Automatic
Control &
Systems
Engineering.

Path Planning Using Mixed Integer Linear Programming

Hong Lu

September, 2021

Supervisor: Dr. Paul Trodden

A dissertation submitted in partial fulfilment of the requirements for the degree of MSc in Robotics.

ABSTRACT

A comprehensive study on the Mixed Integer Linear Programming (MILP) path planning is carried out under the mathematical programming framework. Various control architectures for different purposes are studied, implemented and compared in this paper. The concept of the MILP is introduced at the beginning, then its combination with the path planning is illustrated in detail along with its applications under separated scenarios considering various requirements, such as collision avoidance between the agents, obstacle avoidance and waypoint handling when multiple tasks existed. Moreover, iterative algorithm is also studied from two different perspectives in this paper which significantly reduce the computation time. Apart from global planning, the control architecture with the combination of the local planning scheme is also studied under the framework of Receding Horizon Control. The different resolutions of the MILP-RHC enables the vehicle not only to navigate through the environment safely and without falling into the entrapment using the high-level map information, but also to merely consider the low-level sensed environment. Such online fashioned planning relieves the computational burden significantly. In the last section, the extended capabilities of the MILP path planner is also studied both in finer trajectory generation for Quadrotor UAV and in 3D climber as well. Extensive simulations and results are shown in each section for the MILP path planning studies.

ACKNOWLEDGEMENTS

I hereby want to sincerely thank several people who made this come to true during this pandemic period. First, my advisor Dr. Paul Trodden introduced me into this topic and also into some insightful papers which appeared to be the critical starter for this whole project. The efficient guide provided by Dr. Paul Trodden also helped a lot to this project. I would also want to thank my parents Mr. Lu, Jinhua and Ms. Xu, Qunsuo, along with Miss Tian, Rui, for their unconditional support and care during this period.

Contents

Chapter 1

Introduction

1.1 Background and Motivation

The industries all over the world is undergoing intelligent transformation, through manufacturing to transportation, many automatic toolboxes have been applied to the applications along with the emergence of the artificial intelligence. Among various domains, the appearance of the autonomous mobile robots is the most outstanding one. Starting from the DARPA Urban Challenge in 2003, with the foundations of different autonomous vehicle companies, like Waymo, Tesla, Pony.ai and Baidu, etc., the demands of the relative techniques in image and video processing, decision and planning, mobile computing increased to a great extent. In general, according to [38], the main components of the autonomous vehicle software system includes perception, planning and control. Moreover, for multiple vehicles, the communication and coordination techniques are also need considering.

This project mainly concentrates on the component of the planning for the autonomous vehicles where many techniques emerged in the last decades. Planning algorithms firstly appeared in the graph theory as 'routing' problem, many classic algorithms such as Dijkstra's, A* and D* [11, 20, 49] can be applied for finding the shorted paths between start and terminal points. After decades of the research, the more specific domains named as motion planning is proposed. Since spatial planning is introduced in [33], the planning problem is introduced with the obstacles and generalized as finding a collision free path from the start to the terminal point in the configuration space. More interestingly, in [6], the complexity of the motion planning is proposed and its hardness

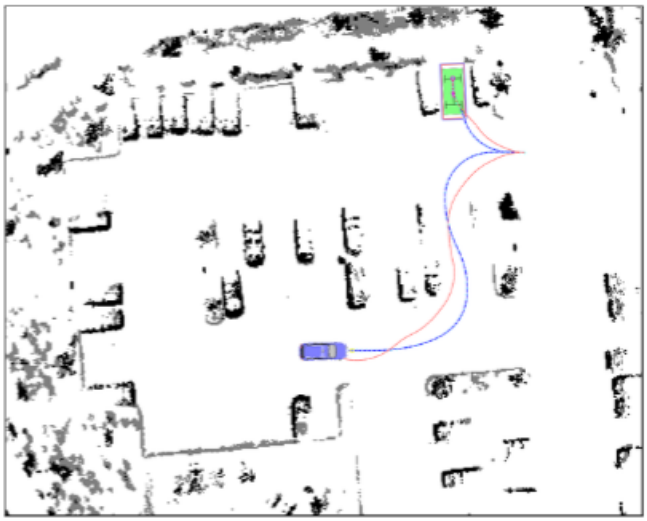


Figure 1.1: Practical techniques for car parking in [12]

is proved by Canny. Currently, many algorithms are based on the classical thoughts including Road Map, Cell Decomposition, Artificial Potential Field and Mathematical Programming. Technically, these methods are not mutually exclusive, many work has combined them into the more sophisticated planner. Many developments [2] have been made using mathematical programming in the motion planning and this motivates the deeper look into this topic.

1.2 Aims and Objectives

As illustrated in Sec. 1.1, the methods for the motion planning are varied. This project aims to design and implement a path planner for the robot(s) using Mixed Integer Linear Programming (MILP), which belongs to the mathematical programming, with various avoidance constraints for single or multiple tasks appointment. The main objectives for reaching such aim are as follows:

- To model the motion control problems of the vehicle, including kinematics and motion constraints, into MILP model.
- To develop various optimization objectives for single and multiple vehicle(s) in various tasks assignments.
- To improve the efficiency and robustness of the simple MILP path planner. Iterative MILP planner is studied for the computation efficiency and the uncertainties

would also be incorporated into the model.

- Demonstrate and validate through MATLAB or ROS based simulation.

1.3 Report Overview

The organization of the following chapters are as follows: Chapter. 2 provides the literature review in the mathematical programming applied in the motion planning domain and emphasize its advantages and disadvantages from various aspects along with the analysis of the methods this report adopted.

Chapter. 3 studied the application of the MILP to the path planing problems where the vehicle dynamics and environment are modelling using MILP. The relative objective functions are proposed and studied. Also, the refinement iterative MILP algorithm is also studied from two different perspectives for relieving the burden of computing.

Chapter. 4 studied the MILP-RHC (Mixed Integer Linear Programming - Receding Horizon control) architecture where the local constrained optimal path is planned using MILP within the framework of RHC. The RHC technique can better handle the uncertainty while the global information is unknown. To improve the robustness, high-level abstracted global information is also added to the planner, thus the controller with various resolutions are studied in this section as well.

Chapter. 5 studied the extended capabilities of the MILP path planner. The MILP path planning is adopted as the component of the quadrotor trajectory generation for finer trajectory generation in this section. More interestingly, a three dimensional application as the city climber is also simulated in this section with the help of the MILP path planner.

Chapter. 6 concludes the work in this project and the future work is proposed as well.

1.4 Project Management

This section provides the summary of the project management where the original gantt chart is shown in Fig. 1.2. The revised gantt chart which is actually conducted during the project is shown in Fig. 1.3.

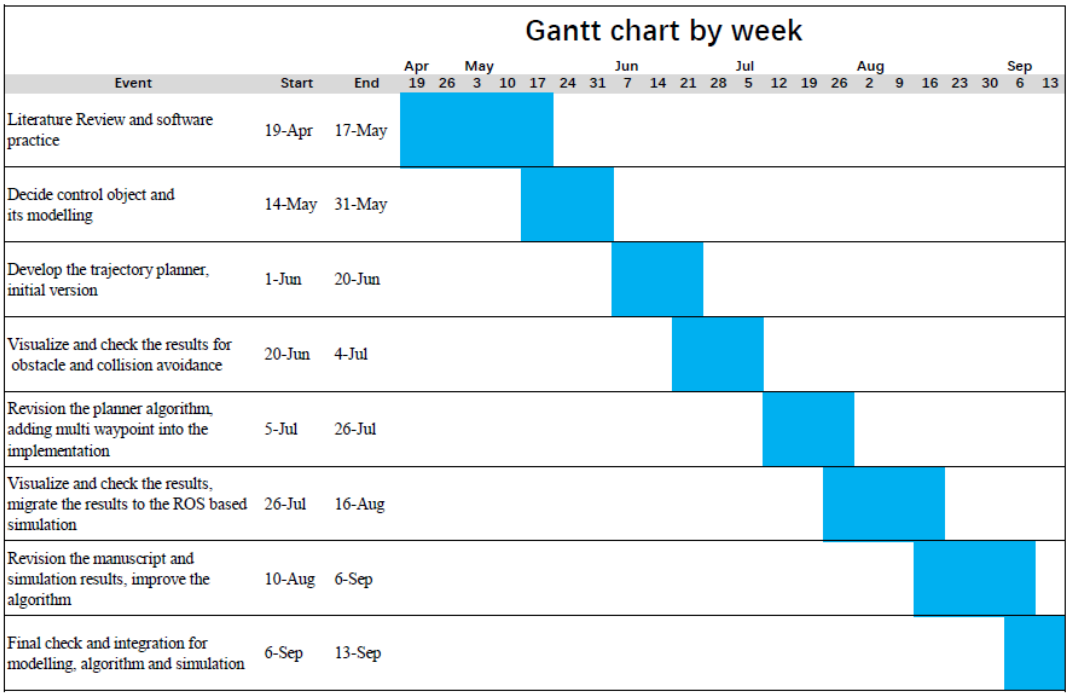


Figure 1.2: Planned Gantt Chart at the start of the project

The actual execution of the project amends more literature review time after receiving the feedback from the supervisor. The start date of the manuscript started earlier than the planned date which gave the abundant time for the revision of the layout and the content of the project. The project was carried out according to the planning gantt chart strictly which reflects that the original consideration for the workload is reasonable and considerate.

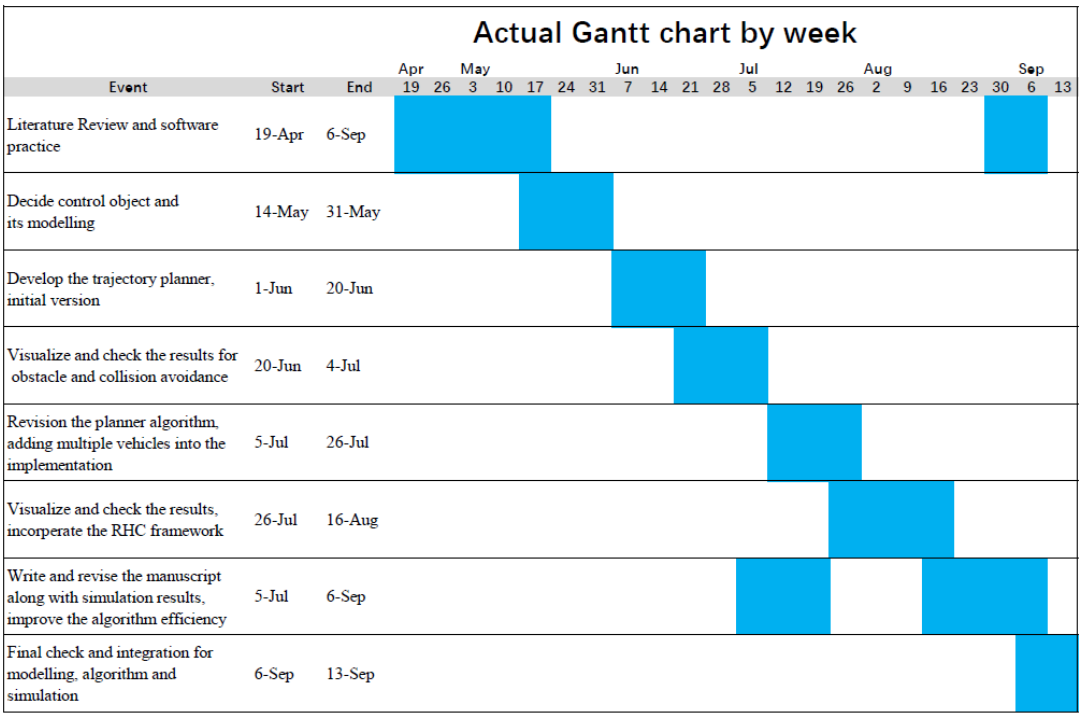


Figure 1.3: Actual Gantt Chart conducted during the project

Chapter 2

Literature Review

This chapter provides the literature review in the motion planning domain as it solves the basic problem as how the mobile robot(s) moves safely and successfully in the free configuration space. During the last two decades, many studies have been conducted and come along with plenty of applications. The first section would offer as the review of the existing planning algorithm. The second section would focus on the mathematical programming for the vehicle planning issues and show the common framework of such method, moreover, it would also focus on the methods that handle the uncertainties within the mathematical programming domain. The last section would review the applications of the planning algorithm to illustrate its functions during the whole autonomous loop.

2.1 Existing algorithms for the planning

The core of the planning algorithm is to optimize the objective functions, though some might be implicit, subject to certain constraints. These constraints are aroused from the certain requirements such as collision avoidance, obstacle avoidance or waypoint appointments, etc. In various environment settings, the given information of the vehicles or environments are different. Each planning algorithm handles various observations of the configuration space.

The discrete planning problems are the simplest to describe as the whole state space would be finite in usual cases. Each discrete state could be actually marked by an unique integer. No uncertainties need considering in the discrete planning problems

as the probability theory could be avoided as all the models are completely know and detectable. Without losing generality, most planning problems could be discretized into the grid search problem. The search purpose on these grid systems can also be defined as feasibility search problem and optimal search problem as described in [31]. The general search for the feasible plans is systematic where the visited node is marked until the goal is searched in the grid space. An insightful abstraction in [31] suggests that the differences between each search algorithms exist in the sort function for priority queue Q. The priority queue is normally a FIFO (First-In First-Out) queue. Thus the one has waited longest in the queue would be chosen for the search. Then the success states of the chosen one would be search, the iteration ends when the goal is reached, otherwise the search continues when the Q is not empty. Such method seems to just give the result that whether the goal could be reached instead of generating a plan for reaching the goal. The Breath-first search [25] adopts the FIFO queue as Q which selects states as the first-come, first-serve principle. This means whichever plan with k step, the k+1 step would be explored. Such method of the searching could cause computation burden because that the systematic search would result in the wave like visualization as many wavefront states bring out unnecessary computations. Thus, the Depth-first search could be adopted as searching in the certain direction by changing Q as a stack, following the LIFO (Last-In First-Out) principle. This means the last visited state would be chosen for the front line. Such aggressive behavior seems desirable for handling the long plan. However, depth-first search might miss out the remaining larger search space as the iterations continues.

Another effective single-source shortest paths method is Dijkstra's algorithm [11]. This algorithm is based on the graph where the environment is abstracted as nodes and edges where the cost of the path is the sum of the edges' length from start to the goal. With the metric of the edge length, the priority queue Q is adopted for sorting the cost-to-come from the source node of each node within the graph. Because the queue content represents the path information, the search direction can efficiently explore the free configuration space as each time the node with shortest cost-to-come would explored instead of FIFO principle.

To combine with the cost-to-go, the A* [20] proposed that the planning algorithm should also corporate the empirical distance from the given state to the goal. The ar-

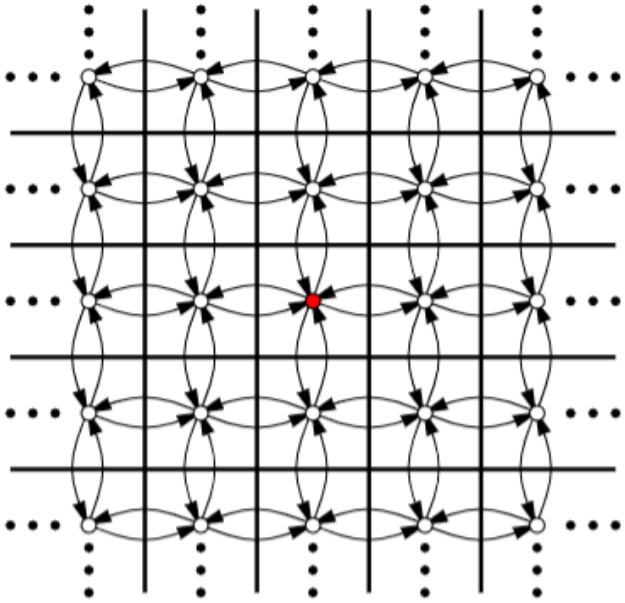


Figure 2.1: The state transition in 2D grid movement problem in [31]

chitecture of the A* is exactly same to the Dijkstra's one, the only difference is that the priority queue Q where in A*, the cost-to-go value is also considered combined with the existed cost-to-come in Dijkstra's algorithm.

Moreover, there also exists the optimal planning problem where in discrete scenario, [30] categorized as the discrete optimal planning and its method is referred as 'value iteration'. Its solution commonly could be described in 'Dynamic Programming' which is similar to that of mathematical programming.

2.2 Mathematical programming for the planning problems

Mathematical programming has explicitly presented the objective function and the corresponding constraints and the solver for that could be pure mathematical challenge where many improvements are proposed. In [2], mathematical programming categories are proposed for multi-vehicle problems and the general framework is proposed as in (2.1):

$$\begin{aligned} \textbf{minimize} \quad & \Phi(x,y) \\ \textbf{subject to} \quad & \Omega_i(x,y),\ i = 1...m \\ & x \in \mathbb{R}^{n_1}, y \in \mathbb{Z}^{n_2}, \end{aligned} \tag{2.1}$$

where the equation represents that the objective function $\Phi(x,y)$ subject to a set of constraints $\Omega_i(x,y) \leq 0$, where x and y are continuous and discrete variables for the decision, respectively. Moreover, Ω represents the mapping from the decision variable space to the constraints' space: $\mathbb{R}^{n_1} \times \mathbb{Z}^{n_2} \to \mathbb{R}^m$. According to [1], the categories happened when there are different values for n_1 and n_2. When $n_1 = 0$, the pure integer optimization problem is represented. When $n_2 = 0$, (2.1) became the continuous optimization problem. When $n_1, n_2 > 0$, the mixed integer optimization problem is presented. The nonlinearity of the objective function or of any constraint would lead the problem to be nonlinear. Among these, the Mixed-Integer Linear Programming (MILP) is well studied for the favorable theoretical guarantees and efficient computation. To be more specific, the general constraints could be replaced by more practical terms shown in (2.2):

$$\begin{aligned} \textbf{minimize} \quad & \Phi(x,y) && (Objective\ Function) \\ \textbf{subject to} \quad & \mathscr{K}(x,y) && \leq 0, \quad (Kinematics) \\ & \mathscr{D}(x,y) && \leq 0, \quad (Dynamics) \\ & \mathscr{C}(x,y) && \leq 0, \quad (Obstacle\ and\ Collision\ Avoidance) \\ & \mathscr{H}(x,y) && \leq 0, \quad (Communication) \\ & \mathscr{O}(x,y) && \leq 0. \quad (Other\ constraints) \end{aligned} \tag{2.2}$$

Under this mathematical programming framework, once the constraints are constructed, only appropriated objective function needs to be proposed. Thus the mathematical programming could handle the multiple complex constraints simultaneously. The constraints in the reality are listed as follows in general, such as obstacle avoidance, collision avoidance, communication topology, dynamics constraints and other constraints listed in (2.2). In this section, various optimization objective functions and constraints are reviewed. In [45], the safe vehicle trajectory generation under receding horizon framework is proposed. The backup trajectories are planned during these planning episode. In [10], A semi-definite programming for multi-robot tracking has been

formulated for the cluttered environment with obstacles. The work utilized the second least eigenvalue of the Laplace matrix of the connectivity graph and yielded the optimal robots configurations in the certain time steps. The whole problem was formulated in MINLP with connected communication graph at all time. In [41], the fixed-wing aircraft was studied as the trajectory planning constrained to the turning rate constraint. Moreover, more than one objective function are proposed such as time consumption and control input. This work also shown that under appropriated modification in the mathematical programming framework, many cost functions could be solve efficiently. In this work, only the global planning and one-time fashion was adopted, such fashion was also implemented in [46, 8]. To reduce the computation burden of the global planning, more techniques were proposed cooperating with the mathematical programming. In [13], the iterative methods were proposed for minimising the control input of the omni-directional vehicle for moving in the two dimensional configuration space. Instead of using fixed discretized time steps, the time selection algorithm was proposed by using feasibility and optimality in a flexible manner. Such thought of using variable time step is similar to that the discrete planning illustrated in [31]. The iterative method first formulate the objective as the feasibility problem as to check the first iteration's possibility as the solution. Then the optimal objective function would be set and new time step would be selected under the environmental constraints. In the discrete space and time planning method, two factors for the collision avoidance are time discretization and obstacle inflation. Former is to actively avoid the collision by selecting the new time step while the later one is passively growing the obstacle to 'push' the path away from the 'real' obstacles.

To handle with the uncertainties of the environment, the optimization is carried out within the certain horizon, called as 'planning horizon' [23]. The various cost functions set within this planning horizon decided how the robot handled the constraints of the tasks. In [4, 50], several cost functions had been proposed and studied where the collisions are considered along with the entrapment escape capability. One of the cost functions cooperated the high-level abstracted global information to guide the robot(s) through the cluttered environment. In [27, 28, 45], hard safety constraint was considered as the priority. The backup route and corner scenario had been considered seriously as the robot was guided to maneuver to gain more horizons behind the corners.

The assignment of the tasks or waypoints was also considered as the constraints in [51, 18, 3]. For applications of autonomous Unmanned Aerial Vehicles (Quadrotor UAVs), first illustrated in [32], many aspects have been studied for efficient path and trajectory generation [9, 44]. How to define the efficiency, like energy consumption or risk minimization, both high depend on the planned path since this determines the flight fuel cost and risk exposure. This makes the path planning highly coupled with the objectives. Moreover, in [15, 16], the energy consumption models were under the wind disturbance modelled in MIQP which considered the trade-off between the time and energy consumption.

One advantage of the mathematical programming is that after formulating the problem with proper constraints, the mathematical models could be solved accordingly using the respective solvers and benefit from it if the model is suitable. For linear programming, there exists the open source solver like the GLPK [34], also there are many others could deal with nonlinear programming as well such as CPLEX [7], Gurobi [19], and COPT, etc. Though using the MP method for multiple vehicle motion planning problem could result in the large number of decision variables and constraints, which directly affect the solution time and the solutions are not necessarily optimal globally. However, with the improvement of the solver techniques and some appropriate modifications, the mathematical programming framework could be applied for wider range of MVMP problem is anticipated. The generality of the mathematical programming enables the framework to copy with various time and space discretization for different resolutions and also it can modelling the non-convex area uniformly where in [47], the multiple vehicles' collision avoidance were modelled as the convex velocity space selection for the optimization and in [41], the vehicle's nonlinear dynamics was estimated by the piece-wise linearization for encoding the convexity. Under the MP framework, the non-convexity could be handle both by problem modelling and solver algorithms which provides more assistance in solving the issues.

