are contained within the bounding box and through a fixed intermediate point of the trajectory.
For each interpolation point, we impose an inequality constraint based on a rectangular bounding box:

$$\begin{aligned} x_{lowerbound} &\leq p_{x_t\, upperbound} \\ y_{lowerbound} &\leq p_{y_t\, upperbound} \end{aligned} \tag{29}$$

The rectangle ensures that each formation formed with that point is within the convex region defined by the hyperplane for each interpolated point. We add two more inequality constraints in the x and y directions, for a total of 2(k-1) inequality constraints:

$$\begin{aligned} [1, t_i, {t_i}^2, ..., {t_i}^n]] \cdot p &\leq p(t_i) + r \\ [-1, -t_i, -{t_i}^2, ..., -{t_i}^n]] \cdot p &\leq -p(t_i) + r \end{aligned} \tag{30}$$

The size of the bounding box determines the margin of time adjustment, which in turn affects the feasible space for trajectory smoothing. The trajectory optimization method considering the bounding box constraint is presented in Algorithm 6.

4.4.3 Trajectory Error

In the previous section, based on the minimum Snap under the equality constraint, we introduced the bounding box constraint to secure the trajectory. Still, the trajectory is often at the edge of the bounding box. We want the trajectories to be as close to the path as possible, i.e., as straight as possible between waypoints and as smooth as possible at the corners. In this subsection, we improve the quadratic programming equation. In addition to the term that minimizes Snap, we add an error term between the original trajectory and the desired trajectory:

$$\min f(p) = \min \int_0^T (p^{(4)}(t))^2\, dt + \lambda \int_0^T (p(t) - p_e(t))^2\, dt \tag{31}$$

Algorithm 6: Given a set of obstacles O, the collection of convex regions $P\{A, b\}$ defined by separating hyper-planes, the original trajectory segments $OrigTrajPoint$, along with the set of equality constraints of Minimum Snap $EqCon$, return safe optimized trajectories $OptTrajPoint$ without collisions with obstacles.

```
OptTrajPoint ← MinimumSnap(OrigTrajPoint, EqCon)
foreach OptTrajPoint_i ∈ OptTrajPoint do
    if OptTrajPoint_i ∈ O then
        foreach P_j{A_j, b_j} ∈ P{A, b} do
            if OptTrajPoint ∈ P_j{A_j, b_j} then
                IneqCon ← CalR(A_j, b_j)
            else
                IneqCon_i ← ∅
            end
        end
    else
        IneqCon_i ← ∅
    end
end
return OptTrajPoint ← MinimumSnap(OptTrajPoint, IneqCon, EqCon)
```

where $p_e(t)$ is the expected trajectory, defined as follows:

$$p_e(t) = [1, t, t^2, ..., t^n]p_o \tag{32}$$

p_o is the parameter of the smoothed trajectory, and the error term of each segment is defined as a straight line between two endpoints, i.e., the parameter of the trajectory segment p_{o_i} between the two endpoints $p_{t_{i-1}}$ and p_{t_i} of the ith trajectory segment is defined as follows:

$$p_{o_i} = [p_{t_i}, -\frac{p_{t_i} - p_{t_{i-1}}}{t_i - t_{i-1}}ti - 1, \frac{p_{t_i} - p_{t_{i-1}}}{t_i - t_{i-1}}, 0, 0, ..., 0] \tag{33}$$

λ is the weight of the error term, which is used to adjust the smoothness and trajectory tracking error. The larger λ is, the more critical the trajectory tracking effect is and the closer the optimized trajectory is to the original trajectory. Still, the peak velocity acceleration becomes too large, leading to a violation of the goal of smoothing the trajectory, so a trade-off is needed. After introducing the error term, we let:

$$H_i = \int_{t_{i-1}}^{t_i} [1, t, t^2, ..., t^n][1, t, t^2, ..., t^n]^T \, dt \tag{34}$$

Removing the constant term, the new QP objective function is expressed as follows:

$$\begin{aligned} \min \int_{t_{i-1}}^{t_i} (p^{(4)}(t))^2 \, dt + \lambda \int_{t_{i-1}}^{t_i} (p(t) - p_e(t))^2 \, dt \\ = \min p^T Q_i p + \lambda p^T H_i p - \lambda {p_o}^T H_i p \\ = \min p^T (Q_i + \lambda H_i) p + -\lambda {p_o}^T H_i p \end{aligned} \tag{35}$$

Similar to the previous solution method, the corresponding equality and inequality constraints are constructed to solve for the optimal trajectory.

5 Multi-robot Formations Motion Planning

5.1 Overview

In multi-robot motion planning, the formation's geometric constraints need to be considered. Furthermore, the motion planning method for robot formations depends on specific task requirements, such as task assignment, formation, formation maintenance, and cooperative motion. In this section, we propose a distributed multi-robot planning approach where each robot or group of robot formations updates its motion strategy in real-time based on local and environmental information. In Sections 5.2 and 5.3, we combine consistency control with a modified artificial potential field method and trajectory tracking control method, respectively, to solve the robot formation and formation maintenance problems and, incidentally, the multi-robot task assignment problem. We introduce the local path planning method in section 5.4. Finally, in Section 5.5, we present the time-configuration space to realize the simultaneous motion planning of multiple robot formations on set routes.

5.2 Multi-robot Task Allocation and Formation

5.2.1 Problem definition

For the problem of multi-robot cooperative target handling, we first need to assign robots according to several targets of different shapes and sizes, which requires multiple robots to reach the specified target point to form a formation. As shown in Fig. 6, a set of robots in any position is placed in a complex environment with obstacles. After completing the task assignment, the robots need to move to the specified area to form a formation and avoid internal motion conflicts between robots and unreachable targets. We need to design a scheduler that, given several tasks and a number of robots, assigns a specific number of robots and formations for each particular task. Secondly, we design a real-time path planner under the assumption that the robots can perceive information about their surroundings, being the robots able to update their paths according to the constantly changing environment.

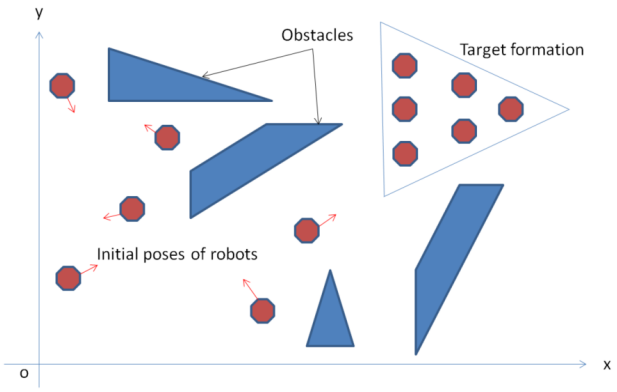


Figure 6: Motion planning for multi-robot formation

5.2.2 Multi-robot Task Allocation

As shown in Fig. 7, given a number of targets, we need to reasonably assign tasks to multiple robots based on their initial positions. Multi-robot task assignment is not the focus of this project, so we assign a robot to the location of the nearest target array to it based on the greedy idea. We define the cost of each robot to the target point as the Euclidean distance and calculate the cost of each robot to all target points to obtain the cost matrix. We apply the Hungarian algorithm (Kuhn 1955), which takes the cost matrix as an input and outputs the index at which each robot should reach the target point.

5.2.3 Motion Planning for Multi-robot Formation

After completing the task assignment, each robot has a specific target point. Due to the complexity of robot formations, it is difficult to apply the multi-robot motion planning approach directly to multi-robot formations. In this subsection, we will focus on the multi-robot motion planning problem

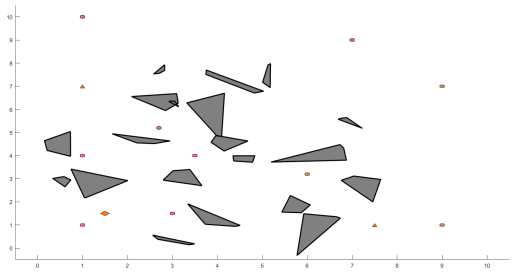


Figure 7: Multi-robot task allocation

of setting the target location point at the target formation location. We assume in the problem definition that each robot can perceive limited local information about its surroundings, which requires the robot to perform motion planning in real-time based on the local environment and environmental information about its surroundings. Since multi-robot motion planning is an extension of single-robot motion planning, it is equivalent to a real-time motion planning problem for each robot. Therefore, we use a distributed control structure, i.e., we expect each robot to be able to update its motion strategy in response to the changing environment in the localization. The artificial potential field method can better plan the robot's kinematic parameters based on local information. Still, it has limitations, such as unreachable targets, excessive gravitational force, and local minima. So we propose a modified version of the artificial potential field method that successfully allows a single robot to reach a target point in a dynamic environment. In addition, when a robot approaches the target position of a formation, it may not be able to reach the target point due to the unreachability of the robot caused by other robots that arrived first and occupied the target point. Instead of using a priority-setting strategy to solve the above problem, we use a leader-follower algorithm based on consistency control to enable a group of robots to reach the target position in the desired formation when they approach the target formation, avoiding the jamming phenomenon.

1. Improved Artificial Potential Field Method

To address the problem of excessive gravitational force, a threshold can be set to limit the size of the gravitational force and avoid large gravitational forces due to the distance from the target point:

$$U_{att}(q) = \begin{cases} \frac{1}{2}\epsilon dis(q, q_{goal})^2 & dis(q, q_{goal}) \leq d_{threshold} \\ d_{threshold}\epsilon p(q, q_{goal}) - \frac{1}{2}\epsilon (d_{threshold})^2 & dis(q, q_{goal}) > d_{threshold} \end{cases} \tag{36}$$

$d_{threshold}$ gives a threshold value that limits the distance between the target and the object. The corresponding gravitational force is:

$$F_{att}(q) = \begin{cases} \epsilon dis(q, q_{goal}) & dis(q, q_{goal}) \leq d_{threshold} \\ \dfrac{d_{threshold}\epsilon dis(q, q_{goal})}{dis(q, q_{goal})} & dis(q, q_{goal}) > d_{threshold} \end{cases} \tag{37}$$

By introducing a new repulsive force function (Hanli et al. 2021), the problem that the target is unreachable due to the presence of obstacles near the target point can be solved:

$$U_{rep}(q) = \begin{cases} \frac{1}{2}\eta(\dfrac{1}{dis(q, q_{obs})} - \dfrac{1}{d_0}^2)dis(q, q_{goal})^2 & dis(q, q_{obs}) \leq d_0 \\ 0 & dis(q, q_{obs}) > d_0 \end{cases} \tag{38}$$

The effect of distance from the target point is based on the original repulsive force field. It can be intuitively seen from the formula that when the object is close to the target, although the repulsive field increases, the distance decreases so that the repulsive field can be dragged to a certain extent. The corresponding repulsive force then becomes:

$$F_{rep}(q) = \begin{cases} F_{rep1} + F_{rep2} & dis(q, q_{obs}) \leq d_0 \\ 0 & dis(q, q_{obs}) > d_0 \end{cases} \tag{39}$$

where

$$F_{rep1} = \eta \frac{1}{dis(q, q_{obs})} - \frac{1}{d_0} \frac{dis(q, q_{goal})^2}{dis(q, q_{obs})^2}, F_{rep2} = \eta (\frac{1}{dis(q, q_{obs})} - \frac{1}{d_0})^2 dis(q, q_{goal}) (40)$$

2. Local Minimum Problem

When the robot's surroundings contain obstacles close to each other, the chaotic potential field can produce local minima of the potential field beyond the goal point, causing the robot to oscillate or even stop moving. There are usually two ways to solve this problem: one is to use the global information of the map, i.e., to set the corresponding potential parameters on all obstacles to avoid or minimize the possibility of falling into local minima as much as possible, but this is not practical in complex maps. Establishing a stable multi-potential field in dynamic environments is difficult, and there is a risk of re steering the robot back to the local minima point. Another approach is using heuristics such as genetic or evolutionary algorithms to find the global minima directly. However, in dynamic and real-time environments, prior knowledge is limited, and the limited heuristic capability makes it difficult to guarantee the accuracy of the global minima.

A simple and effective approach to the local minimum problem is that a random perturbation can be added. However, in most cases, adding a slight perturbation will cause local oscillations, and the oscillations will not stop until the local minimum is jumped. This is undesirable in practical engineering, and in the control module we also want to avoid oscillations as much as possible.