Chapter 3

MILP Path Planning

3.1 Introduction

Mixed Integer Linear Programming (MILP) is the subset of the Linear Programming, it means that some variables are constrained to be integer only. The standard formulation is as follows:

$$\begin{aligned} \min_{\mathbf{x},\,\mathbf{y}} \quad & \mathbf{z} = \mathbf{c}^\top \mathbf{x} + \mathbf{d}^\top \mathbf{y} \\ s.t. \quad & \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} \left\{ \begin{array}{c} \leq \\ = \\ \geq \end{array} \right\} \mathbf{b}, \\ & \mathbf{x}_{min} \leq \mathbf{x} \leq \mathbf{x}_{max}, \\ & \mathbf{y}\ integer. \end{aligned} \tag{3.1}$$

Where in this formulation, **x** is a vector of bounded continuous real variables and **y** is a vector of discrete integer variable. The purpose of this problem is to minimize **z**, which depends linearly on **x** and **y**. **A**, **B**, **b**, **c**, **d** are problem parameterized matrices and vectors, respectively. The reason of using the binary variables are that they could encode the non-convexity [52]. These decisions are varied but similar in many scenarios. In collision avoidance, the robot must either be 'left' or 'right' of the other robot. In Task allocation scenario, the decision variables encodes the choices under certain practical constraints, which is also the convex sub-problem. The MILP inherently retains the complexity of the problem as they are $\mathscr{NP}$-complete [14].

3.1.1 MILP cases

This section introduces some domains which MILP had been applied to and gives readers with the understanding of its functions in the practical issues.

Logistics & Supply Chain

In this topic, only a small and simple problem, the assignment, would be discussed and illustrated in the industry of the logistics. Let N denote the number of the factories and also the number of the trucks for the transportation. We have a set C records the transportation cost of assigning agent i to task j, denoted as $c_{ij}, i \in [1...N], j \in [1...N]$. The optimization objective is to minimize the transportation cost by appropriately assign the task to the agents. Such problem could be solved by introducing a binary variable **b** for the task assignment decision, let it be denoted as **z**. The whole assignment problem can be formulated as follows:

$$\begin{aligned} \min_{\mathbf{z}} \quad & \sum_{i=1}^{N}\sum_{j=1}^{N} c_{ij}\, b_{ij} \\ s.t. \quad & \sum_{i=1}^{N} b_{ij} = 1, \quad \forall j \in \{1...N\} \\ & \sum_{j=1}^{N} b_{ij} = 1, \quad \forall i \in \{1...N\} \\ & b_{ij} \in \{0,1\}, \quad \forall i, j, \end{aligned} \tag{3.2}$$

where b_{ij} is the binary decision variable to determine whether the agent i would be assigned with task j. If $b_{ij} = 1$, then the agent i would be assigned to task j. Otherwise, the agent i would not be assigned task j. Moreover, only one task could be assigned to the one agent and one agent could only accept one task, which represented in the constraints as that the sum of each column and row of the **b** equals to one. Then the objective function could be formulated as the sum of the element-wise multiplication of the cost matrix **C** and decision matrix **b**. In this issue, the binary variable b functions as the mask where the optimal entry (i,j) is masked in 1 and 0 for others. Moreover, the resource constraint could also be added as the issue of transportation regulation as:

$$\sum_{i=1}^{N}\sum_{j=1}^{N} r_{ij}\, b_{ij} \leq R, \tag{3.3}$$

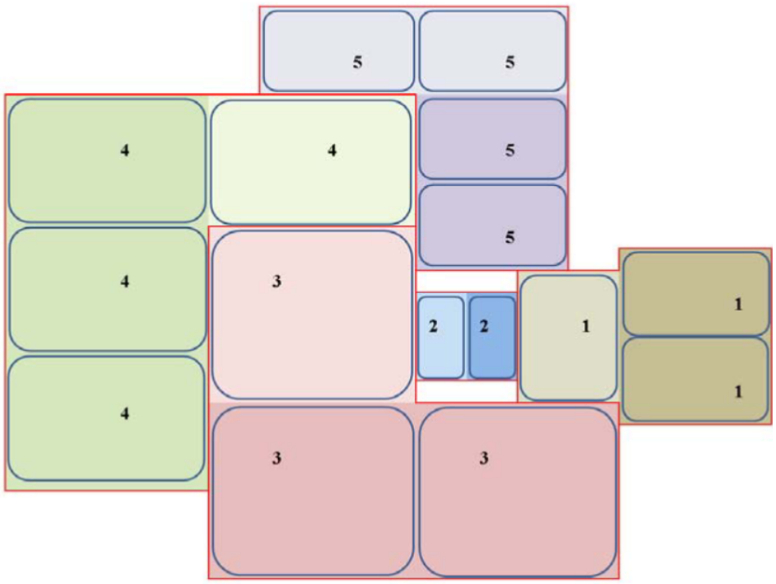


Figure 3.1: Layout solution in [5] where digits denote the different categories. The optimized layout is L-T shaped structure with same category.

where r_{ij} denotes the resource load to assign the agent i with task j, and R represents the overall restriction for the resources. Since there would be many various regulations and budget calculation in the domain of the logistics, the MILP can flexibly cooperate the several constraints to generate the decisions.

Layout Problem

Layout problem widely covered the domain of construction, urban design, electronics and chemistry, etc. A department layout problem is studied in [5] using MILP. Each department is approximated using the rectangle with the given width and length for certain category. The objective function is to minimize the total flow between various departments where the departments are placed in the certain facility. The transportation cost between each department is given and each department cannot overlap with each other. Its final result is shown in Fig. 3.1 where each category stayed compact and the occupancy ares of the all departments is efficient. f_{ij} means the flow between department i and j, x_i, y_i means the centroid of the i-th department in the x and y direction, x_{ij}, y_{ij} denotes the distance between i-th and j-th department in the x and y direction. Each department is defined by two points, up-right and bottom right vertex of the rectangle, as x^h, y^h and x^l, y^l. A and B are the dimension of the facility along the x and y

axis. The decision variable between two rectangles along two directions are defined as z_{ij}^x and z_{ij}^y, these two variables mean the department i precedes department j in the x and y directions, respectively. The formulation of the layout planning is as follows:

$$
\begin{aligned}
\min_{\mathbf{x},\mathbf{y},\mathbf{z}} \quad & \sum_{i\in I} \sum_{j\in I, j\neq i} f_{ij}(x_{ij}+y_{ij}) & & \\
s.t. \quad & x_{ij} \geq x_i - x_j & \forall i,j,i\neq j \quad & (1) \\
& x_{ij} \geq x_j - x_i & \forall i,j,i\neq j \quad & (2) \\
& y_{ij} \geq y_i - y_j & \forall i,j,i\neq j \quad & (3) \\
& y_{ij} \geq y_j - y_i & \forall i,j,i\neq j \quad & (4) \\
& x_i = \tfrac{x_i^h + x_i^l}{2} & \forall i \quad & (5) \\
& y_i = \tfrac{y_i^h + y_i^l}{2} & \forall i \quad & (6) \\
& x_i^h \leq x_j^h + A(1-z_{ij}^x) & \forall i,j,i\neq j \quad & (7) \\
& y_i^h \leq y_j^h + B(1-z_{ij}^x) & \forall i,j,i\neq j \quad & (8) \\
& z_{ij}^x + z_{ji}^x + z_{ij}^y + z_{ji}^y \geq 1 & \forall i,j,i\neq j \quad & (9) \\
& z_{ij}^x, z_{ij}^y \in \{0,1\} & \forall i,j,i\neq j \quad & (10)
\end{aligned}
\tag{3.4}
$$

where the objective function in [5] intends to minimize the total costs in the facility which is calculated by multiplying the total flow with the distance between each department and taking the sum over it. Constraints (1)-(4) define the pair-wise distance along the x and y axis between each departmental centroid. Constraints (5)-(6) define the centroid of each department as the average point of the bottom-left and up-right rectangle points. Constraints (7)-(8) guarantee the precedence relationship exists between departments i and j along the x and y axis. The facility width and length here is served as relaxed 'Big Number'. Constraints (9) ensures that at least in one direction the departments are in precedence relationship so they won't overlap with each other and constraint (10) defines the binary decision variables. The binary variable z_{ij} encodes the layout direction and precedence relationship between each component and the arrangement is optimized according to the flow cost.

3.1.2 Solutions to the MILPs

As illustrated in [40], there is no certain requirement for the deep understanding in the MILP solution algorithms. Once the problem is formulated in the MILP, it is

readily solved as many commercial packages like CPLEX [7] or open source project like GLPK [34] are already on the shelf. Mathematical programming languages like AMPL [17] can also abstract the complicated formulations into the easier language using specific syntax.

Branch and Bound

The Branch and Bound algorithm has been adopted in many solvers like CPLEX, Gurobi and COPT [7, 19]. Firstly, the integer constraints are removed as the formulation is relaxed to the LP problem. Then the LP is solved, if the results satisfy the original integer requirements, then terminate. If not, the branch operation is introduced as certain integer value would be selected, then in each direction it generates new subproblem. Thus, the solving structure is tree-like, introduced with the node description for the tree nodes. A node called 'incumbent' means the current best solution for the MILP is within this node during the solving process. A node called 'fathomed' means the node is no longer need exploring by the solver. Such happened either this branch has no solution or it does have but the corresponding objective value is worse than that of the 'incumbent' node did. The search terminates when all the branches are evaluated. In general, the branch-and-bound algorithm cannot guaranteed to search the entire solution space which is consisted with its $\mathscr{NP}$-complete property. However, with the help of appropriate choice of the heuristic, the globally optimal solution could be found.

Heuristic

Currently, there are various algorithms based on heuristic methods perform well on the optimization problems. The dominated ones including simulated annealing [37] and genetic algorithms [43]. The random search methodology reduces the susceptibility of the solver to 'fall' in local minima. The Heuristic methods are persuasive currently as they are useful in solving large scale optimization problems as the computation capacity surges in the modern age.

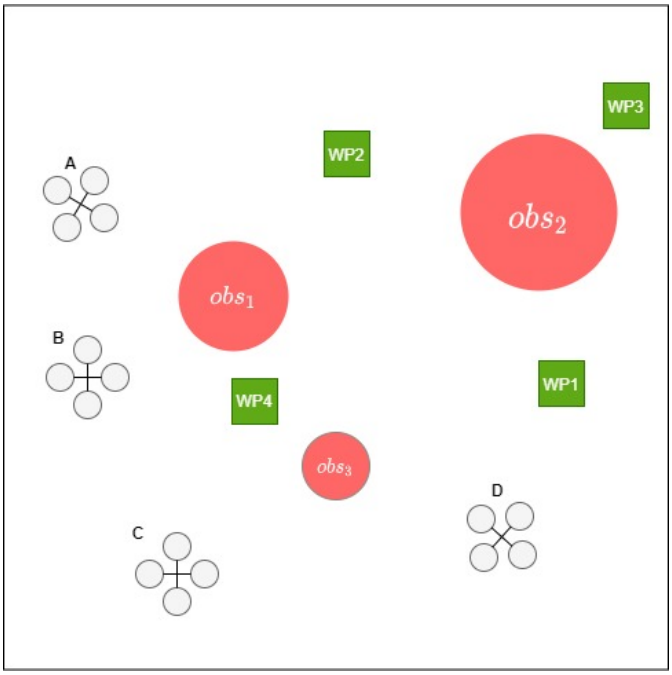


Figure 3.2: Example two-dimensional scenario for multiple vehicles path planning problem where each vehicle is assigned with the waypoint area to reach .

3.2 Problem Statement

This section give the illustration of the the problem statement for the quadrotor case. As quadrotor can generate constant force perpendicularly, thus it has capability to hover in the air. The problem can transform to 2D quite easily. To translate in the plane, the quadrotor uses the force difference generated by each paired rotors for the movement. Because the rotor speed, which is positively proportional to the force generated, can be easily controlled by the Electronic Speed Control (ESC), the quadrotor platform can respond omni-directionally in the 2D plane. The example scenario is shown in Fig. 3.2. There are several problems can be solved for this scenario:

- Each robot was assigned with the specific destination target, such as robot A was assigned to reach Waypoint 1 for finish the task. The assignment itself can also formulate as the MILP problem to solve, though in this paper it is not studied as all the assigned way-points are explicitly given to each robot.

- **Collision Avoidance** During the movement to the Waypoint, each robot should not collide with each other, thus the safety zone around the agent should be estimated.

- **Obstacle Avoidance** When the robot moves in the environment with the obstacles, it must not drive into the obstacles, thus the safe distances between robots and obstacles should be remained.

- For each robot, there might be more than one way-points to visit. The order of the visit is not assigned which is instead generated by the optimization process. On the other hand, the Waypoint assignment could also be automated optimized without pre-allocated for the minimum mission completion time.

- The whole movement could happen under the exterior or interior disturbance, which means the uncertainty could occur. Such as the exterior wind disturbance can exerted in the environment.

Section 3.5 develops the model for solving collision and obstacle avoidance. Section 3.6 develops the model for handling the Waypoints under various scenarios. Some exterior disturbances such as wind uncertainty is also added to test the algorithm's robustness.

3.3 Notations for Problem Formulation

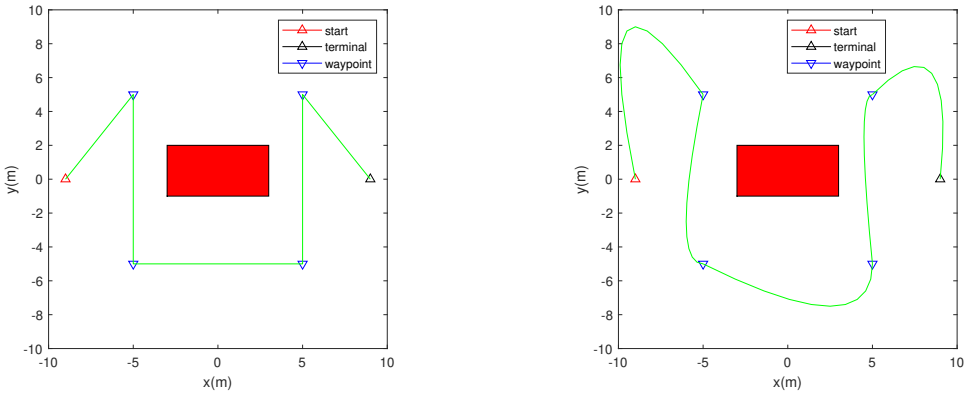
This section provides the notations that would be used to formulate the problem into MILP.

s	vehicle state
u	control input
T	time steps
N_v	number of velocity approximation constraints
N_u	number of control input approximation constraints
N_o	number of obstacle approximation constraints
D	large number for logical constraints for collision avoidance
H	large number for logical constraints for obstacle avoidance
M	large number for logical constraints for waypoints handling

Table 3.1: Notations used in MILP formulation

3.4 Vehicle Dynamics

In the planning, the dynamics should be considered for various vehicles as the generated path should be feasible to some extend. The planning result of Fig. 3.3 (b) represents a much more feasible path. In reality, the dynamics for any type of aerial vehicle are typically non-linear, to capture the non-convexity of the original dynamics, the approximation should be adopted as either linear or piece-wise linear in order to include them into the MILP format.



((a)) Planning without considering dynamics ((b)) Planning considering dynamics

Figure 3.3: Example path planning results with and without considering vehicle dynamics

3.4.1 State Equations

The state equations used in the MILP are linearized and discretized model of the original dynamics. Let (x_t, y_t) denote the position of the vehicle at the time-step t and $(\dot{x}_t, \dot{y}_t)$ as its velocity, the state vector at time-step t can be formed as $\mathbf{s}_t = (x_t, y_t, \dot{x}_t, \dot{y}_t)^\top$. Each vehicle's control forces is assumed as $\mathbf{u}_t = (u_x, u_y)^\top$ under certain input magnitude limits. So the state equation can be written as:

$$\mathbf{s}_{t+1} = \mathbf{A}\mathbf{s}_t + \mathbf{B}\mathbf{u}_t, \tag{3.5}$$

or

$$\begin{bmatrix} x \\ y \\ \mathrm{v}_x \\ \mathrm{v}_y \end{bmatrix}_{k+1} = \begin{bmatrix} 1 & 0 & \triangle t & 0 \\ 0 & 1 & 0 & \triangle t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ \mathrm{v}_x \\ \mathrm{v}_y \end{bmatrix}_k + \begin{bmatrix} (\triangle t)^2/2 & 0 \\ 0 & (\triangle t)^2/2 \\ \triangle t & 0 \\ 0 & \triangle t \end{bmatrix} \begin{bmatrix} \mathrm{u}_x \\ \mathrm{u}_y \end{bmatrix}_k \tag{3.6}$$

where **A** and **B** are the discretized system matices. In this case, the initial condition is specified as $s_0 = s^{init}$ where s^{init} is the initial state of the vehicle.

3.4.2 Velocity and Control Input Constraints

The limitations should be put into the velocity and acceleration for encoding the vehicle capabilities into the MILP form. The maximum speed in the x-y plane are denoted as V_{max}. Though such constraint can not be adde in the MILP directly, the circle can be approximated by N-sided polygons, as displayed in the Fig. 3.4.

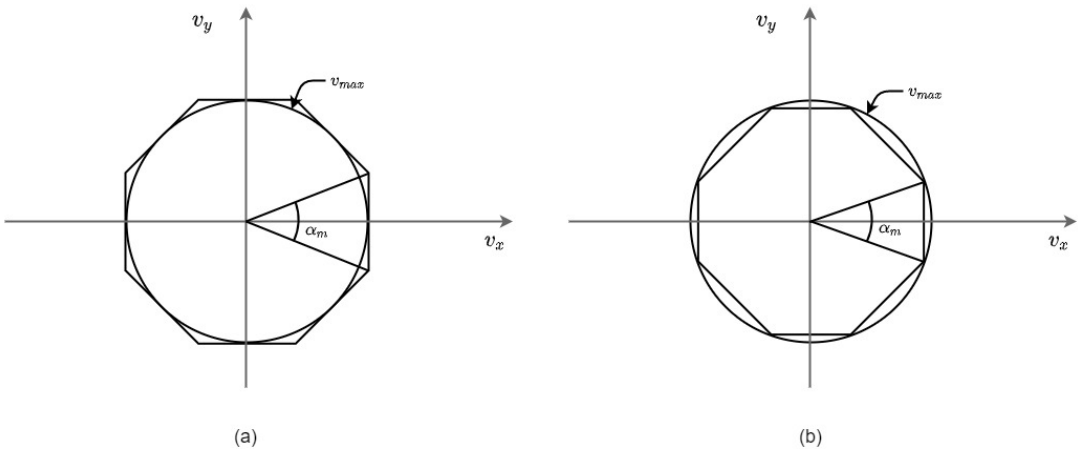


Figure 3.4: The velocity circle is approximated using 8-sided polygon. (a) The overestimated case as some velocities are infeasible. (b) The underestimated case where some feasible velocities is not encoded.

The resulting 2D velocity constraints are:

$$\begin{aligned} &\forall n \in [1 \ldots N_v] \\ &\cos(\frac{2\pi n}{N_v})v_x + \sin(\frac{2\pi n}{N_v})v_y \leq V_{max}, \end{aligned} \tag{3.7}$$

where the M denotes the number of the polygon sides for the approximation. However, such linearization results in the case that some unfeasible velocities are involved [41]. Such overestimated case is solved as we substitute the V_{max} with $V_{max}^{'}$ where

$$V_{max}^{'} = \cos(\frac{\pi}{N_v})V_{max},$$

thus Eqn. 3.7 can be rewritten as:

$$\begin{aligned} &\forall n \in [1 \dots N_v] \\ &\cos(\frac{2\pi n}{N_v})v_x + \sin(\frac{2\pi n}{N_v})v_y \leq \cos(\frac{\pi}{N_v})V_{max}, \end{aligned} \tag{3.8}$$

Similarly, the control input with limit U_{max} can be written as follows:

$$\begin{aligned} &\forall n \in [1 \dots N_u] \\ &\cos(\frac{2\pi n}{N_u})u_x + \sin(\frac{2\pi n}{N_u})u_y \leq \cos(\frac{\pi}{N_u})U_{max}. \end{aligned} \tag{3.9}$$

3.5 Collision and Obstacle Avoidance

3.5.1 Collision Avoidance

Collision avoidance between each robot is enforced using the constraint containing the safe distance. Let r denote the minimum allowed distance for safety, the constraints for collision avoidance are:

$$\begin{array}{llcl} \forall t \in [1 \dots T] & \forall p,q \mid q > p & & \\ & x_{tp} - x_{tq} & \geq & r - Dc_{tpq1} \\ \textit{and} & x_{tq} - x_{tp} & \geq & r - Dc_{tpq2} \\ \textit{and} & y_{tp} - y_{tq} & \geq & r - Dc_{tpq3} \\ \textit{and} & y_{tq} - y_{tp} & \geq & r - Dc_{tpq4} \\ \textit{and} & \sum_{k=1}^{4} c_{tpqk} & \leq & 3. \end{array} \tag{3.10}$$

where c_{tpqk} is the element of the set of the decision variables, whose value is 0 or 1 (binary). D is a positive number that is much larger than any value of the position or velocity in the scenario, often referred as 'Big D', although in many other papers such parameter is called as 'Big M' [13] or 'Big R' [41]. It is clearly that when the

c_{tpqk} equals to 1, the k-th constraint is relaxed. It means in the k-th direction, there's no distance requirement. However, at least in one direction the safe distance should be activated which means the c_{tpqk} equals to 0. Such constraint reflects on the sum of the set of the variable c_{tpqk} that should satisfy $\sum_{k=1}^{K} c_{tpqk} \leq K-1$. In the x-y plane, we can simplify the K to 4 which indicates two directions in x and y axis, respectively.

3.5.2 Obstacle Avoidance

Similar to [13], the obstacle in this paper is modelled as the circle. To encode the circle into MILP, the same approximation for the overestimated case in Sec. 3.4.2 can be adopted. Thus, the obstacle is modelled as follows defined by a set of N_o inequalities:

$$\begin{aligned} \mathscr{O} = \{(x,y) | \cos(\frac{2\pi n}{N_o})(x-p_x) \quad & + \sin(\frac{2\pi n}{N_o})(y-p_y) \leq R_o, \\ & \forall n \in [1 \dots N_o].\}, \end{aligned} \tag{3.11}$$

where (p_x, p_y) denotes the coordinate of the obstacle centroid, R_o denotes the radius of the obstacle. To encode the obstacle avoidance into the MILP, it must be converted to an equivalent set of linear inequalities. For (x_t, y_t) represents the vehicle's position at time-step t, the avoidance condition should be expressed as $(x_t, y_t) \notin \mathscr{O}$. Similar to the collision avoidance, the obstacle avoidance constraints can be written as follows:

$$\begin{aligned} & \forall n \in [1 \dots N_o] \\ & (x_t - p_x)\cos(\frac{2\pi n}{N_o}) + (y_t - p_y)\sin(\frac{2\pi n}{N_o}) > R_o - Hb_n, \\ \textit{and} \quad & \sum_{n=1}^{N_o} b_n \leq N_o - 1. \end{aligned} \tag{3.12}$$

where decision binary variable b_i, $\forall i \in [1 \dots N]$ is introduced to indicate the activation of the constraint. H is here is a positive sufficiently large number as well, if the b_i equals to 1, then the i-th constraint is relaxed. However, to avoid the obstacle, at least one constraint should be satisfied, as the sum of all the binary variable b_i, $\forall i \in [1 \dots N]$ should less or equal to N-1, where N denotes the number of the approximation polygon sides.

3.6 Waypoint Handling

This section, the Waypoint handling would be introduced and formulated into the MILP. The number of the waypoints should greater or equal to 1, when the number equals to 1, the waypoint can be treated as the robot's terminal point. We continue letting (p_x, p_y) denote the coordinate of the robot in the 2D plane. The coordinate of the k-th waypoint is denotes as (W_x^k, W_y^k). The constraints for Waypoint visiting can be written as:

$$
\begin{aligned}
& \forall i \in [1 \ldots T] \quad \forall k \in [1 \ldots W] \\
& \quad p_x - W_x^k \quad \leq \quad M(1 - t_{ik}) \\
\textit{and} \quad & p_x - W_x^k \quad \geq \quad -M(1 - t_{ik}) \\
\textit{and} \quad & p_y - W_y^k \quad \leq \quad M(1 - t_{ik}) \\
\textit{and} \quad & p_y - W_y^k \quad \geq \quad -M(1 - t_{ik}) \\
\textit{and} \quad & \sum_{i=1}^{T} t_{ik} \quad = \quad 1.
\end{aligned}
\tag{3.13}
$$

$$
\forall i \in [1 \ldots T] \; \forall k \in [1 \ldots W] \; \sum_{i=1}^{T} t_{ik} = 1. \tag{3.14}
$$

Similar to Eq. 3.10 3.12, the constraints require that the vehicle visits each Waypoint exact once using the binary decision variable $t_{ik}, \forall i \in [1 \ldots T] \; \forall k \in [1 \ldots W]$. The sum of the variable should equals to 1 means during the whole planning time horizon, only once the constraints for k-th waypoint visit would be activated.

3.7 The Objective Function

This section introduces how to use the MILP constraints to include the finishing time to the formulation. We want to solve the vehicle p achieving the final point before the maximum time-step T. First, let us introduce the binary decision variable f_i to indicate the finish state of each time step. When $f_i = 1$ means the vehicle achieves the final position, and 0 otherwise. The MILP constraints for final point reaching is as follows:

$$
\begin{array}{lllll}
\forall i \in [1 \ldots T] & & & & \\
 & p_{xi} - x^f & \leq & & R(1-f_i) \\
\textit{and} & p_{xi} - x^f & \geq & - & R(1-f_i) \\
\textit{and} & p_{yi} - y^f & \leq & & R(1-f_i) \\
\textit{and} & p_{yi} - y^f & \geq & - & R(1-f_i) \\
\textit{and} & & & &
\end{array}
\tag{3.15}
$$

$$
\sum_{i=1}^{T} f_i = 1. \tag{3.16}
$$

where R is the same sufficiently large, positive number. It can been seen that if $f_i = 1$, Eqn. 3.15 forces the vehicle to equal to the final state. Eqn. 3.16 constraints that the vehicle to the finish point at some time-step. Thus the minimum time solution for the vehicle can be formulated as:

$$
\min_{\mathbf{s},\mathbf{u},\mathbf{c},\mathbf{f}} J = \sum_{i=1}^{T} T_i f_i \tag{3.17}
$$

where T_i is the actual time elapsed at step i. Eqn. 3.17 can also be extend to the multiple vehicles scenario very easily. We can modified the binary decision variable f_i into f_{ip}, as two-dimensional binary array where p indicates each vehicle. Eqn. 3.15, Eqn. 3.16 and Eqn. 3.17 can be rewritten as follows, respectively:

$$
\begin{array}{llllll}
\forall p \in [1 \ldots N] & \forall i \in [1 \ldots T] & & & & \\
 & x_{pi} - x_p^f & \leq & & R(1-f_{ip}) \\
\textit{and} & x_{pi} - x_p^f & \geq & - & R(1-f_{ip}) \\
\textit{and} & x_{pi} - y_p^f & \leq & & R(1-f_{ip}) \\
\textit{and} & x_{pi} - y_p^f & \geq & - & R(1-f_{ip}) \\
\textit{and} & & & &
\end{array}
\tag{3.18}
$$

$$
\forall p \in [1 \ldots N] \sum_{i=1}^{T} f_{ip} = 1. \tag{3.19}
$$

$$
\min_{\mathbf{s},\mathbf{u},\mathbf{c},\mathbf{f}} J = \sum_{p=1}^{N} \sum_{i=1}^{T} T_i f_{ip} \tag{3.20}
$$

There's need to mention that Eqn. 3.19 enforces the logic that each vehicle should reach the final point before T, but does not enforce they to reach the destination at the same time-step. Thus the objective can be formulation as minimize the total reach time of all vehicle p. Unfortunately, according to [41], Eqn. 3.20 leads to an inefficient formulation since there would be multiple solutions stops at each time step. Such problem can be remedied by adding the states and inputs as small input penalty to the cost function.

$$\min_{\mathbf{s},\mathbf{u},\mathbf{c},\mathbf{f}} J = \sum_{p=1}^{N} \sum_{i=1}^{T} \{T_i f_{ip} + \varepsilon(|\mathbf{u}_{x_{ip}}| + |\mathbf{u}_{y_{ip}}|)\} \tag{3.21}$$

where ε is a positive number and small enough. The whole problem is to solve Eqn. 3.21 subject to the constraints in Eqns. (3.10), (3.12), (3.13), (3.15) and (3.16).

3.8 Simulation and Results

3.8.1 Implementation Details

The problem is optimized by the MILP solver, based on branch-and-bound algorithm, implemented in the GLPK [34]. The formulation of the problem is illustrated in AMPL [17] where GLPK accepts to interpret the subset of the AMPL. The single vehicle planning problem is demonstrated in the Appendix. 7.2, the implementation in AMPL is straight forward as long as the parameters and variables are defined clearly at the first place. The solver should also be provided with the data file to instantiates the model file, this task is achieved by the MATLAB file as the user define the environment and respective constraints. The example of the single vehicle planning data generation script is shown in Appendix. 7.1. After the solver completed the optimization, the script is written to analyse the output. The variables, such as planned positions, velocities, are extracted to the MATLAB space and then the plotting utilities can been used for figures. The simulation is based on the Windows PC equipped with 8xCPU@2.6GHz and 16G RAM.

3.8.2 Example: Single Vehicle with Obstacle Avoidance

In this example, the single vehicle with capacity for obstacle avoidance is presented under different time differences. The uniformly gridding methods is used as dt = 1s in

Fig. 3.5 and dt = 0.5s in Fig. 3.6. The global planning with fined time discretization is shown in Fig. 3.7 and the vehicle's state is illustrated in Fig. 3.8. This example mainly shows the differences of adopting various dt values. The smaller the dt is, the finer of the planned path, as a result, there would be less potential collisions between each path vertex. The more efficient algorithm would be introduced in Sec. 3.9.

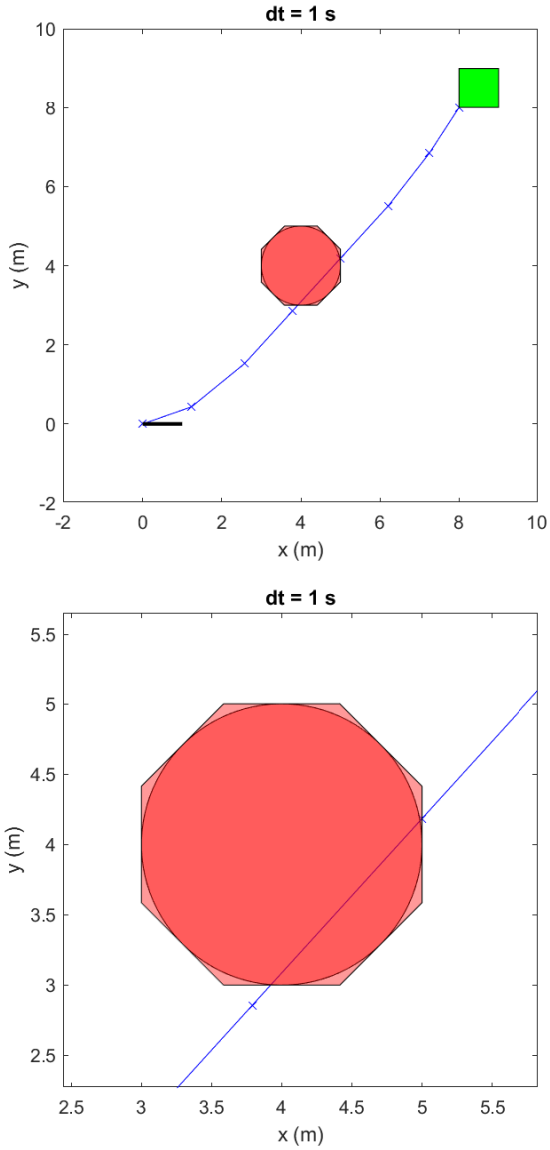


Figure 3.5: Planning result with dt = 1s. The first figure shows the planning result within global horizon, the position of each planning time step is outside of the obstacle. However, due to the poor uniformly gridding, the result might cross the obstacle. The second figure shows the detailed of the planned waypoints.

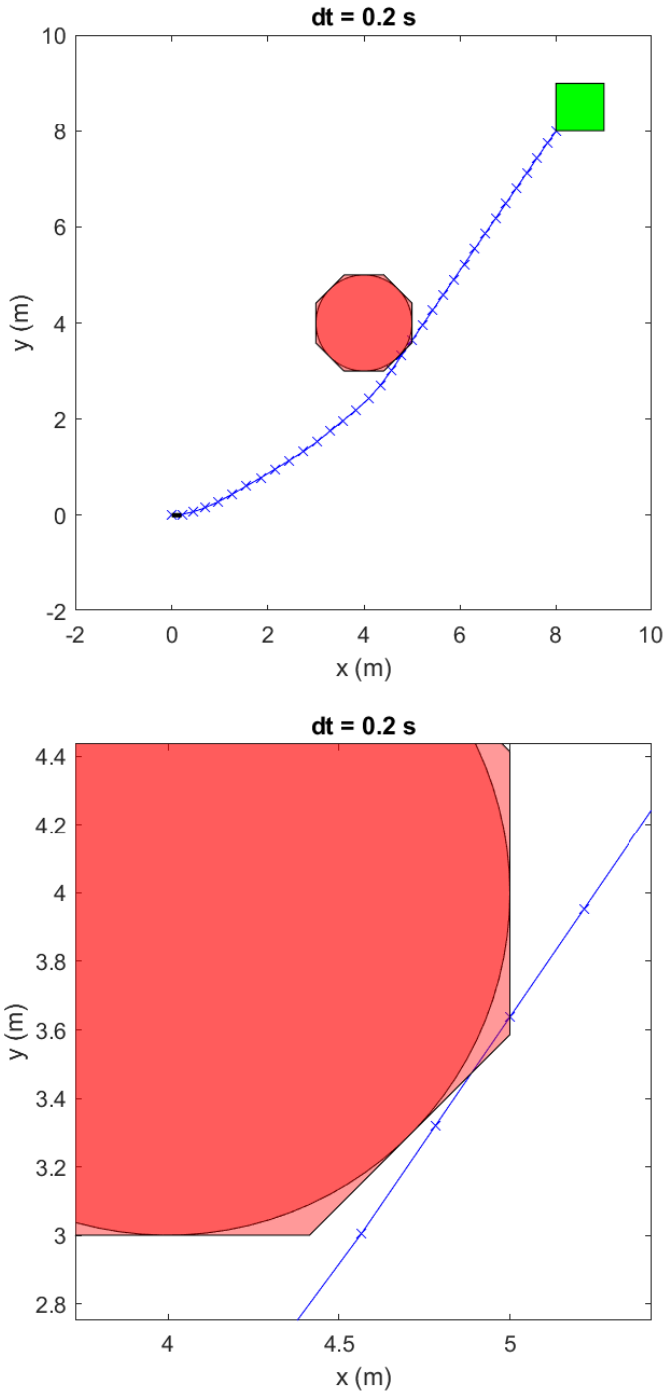


Figure 3.6: Planning result with dt = 0.2s. With finer discrete time step, the path didn't collide with the obstacle, however, with the increase in the number of the planning points, denoted in blue cross, the computation time also increased.

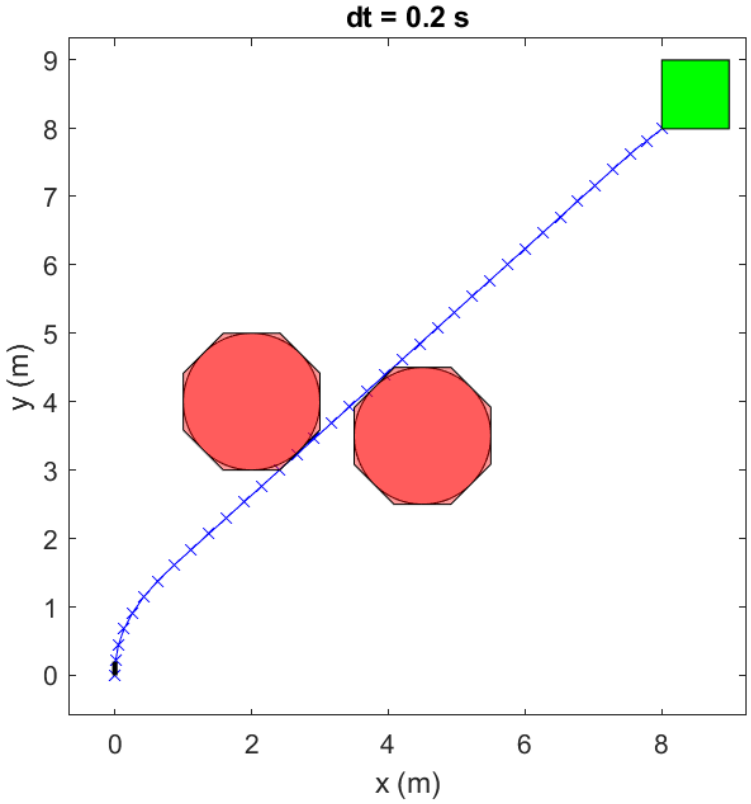


Figure 3.7: Planning result with fined time discretization as $dt = 0.2s$ through narrow corridor between two nearby obstacles.

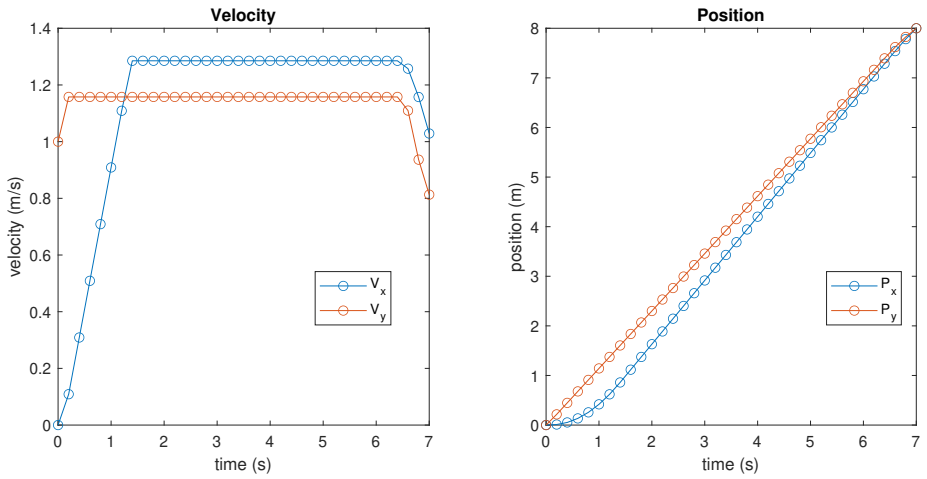


Figure 3.8: The state, velocity and position, of the vehicle during the planning horizon. The final magnitude of the velocity is not restrict to zero in this scenario.

3.8.3 Example: Multiple Vehicles with Collision Avoidance

In this example, multiple vehicles were assigned with final point to reach, respectively, such as vehicle 1 should reach the Final 1 illustrated in the Fig. 3.9. Two different approximations for the collision avoidance distance is adopted in this example. In Eqn. 3.10, four directions are adopted in the the constraints. Actually, more constraints could be put into the avoidance constraints like in Eqn. 3.12. The 8-side approximation is also adopted for safe collision avoidance shown in the figure. Both snapshots of the scenarios show the collisions are free under the appropriated modelling.

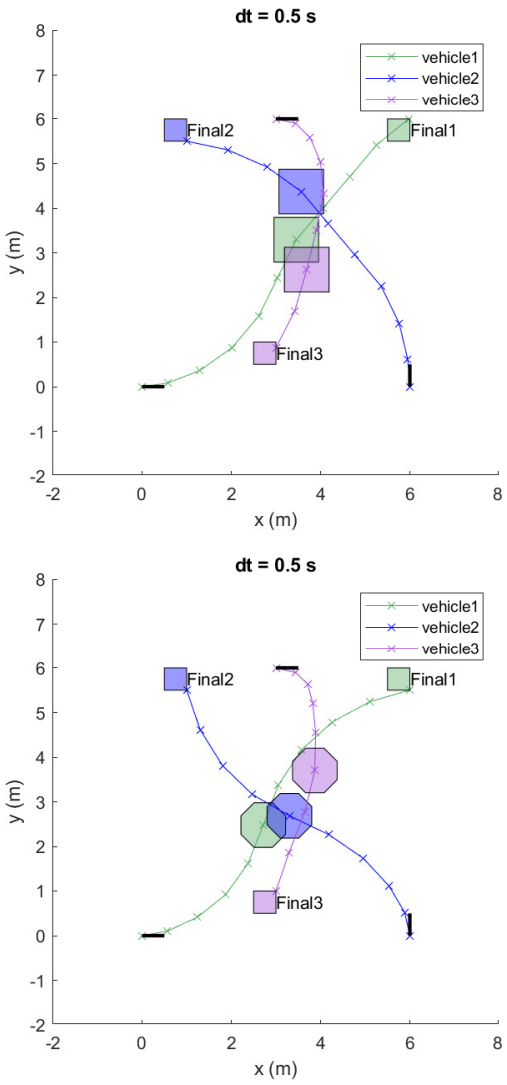


Figure 3.9: The multiple vehicles with collision avoidance both with four-side and eight-sided approximation. Each vehicle's position is out of the colored safety region of other vehicles' at each time step. This demonstrated that the safety region could be flexible to the users' definitions

3.8.4 Example: Single Vehicle with Waypoint Handling

This example demonstrates the single vehicle handling with multiple waypoints. In Fig. 3.10, the optimizer generates the minimum-time path for the vehicle to visit all the waypoints, the order of visit is not pre-assigned. Also, obstacle avoidance could be cooperated, shown in Fig. 3.11.