Since the formation of local minima is due to the joint formation of repulsive and gravitational potential fields generated by several obstacles, the robot is bound to be not far from the obstacles. In other words, instead of relying on a priori information about the surrounding dynamic environment or obstacles, we move around the edges of the nearest perceived obstacles to escape the local minima point. When determining that the robot is stuck in a local minimum, we set the gravitational force exerted on it by the target to zero and decide whether or not it is possible to reach the space where the local minimum is avoided, and if so, make it move against the edge side of the nearest obstacle (the normal direction of the equipotential plane of repulsion) in the direction pointing to the side close to the goal point. Gravitational and repulsive forces are restored until the robot gets rid of the local minimum. See Algorithm 7 for the calculation of the repulsive force and Algorithm 8 for the process of escaping the local minima.

We use the following two judgment conditions to determine whether the robot is trapped in a local

Algorithm 7: Function: $[replusion, L_{rep}] = ComputeRepulsion(O, Pose_{robot}, r_{det}, Pose_{end})$
Given a set of obstacles O, pose of the agent $Pose_{robot}$, detection radius r_{det}, and the pose at the goal point $Pose_{end}$, output repulsion $replusion$, and a list L_{rep} recording the distance of the robot from the obstacles.

```
Calculate the distance from each obstacle to the agent
distance ← 0
if dis(Pose_robot, Pose_end) < threshold then
    r_det = r_det
else
    r_det = r_det/2
end
foreach O_i ∈ O do
    distance ← distance + dis(O_i, Pose_robot)
    if distance < r_det then
        replusion ← F_rep
        L_rep ← distance
    else
        replusion ← 0
        L_rep ← Inf
    end
end
```

Algorithm 8: Function: $Vel_{agent} = LocalMinimal(O, vel_{max}, L_{rep}, Goal_{agent})$
Given a set of obstacles O, maximum velocity vel_{max} ,repulsive force list of obstacles L_{rep} and goal pose of the agent $Goal_{agent}$, follow the strategy of moving along the obstacles, output the appropriate velocity of the agent Vel_{agent}.

Find the obstacle edge that generates the maximum repulsive force
$edge_{repmax} \leftarrow \max(L_{rep})$
foreach $o_i \in O$ **do**
  if $edge_{repmax} \in o_i$ **then**
    $k \leftarrow CalculateSlope(endpoint_l, endpoint_r)$
    break
  end
end
if $dis(endpoint_1, goal) < dis(endpoint_2, goal))$ **then**
  $Vel_{agent} \leftarrow vel_{max}, arctan(endpoint_2, endpoint_1)$
else
  $Vel_{agent} \leftarrow vel_{max}, arctan(endpoint_1, endpoint_2)$
end

minimum.

$$|F_{att}(q) + \sum F_{rep}| < \epsilon \tag{41}$$

where ϵ is a very small positive number, indicating that the combined force on the robot is approximately zero.

$$|x_p - x_c| < \alpha s \tag{42}$$

where *alpha* is a positive number between 0 and 1, x_p is the state of the robot at any point during its motion, and s denotes the total number of miles the robot has traveled from x_p to the current position x_c, indicating that the robot's displacement velocity is minimum after a certain number of miles. It is used to detect whether the robot oscillates around the local minimum.

3. Motion Conflict in Formation

During the formation process, each robot needs to avoid obstacles in the environment and prevent collisions with other robots. Therefore, we construct a temporal configuration space for the robots, i.e., the state information of each robot includes not only the robot's position and orientation but also time slice information. In this temporal configuration space, each robot treats other robots in the same time slice as obstacles in that space. We set a safe distance as an obstacle avoidance condition, which needs to be added to the calculation of repulsion when the distance between any two robots enters within the safe threshold.

Introducing a temporal configuration space can solve the collision problem between robots. How-

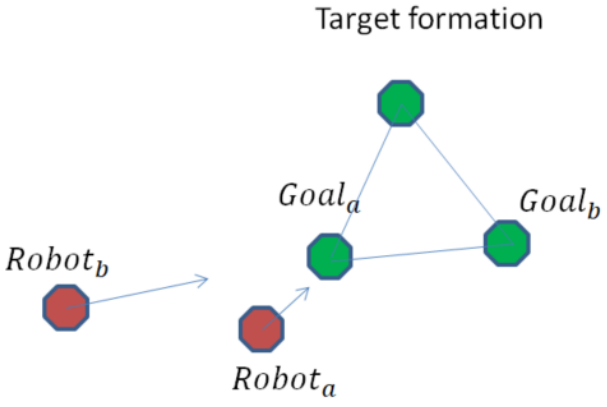


Figure 8: The robot that reaches the position first prevents the other robots from reaching the goal position

ever, when robots reach the formation's target position, the robots will still repel each other, causing motion conflicts. Specifically, the first robot to reach the expected position of the formation may prevent other robots from reaching the expected goal. As shown in Fig. 8, $Robot_a$ and $Robot_b$ approach the desired formation positions $Goal_a$ and $Goal_b$ simultaneously, and since $Robot_a$ is closer to the target position, it will reach the end first and stop moving. Since $Robot_b$ will see $Robot_a$ as an obstacle at this point, $Robot_b$ will never reach the goal point due to the repulsive force. In a worse case, if $Robot_b$ is the last robot to reach this particular queue, the repulsive force on $Robot_b$ from the other positions in the queue will be very high and eventually cause stagnation and jamming. In this case, we consider that $Robot_a$ causes a motion conflict for $Robot_b$. This motion conflict tends to occur in the final stage of the formation.

We combine a consensus-based leader-follower algorithm to solve the above problem with an improved artificial potential field approach. We assume that each robot senses local information in real-time, i.e., each robot can sense the geometric position relative to other neighboring robots, represented by the relative distance in the global coordinate system, as shown in equation (43). Based on the adjacency matrix, we represent the relative position of each robot as a topological map, thus associating all robots together. Fig. 9 illustrates a set of robot formations, where the bidirectional arrows indicate the formation constraints that need to be satisfied between any two robots. When the formation relationships of all these robot pairs are satisfied, the entire formation converges, and the above process will be achieved through consistency control. The motion conflict generally occurs at the stage when the formation of the formation is about to take place, which ensures the feasibility of implementing leader-follower control.

$$\Delta x_{i,j} = x_j - x_i,\ \Delta y_{i,j} = y_j - y_i \tag{43}$$

$$0 \le v_i \le v_{max},\ |a_i| \le a_{max} \tag{44}$$

Within the leader-follower approach framework, the follower's control input is determined by the state of the leader and the formation information of the neighboring robots. We model each group of robot formations as a formation graph in which the nodes are the poses of the robots, the edges represent the relative distance of each robot from the other robots in the formation, and each follower keeps a fixed distance and angle from the leader to maintain the formation. As the robots are subject to gravitational and repulsive interactions before the implementation of consensus control, each robot can move towards a predetermined target point and maintain a certain distance from the other robots. Using two different control strategies, each group of robots can eventually form a specific formation. In addition, considering the feasibility of consensus control, We limit the velocity and acceleration of our robots, as shown in equation (43), to solve some unexpected situations, such as avoiding some moving obstacles.

4. Consensus Control Based on Graph Theory

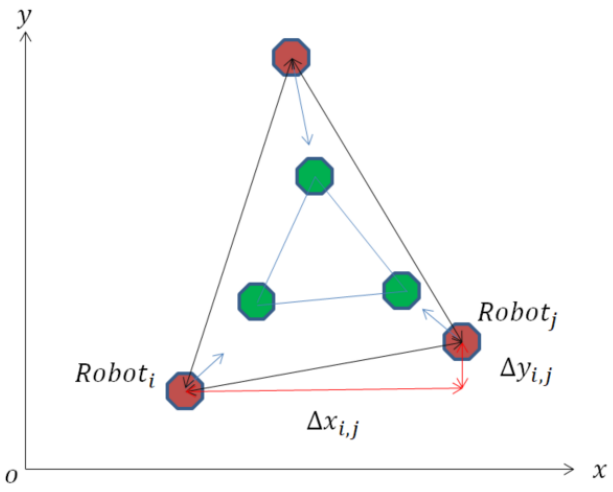


Figure 9: Formation constraint nets for a particular formation

This project uses distributed control to implement leader-follower-based first-order consensus control.

The algorithm for follower formation is as follows:

$$u_i(k) = \epsilon \sum_{j \in N_i} a_{ij} w_{ij} (x_j k - x_i(k) - r_{ij}(k)) \tag{45}$$

where

$$w_{ij} = 2 - e^{-(x_{ij}k - r_{ij}(k))^2} \tag{46}$$

epsilon is a constant greater than zero, and r_{ij} denotes the relative distance between robots i and robots j. We use formation error and tracking error to measure the stability of the formation. When the formation converges to stability, both errors will converge simultaneously, and w_{ij} reflects the difference between the formation relationship of the actual motion and the desired formation relationship during the whole motion. The following equation defines the control input for the leader N:

$$u_N(k) = mD(k) + \sum_{i \in N_i} a_{Ni} r_{Ni}(k) \tag{47}$$

where r_{Ni} is the relative distance between leader N and follower i, m is a constant, and $D(k)$ is the distance between the leader and the target point at k moments. The leader's control input is determined by the distance between the neighboring robot and its target point. For multiple robots about to form a formation, with the introduction of consensus control, the 'target point' of each robot is determined by the position of neighboring robots so that the gravitational force is dynamically variable. This may avoid the problem of unreachable goals due to motion conflicts.

5. Combination of Consensus Control and Artificial Potential Field

We combine the artificial potential field method with the consistency control. At each moment, the distance between the leader and the target point is calculated as the leader's gravitational force and does not exceed a set threshold. At the same time, the repulsive force on the robot is calculated for all obstacles within the robot's detection range so that the robot continues to move towards the target point while avoiding the obstacles. The force of the leader is defined in equation (48).

$$F_{leader} = \epsilon_l (mD(k) + \sum_{i \in N_i} a_{Ni} r_{Ni}(k)) + F_{rep}(q_{leader}) \tag{48}$$

The gravitational force of the follower is determined based on the position error information of the robots connected to it in the relational constraint network, causing it to move toward a position that converges with the formation. Similar to the leader, the gravitational force is weighted together with the repulsive force of the detected obstacles as the velocity of the follower. Each robot updates the positions of all robots according to the velocity command. Define the force on the follower as shown in equation (49).

$$F_{follower_i} = \epsilon_f (\sum_{j \in N_i} a_{ij} w_{ij} (x_{ij}(k) - r_{ij}(k))) + F_{rep}(q_{follower_i}) \tag{49}$$

The motion planning algorithm for formation generation is shown in Algorithm 9. The inputs to the algorithm include the initial poses of several robots and the corresponding target positions of each robot, as well as information about obstacles in the environment. We first apply a modified artificial potential field method to achieve distributed control of all robots. In the time-configuration space, for each robot at a specific time slice, the poses of the other robots will be considered as obstacle positions added to the list of obstacles, and the corresponding repulsive forces will be calculated. At each time slice, we traverse all robots, update their velocity and positional information, and determine whether any robot falls into a local minimum. When the entire robots converge to a certain distance, i.e., close enough to their respective target points, we initiate consistency control to avoid motion conflicts. For each robot formation, we randomly select one robot as the leader and the rest as the followers. We define the adjacency matrix as the all-1 matrix of N×N, where N is the number of robots that make up the robot formation. The formation can be formed when the whole robot is less than a certain threshold from the respective target position, and the cycle can be stopped.

Algorithm 9: Function: A combination of artificial potential field method and one-order consensus control.
Given a set of obstacles O, the number of the agents N, current poses of the all robots $P_{current}$, the relative position matrix of the formation F_{delta}, the current velocity of the leader V_{leader}, the goal poses of followers $G_{followers}$, set of goal poses for the leader G_{leader}, and the detection radius r_{det}, output the new poses of the robots P_{next} and the velocity of the agents V_{agents} at next time step.

Kinematic parameters$\leftarrow (v_{max}, a_{max}, k_{leader}, k_{follower}, dt)$
AdjacentMatrix$\leftarrow I_{N\times N}$, EdgeWeightMatrix$\leftarrow \varnothing$,SumWeightMatrix$\leftarrow zeros_{(N-1)\times 1}$
$flag \leftarrow false$
#If flag is true, start consensus control
if $dis(P_{current_{followers}}, G_{followers} < threshold)$ **then**
  $flag \leftarrow true$
end
$V_{leader} \leftarrow k_{leader} dis(G_{leader}, P_{current_{leader}})$
foreach $other\ followers \in follower_i$ **do**
  $O \leftarrow P_{current_i}$
end
$replusion_{leader} = ComputeRepulsion(O, Pose_{current_{leader}}, r_{det})$
$V_{leader} \leftarrow V_{leader} + replusion_{leader}$
if $distance > dis_{threshold}$ *and* $V_{leader} < V_{threshold}$ **then**
  $V_{leader} \leftarrow -1 + 2 \times [0,\ 1]$
end
if $V_{leader} > v_{max}$ *or* $a_{leader} > a_{max}$ **then**
  $V_{leader} \leftarrow v_{max}$, $a_{leader} \leftarrow a_{max}$
end
$P_{next_{leader}} \leftarrow Pose_{current_{leader}} + V_{leader} d_t$
foreach $follower_i \in followers$ **do**
  if $flag\ is\ false$ **then**
    foreach $other\ follower_j$ **do**
      if $A_{ij} = 1$ **then**
        $sum_{delta} \leftarrow 0$
        $sum_{delta} \leftarrow sum_{delta} + edge_{weight}(P_{current_j}, F_{delta_{i,j}})$
      end
      if $sum_{delta} > distance_{threshold}$ **then**
        $sum_{delta} \leftarrow distance_{threshold}$
      end
    end
    $V_{follower_i} \leftarrow k_{follower} V_{leader} + \gamma sum_{delta}$
  else
    $V_{follower_i} \leftarrow k_{follower} dis(P_{current_i}, G_{followers_i})$
  end
  foreach $other\ robots$ **do**
    $O \leftarrow P_{current_i}$
  end
  $[replusion_{follower_i}, L_{rep}] = ComputeRepulsion(O, P_{current_i}, r_{det})$
  if $follower_i\ gets\ stuck\ in\ a\ local\ minimum$ **then**
    $V_{follower_i}, dir \leftarrow LocalMinimal(O, v_{max}, L_{rep}, Goal_{followers_i}))$
  else
    $V_{follower_i} \leftarrow V_{follower_i} + replusion_{follower_i}$
  end
  $P_{next_i} \leftarrow P_{current_i} + V_{follower_i} d_t$
end

5.3 Trajectory Tracking and Formation Holding

After the robots form a specific formation, the global path planner enables the multi-robot formation to compute a feasible global trajectory. In this section, we tackle the problem of planning the movement of multiple robots in formation on a set route. The global path planner computes a feasible global trajectory after the robots form a given formation. The robots are not only required to move collaboratively along the given global path but also to maintain the formation during the movement. We use the navigator-follower algorithm to implement multi-robot motion planning. We select the geometric center of the formation as the virtual leader and apply pure pursuit control to achieve trajectory tracking. When a moving obstacle approaches the robot formation, we use the dynamic window method to make the navigator robot change its motion strategy to avoid the collision. For the followers, the approach is similar to the formation, and we apply consistency control to make them follow the navigator to maintain a specific formation in real-time. In addition, boundary constraints are introduced on the velocity and acceleration of the robot to conform the kinematic parameters to the robot's motion.

Pure Pursuit Control is more robust at low speeds and can achieve better path tracking. We introduce PID control to calculate the acceleration from the current velocity to the desired velocity. A smaller pre-targeting distance enables the robot to track the path more accurately. A larger pre-sighting distance allows the robot to follow the path more smoothly, but it cannot track the original path accurately, and understeer can occur at large corners. In addition, a significant speed change in the robot will affect the forward-looking distance calculation of the pure tracking algorithm, leading to speed drift, which is not conducive to motion control. At low speeds, the path changes hardly affect the corner. When the path change at low velocity is relatively large, the dynamic adjustment of the foresight distance parameter can ensure a slight change in the turning angle. We first find the closest point to the current point in the target path according to the robot's current position and then calculate the closest point to this point from the forward-looking distance as the goal point. If the distance between two points is less than the forward-looking distance, the path point continues to be sampled backward until the distance exceeds the forward-looking distance. If the next path point is too far from the current point, we introduce a direction-override constraint to prevent the robot from backtracking.

5.4 Local Path Planning

When a dynamic obstacle approaches the robot formation, the navigator should respond accordingly to change the established motion strategy of trajectory tracking and plan a new local path immediately around the obstacle. The robot will re-detect the obstacle's position after updating the motion state. If there is no collision with the robot, it will continue to run along the trajectory from that position. In this section, we apply the dynamic window approach to plan a relatively optimal local path for the robot by considering the distance to the obstacle, the orientation of the goal point, and the velocity.

Since the robot in this project is holonomic, future trajectories can be predicted based on the kinematic model and given velocities. Therefore, only kinematic constraints are considered for sampling several velocities and angular velocities. The trajectory is projected and then scored for each possible trajectory using the evaluation function to select the optimal solution. We consider the velocity and acceleration constraints to determine the robot's velocity range, sample the velocity with a specific velocity resolution, and then use equation (50) as the basis for the evaluation.

$$E(v,\omega) = \epsilon(w_1 orientation(v,\omega) + w_2 distance(v,\omega) + w_3 velocity(v,\omega)) \tag{50}$$

where $orientation(v,\omega)$ is the angular difference between the robot and the next potential target path point, $distance(v,\omega)$ is the distance between the robot and the dynamic obstacle, and $velocity(v,\omega)$ is the velocity corresponding to the trajectory segment. The evaluation function is defined by the motivation to make the cart travel as fast as possible towards a potential target with a safe trajectory. We iterate through all the angular velocity velocities to calculate the corresponding evaluation functions and normalize them, assign appropriate weights to each one and select the optimal solution to publish.

We combine pure pursuit control with the dynamic window method. Whenever we calculate the

velocity and angular velocity required for trajectory tracking, we use the dynamic window to predict the robot's position after passing through all possible trajectories in the future and determine if there will be a collision with an obstacle. If it is a safe trajectory, the position is updated using the kinematic parameters calculated from the trajectory tracking. Otherwise, the dynamic window method is applied to select the best path for the robot. The above method is shown in Algorithm 10.

Algorithm 10: Combine the pure pursuit algorithm and DWA algorithm for motion planning.
Given the current robot's velocity $v_{current}$, angular velocity $\omega_{current}$, position $P_{current}$, the set of the midpoint of the trajectory W, the forward looking distance L, and the weight of the evaluation function,update the robot's motion parameters $(v_{new}, \omega_{new}, P_{new})$ according to the local information.velocity, angular velocity and position.

#Initialization
$P_{current} \leftarrow$ Position of the robot at the current moment
$(v_{current}, \omega_{current}) \leftarrow$ The velocity and angular velocity of the robot at the current moment
while *Robot does not reach the end point* **do**
  if *There are no dynamic obstacles around the robot* **then**
    #Find the nearest waypoint to the robot and the current goal point based on forward looking distance.
    $(P_{nearest}, P_{target}) \leftarrow FindPoint(W, L)$
    #Transformation of goal points from the global coordinate system to the robot coordinate system
    $P_{target} \leftarrow Transform(P_{target})$
    #Calculate the desired steering Angle δ
    $\delta \leftarrow CalculateAngle(P_{target}, L, l, \alpha)$
    #Update the state of the robot according to the movement per unit time
    $(v_{new}, \omega_{new}) \leftarrow UpdateState(P_{current}, v_{current}, \omega_{current})$
  else
    #Compute the range of velocity and angular velocity for the current sampling
    $(v_{sample}, \omega_{sample}) \leftarrow DynamicWindow(v_{current}, \omega_{current})$
    foreach $v_i \in v_{sample}$, $\omega_i \in \omega_{sample}$ **do**
      #Simulate a path over time
      $traj_{sample} \leftarrow ComputeTraj(v_i, \omega_i)$
      #Score based on evaluation function
      $EvalMatrix \leftarrow CalculateScore(traj - sample)$
    end
    #Normalize evaluation matrix and select the optimal solution
    $v_{new}, \omega_{new} \leftarrow Optimal(Normalize(EvalMatrix))$
  end
  #Update the position of the robot
  $P_{new} \leftarrow UpdatePos(P_current, v_{new}, \omega_{new})$
  return $v_{new}, \omega new, P_{new}$
end

5.5 Sequential Trajectory Planning in Time-configuration Space

So far, we have achieved a single formation's formation, maintenance, and obstacle avoidance functions. However, in practical scenarios, multiple robot formations are often required to work simultaneously, which requires cooperative control between each robot formation to avoid mutual collisions. In this section, we use sequential trajectory planning in the time-configuration space. We assign static priorities to robot formations based on the Euclidean distance of each robot formation to the target location, i.e., the closer to the target, the higher the priority. We assume that each robot formation can sense the position of other robot formations within a certain distance. When a robot formation approaches another robot formation, the lower priority robot formation will slow down or wait until the higher priority robot formation has passed. The lower priority robots are constantly aware of

their surroundings during this process until they are assured that there are no higher priority robots around, and then continue on their original trajectory.

As shown in Fig. 10, the priority of the robots decreases from top to bottom. At each moment, the

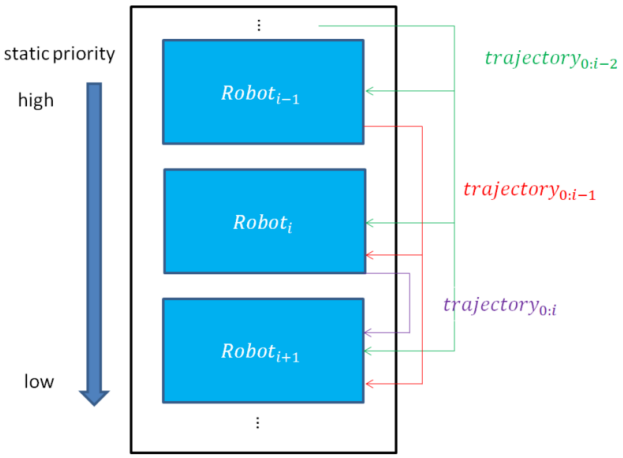


Figure 10: Sequential trajectory optimization

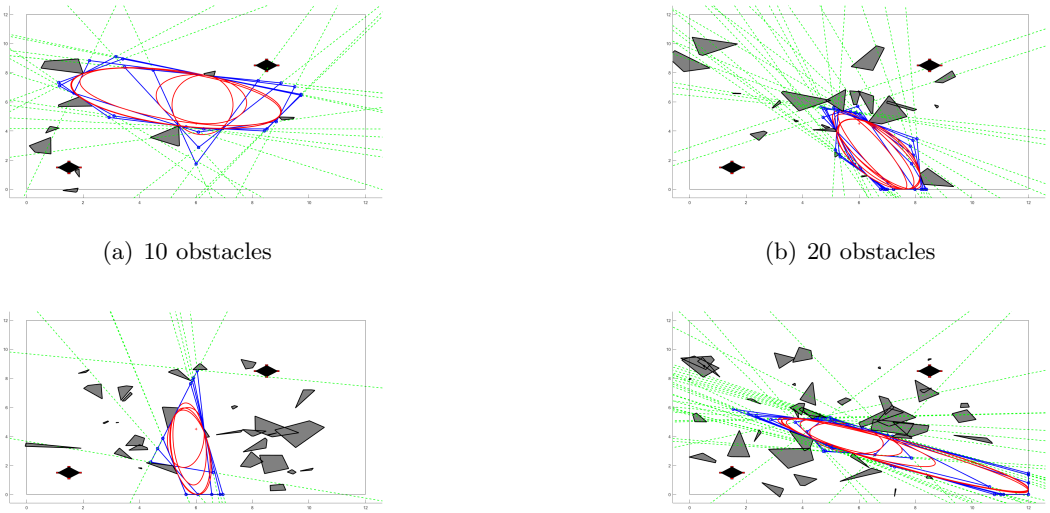
first i robots release their own position information to robots below their priority. For the low-priority robots, all high-priority robots are treated as obstacles. When they come within a safe distance, the low-priority robots avoid stopping their movement to ensure that the high-priority robots have priority to pass.

6 Experiments and Analysis

In this section, we describe in detail all the simulation experiments performed in this project, including experimental hypotheses, evaluation metrics, and experimental results. All simulation experiments were implemented in MATLAB on an Intel i5 processor at 2.3 GHz with 16 Gb of RAM.

6.1 Compute the convex obstacle -free regions

To test the computational performance of convex obstacle-free regions, we conducted five simulation experiments on MATLAB 2017. We make a crucial assumption here: the obstacles are all convex because optimal solutions do not necessarily exist in the non-convex case. We randomly generated 10, 20, 30, and 40 obstacles on a 10×10 map and then randomly selected a point to compute a set of convex regions. We use Matlab's SDPT3 solver (version 4.0) (ApS 2019) to solve the semi-definite programming problem. SDPT3 is a Matlab solver's toolbox for solving semi-definite programming problems with linear equations and matrix inequality constraints. The results of the convex region calculation are shown in Fig. 11, where gray objects represent obstacles, and each convex obstacle-free region consists of blue line segments and their green extensions. As the iterations proceed, the red ellipse becomes larger and larger until convergence. We counted the average computation time and the maximum computation time with different numbers of obstacles, as shown in Table 1, and the results show that the algorithm can converge in a few seconds.