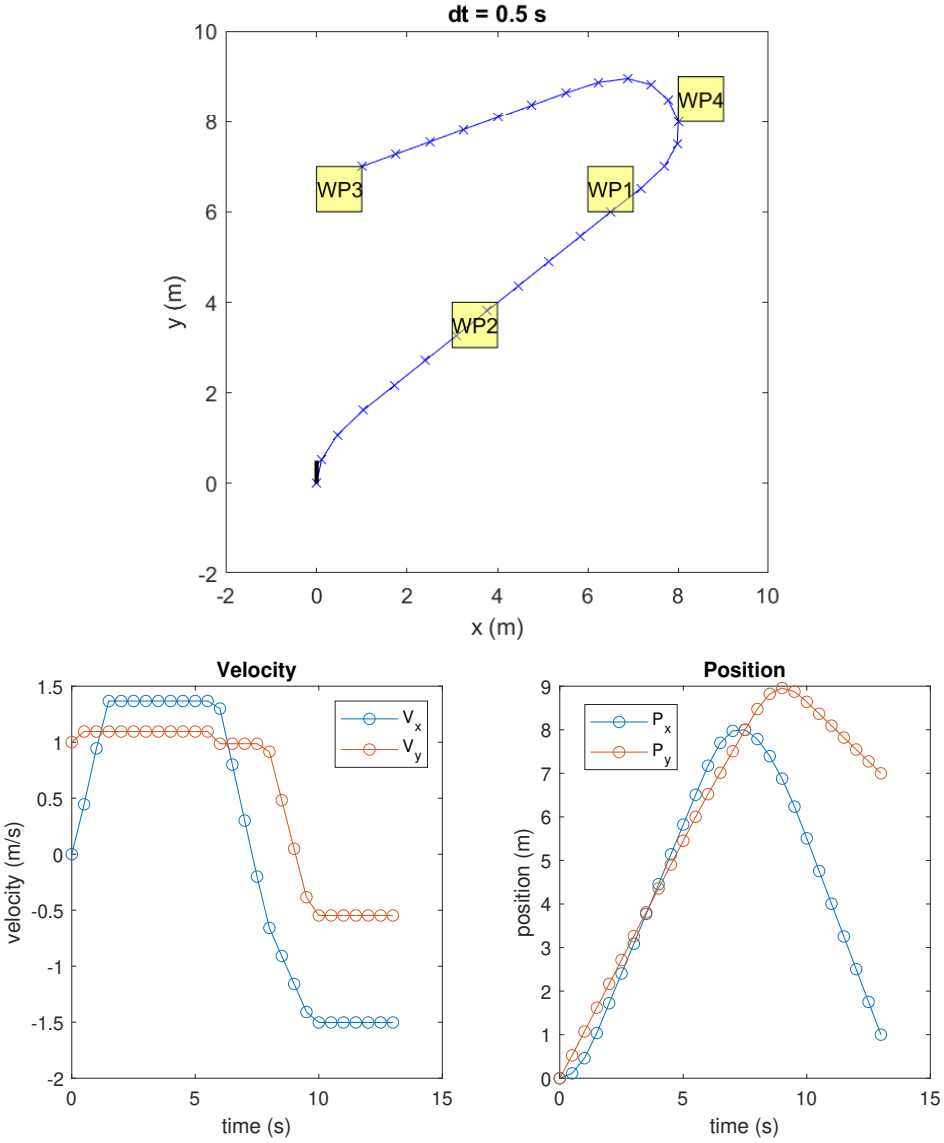


Figure 3.10: Planning result for the vehicle to visit four separated waypoints. The initial position of the vehicle is $[0,0]^\top$, initial velocity is $[1,0]^\top$ as indicated by the black arrow. The order of the visit is optimized by the solver instead of pre-assigning. Blue cross denotes the position of each planning time step.

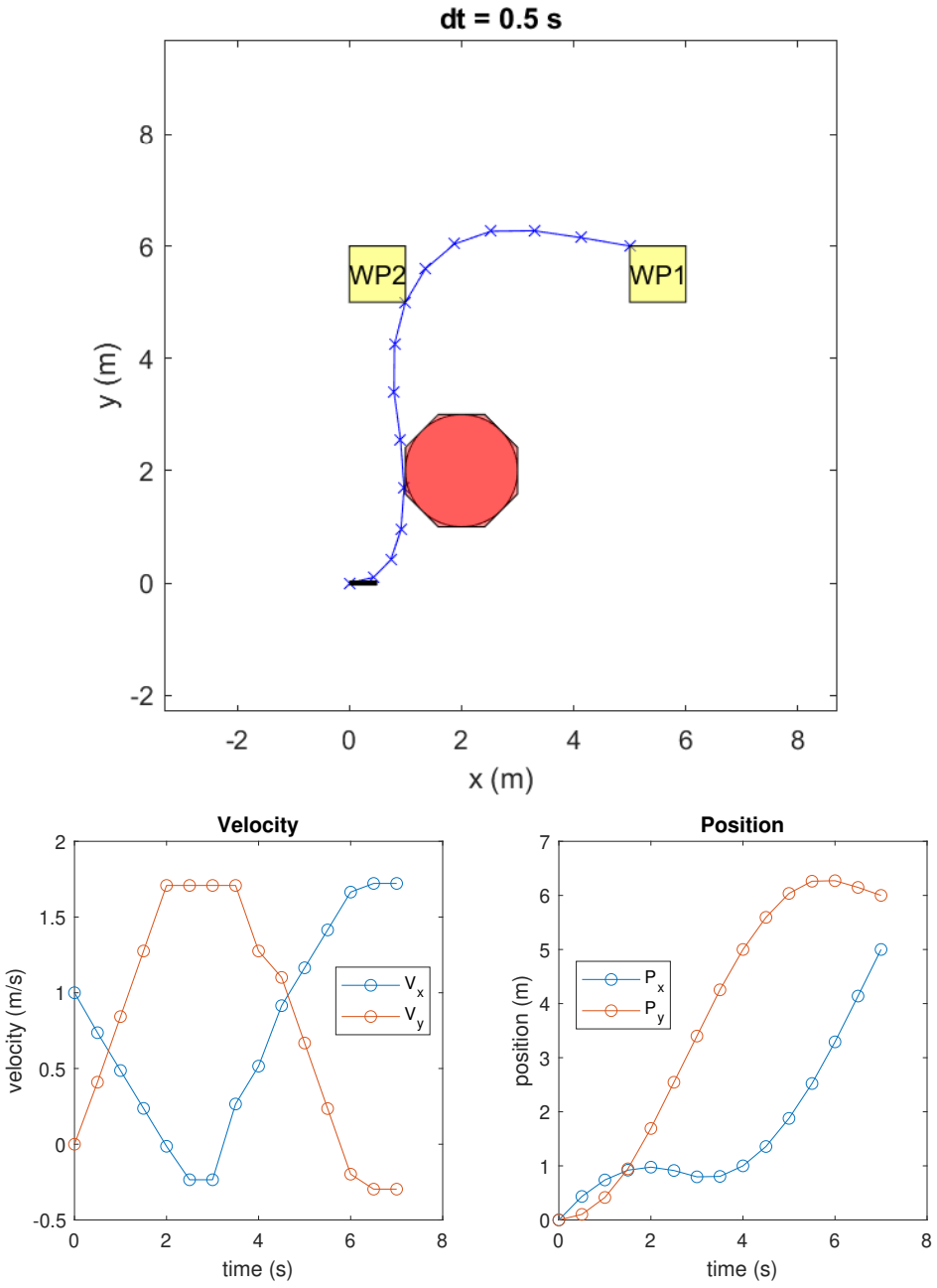


Figure 3.11: Obstacles can be cooperated as well, with fined time dicretization. The planning result is demonstrated with all the successful waypoints visited along with collision-free path.

3.8.5 Example: Single Vehicle with Wind Disturbance

In this example, the exterior wind disturbance is added to the environment. With fined time and appropriate horizon steps, the minimum-time path could be calculated by the optimizer. The exterior disturbance like wind is treated as the exerted force applied in the dynamics of the vehicle in MILP. In this setting, the wind disturbance is perfectly known to the vehicle at each time step where the online model for handling the unknown disturbance is studied in Chap. 4.

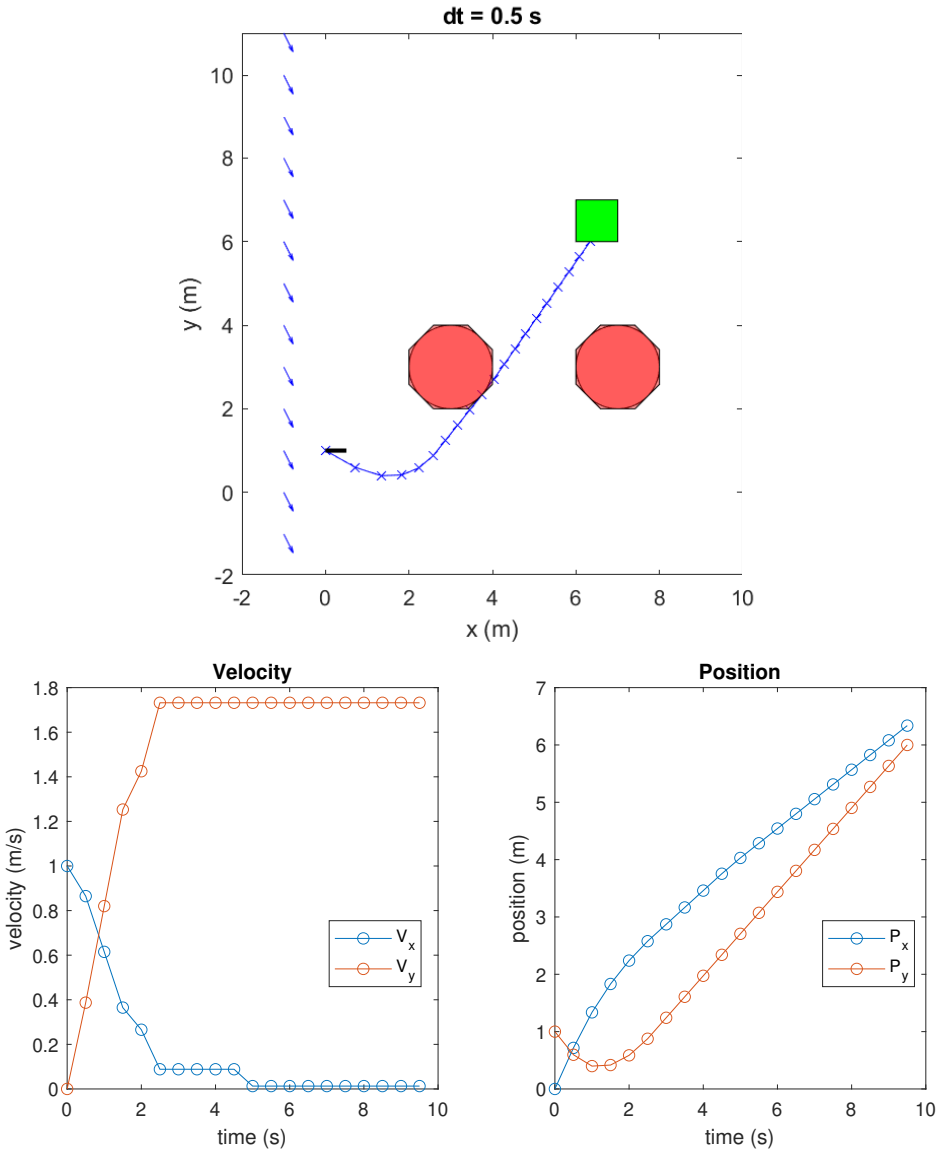


Figure 3.12: Exterior disturbance like wind is Incorporated into the MILP. The wind velocity is $[0.5, -1]^\top$ denoted as blue arrows in the left side. With the perfect knowledge of the disturbance, the vehicle reached the final target against wind and obstacles.

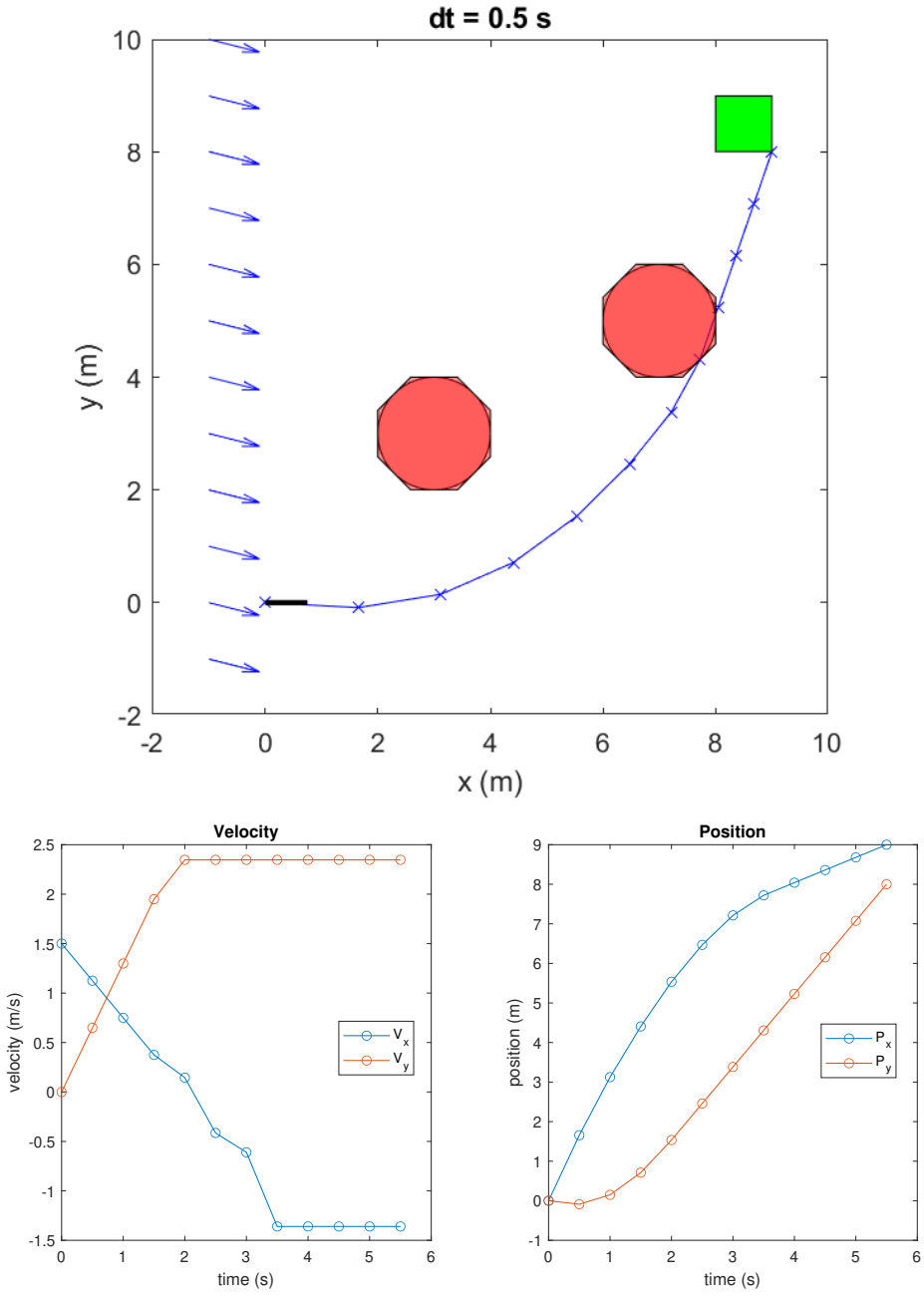


Figure 3.13: Exterior disturbance like wind is Incorporated into the MILP. The wind velocity is $[2, -1]^\top$ denoted as blue arrows in the left side. With the perfect knowledge of the disturbance, the vehicle reached the final target against wind and obstacles.

3.9 Iterative Refinement using MILP

This section would introduce the iterative techniques to conquer the drawbacks of the global one-time MILP planning fashion. To achieve more robust outcomes, the discretized time steps need being quite small, resulting in exponentially increased decision variables both for collision and obstacle avoidance, thus the solving time increased as well [41]. As we shown in the Sec. 3.5, the critical factors for the finer path planning is the finer time gridding or larger obstacle inflation. For each time step, the planner needs to judge the collision situations for obstacles thus the total computation time is proportional to the $N_p \times N_o \times N_T$ where N_p is the number of the collision check, it is default as 4 here described in Sec. 3.5. There are few techniques [13] for reducing the total computation time while maintaining the path safety. One way is to reduce the time step and another is to increase the obstacle inflation for the exchange of decreasing the number of uniform sample time gridding. Thus this section would introduce iterative MILP methods for vehicle control problems. Inspired by [13], we firstly set the objective function as minimize the whole control input as

$$J = \sum_{k=0}^{T-1} (|u_x^k| + |u_y^k|), \tag{3.22}$$

where u_x and u_y are bounded as described in Sec. 3.4.

3.9.1 Iterative time selection algorithm for MILP path planning

It is obvious that the fewer avoidance checks during the path, the less time it would cost, thus it is more advantageous. The iterative time step selection is the natural solution to this thought. The idea is to first solve the MILP with no obstacle considered in the model as shown in the first iteration of the Fig. 3.14, then the path is checked for the collisions. If there are collision exists, the MILP would be augmented by adding the time steps to each collided path where the time step selection happened. Then the augmented MILP is solved and the resulting path is checked for the collisions, repeating the procedure until a collision-free path is planned.

If there are no collisions, the algorithm would be terminated, otherwise, for each collided path i, we compute the time interval $[t_1^i, t_2^i]$ where the trajectory is crossing the obstacle. Then the augmented time step t_{new}^i should be selected in $[t_1^i, t_2^i]$. In this section,

we choose the average value of the interval as $t^i_{new} = (t^i_1 + t^i_2)/2$. The whole procedure for iterative time step selection is shown in Alg. 1.

Algorithm 1: The iterative MILP time selection algorithm

```
Formulate the MILP without obstacle consideration.
Solve the MILP for each obstacle avoidance.
Check the path collisions for each path interval.
while Collision exists do
    For each collision i, calculate the time interval [t_1^i, t_2^i].
    Augment the MILP with newly selected time step t_new^i between each
     collision time interval. Solve the MILP to plan the new path.
    Check the resulting path for collision.
end
```

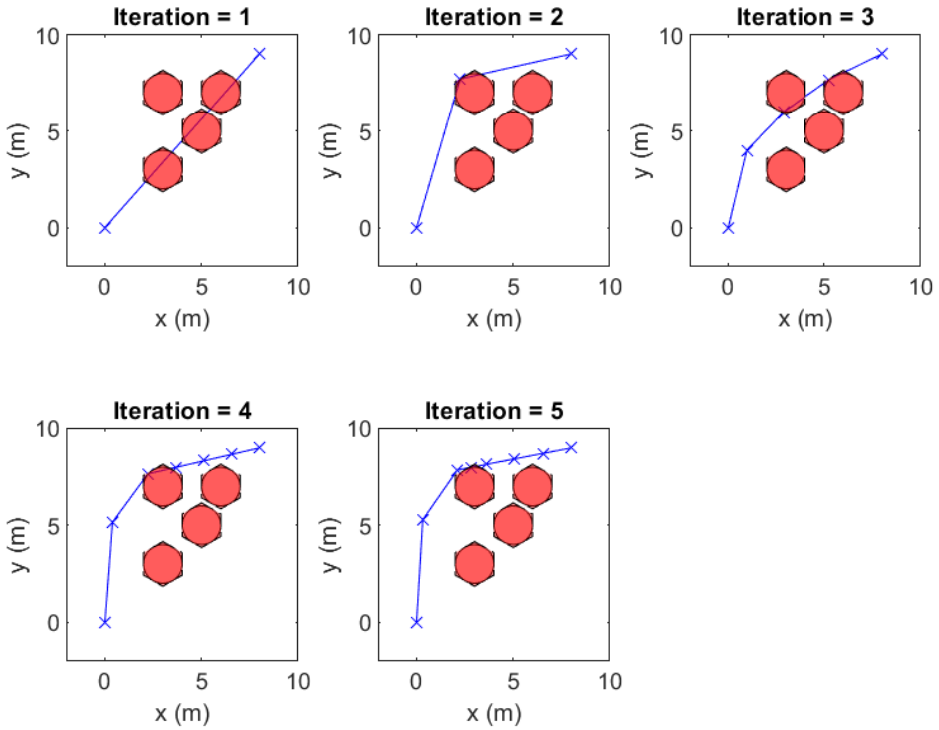


Figure 3.14: The snapshots of each iteration for the iterative time selection MILP method. The red polygons represents the obstacle approximation, the cross represents the augmented time step selected. The start state: $(x_s, y_s, \dot{x}_s, \dot{y}_s) = (0, 0, 0, 1)$.

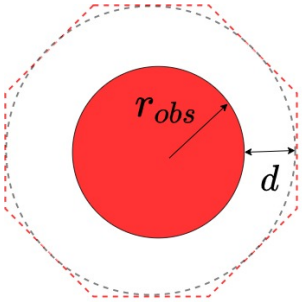


Figure 3.15: Illustration of the obstacle inflation.

3.9.2 Iterative obstacle inflation algorithm for MILP path planning

The another iterative MILP algorithm for obstacle avoidance is to increase the obstacle's buffer size d. The illustration of the obstacle expansion is shown in Fig. 3.15, the origin radius of the obstacle is denoted as r_{obs}, the dotted line represents the overestimated approximation to the obstacle. Here we define the inflation coefficient $\alpha \geq 0$, thus the expanded radius defined as $r_{expanded} = (1+\alpha)r_{obs}$ and $d = \alpha r_{obs}$.

The whole procedure of the obstacle growing algorithm is list in Alg. 2. The first step is the same as that of the time selection algorithm, without considering the obstacle in the environment. The difference is that the method dealing with the collisions when it happened. When the collision happened with the j-th obstacle, the algorithm would expand the d by the certain increment value.

Algorithm 2: The iterative MILP obstacle inflation algorithm

1 Formulate the MILP without obstacle consideration according to the vehicle control objectives.
2 Set obstacle buffer zone d_j for each obstacle.
3 Solve the MILP for each obstacle avoidance with its respective buffer size.
4 Check the path collisions for each path interval.
5 **while** *Collision exists* **do**
6 For each collision i that happened with obstacle j, expand the j-th obstacle's buffer size.
7 Augment the MILP with newly expanded obstacle. Solve the MILP to plan the new path.
8 Check the resulting path for collision.
9 **end**

However, the iterative growing obstacle method still got some problems in planning the good quality path, as the expansion of the obstacles would take up much space in the configuration space and the initial and terminal point might got engulfed by this expansion. This method could be quite efficient for the dense uniformly time step gridding while not that robust for too coarse time step initialization as the it might iteratively cause the more invaluable obstacle growth.

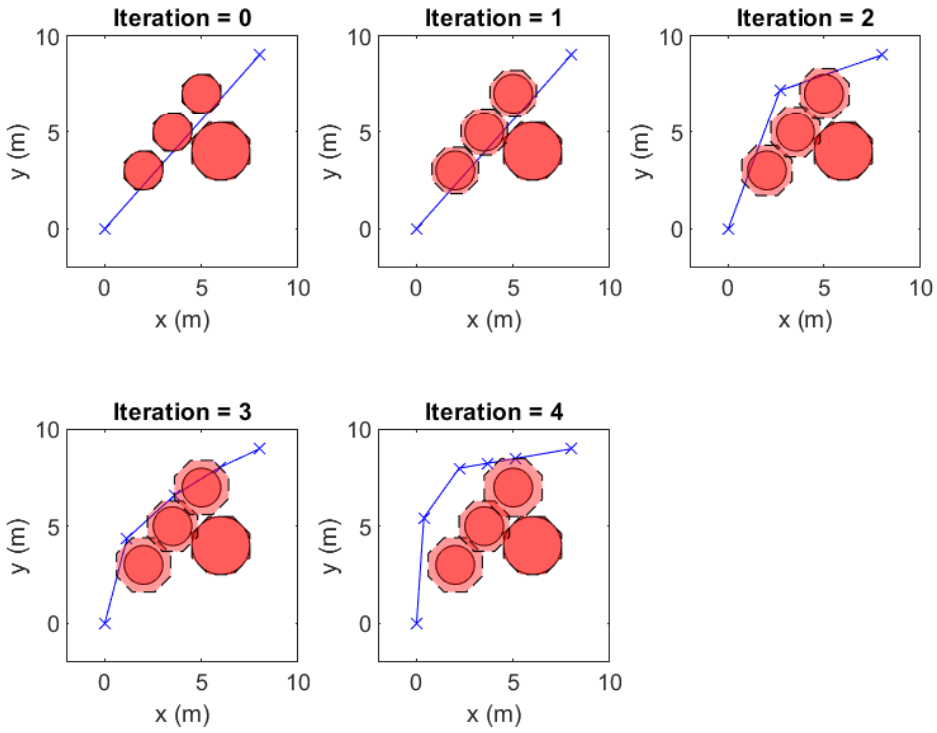


Figure 3.16: The snapshots of each iteration for the iterative obstacle inflation MILP method. The dotted red polygons represents the obstacle approximation with expansion, the cross represents the augmented time step selected. The expansion coefficient equals 0.1. The start state: $(x_s, y_s, \dot{x}_s, \dot{y}_s) = (0, 0, 0, 1)$.

The comprehensive comparison between the global one-time and iterative MILP planning methods are demonstrated in Fig. 3.17. The four figures show the efficiency and differences between various methods. The growing obstacle can also be combined with the uniform gridding, thus, with the appropriate choice of the method combination, the suitable computation time would be achieved.

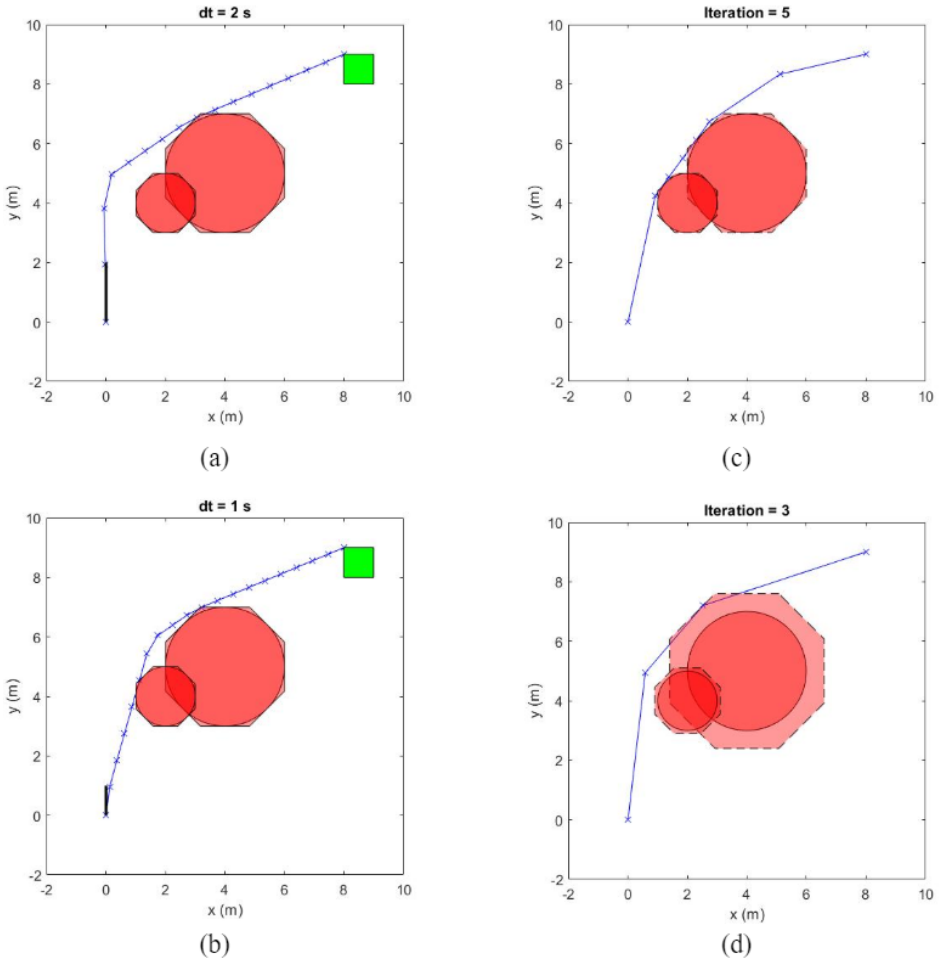


Figure 3.17: The comparison of the MILP methods with minimum control input objective. (a) the result with time step equals to 2s. The path intersected in one interval with the obstacle. (b) the result with time step equals to 1s. The finer gridding makes the path avoid the collision while more avoidance time steps are chosen, thus the computation time arises. (c) the result of iterative time step selection algorithm, the time step is reduced significantly compared to (b). (d) the result of iterative growing obstacle with time step selection algorithm, the number of the timing for avoidance check reduces to four and the path is collision-free. The initial and terminal configuration are all the same as $(x_s, y_s, \dot{x}_s, \dot{y}_s) = (0,0,0,1)$ and the terminal region is restrict to $(x_f^{min}, y_f^{min}, x_f^{max}, y_f^{max}) = (8,8,9,9)$

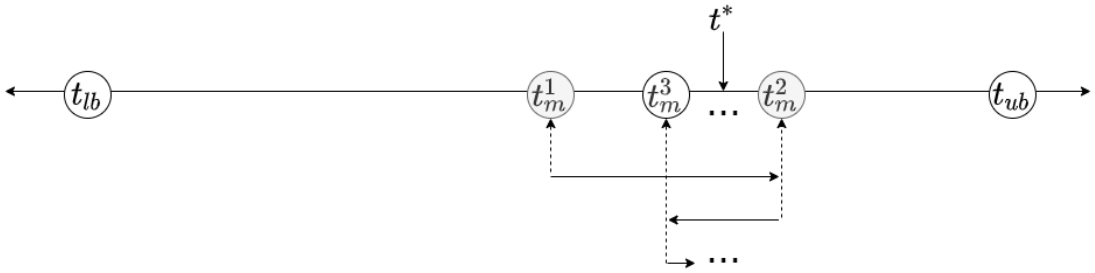


Figure 3.18: The binary search for the optimal minimum time with the tolerance ε

3.9.3 Iterative MILP with minimum time objective

This section the minimum time path planning method is presented. Without uniformly time step gridding, the iterative method can be used to search for the minimum time t^*. The search strategy is the binary search for the optimal minimum time. The illustration of the binary search is shown in Fig. 3.18. We first set the lower and higher bounds for the time as t_{lb} and t_{ub}, respectively. Then, we use the MILP to formulate the feasible problem instead of the optimization problem. The initialization of the t_{lb} and t_{ub} should be appropriated where t_{lb} should be sufficiently small number and t_{ub} cannot be too large as the computation time for the feasibility should also be considered. Then the average value t_m is calculated as $t_m = (t_{lb} + t_{ub})/2$, and the MILP feasibility problem is solved using the t_m. If the the problem can be solved using t_m, then the optimal minimum time must lie in the time interval (t_{lb}, t_m], otherwise, it is lied in the (t_m, t_{ub}]. Then the second average value of the feasible time interval is calculated for searching the optimal minimal time with tolerance of ε whose value is sufficiently small for determine the optimal finish time t^* is within the time interval. The average value t_M is used as the final time value to validate the MILP feasibility, $t_m^1, t_m^2, ..., t_m^k$ is the sequence of the iterative final times for the binary search algorithm. The thought of the binary search is illustrated in Alg. 3. For the time interval $(t_L, t_R]$, if the t_M is infeasible for the MILP, then we should search in the right-half time interval, otherwise, we search in the left-half time interval. The iteration terminated when $t_R - t_L < \varepsilon$. After k iterations, the time interval's length is obviously equals to $(t_{ub} - t_{lb})/2^k$.

We only discuss the iterative MILP minimum time algorithm here and obviously the algorithm is quite suitable for solving the minimum time objective efficiently.

Algorithm 3: The iterative MILP for the minimum time algorithm

Formulate the MILP without objective function.

Set time boundaries as $t_L := t_{lb}$ and $t_R := t_{ub}$

Calculate the average of the feasible time interval as $t_M := (T_L + T_R)/2$.

while $(T_R - T_L) > \varepsilon$ **do**

    if *the path is feasible with the final time* $t_f := t_M$ **then**

        Set $t_R := t_M$.

    else

        Set $t_L := t_M$.

    end

    Set $t_M := (T_L + T_R)/2$.

end

Chapter 4

Receding Horizon Control

4.1 Introduction

Recently, with the emergence with huge numbers of autonomous vehicles, the autonomous navigation techniques are arising in the recent decades. Moreover, with the appearance of the modern software middle-ware like ROS [48], along with high performance computing units, the real-time planning is transformed to be much more robust and demonstrated the adaptive capability to the various changing environments. To cope with the uncertainty caused both in the interior and the exterior, such as plant modelling error and disturbance, etc., more sophisticated control architectures should be used instead of using the global method proposed in the Chap. 3 as the path is planning in once. Though the iterative method is proposed to reduce the number of points where collision check is needed, such technique still can barely handle with the large and complex stationary or dynamic environments under the burden of the high computation demands. One alternative way to overcome such problem is to adopt the Receding Horizon Control(RHC) [23]. Though RHC can handle the real-time planning more sophisticated, there are still several problems remained and many papers are published to overcome them. The idea of the safe guaranteed planner is proposed, in [45], the safe basis states and backup rescue path is paralleled planned. In [28], the corner scenario is considered as the environment dynamics behind the corner is unpredictable within the certain horizons, thus certain maneuvers are required which is considered within the RHC-based planner.

The components and procedure of Receding Horizon Control is introduced in the

next section. Mixed Integer Linear Programming (MILP) is still used for the optimization as it is suitable for encoding the logical constraints. Thus the MILP is used in the RHC framework to relieve the computation burdens and to incorporate feedback. The path planned by the MILP ought to guided by the objective function as pointing to the terminal point but there's no necessary requirement to reach it, since the restriction of the size of the horizon. Multiple objective penalty terms would be discussed as the critical part of the RHC path planner.

4.2 Receding Horizon Path Planner Overview

The Receding Horizon Control, also called Model Predictive Control, designs the input sequence that optimizes the plant's output in the certain horizons. The approach to ensure that the successively planned path can reach the goal is to minimize the estimate of the cost to go from the plan's ending. Thus, how to find the accurate estimation for the cost to go is a critical factor without planning the path all the way to the goal point. This section investigate various cost-to-go functions based on various estimations.

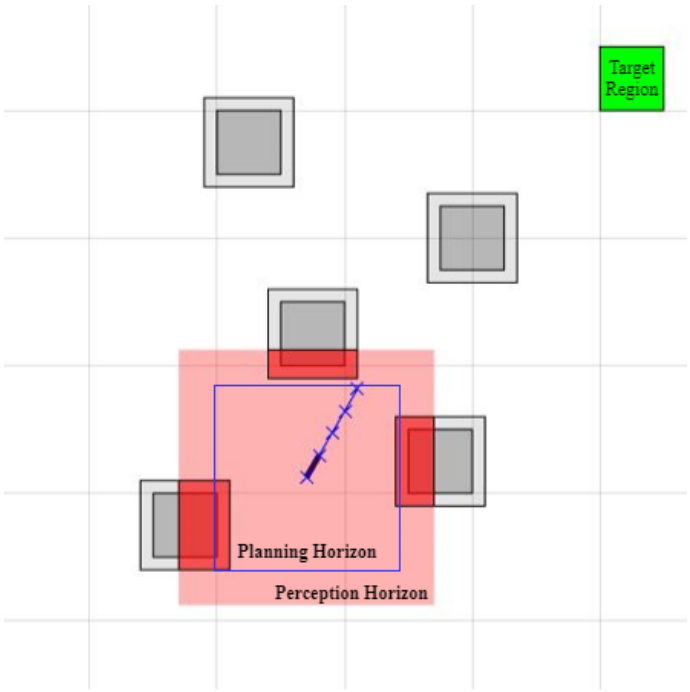


Figure 4.1: Illustration for Receding Horizon Control strategy with perception range of 1.5m and planning horizon with 0.8s. The obstacles are inflated accordingly and the grey parts are unsensed by the vehicle.

The illustration of the RHC is demonstrated in Fig. 4.1 where several horizons are introduced for several purposes. The red area represents the perception area, thus the observed obstacles are label in red with inflation processed. The blue squared are represents the planning Horizon where certain cost function is optimized and the execution horizon is that only one step of the control sequence would be executed. The overall procedure of RHC to be employed is as follows:

1. The path planning formulation in Chap. 3 is used to optimize, starting from current state $\mathbf{s_{cur}}$ and current time t to stop at the time $t+H$ at target state $\mathbf{s_{tar}}$. H is the prediction horizon.

2. Only implement the first step of the control sequence resulting from the optimizer.

3. Repeat until reach the goal.

4.3 Fixed Horizon Minimum Time Controller

The fixed horizon controller was actually presented in [41], and implemented in Chap. 3. We show that such planning strategy actually can fit into the RHC framework where a large fixed horizon H is adopted and the goal is guaranteed to be reach within this horizon. Recap on the minimization objective stated in Sec. 3.7:

$$\begin{aligned} \min_{\mathbf{u}(\cdot)} \phi_1(\mathbf{b_f}, \mathbf{t}) \quad &= \sum_{i=1}^{H} b_{f,i} t_i, \\ s.t. \qquad & Eqn.\ (3.15) \quad (Goal\ achievement) \\ & Eqn.\ (3.16) \\ & Eqn.\ (3.10) \quad (Collision\ avoidance) \\ & Eqn.\ (3.12) \quad (Obstacle\ avoidance) \\ & Eqn.\ (3.6) \quad (Discretized\ dynamics) \end{aligned} \tag{4.1}$$

where H denotes the size of the horizon, and there exists H binary decision variables $b_f \in \{0,1\}$ deciding at which time step the goal is reached. The optimization should be constrained by various planning requirements listed in Eqn. 4.1.

Such formulation optimizes the minimum arrival-time path to the goal. However, experiment shows that the computation burden grows quickly as the path length and environment complexity increases. Thus, to handle this problem, RHC with small horizon

H is adopted and some modifications would also be introduced to handle the additional problems by using such solution.

4.4 Simple Goal Cost Estimation

As we use the small horizon, the goal cannot be reached within the planning horizon, thus the cost function has to add the terminal penalty as the guidance. Before we discuss the cost function, the obstacle inflation is explained as follows. Illustrated in the Fig. 4.2, if the width of the rectangle obstacle is smaller compared to the discretized step length during the planning, the operation of the inflation is required for the vehicle safety. Such scenario happens in [41] while the thin obstacle is appeared in the environment. The inflation scheme is little different from that demonstrated in the Sec. 3.9 where the collision is allowed in the iterative algorithm.

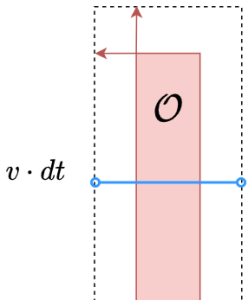


Figure 4.2: Thin obstacle might be 'override' by the path during the planning due to the large time difference. Blue line represents the magnitude of the travel length in one time step for the vehicle.

Here, square obstacles are adopted in the environment and its inflation is illustrated in Fig. 4.3 where the cutting corner scenario is analyzed for deciding the inflation length d. The extreme situation is that the path cut the corner with 45-degree attack angle in one time step with the length of $v \cdot dt$ where the path indeed touch the real obstacle's corner. As a result, the inflation length d should at least taking care of this extreme situation as its minimum value should equal to $\frac{v \cdot dt}{2\sqrt{2}}$. Such technique can also be extended into three dimension very straightforward as illustrated in [28].

As a result, the buffer area of the obstacle is allowed to cut in by the planned path, such happened as shown in Fig. 4.4 (a) (d) and Fig. 4.5 (a) (d) as the snapshots of

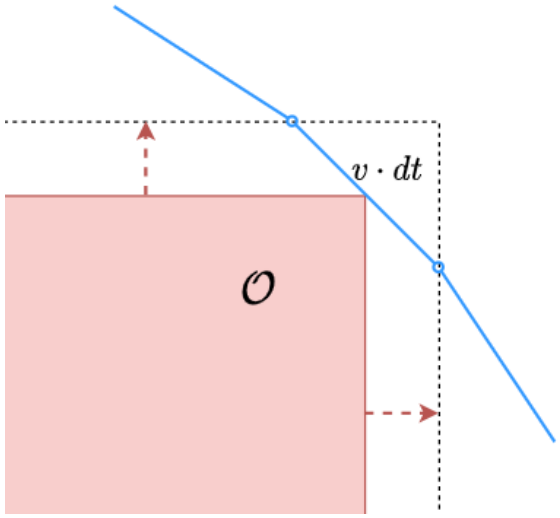


Figure 4.3: Obstacle has to inflate in each corner for avoiding the path result cutting into the corner. Blue line represents the magnitude of the travel length in one time step for the vehicle. Red dotted arrow denotes the inflation of the squared obstacle.

the whole planning. The grey label obstacles are undetected one by the vehicle where the lighter area denotes the inflation of the obstacle decided by the maximum vehicle speed and time difference dt. The red area denotes the perception area of the vehicle and the reddish rectangles are feed into the MILP as obstacles during the planning where the inflation process would prevent the path from colliding with the real obstacle configuration spaces.

MILP path planning is repeatedly applied within the time horizon H as this time window is sliding towards the goal. The optimizer would generate the sequence of control inputs at the time step k to the vehicle as $\{\mathbf{u}(k+i) \in \mathbf{R}^2 : i = 0,1,...,H-1\}$ which gives the future states as $\{\mathbf{x}(k+i) \in \mathbf{R}^2 : i = 0,1,...,H\}$. Only the first step of the plan is executed in this section. Similar to [4], the cost function is the terminal penalty term as follows:

$$\min_{\mathbf{u}(\cdot)} \phi_2(\mathbf{x}(k+H)) = \ell_b(\mathbf{x}_{goal} - \mathbf{x}(k+H)), \tag{4.2}$$

where $\ell_b(\cdot)$ denotes the 1-norm of the distance form the planned path's end to the goal. The constraints are also added as same as the Eqn. 4.1. According to [4], this choice of the terminal penalty can prevent the vehicle from reaching the goal when this

approximation doesn't reflect the real *executable* path. There are two scenarios, one is the line connecting the path's ending point to the goal is penetrating the obstacle as illustrated in Fig. 4.2, another is when the vehicle fell into the concave obstacle, only with the guidance of the terminal penalty, the vehicle would got trapped in this 'local minima'. Such behavior is quite similar to that when the potential field is adopted for the navigation.

The simulation results are shown in Fig. 4.4 and Fig. 4.5 with detection range of 1.5m and 2m, respectively. The planning horizon is 5 time steps that is 1s with dt equals to 0.2s. The whole computational time for the planning is reduced using the online fashion since only the surrounding obstacles are considered for MILP. More sophisticated planner would demonstrate in the next section to overcome the concave obstacles with the incorporation of the abstraction of the global information.

4.5 Modified Cost Point Estimation and RHC

As described in Sec. 1.1, motion planning problem is proved to be $\mathscr{NP}$-hard. There remains many well-known techniques to be applied in such problems, such as Probabilistic Road Maps [24], Rapidly-exploring Random Trees [30] and Cell Decomposition methods [50]. All of these techniques are aimed to reduce the dimensionality of the problem by sampling the possible control actions. However, most of these results are not optimal though many iterative algorithms have been proposed to prove that it can reach asymptotically optimal under certain constraints. Thus, abstracted global information should be incorporate when the planning phase is conducting to avoid the local minima stated in the former section. As a result, similar to [27, 28], the planning part consists of cost estimation and a path planning phase.

The resolution of the control architecture is shown in Fig. 4.6. Similar to that in Chap. 4, within the planning horizon H, the MILP optimized according to the vehicle dynamics for H steps. This cost estimates the time to reach the goal from the cost point x_{vis}, whose cost-to-go is abstracted previously. This control architecture includes two different levels of resolutions which handles different levels' challenges and exploits the planning problem's structure. On the global sense, which can also referred as long time-scale, a successful controller only need to decide which corridor to go through

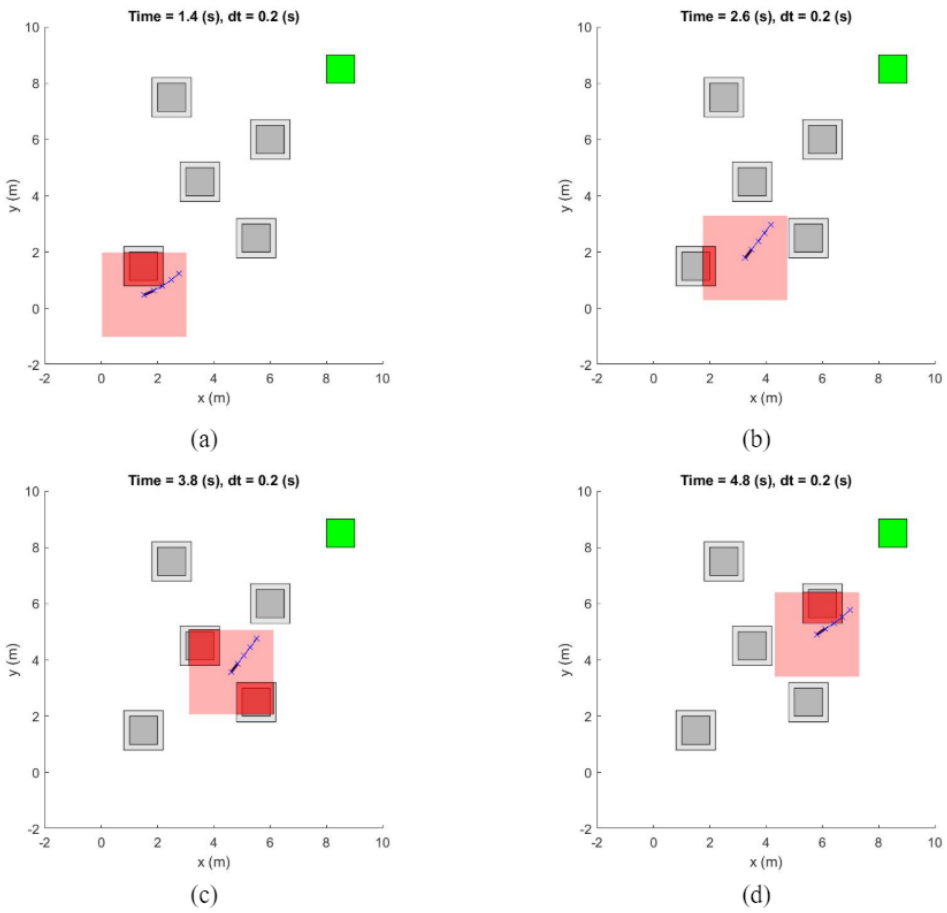


Figure 4.4: The snapshots of the planning scenarios using RHC while the detection range is 1.5m. (a) The snapshot when time elapsed is 1.4 s. (b) The snapshot when time elapsed is 2.6s. (c) The snapshot when time elapsed is 3.8s. Due to the restriction of the detection range, not many obstacles are detected in the relatively cluttered area. (d) The snapshot when time elapsed is 4.8s.