(a) 10 obstacles

(b) 20 obstacles

(c) 30 obstacles

(d) 40 obstacles

Figure 11: Results of generating convex obstacle-free regions

Table 1: Computation time for convex obstacle-free regions using SDPT3 solver

Number of obstacles	10	20	30	40
Average computation time (s)	3.1480	5.2446	5.5238	6.6431
Maximum computation time (s)	4.0764	7.0227	5.9262	7.9501

6.2 Global Path Planning with Heuristic Graph Search

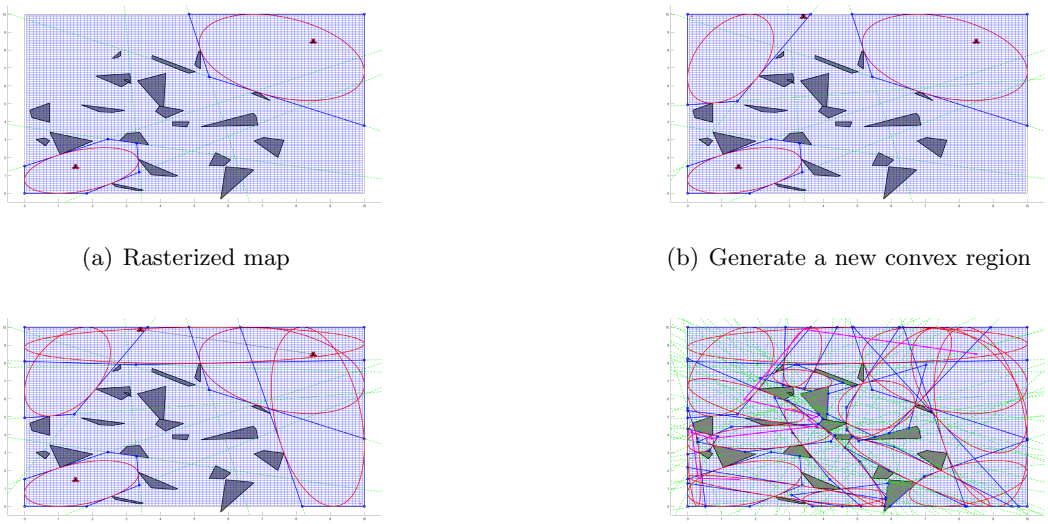
We mentioned in Section 4.3.1 to define solving feasible configurations under queueing constraints as quadratic optimization problems under inequality constraints. We choose Matlab's built-in optimization function fmincon to solve it. The minimization objective function is quadratic, and the linear inequality constraints on the parameter values are defined according to the hyperparameters A and b of the hyperplane to ensure that each vertex of the formation is in the convex region. The range of parameter values is set to the boundaries of the map. We performed five tests with 10,20,30, and 40 obstacles, respectively, and counted the time to calculate the optimal configuration using the fmincon function, as shown in Table 2. The results show that the optimal configuration can be found within half a second.

Table 2: Computation time for optimal configuration using fmincon function

Number of obstacles	10	20	30	40
Average computation time (s)	0.0885	0.0546	0.0798	0.1348
Maximum computation time (s)	0.1504	0.1152	0.1185	0.3636

We test our heuristic graph search algorithm on a 10×10 map consisting of 10 random convex obstacles. First, we rasterize the two-dimensional space into a grid map and then compute the convex regions corresponding to a given initial location and target configuration as shown in Fig. 12 (a). According to Equation (28) we obtain the evaluation matrix as shown in Fig. 13. We select a point far from the existing set of obstacles and convex regions as a new seeding point to generate a new convex region, as shown in Fig. 12 (b). We perform a graph search in the existing convex region with all feasible configurations denoting the vertices in the graph and the solid black lines denoting the safe path line segments connecting the two configurations and serving as the edges of the graph, as shown in Fig. 12 (c). The above process is repeated until we find a feasible shortest path, as shown by the solid pink line in Fig. 12 (d).

We conducted simulation experiments in four sets of environments consisting of different numbers



(a) Rasterized map

(b) Generate a new convex region

(c) Perform graph search in the existing convex region

(d) Find the first feasible global path

Figure 12: The process of generating the first global path by heuristic graph search

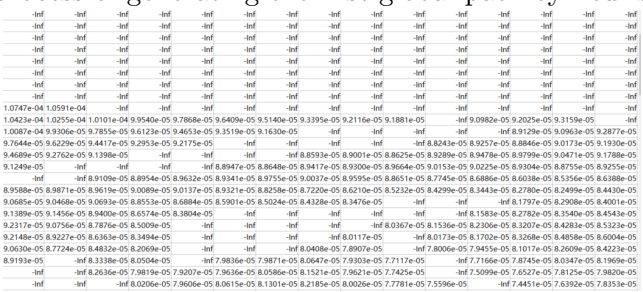
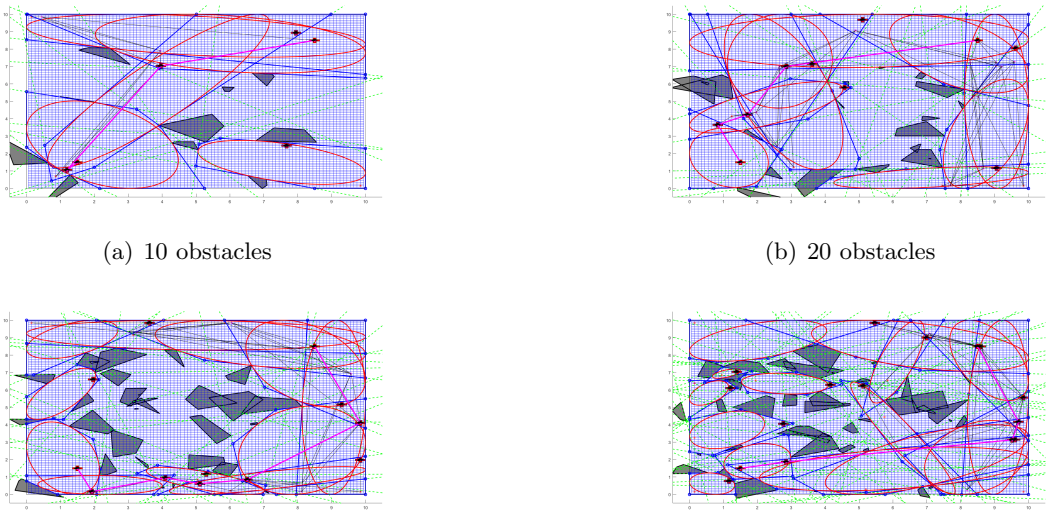


Figure 13: Evaluation matrix

of randomly generated obstacles. In the presence of global paths, the planner can explore a small number of convex regions to find the first feasible path, as shown in Fig. 14.

In addition, we compared the performance of five groups of random seeding and heuristic seeding in the same scenarios with the number of obstacles of 10, 20, and 30, respectively, using finding the first feasible path as the condition for iterative termination. We evaluate the metrics in path length and running time, as shown in Fig. 15, where each error bar is the standard deviation. The experimental results show that as the number of obstacles increases, the latter takes significantly less computation time than the former and can generate feasible paths more efficiently. However, for the path length, in some cases, blind sampling can even find a shorter path than heuristic sampling, which still requires



(a) 10 obstacles

(b) 20 obstacles

(c) 30 obstacles

(d) 40 obstacles

Figure 14: The first feasible path found by the global path planner

(a) Comparison of running times

(b) Comparison of path lengths

Figure 15: Comparison of blind search and heuristic search

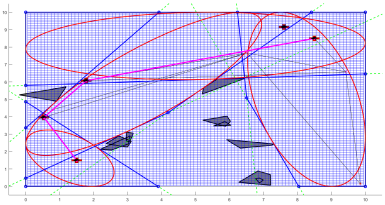
searching within a specific area of the convex region to find a shorter path.

We show in subsection 4.4.3 that our method not only finds the first feasible global trajectory quickly but also has the potential to find the globally optimal path. We validate our expectations using a heuristic graph search under different iterative conditions. We tested five groups on 10, 20, and 30 obstacle maps with the iteration conditions set to find the first feasible global trajectory. We searched 80% of the convex region of the global map, respectively. The experimental results are shown in Fig. 16, and it can be intuitively seen that we successfully found the globally optimal path. Similarly, we compared the path length and running time, as shown in Fig. 17, where the latter found a shorter path; however, in some cases, the difference in path length between the two was slight, yet it took 3 5 times longer to find a globally optimal path. To summarize, we have to make a trade-off between the two iterative conditions.

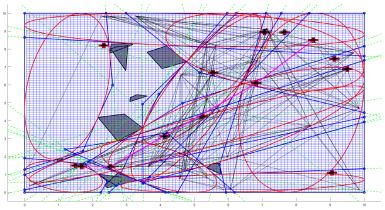
6.3 Trajectory Optimization

In this subsection, we test the performance of trajectory optimization with the introduction of bounding box constraints and trajectory errors, respectively, given feasible global paths generated by the global path planner. First, we apply Minimum Snap under the equation constraint to optimize the trajectory. We solve the quadratic programming equation using the quadprog function provided by MATLAB. The pink trajectory in Fig. 18 (a) is the trajectory before optimization, and the black trajectory in Fig. 18 (b) is the trajectory after smoothing. The experimental results show that the optimized trajectory is smoothed but re-encounters the obstacle.

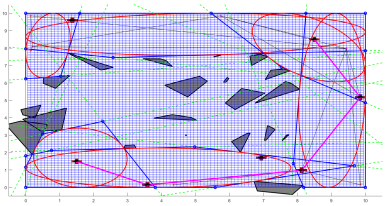
The optimization results after introducing the inequality constraints defined by rectangular bounding boxes are shown in Fig. 19, where the green points are the intermediate points interpolated in, the intermediate points violating the constraints are added with brown bounding boxes, and the black



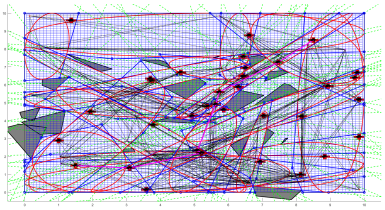
(a) The first feasible path found in the case of 10 obstacles



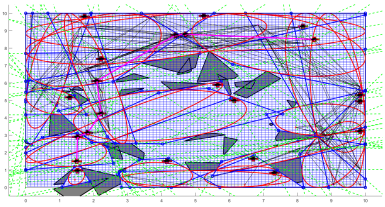
(b) The global optimal path found for the case of 10 obstacles



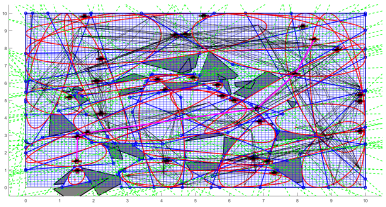
(c) The first feasible path found in the case of 20 obstacles



(d) The global optimal path found for the case of 20 obstacles



(e) The first feasible path found in the case of 30 obstacles



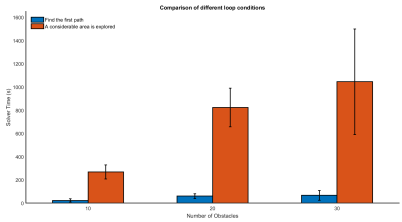
(f) The global optimal path found for the case of 30 obstacles

Figure 16: Global feasible paths found under different iteration conditions

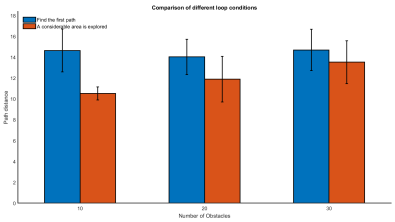
trajectories are the optimized trajectories. Our simulation results prove that only a finite number of interpolated points need to be constrained to generate safe and smooth trajectories, consistent with the results we expected in subsection 4.4.2.

Finally, we tested the effect of introducing the trajectory error term on the optimization results. First, we compared the trajectories generated under six weights λ. The results shown in Fig. 20 indicate that the smaller the λ, the smoother the trajectory, but the more it deviates from the original trajectory, and vice versa. After making the trade-off, we choose λ of 1000. Fig. 21 (a) shows the trajectory generated by the minimum snap method considering only the equation constraint, and Fig. 21 (b) shows the trajectory after introducing the trajectory error. The experimental results show that the introduction of the trajectory error makes the optimized trajectory fit the original trajectory more closely and meet our expectations.