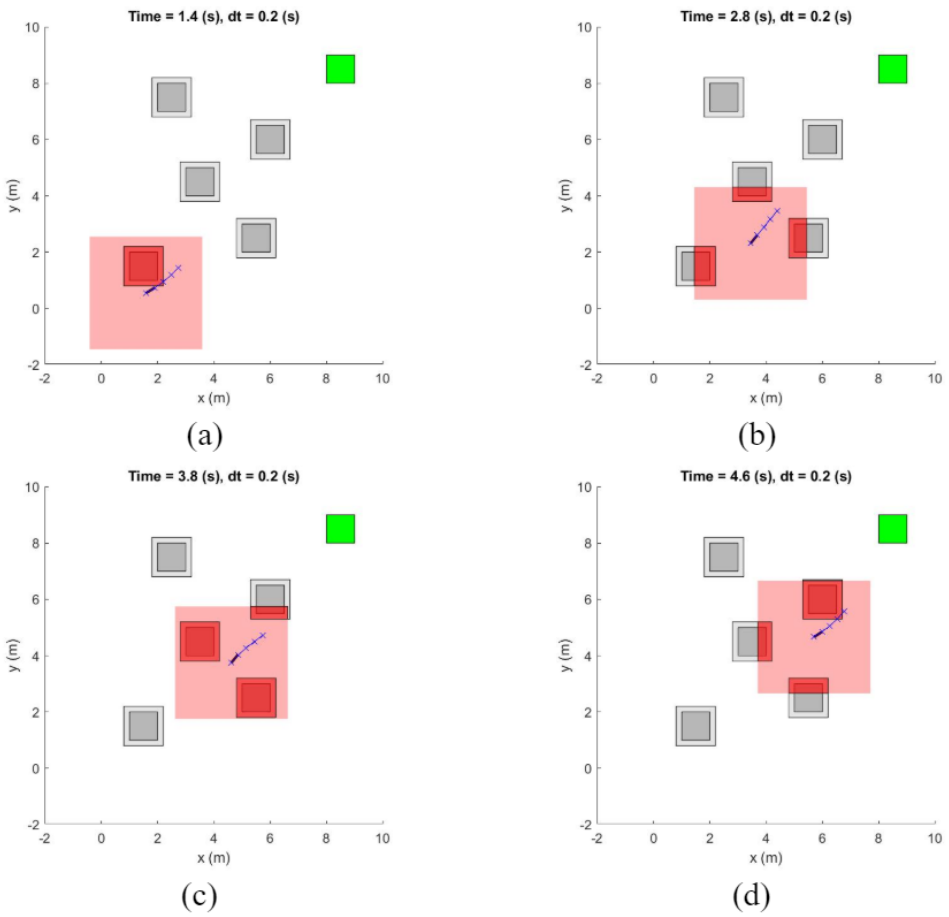


Figure 4.5: The snapshots of the planning scenarios using RHC while the detection range is 2m. (a) The snapshot when time elapsed is 1.4 s. (b) The snapshot when time elapsed is 2.8s. (c) The snapshot when time elapsed is 4.2s. With the increase in the detection range, more area is included for the MILP optimization. (d) The snapshot when time elapsed is 4.8s.

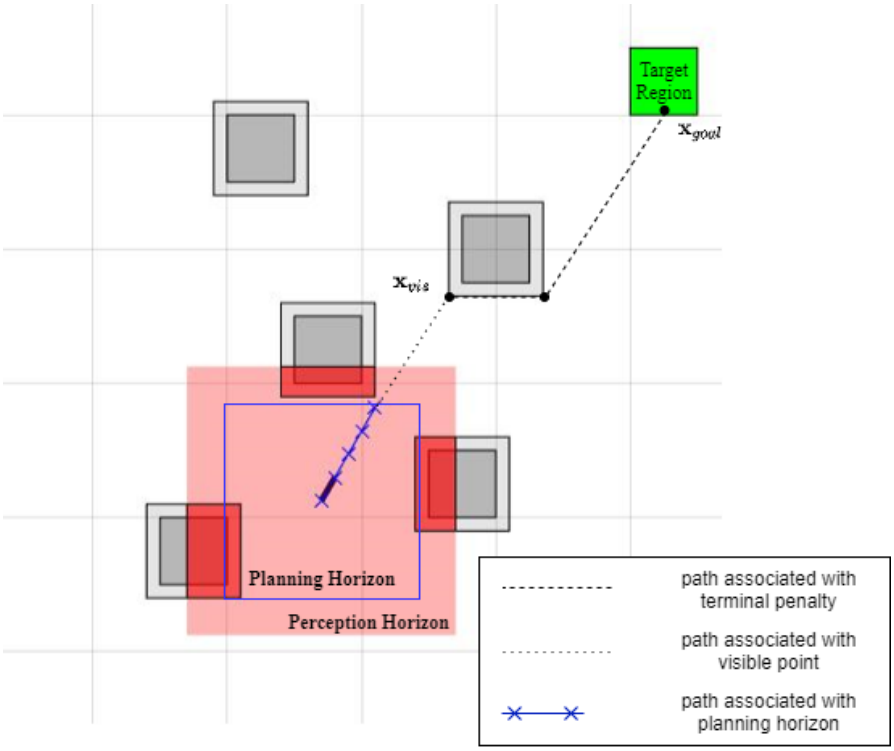


Figure 4.6: Resolution levels for modified RHC planner

or which obstacle to passage by under the objectives for shortest-distance or minimum time requirements. On a short time-scale, which is also refer to local sense, a successful controller should plan the kinodynamically suitable path for the vehicle to deal with the nearby obstacles. These various resolution levels of RHC make the planner more robust and adjustable to the environment.

According to the modified RHC strategy, the modified obejctive can be given as follows:

$$\min_{\mathbf{u}(\cdot)} \phi_3(\mathbf{x}(k+H)) = \min_{\mathbf{u}(\cdot)} \frac{\ell_b(\mathbf{x}_{vis} - \mathbf{x}(k+H))}{v_{max}} + C_{vis}, \tag{4.3}$$

where C_{vis} denotes the estimated cost from $\mathbf{x}_{vis}$ to the goal which could be re-estimated very quickly if the environment is changing.

4.5.1 Cost Map

To build up the cost map which includes the set of cost estimate points, the environment needs abstracted. This section, the 2D obstacle is considered for building up the cost map. The corner points of the inflated rectangles, along with the start the goal points

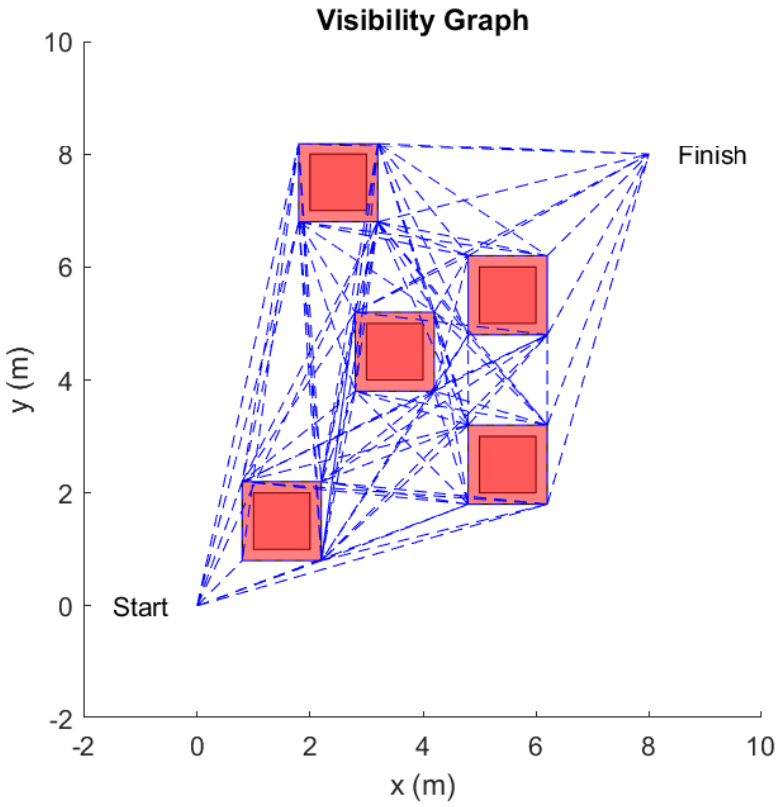


Figure 4.7: Visibility graph

are selected for being the estimation vertices.

The Visibility Graph is built as the first step for the coarse cost map computation in an obstacle field. This is a good approximation because of its fast computation and the straight-line estimation without considering the dynamics which can be easily rerun under the changing environment. Fig. 4.7 shows the connectivity of the estimate points when the obstacles are appeared in the environment, both start and goal points are connected into this graph. Each edge should not collide with the obstacles as each vertex is 'visible' to each other. Such result can easily be applied into the shortest-path algorithm for the cost estimation for each visibility point $\mathbf{x}_{vis}$.

The shortest path is planned using the goal point sourced Dijkstra's algorithm, as a result, we can find out the shortest path from the goal to each vertex in the visibility graph. The coarse shortest distance path is the one with goal and start point as two terminals. Fig. 4.8 shows the shortest path extracted from the visibility graph which is sourced from the goal point.

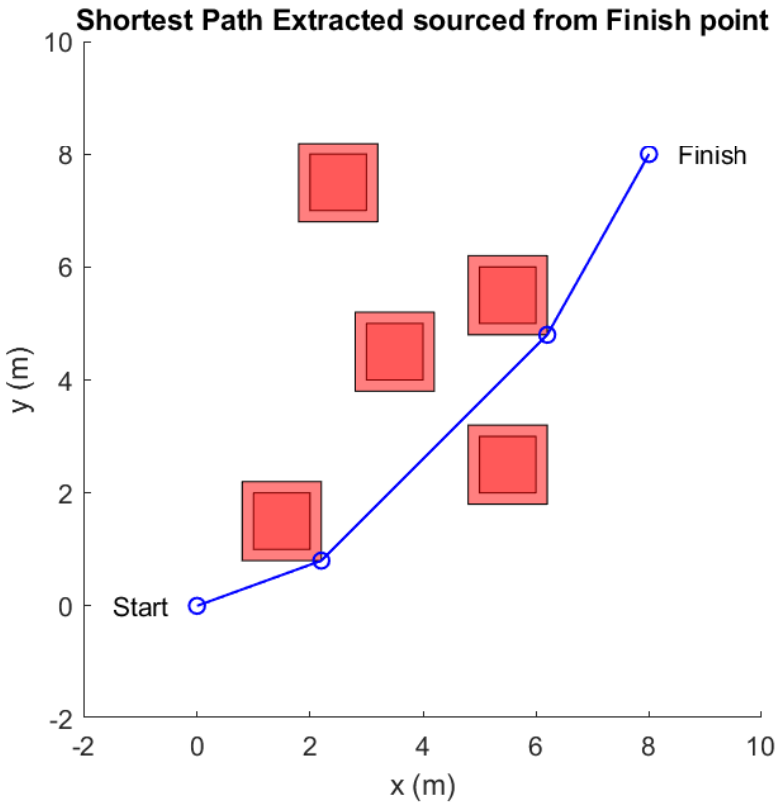


Figure 4.8: Shortest path sourced from the goal to the start point

4.5.2 Example: Local minima environment

With the help of the long-time scale estimation, the visibility estimation constraint ensures that the length between the planning horizon's end to the visibility point is the *doable* path. Also, the vehicle whould not fall into the entrapment because the shortest path guaranteed that the guidance path would not got 'trapped' in the local minima area. This can be proved by the contradiction: if the shortest path falls into the entrapment, then the additional effort would be made to escape from it, thus, such path is not the shortest. This section provides the efficiency for the vehicle to avoid the entrapment.

Implementation details

The problem is optimized by the MILP solver, based on branch-and-bound algorithm, implemented in the CPLEX [7]. The formulation of the problem is illustrated in AMPL [17]. The example of the single vehicle planning Receding Horizon configuration and data generation script are shown in Appendix. 7.3. After the solver completed the optimization, the script is written to analyse the output. The variables, such as planned

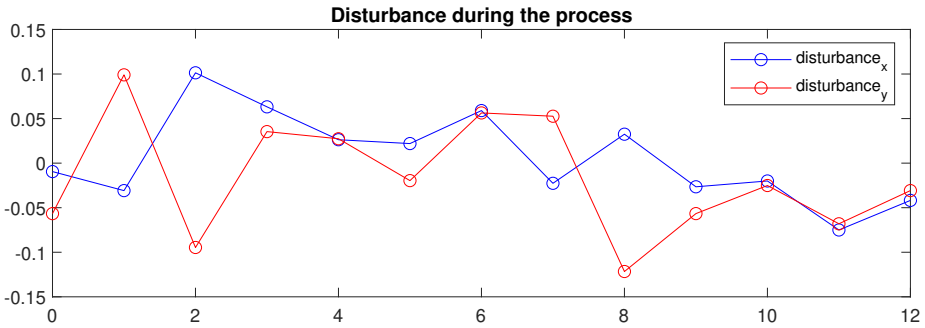


Figure 4.9: Disturbance of the whole process

positions, velocities, are extracted to the MATLAB space and then the plotting utilities can been used for figures. The simulation is based on the Windows PC equipped with 8xCPU@2.6GHz and 16G RAM. Due to the restrictions of the variables size of the demo version of the AMPL, only 500 variables are available for the whole model. Thus, the entrapment scenario is given with large time difference dt and small time step for shrinking down the size of the decision variables. The exterior disturbance generated in the x, y axis are demonstrated in the Fig. 4.9, repectively. The disturbance are bound around the zero with margins of $0.1 * u_{max}$ which shows in the Appendix. The multiple planning results using RHC is shown in Fig. 4.10. The dotted represents the inflation of the obstacles which has been stated in Sec. 3.9 where the green zone represents the goal area. All of these paths avoid the entrapment area with the guidance of the visibility estimate points with the help of the global shortest path.

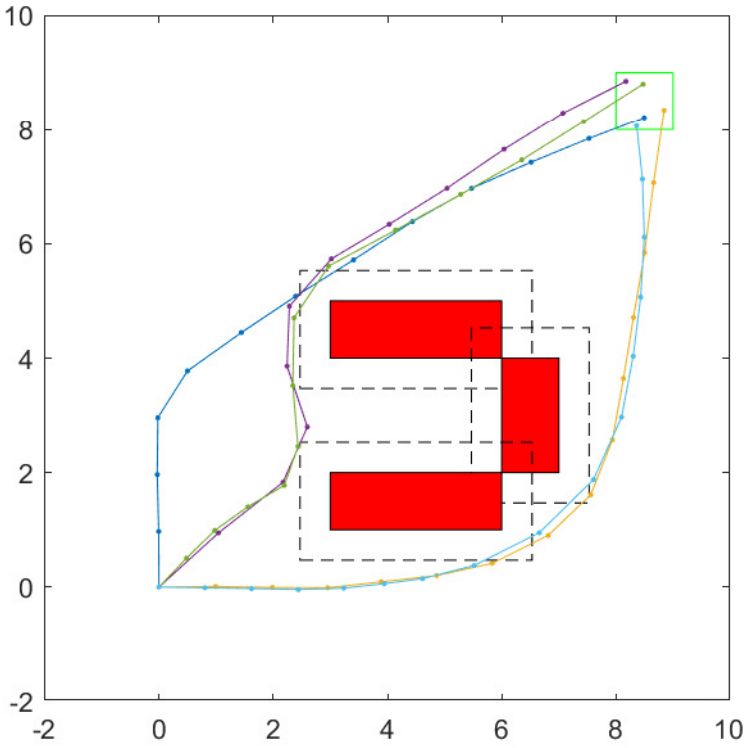


Figure 4.10: Various planning results don't fall into the entrapment area. The dotted rectangles represent the inflated obstacles according to the discrete time step of the simulation for obstacle avoidance as the hard safety requirement. These obstacles form an entrapment where vehicle would fall into using cost function shown in Sec. 4.4.

Chapter 5

Extended capabilities of MILP

5.1 Component of Trajectory Generation

This section will introduce how the finer trajectory can be generated after the coarse path, yet constrained to the object's dynamics, planned by the MILP using appropriate approximation. The simulation shows that the higher-degree approximation of the dynamics can generate more feasible path for trajectory to fit. In [8], the three dimensional dynamics of the mass-point model is presented and online trajectory generation is introduced. In [9], the the convex partition is adopted for the safe region segmentation in the environment, moreover, in [29], the safety corridor is also proposed for the quadrotor UAVs as the piece-wise flight corridor is generated to guarantee the obstacle avoidance.

Quadrotor UAV platforms are widely been applied both in the indoor and outdoor environment and various testbeds had been proposed in [21, 26, 36]. Multiple trajectory generation techniques for the quadrotor were also proposed in the last few years, such as minimum-snap [35] trajectory allows the micro quadrotor to fly in the highly constrained indoor environment. In [22], the swarm trajectory generation algorithm is also proposed for the safe distributed trajectories for the quadrotors.

Path planning, as the 'upstream' for the trajectory planning, its quality is critical for the final generated trajectory. The simulation and comparison is given in the Sec. 5.1.3 to show the fined approximated path is quite similar to the final generated trajectory based on the quadrotor UAV. The first two parts briefly introduced the quadrotor kinematics and dynamics.

5.1.1 Quadrotor Kinematics

Commonly, the kinematics of a quadrotor could be illustrated using few frames. The inertial frame, $\mathscr{A}$, defined by a_1, a_2 and a_3 with a_3 pointing upward. The body frame, $\mathscr{B}$, defined by b_1, b_2 and b_3 where actually b_1 and b_2 constructed the X-Y plane of the quadrotor and b_3 is perpendicular to this plane. The center of the body frame is attached to the center of the mass, C, of the quadrotor, shown in Fig. 5.1.

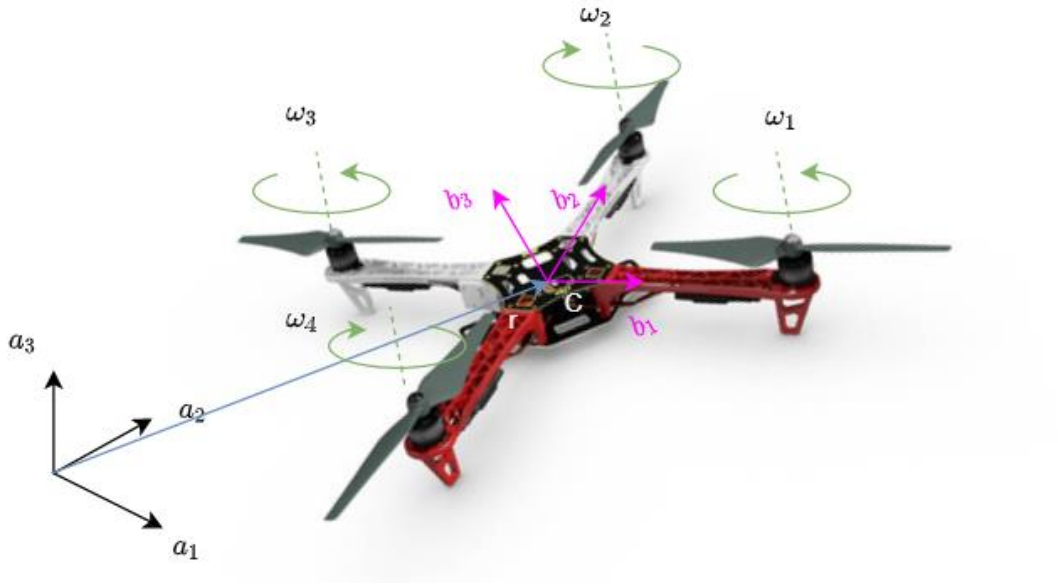


Figure 5.1: Body frame b_i and inertial frame a_i for the quadrotor. The pair of the rotors rotate in the same direction, ω_2 and ω_4 rotate counterclockwise while ω_1 and ω_3 rotate clockwise

To represent the displacement of the quadrotor in the inertial frame, vector $\mathbf{r}$ is given and defined as $[x_b, y_b, z_b]^\top$. In this work, to model the rotation of the quadrotor, though there exist many conventions of using Euler angles, the Z-X-Y Euler angles is used to model the rotation in the inertial frame. To get from $\mathscr{B}$ to $\mathscr{A}$, first rotate about a_3 by the yaw angle, ψ, then rotate about a_1 by the roll angle, ϕ and finally rotate about a_2 by the pitch angle, θ. The rotation matrices for these axes are shown below, respectively.

$$\mathrm{R}_z(\psi) = \begin{bmatrix} c(\psi) & -s(\psi) & 0 \\ s(\psi) & c(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{5.1}$$

$$\mathrm{R}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c(\phi) & -s(\phi) \\ 0 & s(\phi) & c(\phi) \end{bmatrix}, \tag{5.2}$$

$$\mathrm{R}_y(\theta) = \begin{bmatrix} c(\theta) & 0 & s(\theta) \\ 0 & 1 & 0 \\ -s(\theta) & 0 & c(\theta) \end{bmatrix}, \tag{5.3}$$

According to the Z-X-Y Euler angles sequence, the rotation matrix for transforming coordinates from $\mathscr{B}$ to $\mathscr{A}$ is produced [39]:

$$^{\mathscr{A}}[R]_{\mathscr{B}} = \begin{bmatrix} c(\theta)c(\psi) - s(\phi)s(\psi)s(\theta) & -c(\phi)s(\psi) & s(\theta)c(\psi) + s(\phi)s(\psi)c(\theta) \\ s(\psi)c(\theta) + c(\psi)s(\phi)s(\theta) & c(\phi)c(\psi) & s(\psi)s(\theta) - c(\psi)s(\phi)c(\theta) \\ -c(\phi)s(\theta) & s(\phi) & c(\phi)c(\theta) \end{bmatrix}, \tag{5.4}$$

where $c(\cdot)$ and $s(\cdot)$ denotes cosine and sine function, respectively. Moreover, the angular velocity of the robot in the body frame are given by p, q and r, these values are related to the derivatives of the roll, pitch and yaw angles as follows:

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} c(\theta) & 0 & -c(\phi)s(\theta) \\ 0 & 1 & s(\phi) \\ s(\theta) & 0 & c(\phi)c(\theta) \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}. \tag{5.5}$$

5.1.2 Quadrotor Dynamics

Newton's Equations of Motion

Let $\mathbf{r}$ shown in Fig. 5.1 denote the position of the center of mass C in $\mathscr{A}$. Then the equation regarding the acceleration of the mass is illustrated as:

$$m\ddot{\mathbf{r}} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + ^{\mathscr{A}}[R]_{\mathscr{B}} \begin{bmatrix} 0 \\ 0 \\ F_1 + F_2 + F_3 + F_4 \end{bmatrix}, \tag{5.6}$$

where each rotor generates the force F_i in the $\mathbf{b}_3$ direction. The gravity is always applied in the direction of $-\mathbf{a}_3$. Here we can define the first input $u_1 = \sum_{i=1}^{4} F_i$ as the total force produced by the rotors.

Euler's Equations of Motion

According to the Fig. 5.1, the brief body frame with local coordinate is demonstrated in Fig. 5.2. The torque $\tau_{\mathbf{b}_i}$ along three body axes are calculated as follows:

$$\begin{aligned} \tau_{\mathbf{b}_1} &= d(f_2 - f_4), \\ \tau_{\mathbf{b}_2} &= d(f_3 - f_1), \\ \tau_{\mathbf{b}_3} &= -M_1 + M_2 - M_3 + M_4, \end{aligned} \tag{5.7}$$

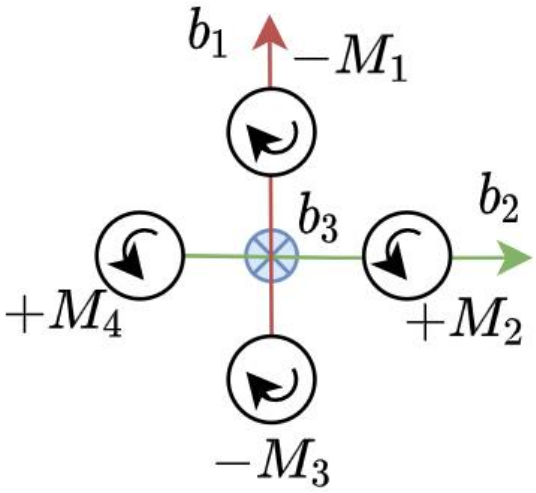


Figure 5.2: Quadrotor body frame with propeller rotations

where d denotes the length of the identical quadrotor arm. f_i denotes the force generated by each propeller. The angular acceleration using the Euler equations is as follows:

$$\mathscr{I} \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} d(f_2 - f_4) \\ d(f_3 - f_1) \\ -M_1 + M_2 - M_3 + M_4 \end{bmatrix} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \mathscr{I} \begin{bmatrix} p \\ q \\ r \end{bmatrix}, \tag{5.8}$$

where $\mathscr{I}$ denotes the moment of inertia for the quadrotor and defined as:

$$\mathscr{I} = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}.$$

The Eqn. 5.9 can be rewrite as:

$$\mathscr{I} \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} 0 & d & 0 & -d \\ -d & 0 & d & 0 \\ \gamma & -\gamma & \gamma & -\gamma \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \mathscr{I} \begin{bmatrix} p \\ q \\ r \end{bmatrix}, \tag{5.9}$$

where γ indicates the relationship of the lift and drag defined as $\gamma = \frac{k_M}{k_F}$. Accrodingly, we can define the second input u_2 as:

$$u_2 = \begin{bmatrix} 0 & d & 0 & -d \\ -d & 0 & d & 0 \\ \gamma & -\gamma & \gamma & -\gamma \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix} \tag{5.10}$$

5.1.3 Trajectory generation based on MILP path planner

As the 'upstream' of the trajectory planning module, the path planner module is critical and [21, 36, 29, 22] proposed various solutions for incorporating these two modules together for more accurate and agile outcome. Thanks to the capability of the quadrotor that can hover in the air, more stable and safe planning control strategy can be adopted. Fig. 5.3 demonstrates the components that our approach to generate the feasible and minimum-snap trajectory. We first use the user input to build up the map with obstacles, start and goal configurations. Then the path and trajectory planner are in cascade structure. Finally the desired trajectory is executed by the controller to output the control input to the quadrotor. The dotted components below are adopted in the local planning scheme such as MPC planner, where only the restrictively local information is given to the vehicle. In this simulation, the global planning scheme is adopted.

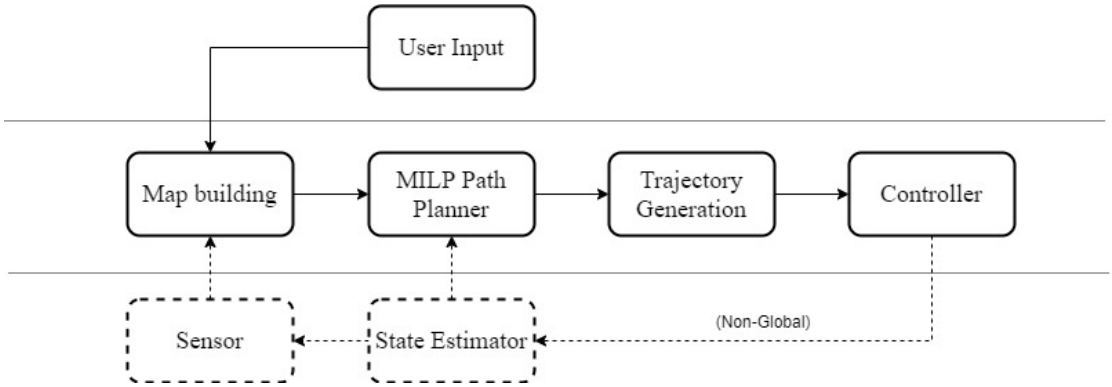


Figure 5.3: Components of the quadrotor trajectory generation. Due to the limitation of linear approximation of using MILP, the MILP path planning module could serve as the fast, rough planning module in the trajectory planner while finer kinodynamical trajectory is token care of in the succeed module.

For the low level control, Fig. 5.4 shows the position and attitude loops according to [42]. The control problem is to determine the inputs u_1, u_2 to hover or to follow the desired trajectory z^{des}. The position and orientation control is describe in [36] which we won't discuss much in this section.

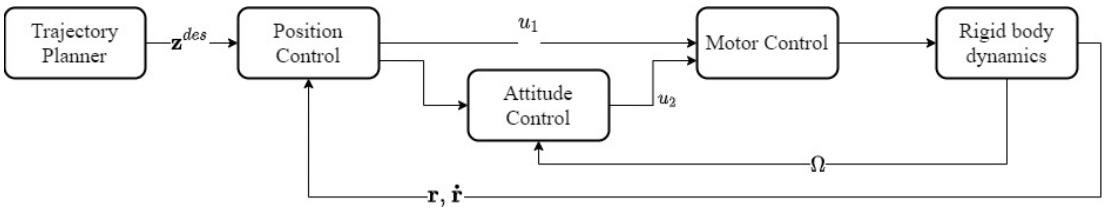


Figure 5.4: The position and attitude control loops for the Quadrotor UAV

The simulations are intended to verify the validity of MILP path planning as a component for further trajectory planner. Testing scenario is set to be similar to the woods and the quadrotor is restricted in the yaw angle movement. The comparison of the two results is shown in Fig. 5.5, the path with green crosses is MILP result with 10-side approximation and 0.2s in dt for the quadrotor, the blue dotted path is the real trajectory generated by the controller while the red dotted is the desired trajectory based on the former planned MILP path.

The path is configured by the user input as the start point is $[0,0,0]^T$, the goal with $[8,8,2]^T$. The wood-like obstacles are scattered in the environment. The velocity and position information of the trajectory planner are provided in the Fig. 5.6. Moreover,

the snapshots of the quadrotor UAV (crazyflie) during the operation is shown in the Fig. 5.7.

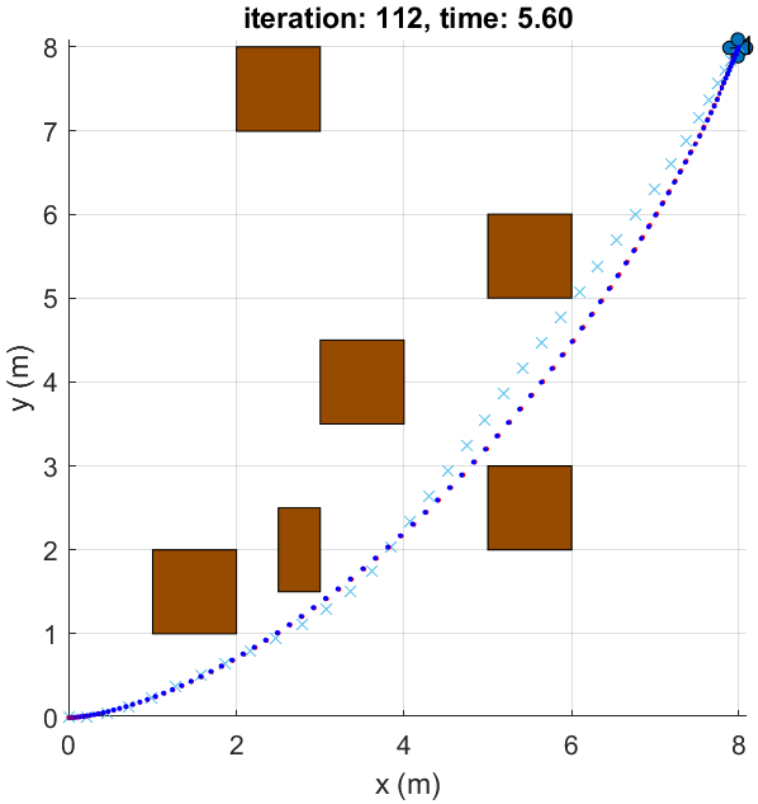


Figure 5.5: Executed Quadrotor UAV trajectory, in blue dotted line, is generated based on the MILP path planning result, in green crossed line.

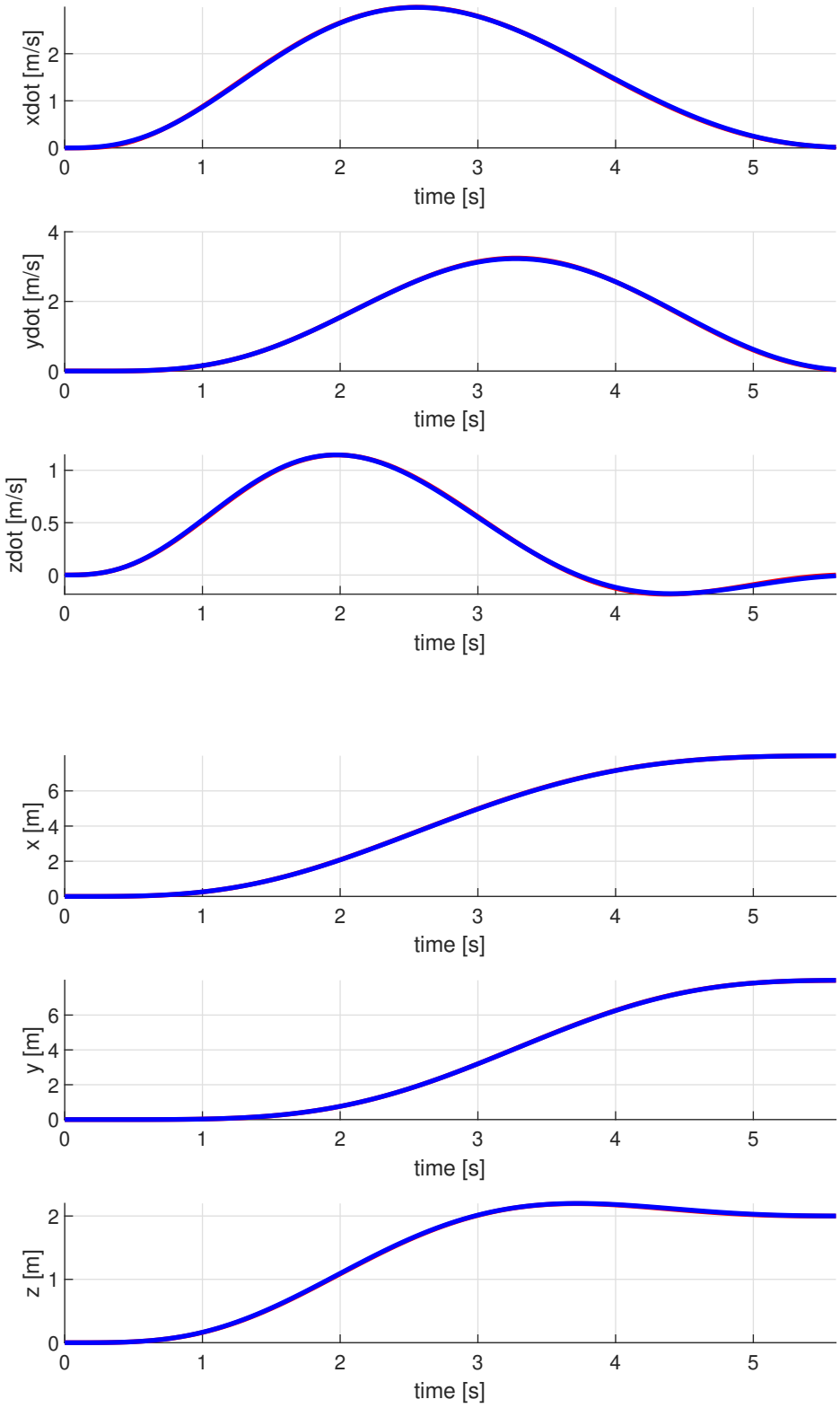


Figure 5.6: Velocity and position of the generated trajectory

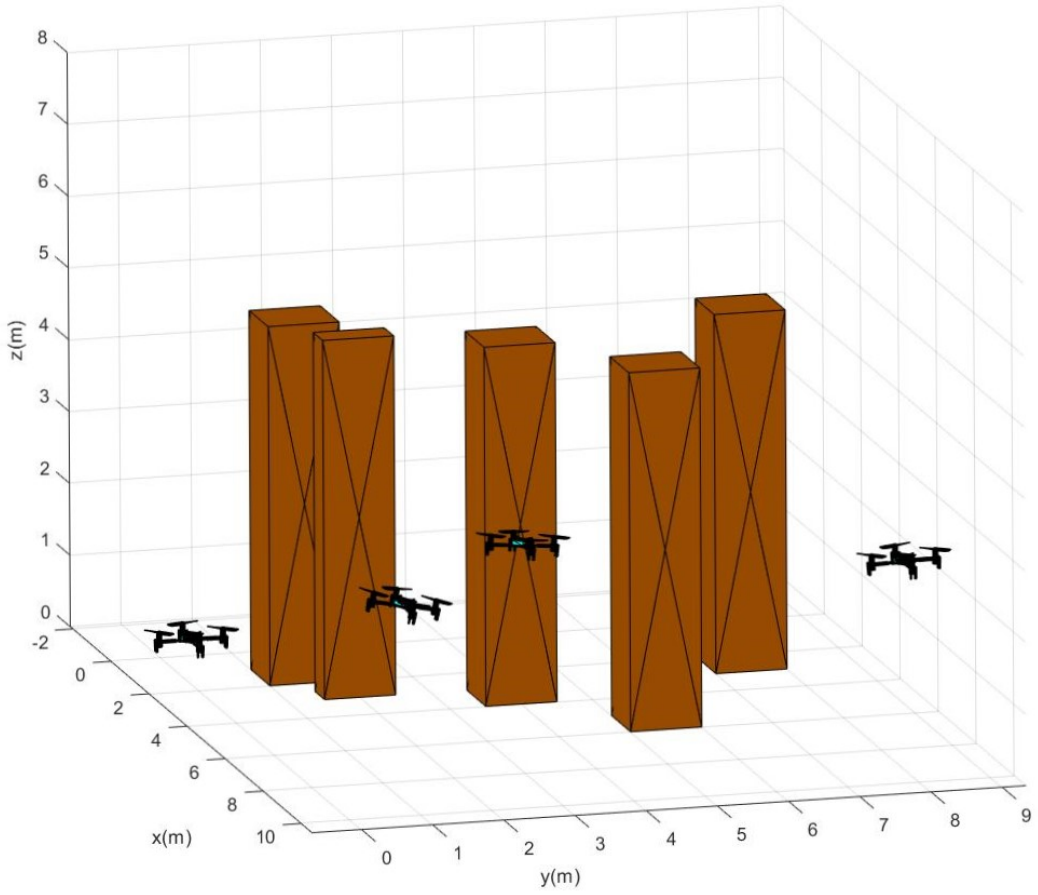


Figure 5.7: Snapshots of the crazyflie along the trajectory in the wood scenario. The roll and pitch angles are shown during the process. The wood in the bottom right is zoomed in in order to show the trajectory clearly.

5.2 Extension to the 3D applications

The 2D implementation of the MILP path planning can be easily extended to the 3D application, inspired by [53], the city 3d climber application is introduced in this section. The intuitive is simple and clear as the 2D planning can be transformed to the 3D result using folding as illustrated in Fig. 5.8. The more constraints could be added to the MILP formulation described in the Sec. 3. For example, the disjunction point along the path between two surfaces should be specified, in Fig. 5.8, the separation point is specified as $[4,4]^\top$ in the 2D environment, yet the velocity should also be constrained according to the vehicle's dynamics constraints. We can also define multiple waypoints for the vehicle to track and convert its path to the 3D results. The order of the waypoints can be pre-assigned or optimized as described in 3.6. After we folded the 2D path along the intersected line, the 3D results are demonstrated in the Fig. 5.9 and 5.10 in front and rear views, respectively.

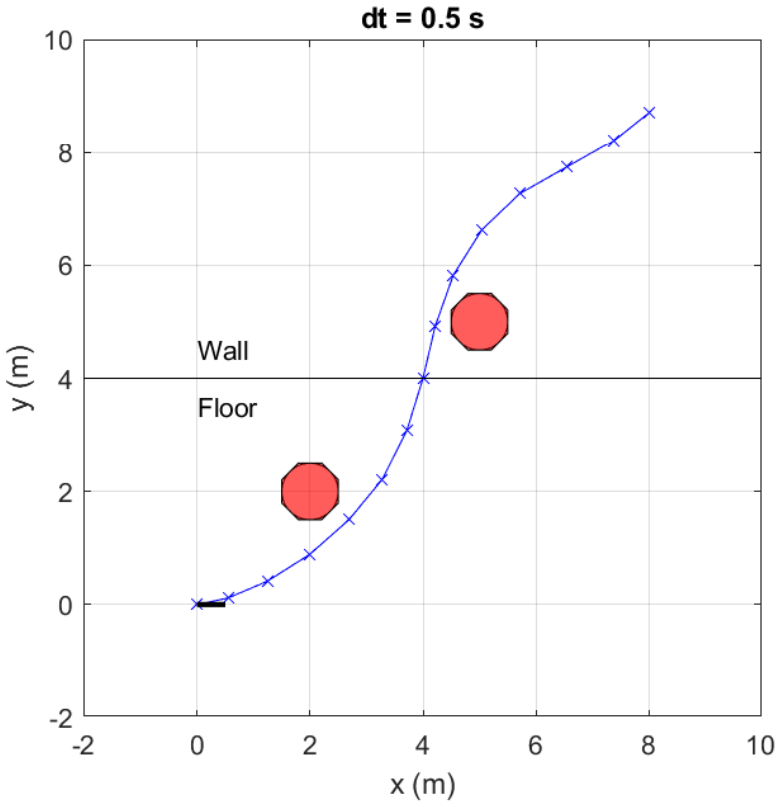


Figure 5.8: Planned Path in 2D with specific disjunction point $(4,4)^\top$ on the predefined wall and floor separation line.