Comparing our proposed method with the minimum snap method, as shown in Fig. 22, our approach fits the original trajectory more closely, and the trajectory error is smaller. In addition, we counted the additional computation time required for the three steps of initializing the bounding box,



(a) Comparison of running times



(b) Comparison of path lengths

Figure 17: Comparison of running time and path length under different iteration conditions

(a) Trajectories before smoothing

(b) The smoothed trajectory

Figure 18: Optimization results of Minimum Snap under equality constraints

(a) Trajectory optimization under equality constraints

(b) Trajectory optimization under equality and inequality constraints

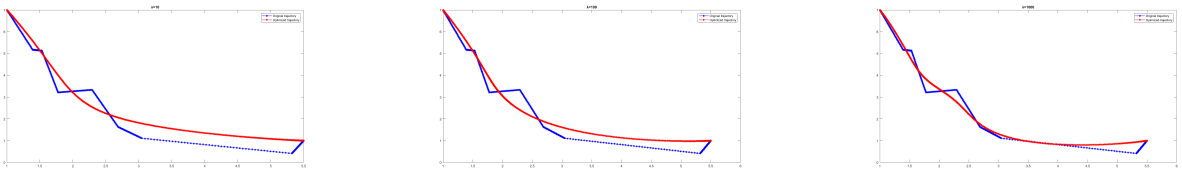
Figure 19: Comparison of trajectories before and after applying bounding box inequality constraints

generating the bounding box, and introducing the trajectory error term, respectively, as shown in blue, red, and yellow in the histogram of Fig. 23. We performed nine trajectory optimizations during the experiments; each optimization time was less than 0.5 s. The results indicate that our method is suitable for real-time path planning scenarios.

6.4 Task Allocation and Formation Generation

This subsection shows simulation results for multi-robot task allocation and formation generation. We test whether our algorithm can solve the local minima and motion conflict problems better than existing methods.

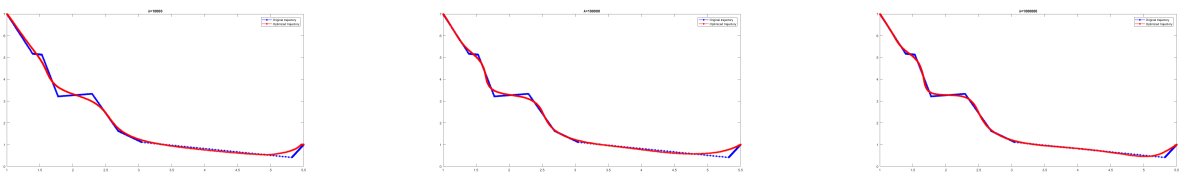
First, we are given ten robots with random initial poses and three objects in a 10×10 map consisting of several obstacles. We assign the robot to the position of the nearest target formation based on the idea of greed. We use the Euclidean distance of each robot to the target point as the cost matrix, as



(a) λ=10

(b) λ=100

(c) λ=1000



(d) λ=10000

(e) λ=100000

(f) λ=1000000

Figure 20: Optimized trajectories computed for different λ values

(a) Trajectory optimization under equality and inequality constraints

(b) The optimized trajectory after introducing the trajectory error

Figure 21: Comparison of optimized trajectories before and after introducing the error term

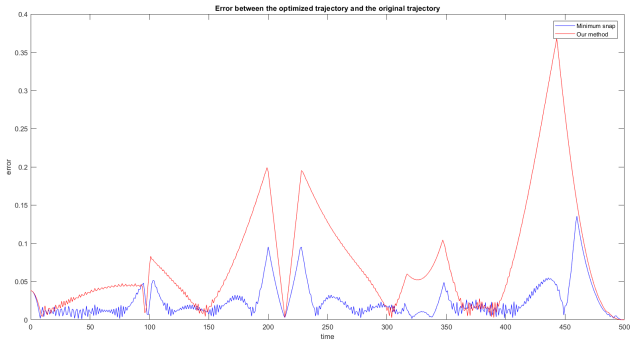


Figure 22: Trajectory error comparison between Minimum Snap and our method

shown in Fig. 24 (a). We apply the Hungarian algorithm to get the index of each robot that should reach the target point, and the task assignment results are shown in Fig. 24 (b).

To perform the task, we assign the ten robots into three formations consisting of three, three, and four robots. Subsequently, we execute the distributed control algorithm, as shown in Fig. 25 (a), one robot in the pink formation is trapped in a local minimum due to the target point blocking the side of the obstacle, resulting in its inability to reach the endpoint, and its position error curve is shown in Fig. 25 (b). After we apply a slight disturbance to the pink robot, as shown in Fig. 25 (c), the robot can gradually move away from the local minimum point by oscillating and finally successfully reach the endpoint. However, we can see from the position error curve (Fig. 25 (d)) and velocity curve (Fig. 25 (e)) that although the robot can converge to the target position, local oscillations occur.

We tested our proposed algorithm in the same scenario. As shown in Fig. 26 (a), when the robot determines that it is stuck in a local minimum and is likely to oscillate, it begins to move along the edge of the nearest obstacle until it is away from the local minimum. From Fig. 26 (b), we can see that the position error can converge to zero directly without oscillation, proving our algorithm's effectiveness.

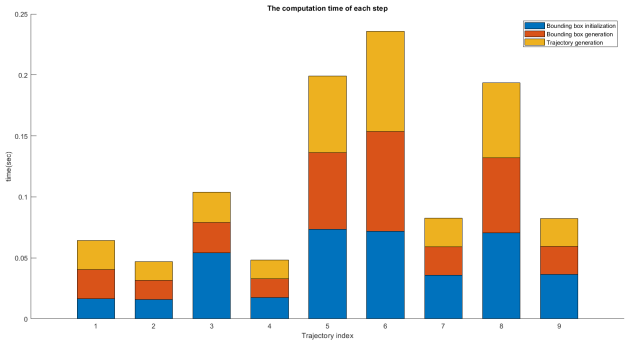


Figure 23: Additional computation time for our method

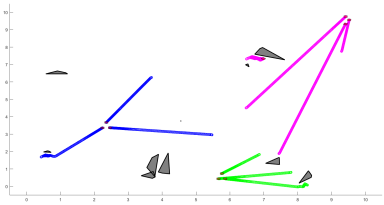
	1	2	3	4	5	6	7	8	9	10
1	7.7106	7.6809	7.5424	7.5727	7.2853	7.4269	7.3606	1.3594	1.2605	1.3558
2	4.4281	4.3517	4.2323	4.3108	3.4714	3.6017	3.5252	5.1016	4.9605	5.0290
3	5.9174	5.7996	5.7217	5.8410	4.1817	4.2363	4.1364	6.1933	6.0394	6.0718
4	2.2770	2.1536	2.2278	2.3474	1.3265	1.1919	1.2697	9.8703	9.7335	9.8063
5	11.4643	11.4521	11.3112	11.3235	11.2116	11.3554	11.2918	2.8586	3.0123	2.9769
6	2.3326	2.4160	2.3054	2.2178	3.7554	3.8785	3.9093	7.7192	7.6279	7.7236
7	5.7667	5.6773	5.5676	5.6588	4.5004	4.6055	4.5149	4.6111	4.4584	4.5034
8	7.4453	7.3993	7.2653	7.3122	6.7859	6.9198	6.8455	1.7976	1.6505	1.7122
9	3.4664	3.3769	3.2671	3.3595	2.4115	2.5449	2.4715	6.1395	6.0011	6.0724
10	9.4179	9.4178	9.2764	9.2765	9.3775	9.5269	9.4732	2.1998	2.3068	2.3517

(a) Cost matrix

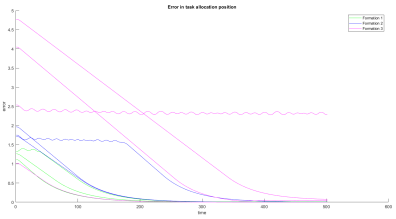
	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	1	0	0	0
2	0	0	0	0	0	0	0	0	0	1
3	0	0	0	0	0	0	0	1	0	0
4	0	0	0	0	1	0	0	0	0	0
5	0	0	0	0	0	0	0	0	1	0
6	0	0	0	0	0	1	0	0	0	0
7	0	0	0	1	0	0	0	0	0	0
8	0	0	1	0	0	0	0	0	0	0
9	0	1	0	0	0	0	0	0	0	0
10	1	0	0	0	0	0	0	0	0	0

(b) Task allocation results

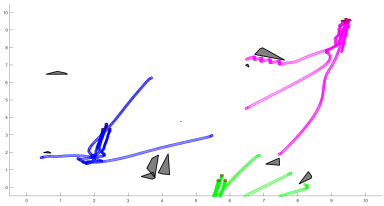
Figure 24: Results of task allocation by applying the Hungarian algorithm



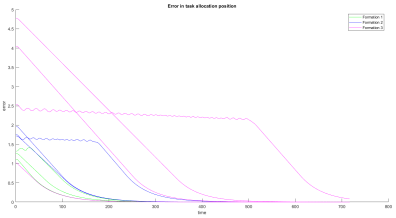
(a) The case of getting stuck in a local minimum



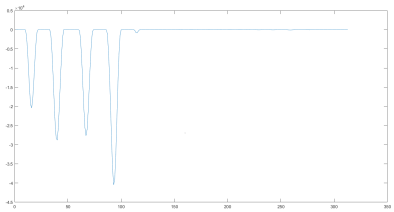
(b) Position error curves under local minima



(c) The result after adding the random perturbation



(d) Position error curve after adding the random perturbation



(e) The velocity curve after adding the random perturbation

Figure 25: Add random perturbations to solve the local optimum problem

In addition, we tested the algorithm's robustness in four different scenarios, and the formation generation process and position error curves are shown in Fig. 29. The experimental results show that all robots can converge to the desired formation position without local oscillations and motion conflicts.

6.5 Trajectory Tracking and Formation Holding

This section demonstrates experiments in which multiple robots track a set trajectory and maintain a formation. Considering our proposed formation control algorithm, we make the following assumptions:

- Robot formations are not allowed to change formations to avoid obstacles.
- All robots in each formation can sense the relative distance to the virtual leader.

We perform the simulation using a diamond-shaped formation of four robots on a 10×10 map consisting of 20 random obstacles, given a global trajectory computed by a path planner. Fig. 29 (a) shows the global smooth feasible trajectory planned by the global path planner for the robot formation. Fig. 29 (b) shows the tracking results based on the trajectory of the formation. The robot formation starts from the starting point and tracks the trajectory in a specific formation until it reaches the

(a) Escape from local minima and avoid oscillations

(b) The position error converges directly to zero

Figure 26: The robot escapes from the local minima under the requirement that no oscillations occur

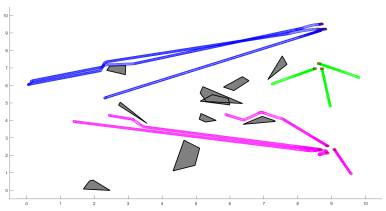
destination. The solid red line represents the leader's tracking trajectory, and the other colored lines represent the followers' tracking trajectories, which results from applying consensus control to the four control objects. The error of leader trajectory tracking is shown in Fig. 29 (c), and the tracking error is controlled within 0.05, which shows a good tracking effect. The formation keeping error of the follower is shown in Fig. 29 (d), and we control the error within 10^{-16}, which maintains the formation almost perfectly. In addition, the first-order consistency control solves the consistency problem of formation speed. As shown in Fig. 29 (e), the solid red line represents the speed of the virtual leader, and the solid black line represents the speed of the follower, and the latter can better maintain the speed convergence with that of the leader.

6.6 Local Path Planning

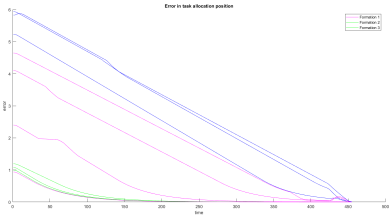
This section demonstrates simulation experiments for local path planning of robot formations. We tested the robustness of the motion planning algorithm in two scenarios. In a 10×10 map consisting of 20 randomly generated obstacles, we simulated two separate triangular formations of three robots. We used the global path planner beforehand to plan a smooth and feasible global trajectory, as shown in Fig. 29 (a), where the red rectangles represent the detected obstacles. Applying our motion planning algorithm, we synthetically tested the effectiveness of trajectory tracking, formation control, and real-time obstacle avoidance. When the leader detects an obstacle by implementing the dynamic window method, the robot formation successfully avoids the obstacle and then quickly returns to the trajectory tracking state. When the virtual leader avoids the obstacle, it can be seen from Fig. 29 (b) that the followers can maintain the formation well. Fig. 29 (c) shows the tracking velocity profile of the followers, which quickly converges to the same state as the leader after a sudden change when the obstacle is avoided. Fig. 29 (d) and Fig. 29 (e) show the formation position error and formation velocity error respectively. A similar example is shown in Fig. 30. In summary, our motion planning algorithm can solve the tasks of trajectory tracking, formation keeping, and dynamic obstacle avoidance well.

6.7 Multi-robot Collaboration

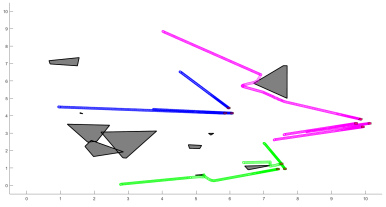
In this subsection, we demonstrate collaborative simulation experiments of multi-robot formations moving objects. We tested the performance of sequential trajectory planning in three robot formations consisting of three, three, and four robots, respectively, in a 10×10 map consisting of 20 randomly generated obstacles. The three robot formations are shown in 31 (a) are represented in pink, blue, and green, respectively. We assigned priority to the robot formations based on their distance from their respective target configurations, i.e., the green formation had the highest priority, the blue the second highest, and the pink the lowest. We first apply the global path planner and local path planner to plan feasible tracks for each group of robot formations, as shown in Fig. 31 (b). After that, we plan the motion of all robot formations by sequential trajectory planning in the temporal configuration space. The respective trajectories planned in this space are shown in Fig. 31 (c). At each time, we ensure that the robot formations are within a safe distance. The leader trajectory tracking position error, follower formation holding position error, and follower formation holding velocity error curves for each formation are shown in Fig. 31 (d), Fig. 31 (e), and Fig. 31 (f), respectively.



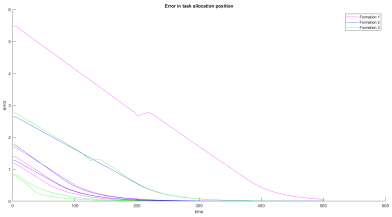
(a) Scenario 1



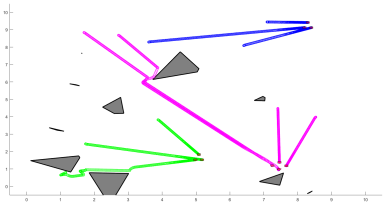
(b) Position error curve 1



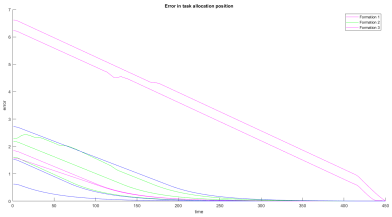
(c) Scenario 2



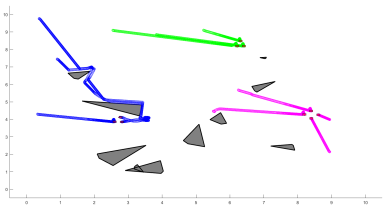
(d) Position error curve 2



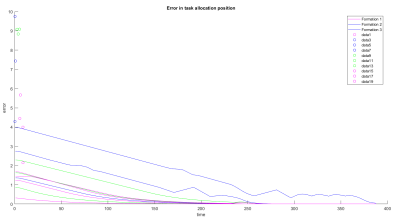
(e) Scenario 3



(f) Position error curve 3

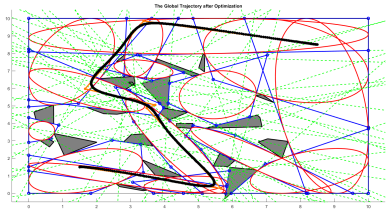


(g) Scenario 4

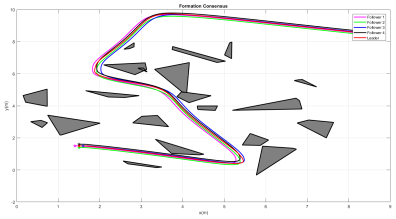


(h) Position error curve 4

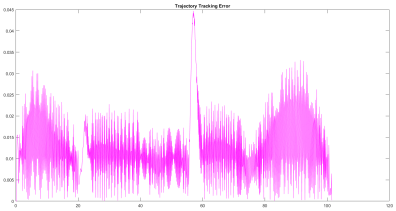
Figure 27: Formation generation results in different scenarios



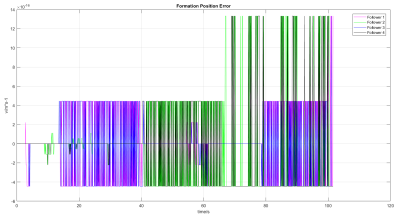
(a) The original smooth global trajectory



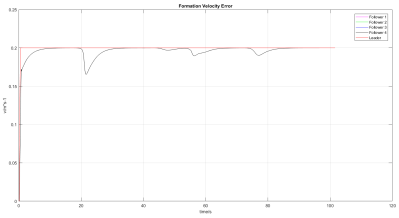
(b) Trajectory tracking result



(c) Position error of trajectory tracking (leader)

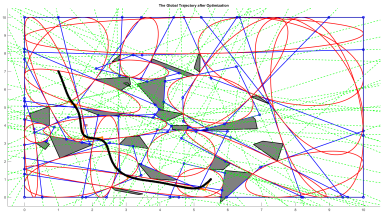


(d) Formation control position error (followers)

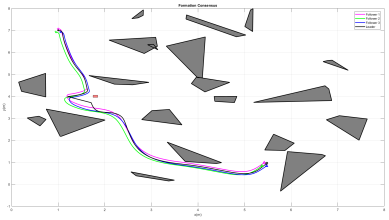


(e) Formation control speed error (followers)

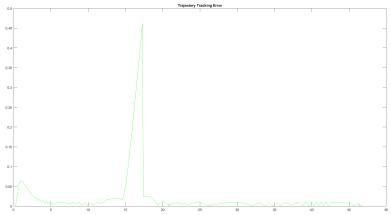
Figure 28: Experimental results of trajectory tracking and formation holding



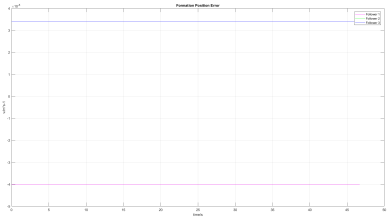
(a) The optimized global trajectory for formation



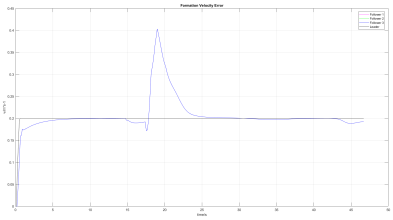
(b) Local trajectories avoid dynamic obstacles



(c) Position error of trajectory tracking (leader)

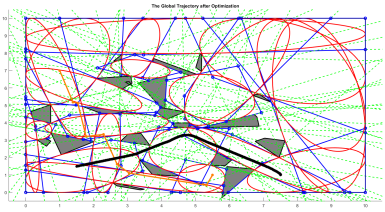


(d) Formation position error (followers)

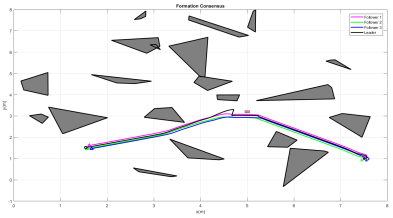


(e) Formation velocity error (followers)

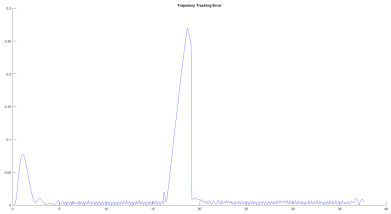
Figure 29: Experimental results of local path planning: Scenario 1



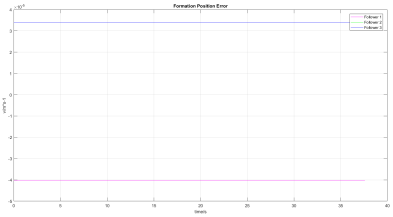
(a) The original smooth global trajectory



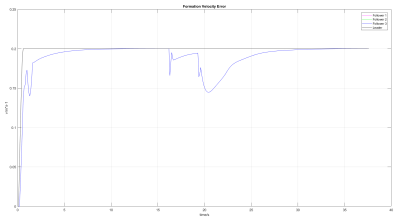
(b) Local trajectories avoid dynamic obstacles



(c) Position error of trajectory tracking (leader)

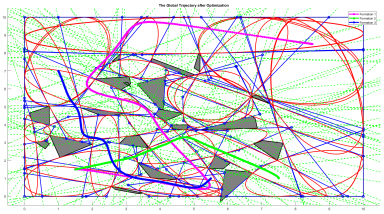


(d) Formation position error (followers)

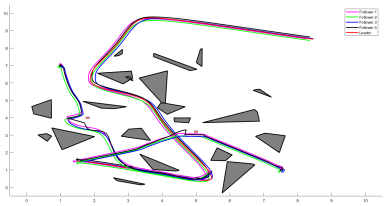


(e) Formation velocity error (followers)

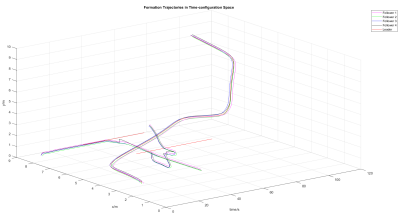
Figure 30: Experimental results of local path planning: Scenario 2



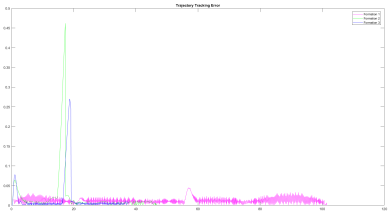
(a) The original smooth global trajectory



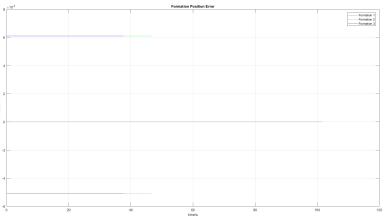
(b) Local trajectories avoid dynamic obstacles



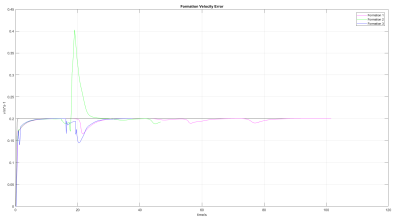
(c) Trajectories under time-configuration space



(d) Position error of trajectory tracking (leader)



(e) Formation position error (followers)



(f) Formation velocity error (followers)

Figure 31: Experimental results of sequential trajectory optimization

7 Discussion

Overall, the multi-robot scheduling system designed in this project can successfully implement multi-robot scheduling, formation generation, and formation maintenance to collaborative object handling in cluttered environments. The success of global trajectory generation in this project depends on the set of obstacle-idle regions. We require each convex region to be separated from all obstacles because all obstacles within a specific range are known. In a practical scenario, this would require a human operator to manually enter the vertices of all obstacles, which is tedious and impractical in a complex environment. The heuristic seeding method proposed in this project to speed up the computation of feasible configurations is described in Algorithm 3. We restrict the seeding points to outside the existing convex region and away from the obstacle. The reasons why the heuristic proposed in this project is more effective in most cases than random search and allows for faster generation of global path planning for robot formations are as follows:

- The sampling process is separated from checking whether the configuration is feasible.
- Each sampling point is guaranteed to generate a new convex region.

In the global path planning simulation experiments considering formation constraints (subsection 6.2), we were able to obtain feasible global paths for any formation consisting of any number of robots with high computational efficiency mainly based on the following reasons:

- We assume that all obstacles in the environment are convex, which ensures that our computation of the barrier-free space based on convex optimization can cover the entire space. In the non-convex case, there is not necessarily an optimal solution.
- We assume that the robot's dynamics are ignored, simplifying the control and constraint optimization problem.
- We use convex packets as formation constraints. The number of formation constraints we define is independent of the number of robots that make up the formation and is related to the number of vertices in the robot formation. For example, a triangular formation of several robots would introduce three formation constraints.

In addition, we can ensure that the generated trajectories do not intersect with the obstacles for the following reasons:

- The pre-existing convex region and the newly generated convex region are completely contained in the time-configuration space.
- Each trajectory segment of the formation before the optimization is a straight line, so the movement of the formation between two configurations is also contained within the convex region (the line connecting any two points within the convex polygon still lies within the convex polygon).

We require every trajectory segment to fall in the intersection region of two convex regions. However, this can lead to discarding segments that may not intersect the obstacle but fall within the intersection region of the other convex region, missing the narrow region and leading to global path planning failure. The so-called global optimum in this project is based on the premise that this restriction is not violated. Suppose we generate more feasible configurations, increase the number of trajectory segments, or assign each to a specific convex region. In that case, we can break this restriction to obtain a generalized global optimal path, but this will increase the complexity of convex optimization, which is contrary to the motivation of designing an efficient path planner in this project. We always have to trade between the optimal global solution and the fast feasible solution in path planning problems.
In the formation generation simulation experiments in subsection 6.4, the robots do not fall into local minima and are guaranteed not to collide with other robots. The advantage of the motion planning method proposed in this project is that it decouples the problem into:

- In the time-configuration space set the formation center to track a given global trajectory for the virtual leader.
- Consensus control is applied to the actual presence of robots to maintain a specific formation.

For the formation generation problem, We introduce consensus control to resolve motion conflicts, but it will not work in some extreme cases. For example, if the formation goal position is close to obstacles, our approach cannot guarantee the early convergence of the robots and may still lead to formation failure. Therefore, we need a dynamic prioritization strategy to solve the multi-robot formation generation problem and dynamically assign navigators according to the actual situation. In addition, we use first-order consistency control to maintain the formation and optimize the zero-order and first-order derivatives of the robot position. Still, in the trajectory optimization part, we constrain the fourth-order derivatives of the trajectories. Theoretically, we can implement higher-order consistency control. Still, in this project, the multi-robot handling process is carried out at low speed, and the velocity consistency is already able to ensure the smooth movement of the formation; our experimental results prove our assumptions are correct.

8 Conclusion

To conclude, this project has designed a relatively complete multi-robot cooperative control system in the context of object handling and has shown good robustness. Our proposed framework can introduce additional constraints and goals, such as the need for the robot to visit multiple mandatory waypoints in some cases, to visit target points in a specific order, etc. The global path planner proposed in this project combines a sampling-based approach and a nonlinear constraint on work area formation to compute global paths that consider formation constraints efficiently. Considering the real-time nature, one may also consider a more efficient and intelligent, but not optimal, algorithm to compute convex regions quickly, such as artificially giving suitable regions and significant points to find feasible convex regions more quickly. To cope with the clustered environment, we need a more efficient heuristic to select the sampling points used to generate the convex regions., such as selecting seeding points in the result of a sampling-based path planner (RRT star). Our approach can be extended for formation control to convex multi-robot formations of arbitrary size and an arbitrary number of robots due to the introduction of formation constraints. Since we minimize the function of the snap parametrization, the global trajectory can satisfy random position, velocity, and acceleration constraints. In addition, we use a new QP formulation and introduce a bounding box to minimize the tracking error, allowing the trajectory to slow down on the time scale to improve safety and robustness. Future research should include consideration of dynamical feasibility, i.e., introducing dynamical constraints on the robot in the optimization problem and predicting dynamic obstacle motion uncertainty in local path planning.

References

Abbas, M. A. (2011), Non-linear model predictive control for autonomous vehicles, PhD thesis.

Agarwal, P., Kumar, S., Ryde, J., Corso, J. & Krovi (2013), *Towards A Swarm of Agile Micro Quadrotors*, pp. 217–224.

Akin, H. L., Amato, N. M., Isler, V. & Van der Stappen, A. F. (2015), *Algorithmic Foundations of Robotics XI: Selected Contributions of the Eleventh International Workshop on the Algorithmic Foundations of Robotics*, Vol. 107, Springer.

Alonso-Mora, J., Baker, S. & Rus, D. (2017), 'Multi-robot formation control and object transport in dynamic environments via constrained optimization', *The International Journal of Robotics Research* **36**(9), 1000–1021.

Alonso-Mora, J., Montijano, E., Schwager, M. & Rus, D. (2016), Distributed multi-robot formation control among obstacles: A geometric and optimization approach with consensus, *in* '2016 IEEE international conference on robotics and automation (ICRA)', IEEE, pp. 5356–5363.

ApS, M. (2019), 'Mosek optimization toolbox for matlab', *User's Guide and Reference Manual, Version* **4**.

Arai, T. & Ota, J. (1992), Motion pianning of multiple mobile robots, *in* 'Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems', Vol. 3, IEEE, pp. 1761–1768.

Ayanian, N., Kallem, V. & Kumar, V. (2011), Synthesis of feedback controllers for multiple aerial robots with geometric constraints, *in* '2011 IEEE/RSJ International Conference on Intelligent Robots and Systems', IEEE, pp. 3126–3131.

Barfoot, T. D. & Clark, C. M. (2004), 'Motion planning for formations of mobile robots', *Robotics and Autonomous Systems* **46**(2), 65–78.

Barraquand, J., Langlois, B. & Latombe, J.-C. (1992), 'Numerical potential field techniques for robot path planning', *IEEE transactions on systems, man, and cybernetics* **22**(2), 224–241.

Bien, Z. & Lee, J. (1992), 'A minimum-time trajectory planning method for two robots', *IEEE Transactions on Robotics and Automation* **8**(3), 414–418.

Broggi, A., Bertozzi, M., Fascioli, A., Bianco, C. G. L. & Piazzi, A. (2000), 'Visual perception of obstacles and vehicles for platooning', *IEEE Transactions on Intelligent Transportation Systems* **1**(3), 164–176.

Buckley, S. J. (1988), *Fast motion planning for multiple moving robots*, IBM Thomas J. Watson Research Division.

Cazzato, D., Cimarelli, C., Sanchez-Lopez, J. L., Voos, H. & Leo, M. (2020), 'A survey of computer vision methods for 2d object detection from unmanned aerial vehicles', *Journal of Imaging* **6**(8), 78.

Chen, J., Liu, T. & Shen, S. (2016), Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments, *in* '2016 IEEE international conference on robotics and automation (ICRA)', IEEE, pp. 1476–1483.

Chen, Y. Q. & Wang, Z. (2005), Formation control: a review and a new consideration, *in* '2005 IEEE/RSJ International conference on intelligent robots and systems', IEEE, pp. 3181–3186.

Coulter, R. C. (1992), Implementation of the pure pursuit path tracking algorithm, Technical report, Carnegie-Mellon UNIV Pittsburgh PA Robotics INST.

Cowan, N., Shakerina, O., Vidal, R. & Sastry, S. (2003), Vision-based follow-the-leader, *in* 'Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)', Vol. 2, IEEE, pp. 1796–1801.

Demyen, D. & Buro, M. (2006), Efficient triangulation-based pathfinding, *in* 'Aaai', Vol. 6, pp. 942–947.

Desai, J. P., Ostrowski, J. P. & Kumar, V. (2001), 'Modeling and control of formations of nonholonomic mobile robots', *IEEE transactions on Robotics and Automation* **17**(6), 905–908.

Erdmann, M. & Lozano-Perez, T. (1987), 'On multiple moving objects', *Algorithmica* **2**(1), 477–521.

Feddema, J. T., Lewis, C. & Schoenwald, D. A. (2002), 'Decentralized control of cooperative robotic vehicles: theory and application', *IEEE Transactions on robotics and automation* **18**(5), 852–864.

Ferrari, C., Pagello, E., Ota, J. & Arai, T. (1998), 'Multirobot motion coordination in space and time', *Robotics and autonomous systems* **25**(3-4), 219–229.

Fischer, P. (1993), Finding maximum convex polygons, *in* 'International Symposium on Fundamentals of Computation Theory', Springer, pp. 234–243.

Ge, S. S. & Cui, Y. J. (2002), 'Dynamic motion planning for mobile robots using potential field method', *Autonomous robots* **13**(3), 207–222.

Ge, S. S., Fua, C.-H. & Lim, K. W. (2004), Multi-robot formations: queues and artificial potential trenches, *in* 'IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004', Vol. 4, IEEE, pp. 3345–3350.

Guattery, S. & Miller, G. L. (2000), 'Graph embeddings and laplacian eigenvalues', *SIAM Journal on Matrix Analysis and Applications* **21**(3), 703–723.

Hanli, Y., Qiliang, L. & Weizhen, Z. (2021), Robot obstacle avoidance based on improved artificial potential field method, *in* '2021 International Conference on Control, Automation and Information Sciences (ICCAIS)', IEEE, pp. 769–775.

Harinarayan, K. & Lumelsky, V. J. (1994), Sensor-based motion planning for multiple mobile robots in an uncertain environment, *in* 'Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'94)', Vol. 3, IEEE, pp. 1485–1492.

Hart, P. E., Nilsson, N. J. & Raphael, B. (1968), 'A formal basis for the heuristic determination of minimum cost paths', *IEEE transactions on Systems Science and Cybernetics* **4**(2), 100–107.

He, S., Li, Z. & Xiao, D. (2003), A research on strategy of bus priority, *in* 'Proceedings of the 2003 IEEE International Conference on Intelligent Transportation Systems', Vol. 2, IEEE, pp. 1324–1328.

Herrero-Perez, D. & Matinez-Barbera, H. (2008), Decentralized coordination of autonomous agvs in flexible manufacturing systems, *in* '2008 IEEE/RSJ International Conference on Intelligent Robots and Systems', IEEE, pp. 3674–3679.

Ho, Y.-J. & Liu, J.-S. (2009), Collision-free curvature-bounded smooth path planning using composite bezier curve based on voronoi diagram, *in* '2009 IEEE international symposium on computational intelligence in robotics and automation-(CIRA)', IEEE, pp. 463–468.

Hopcroft, J. E., Schwartz, J. T. & Sharir, M. (1984), 'On the complexity of motion planning for multiple independent objects; pspace-hardness of the" warehouseman's problem"', *The International Journal of Robotics Research* **3**(4), 76–88.

Khamis, A., Hussein, A. & Elmogy, A. (2015), 'Multi-robot task allocation: A review of the state-of-the-art', *Cooperative robots and sensor networks 2015* pp. 31–51.

Krontiris, A., Louis, S. & Bekris, K. E. (2012), Multi-level formation roadmaps for collision-free dynamic shape changes with non-holonomic teams, *in* '2012 IEEE International Conference on Robotics and Automation', IEEE, pp. 1570–1575.

Kuhn, H. W. (1955), 'The hungarian method for the assignment problem', *Naval research logistics quarterly* **2**(1-2), 83–97.

LaValle, S. M. (2006), *Planning algorithms*, Cambridge university press.

Lawton, J. R., Beard, R. W. & Young, B. J. (2003), 'A decentralized approach to formation maneuvers', *IEEE transactions on robotics and automation* **19**(6), 933–941.

Lee, D.-H., Zaheer, S. A. & Kim, J.-H. (2014), 'A resource-oriented, decentralized auction algorithm for multirobot task allocation', *IEEE Transactions on Automation Science and Engineering* **12**(4), 1469–1481.

Leonard, N. E. & Fiorelli, E. (2001), Virtual leaders, artificial potentials and coordinated control of groups, *in* 'Proceedings of the 40th IEEE Conference on Decision and Control (Cat. No. 01CH37228)', Vol. 3, IEEE, pp. 2968–2973.

Lien, J.-M. & Amato, N. M. (2004), Approximate convex decomposition, *in* 'Proceedings of the twentieth annual symposium on Computational geometry', pp. 457–458.

Liu, H., Liu, W. & Latecki, L. J. (2010), Convex shape decomposition, *in* '2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition', IEEE, pp. 97–104.

Mellinger, D. & Kumar, V. (2011), Minimum snap trajectory generation and control for quadrotors, *in* '2011 IEEE international conference on robotics and automation', IEEE, pp. 2520–2525.

Neto, A. A., Macharet, D. G. & Campos, M. F. (2010), Feasible rrt-based path planning using seventh order bézier curves, *in* '2010 IEEE/RSJ International Conference on Intelligent Robots and Systems', IEEE, pp. 1445–1450.

O'Donnell, P. A. & Lozano-Pérez, T. (1989), Deadlock-free and collision-free coordination of two robot manipulators, *in* '1989 IEEE International Conference on Robotics and Automation', IEEE Computer Society, pp. 484–489.

Olfati-Saber, R. & Murray, R. M. (2002), 'Distributed cooperative control of multiple vehicle formations using structural potential functions', *IFAC Proceedings Volumes* **35**(1), 495–500.

Ollero, A., García-Cerezo, A. & Martinez, J. (1994), 'Fuzzy supervisory path tracking of mobile reports', *Control Engineering Practice* **2**(2), 313–319.

Ollero, A. & Heredia, G. (1995), Stability analysis of mobile robot path tracking, *in* 'Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots', Vol. 3, IEEE, pp. 461–466.

Omidshafiei, S., Pazis, J., Amato, C., How, J. P. & Vian, J. (2017), Deep decentralized multi-task multi-agent reinforcement learning under partial observability, *in* 'International Conference on Machine Learning', PMLR, pp. 2681–2690.

Peng, J. & Akella, S. (2005), 'Coordinating multiple robots with kinodynamic constraints along specified paths', *The International Journal of Robotics Research* **24**(4), 295–310.

Ren, W. & Beard, R. W. (2004*a*), 'Decentralized scheme for spacecraft formation flying via the virtual structure approach', *Journal of Guidance, Control, and Dynamics* **27**(1), 73–82.

Ren, W. & Beard, R. W. (2004*b*), 'Formation feedback control for multiple spacecraft via virtual structures', *IEE Proceedings-Control Theory and Applications* **151**(3), 357–368.

Saha, I., Ramaithitima, R., Kumar, V., Pappas, G. J. & Seshia, S. A. (2014), Automated composition of motion primitives for multi-robot systems from safe ltl specifications, *in* '2014 IEEE/RSJ International Conference on Intelligent Robots and Systems', pp. 1525–1532.

Sarmientoy, A., Murrieta-Cidz, R. & Hutchinsony, S. (2005), A sample-based convex cover for rapidly finding an object in a 3-d environment, *in* 'Proceedings of the 2005 IEEE International Conference on Robotics and Automation', IEEE, pp. 3486–3491.

Scharf, D. P., Hadaegh, F. Y. & Ploen, S. R. (2004), A survey of spacecraft formation flying guidance and control. part ii: control, *in* 'Proceedings of the 2004 American control conference', Vol. 4, Ieee, pp. 2976–2985.

Simba, K. R., Uchiyama, N. & Sano, S. (2016), 'Real-time smooth trajectory generation for non-holonomic mobile robots using bézier curves', *Robotics and Computer-Integrated Manufacturing* **41**, 31–42.

Stentz, A., Dima, C., Wellington, C., Herman, H. & Stager, D. (2002), 'A system for semi-autonomous tractor operations', *Autonomous Robots* **13**(1), 87–104.

Sun, D., Wang, C., Shang, W. & Feng, G. (2009), 'A synchronization approach to multiple mobile robots in switching between formations', *IEEE Trans. Robot.* **25**(5), 1074–1086.

Tan, K.-H. & Lewis, M. (1997), Virtual structures for high precision cooperative control, Technical report, Technical Report.

Tournassoud, P. (1986), A strategy for obstacle avoidance and its application to mullti-robot systems, *in* 'Proceedings. 1986 IEEE International Conference on Robotics and Automation', Vol. 3, IEEE, pp. 1224–1229.

Urmson, C., Ragusa, C., Ray, D., Anhalt, J., Bartz, D., Galatali, T., Gutierrez, A., Johnston, J., Harbaugh, S., "Yu" Kato, H. et al. (2006), 'A robust approach to high-speed navigation for unrehearsed desert terrain', *Journal of Field Robotics* **23**(8), 467–508.

Van Den Berg, J. P. & Overmars, M. H. (2005), Prioritized motion planning for multiple robots, *in* '2005 IEEE/RSJ International Conference on Intelligent Robots and Systems', IEEE, pp. 430–435.

Vidal, R., Shakernia, O. & Sastry, S. (2003), Formation control of nonholonomic mobile robots with omnidirectional visual servoing and motion segmentation, *in* '2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)', Vol. 1, IEEE, pp. 584–589.

Wang, S., Chen, L., Hu, H. & McDonald-Maier, K. (2012), Doorway passing of an intelligent wheelchair by dynamically generating bezier curve trajectory, *in* '2012 IEEE International Conference on Robotics and Biomimetics (ROBIO)', IEEE, pp. 1206–1211.

Wit, J., Crane III, C. D. & Armstrong, D. (2004), 'Autonomous ground vehicle path tracking', *Journal of Robotic Systems* **21**(8), 439–449.

Yamashita, A., Arai, T., Ota, J. & Asama, H. (2003), 'Motion planning of multiple mobile robots for cooperative manipulation and transportation', *IEEE Transactions on Robotics and Automation* **19**(2), 223–237.

Yao, J., Xiao, Y., You, P. & Sun, G. (2022), 'The international conference on image, vision and intelligent systems (icivis 2021)'.

Zhao, P., Chen, J., Song, Y., Tao, X., Xu, T. & Mei, T. (2012), 'Design of a control system for an autonomous vehicle based on adaptive-pid', *International Journal of Advanced Robotic Systems* **9**(2), 44.

Zhu, X., Sim, K. M., Jiang, J., Wang, J., Chen, C. & Liu, Z. (2014), 'Agent-based dynamic scheduling for earth-observing tasks on multiple airships in emergency', *IEEE Systems Journal* **10**(2), 661–672.

A Appendix

The repository has a total of 59 MATLAB files, including 16 mat files and 43 m files. The running methods and running dependencies are detailed in the README.md documentation.

A.1 GitLab Repository

Access the GitLab repository using the below link:
https://git-teaching.cs.bham.ac.uk/mod-msc-proj-2021/wxz163

A.2 File Structure

The structure of the repository is shown in the following figure:

```
|  Project_Presentations.m                  <- Demonstrates all the functions that this project integrates.
|  README.md                                <- Instructions for running the code.
|  startup.m                                <- Add the required toolkit to the work path.
|
├─data
|      demo.mat                             <- All data files generated by the demo saved in .mat format.
|      formation_motion.mat                 <- Data files generated by formation control.
|      initialization.mat                   <- All the initialized data used for visualization..
|      obs.mat                              <- Data of generated obstacles.
|      origtraj.mat                         <- The unoptimized global path of the formation.
|      smtraj.mat                           <- The optimized global path of the formation.
|      task_allocation.mat                  <- The result of the task allocation.
|      task_allocation_motion.mat           <- Motion planning results for task allocation.
|
├─sub_function_draw
|      drawbar.m                            <- Plot bar graphs to analyze the performance of global path planning.
|      drawformation.m                      <- Draw the formation according to the shape of the formation.
|      drawformation_control.m              <- Visualization of robot formation keeping.
|      drawformation_control_ta.m           <- Plot the trajectory of each robot during robot task allocation.
|      draw_conv_region.m                   <- Draw the resulting convex obstacle-free regions.
|      draw_diag.m                          <- Plot the analysis results of the path planning simulation.
|      draw_obs.m                           <- Draw the generated obstacle.
|      draw_path.m                          <- Plot the trajectory of each robot during the robot formation hold.
|      draw_path_ta.m                       <- Plot the trajectory of each robot during robot task allocation.
|      draw_smooth.m                        <- Plot the optimized global trajectory.
|
├─sub_function_gp
|      c2v.m                                <- Calculate the vertex coordinates of the formation based on the formation center.
|      Computing_LargeConvexRegions.m       <- Calculate the convex obstacle-free regions.
|      createobstcell.m                     <- Generate the vertices and edges of the obstacle.
|      formation.m                          <- Determine if a feasible configuration exists and return the best configuration.
|      generate_obstacle.m                  <- Randomly generated obstacles.
|      GeoProperties.m                      <- Generate specific robot formations based on the geometric parameters of the formation.
|      Global_planning.m                    <- Heuristic global path planner.
|      Global_planning_orig.m               <- A global path planner based on random search.
|      graphsearch.m                        <- Perform a graph search in the generated graph to return nodes and edges.
|      initial_guess.m                      <- Calculate the obstacle distance matrix for given obstacles.
|      l2T.m                                <- Generate relative positions between robots based on formation shapes.
|      lcon2vert.m                          <- Computes the vertices of a polygon defined by linear equations and linear inequality constraints.
|      r2m.m                                <- Given the initial poses of the robot formation, the coordinates of each vertex of the robot formation.
|      shortestpath.m                       <- Given a number of nodes and graphs, the shortest path is calculated based on the A* algorithm.
|      vert2lcon.m                          <- Computes linear constraints for multiple deformations at a given vertex.
|
├─sub_function_mc
|      apf_con.m                            <- Combined with improved artificial potential field method with consistency control.
|      compute_repulsion.m                  <- Calculation of repulsive forces in the artificial potential field method.
|      DWA_con.m                            <- Combining dynamic windowing method and consensus control for robot formation.
|      Local_Planning.m                     <- Local path planning for robot formations.
|      param_lim.m                          <- Limit the speed and acceleration of the robot.
|      pursuit_con.m                        <- Combination of pure tracking control and consistency control.
|
├─sub_function_ta
|      generate_task.m                      <- Generate a number of random tasks for multiple robots.
|      TaskAllocation.m                     <- Robot task scheduling, assigning the robot's index to each target location.
|
├─sub_function_ts
|      cal_optimized_traj.m                 <- Calculate the polynomial coefficients of the optimized trajectory.
|      cal_poly_ind.m                       <- Computes the index of the polynomial coefficients.
|      cal_Q_Matrix.m                       <- Calculate the Q matrix in Minimum Snap.
|      cal_r.m                              <- Calculate the size of the bounding box according to the inequality constraint of the convex region.
|      cal_time_stamp.m                     <- Calculate the time interval.
|      cal_traj_val.m                       <- Calculates the value of the midpoint of the trajectory.
|      cal_t_vector.m                       <- Calculate the time vector.
|      minimum_snap_box.m                   <- Improved version of Minimum Snap to generate safe and smooth trajectories.
|
└─test
       Computing_LargeConvexRegions_test.m  <- Test and display the computation process of the convex obstacle-free regions.
       global_planning_test.m               <- Test the global path planner and display the global optimal trajectory after smoothing.
       motion_planning_fh_test.m            <- Test and display the process of formation holding.
       motion_planning_ta_test.m            <- Test and display the process of task scheduling and queue formation.
```

Figure 32: The repository file structure

A.3 Code Sources

lcon2vert.m and vert2lcon.m use the resources from Matt J's Analyze N-Dimensional Convex Polyhedra (available at: https://www.mathworks.com/matlabcentral/fileexchange/30892-analyze-n-dimensional-convex-polyhedra). The two files mentioned above are in the sub_function_gp folder in the git repository. The code in Computing_LargeConvexRegions.m is written based on the method proposed by (Akin et al. 2015) for solving convex regions based on SemideFinite Programming. In cal_optimized_traj.m, an improved version of Minimum Snap, the code is my original work, but inspired by (Chen et al. 2016) and (Cazzato et al. 2020). All other files and code are my original work.

A.4 Running the Code

Running the code requires MATLAB2017b or later and relies on the MOSEK Toolbox and Robotics Toolbox. A quick installation of the required Toolbox can be found in the README documentation. Add the subfunction and data files to the working path before running the test code: right-click data, sub_function_draw, sub_function_gp,sub_function_mc, sub_function_ta,sub_function_ts,test folder and select "Add to Path" and then select "Selected" Folders and Subfolders ". The testing part of the project is divided into the following tasks:

1. The first task selects a valid point in the map of randomly generated obstacles and generates the corresponding convex obstruction-free region. The steps to run the generation process are as follows:
   (a) Open Computing_LargeConvexRegions.m in the test folder.
   (b) Run the file to show the process of convex region generation and output the hyperparameters of the separated hyperplane in the command window.
2. The second task is global path planning and trajectory optimization. Follow these steps to run the code:
   (a) Open the global_planning_test.m file.
   (b) Modify different iteration conditions (comment out or uncomment ines 1 and line 2) to visualize the first feasible trajectory or globally optimal trajectory, and the corresponding optimized trajectory. (This may take some time to complete, depending on the performance of the computer. It takes several minutes to compute the global optimal trajectory.)
3. The third task is task allocation for multiple robots. Follow these steps to run the code:
   (a) Open the motion_plannning_ta_test.m file.
   (b) Run file showing animation of task allocation and formation process for multiple robots.
4. The last task is cooperative object handling and formation holding. Follow the following steps to run the code:
   (a) Open the motion_planning_fh_test.m file.
   (b) Running file display animation: Multiple robots maintain formation and follow a global path to move objects.

Directly run Project_Presentation.m file multi-robot task scheduling, 3 robots form formation, keep formation, and work together.
More information about the running code can be found in the README documentation.