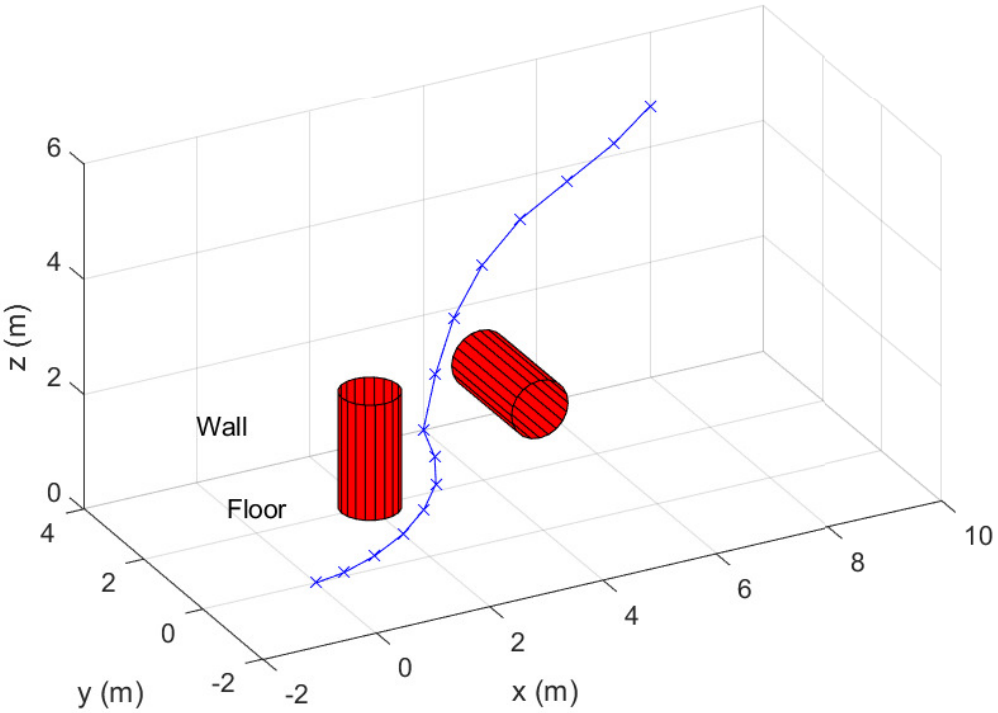


Figure 5.9: Front view of the city climber path

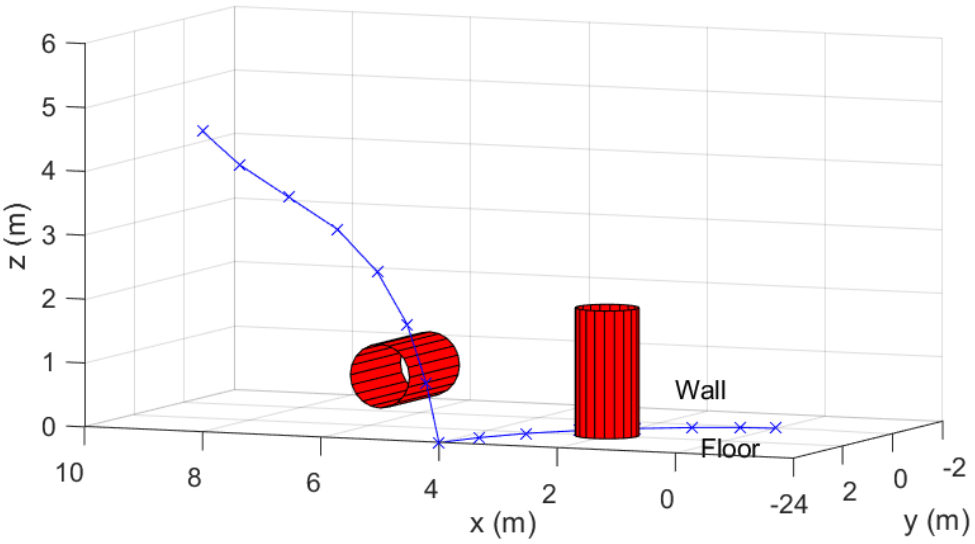


Figure 5.10: Rear view of the city climber path

Chapter 6

Conclusion and Future work

A comprehensive study of the MILP path planning is conducted in this project. The whole layout of the project is from the vanilla's form to more sophisticated one. Extensive simulations are carried out to validate the proposed methods. Various planning patterns are studies such as local planning, global planning, as well as the combination of these two.

In Chap. 3, both global one-time and iterative MILP planner is studied under various practical constraints. The usage of the binary decision variables in the MILP formulation is the critical factor for successfully solving the MILP path planning problem and also in other domains. The computation performance is improved by using the iteration algorithm. However, in the cluttered environment, the number of the newly selected time step would converge to that of the original MILP planner and the growing obstacle method would result in the growing occupation of the free configuration space which would finally lead to the planning failure.

In Chap. 4, an online planning fashion is studied with the introduce of the RHC framework. By only considering the perception horizon, the computation burden would drop a lot compared to the global planning fashion. However, the MILP objective function needs considering deliberately since the the area outside the perception horizon is unknown to the vehicle. Several cost functions are also studied and implemented, finally with the help of the abstracted global information, the vehicle could avoid the entrapment in the environment.

In Chap. 5, the extended capabilities of the MILP is presented. The MILP planner could serve as the rough approximation for the quadrotor UAV and provide the guidance

for the flight. Also, the two dimensional problem could successfully converted to the three-dimensional application by carefully predefining the parameters.

The future extension of this project is wide but also challenged. To overcome the error of the dynamics approximation by using the MILP, more complex formulation needs proposing which would result in nonlinear and larger constraint variable space. This would result in increasing computation time and relying on the advance of the solver techniques. Moreover, the online navigation still cannot deal with the environment successfully sometimes. There are many scenarios such as corner, wall crash and cross road need studying and the unified strategy needs proposing. MILP is suitable for the quick modelling and estimation of the environment because of its preferred properties and existing techniques. Though many methods added many safety constraints in the MILP to compensate its rough approximation of the model, to improve the accuracy and handle various vehicle dynamics, the nonlinear programming needs studying in the future for more robust results. Moreover, the 3D scenario for the planning is different from that of the 2D ones, appropriated modelling needs studied and many constraints also need extending to operate in the 3D environment.

REFERENCES

[1] Pramod Abichandani, Hande Benson, and Moshe Kam. Mathematical programming approaches for multi-vehicle motion planning: Linear, nonlinear, and mixed integer programming. *Foundations and Trends in Robotics*, 2(4):261–338, 2013.

[2] Pramod Abichandani, Gabriel Ford, Hande Y. Benson, and Moshe Kam. Mathematical programming for multi-vehicle motion planning problems. In *2012 IEEE International Conference on Robotics and Automation*, pages 3315–3322, 2012.

[3] Senthil Hariharan Arul and Dinesh Manocha. Dcad: Decentralized collision avoidance with dynamics constraints for agile quadrotor swarms. *IEEE Robotics and Automation Letters*, 5(2):1191–1198, 2020.

[4] J. Bellingham, A. Richards, and J.P. How. Receding horizon control of autonomous aerial vehicles. In *Proceedings of the 2002 American Control Conference (IEEE Cat. No.CH37301)*, volume 5, pages 3741–3746 vol.5, 2002.

[5] Yossi Bukchin and Michal Tzur. A new milp approach for the facility layout design problem with rectangular and l/t shaped departments. *11th IMHRC Proceedings*, 9, 2010.

[6] John F. Canny. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA, USA, 1988.

[7] IBM ILOG Cplex. V12. 1: User's manual for cplex. *International Business Machines Corporation*, 46(53):157, 2009.

[8] Kieran Culligan. Online trajectory planning for uavs using mixed integer linear programming. 07 2007.

[9] Robin Deits and Russ Tedrake. Efficient mixed-integer planning for uavs in cluttered environments. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 42–49, 2015.

[10] Spletzer J. Hsieh A. Derenick, J. An optimal approach to collaborative target tracking with performance guarantees. In *J Intell Robot Syst*, pages 47–67, 2009.

[11] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[12] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Practical search techniques in path planning for autonomous driving. In *Proceedings of the First International Symposium on Search Techniques in Artificial Intelligence and Robotics (STAIR-08*, 2008.

[13] M.G. Earl and R. D'Andrea. Iterative milp methods for vehicle-control problems. *IEEE Transactions on Robotics*, 21(6):1158–1167, 2005.

[14] C. Floudas. *Nonlinear and Mixed-Integer Optimization: Fundamentals and Applications.* Oxford University Press, Inc., 1995.

[15] Yacef Fouad, Laid Degaa, and Mustapha Hamerlain. Energy-efficiency path planning for quadrotor uav under wind conditions. pages 1133–1138, 06 2020.

[16] Yacef Fouad, Nassim Rizoug, Omar Bouhali, and Mustapha Hamerlain. Optimization of energy consumption for quadrotor uav. 09 2017.

[17] Robert Fourer, David M. Gay, and Brian W. Kernighan. Ampl: A mathematical programing language. In Stein W. Wallace, editor, *Algorithms and Model Formulations in Mathematical Programming*, pages 150–151, Berlin, Heidelberg, 1989. Springer Berlin Heidelberg.

[18] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and TomÃ¡s Lozano-PÃ©rez. Integrated task and motion planning. *Annual Review of Control, Robotics, and Autonomous Systems*, 4(1):265–293, 2021.

[19] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2021.

[20] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[21] Gabriel M. Hoffmann, Haomiao Huang, Steven L. Waslander, and Claire J. Tomlin. Precision flight control for a multi-vehicle quadrotor helicopter testbed. *Control Engineering Practice*, 19(9):1023–1036, 2011.

[22] Wolfgang Hönig, James A. Preiss, T. K. Satish Kumar, Gaurav S. Sukhatme, and Nora Ayanian. Trajectory planning for quadrotor swarms. *IEEE Transactions on Robotics*, 34(4):856–869, 2018.

[23] A. Jadbabaie, J. Primbs, and J. Hauser. Unconstrained receding horizon control with no terminal cost. In *Proceedings of the 2001 American Control Conference. (Cat. No.01CH37148)*, volume 4, pages 3055–3060 vol.4, 2001.

[24] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[25] Dexter C. Kozen. *Depth-First and Breadth-First Search*, pages 19–24. Springer New York, New York, NY, 1992.

[26] Vijay Kumar and Nathan Michael. Opportunities and challenges with autonomous micro aerial vehicles. *The International Journal of Robotics Research*, 31(11):1279–1291, 2012.

[27] Y. Kuwata and J. How. Receding horizon implementation of milp for vehicle guidance. In *Proceedings of the 2005, American Control Conference, 2005.*, pages 2684–2685 vol. 4, 2005.

[28] Yoshiaki Kuwata. Real-time trajectory design for unmanned aerial vehicles using receding horizon control. 12 2013.

[29] Shupeng Lai, Menglu Lan, and Ben M. Chen. Efficient safe corridor navigation with jerk limited trajectory for quadrotors. In *2018 37th Chinese Control Conference (CCC)*, pages 10065–10070, 2018.

[30] Steven M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical report, 1998.

[31] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, USA, 2006.

[32] J. Leishman. The breguet-richet quad-rotor helicopter of 1907. 2001.

[33] Lozano-Perez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, C-32(2):108–120, 1983.

[34] A. Makhorin. Glpk (gnu linear programming kit).
Available at `http://www.gnu.org/software/glpk/glpk.html`.

[35] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE International Conference on Robotics and Automation*, pages 2520–2525, 2011.

[36] Nathan Michael, Daniel Mellinger, Quentin Lindsey, and Vijay Kumar. The grasp multiple micro-uav testbed. *IEEE Robotics Automation Magazine*, 17(3):56–65, 2010.

[37] Panos M. Pardalos and Thelma D. Mavridou. *Simulated annealingSimulated Annealing*, pages 3591–3593. Springer US, Boston, MA, 2009.

[38] Scott Drew Pendleton, Hans Andersen, Xinxin Du, Xiaotong Shen, Malika Meghjani, You Hong Eng, Daniela Rus, and Marcelo H. Ang. Perception, planning, control, and coordination for autonomous vehicles. *Machines*, 5(1), 2017.

[39] Caitlin Powers, Daniel Mellinger, and Vijay Kumar. *Quadrotor Kinematics and Dynamics*, pages 307–328. Springer Netherlands, 2015.

[40] A. Richards and J. How. Mixed-integer programming for control. In *Proceedings of the 2005, American Control Conference, 2005.*, pages 2676–2683 vol. 4, 2005.

[41] A. Richards and J.P. How. Aircraft trajectory planning with collision avoidance using mixed integer linear programming. In *Proceedings of the 2002 American Control Conference (IEEE Cat. No.CH37301)*, volume 3, pages 1936–1941 vol.3, 2002.

[42] Francesco Sabatino. Quadrotor control: modeling, nonlinearcontrol design, and simulation. Master's thesis, KTH, Automatic Control, 2015.

[43] Kumara Sastry, David Goldberg, and Graham Kendall. *Genetic Algorithms*, pages 97–125. Springer US, Boston, MA, 2005.

[44] Tom Schouwenaars, Bart De Moor, Eric Feron, and Jonathan How. Mixed integer programming for multi-vehicle path planning. In *2001 European Control Conference (ECC)*, pages 2603–2608, 2001.

[45] Tom Schouwenaars and Eric Feron. Safe receding horizon path planning for autonomous vehicles. 01 2002.

[46] Tom Schouwenaars, Éric Féron, and Jonathan How. Safe receding horizon path planning for autonomous vehicles. In *Proceedings of the Annual Allerton Conference on Communication Control and Computing*, volume 40, pages 295–304. The University; 1998, 2002.

[47] Jamie Snape, Stephen J. Guy, Ming C. Lin, Dinesh Manocha, and Jur Van Den Berg. Reciprocal collision avoidance and multi-agent navigation for video games. In *Multiagent Pathfinding - Papers from the 2012 AAAI Workshop, Technical Report*, volume WS-12-10, pages 49–52, December 2012.

[48] Stanford Artificial Intelligence Laboratory et al. Robotic operating system.

[49] Anthony Stentz. Optimal and efficient path planning for unknown and dynamic environments. *INTERNATIONAL JOURNAL OF ROBOTICS AND AUTOMATION*, 10:89–100, 1993.

[50] Ashleigh Swingler and Silvia Ferrari. A cell decomposition approach to cooperative path planning and collision avoidance via disjunctive programming. In *49th IEEE Conference on Decision and Control (CDC)*, pages 6329–6336, 2010.

[51] Ilknur Umay, Baris Fidan, and William Melek. An integrated task and motion planning technique for multi-robot-systems. In *2019 IEEE International Symposium on Robotic and Sensors Environments (ROSE)*, pages 1–7, 2019.

[52] H. Paul Williams and Sally C. Brailsford. *Computational Logic and Integer Programming*, page 249–281. Oxford University Press, Inc., USA, 1996.

[53] Ronggang Yue, Jizhong Xiao, Shaoping Wang, and Samleo L. Joseph. Modeling and path planning of the city-climber robot part ii: 3d path planning using mixed integer linear programming. In *2009 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 2391–2396, 2009.

Chapter 7

Appendix

7.1 Planner.m

```
function ofile = integer_planner_2D(pos_init, vel_init, pos_final,...
    wind_disturb, obs_info, opbox, num_vel_cst, num_acc_cst, num_obs_cst,...
    acc_max, vel_max, vel_min, num_ts, dt, out_data_file)
% This function is integer planner in 2D environment, and the function
% returns the results from the solver.
%
% ofile = integer_planner_2D(pos_init,vel_init, pos_final, num_vel_cst, opbox,...
%       num_acc_cst, acc_max, vel_max, vel_min, num_ts, dt, out_data_file)

% Version 1.0 : Lu, Hong, 17 July 2021
% Email: hlu39@sheffield.ac.uk
% Last Modified: 22 July 2021

    disp('Mixed Integer Linear Planner for path planning.')
    % assertion for the input
    assert(length(pos_init) == 2,...
        'Dimension of the initial position should be 2')
    assert(length(vel_init) == 2,...
        'Dimension of the initial velocity should be 2')
    assert(length(pos_final) == 4,...
        'Dimension of the final position should be 2')
    assert(length(num_vel_cst) == 1,...
        'Number of the velocity approximation constraint should be a scalar')
    assert(length(num_acc_cst) == 1,...
```

```
25          'Number of the acceleration approximation constraint should be a scalar')
26      assert(length(num_obs_cst) == 1,...
27          'Number of the obstacle approximation constraint should be a scalar')
28      assert(length(acc_max) == 1,...
29          'Maximum acceleration should be a scalar')
30      assert(length(vel_max) == 1,...
31          'Maximum velocity should be a scalar')
32      assert(length(vel_min) == 1,...
33          'Minimum velocity should be a scalar')
34      assert(floor(num_ts) == num_ts,...
35          'Number of the time step should be scalar and integer')
36      assert(length(dt) == 1,...
37          'The time gap value should be a scalar')
38      assert(length(opbox) == 4,...
39          'The length of the opbox should be 4')
40      assert(length(wind_disturb) == 2,...
41          'Dimension of the wind disturbance should be 2')
42
43
44      if ~isempty(obs_info)
45          assert(size(obs_info, 2) == 3,...
46              'Column dimension of the obstacle should be 3, [x, y, radius]')
47          num_obs = size(obs_info, 1);
48      else
49          num_obs = 0;
50      end
51
52      % change the file root according to the custom PC
53      file_root = 'D:\Postgraduate\Dissertation\src\';
54
55
56      filename = [[file_root 'data\'] out_data_file '.dat'];
57      fid = fopen(filename, 'w');
58      ctn = 0;
59      ctn = ctn + AMPLcomment(fid, ['Data file generated for ' out_data_file]);
60
61      % write the wind disturbance
62      wind_x = wind_disturb(1);
63      wind_y = wind_disturb(2);
```

```
64      ctn = ctn + AMPLmatrix(fid, 'wind_disturbance',...
65          [ones(1,num_ts).*wind_x; ones(1,num_ts).*wind_y]);
66
67      ctn = ctn + AMPLscalar(fid, 'epsilon', 0.001);
68
69      % write the circle approximation
70      ctn = ctn + AMPLscalarint(fid, 'n_vel_cst', num_vel_cst);
71      ctn = ctn + AMPLscalarint(fid, 'n_acc_cst', num_acc_cst);
72      ctn = ctn + AMPLvector(fid, 'cos_vel', cos(2*pi*[1:num_vel_cst]/num_vel_cst));
73      ctn = ctn + AMPLvector(fid, 'sin_vel', sin(2*pi*[1:num_vel_cst]/num_vel_cst));
74      ctn = ctn + AMPLvector(fid, 'cos_acc', cos(2*pi*[1:num_acc_cst]/num_acc_cst));
75      ctn = ctn + AMPLvector(fid, 'sin_acc', sin(2*pi*[1:num_acc_cst]/num_acc_cst));
76      ctn = ctn + AMPLscalar(fid, 'cos_vmax', cos(pi/num_vel_cst));
77      ctn = ctn + AMPLscalar(fid, 'cos_amax', cos(pi/num_acc_cst));
78
79      % write obstacle information
80      ctn = ctn + AMPLmatrix(fid, 'obs_centroid', obs_info(:,1:2));
81      ctn = ctn + AMPLvector(fid, 'obs_radius', obs_info(:,3));
82      ctn = ctn + AMPLscalarint(fid, 'n_obs', size(obs_info, 1));
83      ctn = ctn + AMPLscalarint(fid, 'n_obs_cst', num_obs_cst);
84      ctn = ctn + AMPLvector(fid, 'cos_obs', cos(2*pi*[1:num_obs_cst]/num_obs_cst));
85      ctn = ctn + AMPLvector(fid, 'sin_obs', sin(2*pi*[1:num_obs_cst]/num_obs_cst));
86
87      % write the vel and acc limiations
88      ctn = ctn + AMPLscalar(fid, 'acc_max', acc_max);
89      ctn = ctn + AMPLscalar(fid, 'vel_max', vel_max);
90      ctn = ctn + AMPLscalar(fid, 'vel_min', vel_min);
91
92      % write time information
93      ctn = ctn + AMPLscalarint(fid, 'n_ts', num_ts);
94      ctn = ctn + AMPLscalar(fid, 'dt', dt);
95
96      % write pos_init and pos_final
97      ctn = ctn + AMPLvector(fid, 'pos_init', pos_init);
98      ctn = ctn + AMPLvector(fid, 'vel_init', vel_init);
99      ctn = ctn + AMPLvector(fid, 'pos_final', pos_final);
100     ctn = ctn + AMPLvector(fid, 'pos_cst', opbox);
101
102     fclose(fid);
```

```
103
104     sprintf('%d bytes written', ctn)
105     disp(['Data file generation finished. File located at ' filename])
106
107     % call the solver for solutions
108     model_root = [file_root 'mod\'];
109     solver_root = [file_root 'glpk-5.0\w64\'];
110     data_root = [file_root 'data\'];
111     output_root = [file_root 'output\'];
112
113     solver = [solver_root 'glpsol.exe '];
114     model = [model_root 'integer_planner_2D.mod '];
115     data = [data_root 'integer_planner_2D.dat '];
116     ofile = [output_root 'integer_planner_2D.txt '];
117
118     cmd = [solver '--model ' model '--data ' data '--output ' ofile];
119
120     statue = system(cmd);
121
122     if statue == 0
123         disp('solver is called.')
124     else
125         disp('solver is not called.')
126     end
127
128 end
```

7.2 Planner.mod

```
# This is the ampl file for the 2D MILP path planning
# No support for the multi-waypoint assignment
# Support for the single vehicle for one waypoint target (finish target)
# with obstacle avoidance
# This is the part of the code for the partial requirement for
# the degree of M.Sc. Robotics
#
# Author: Lu, Hong
# E-mail: hlu39@sheffield.ac.uk
# If you want to use the code for own purpose, please contact the author for agreement.
# Last Modified: July 23, 2021

# number of time steps
param n_ts integer >2;
# non-integer for the dt
param dt > 0;
# number of obstacles
param n_obs integer >=0;

# wind disturbance
# wind disturbance at each time step in m/s
param wind_disturbance{1..2, 1..n_ts};

# small weighting on control input
param epsilon >0;

# obstacle information centroid and radius
param obs_centroid{1..n_obs, 1..2};
param obs_radius{1..n_obs};
# coefficient for obstacle inflation
param obs_inflation >= 1;

param n_vel_cst integer >=2; # number of the velocity constraints
param n_acc_cst integer >=2; # number of the acceleration constraints
param n_obs_cst integer >=2; # number of the obstacle constraints


```

```
38 param acc_max >0; # maximum acceleration
39 param vel_max >0; # maximum velocity
40 param vel_min >0, <= vel_max; # minimum velocity
41
42 param pos_init{1..2};
43 param vel_init{1..2};
44 # final position circle [xmin xmax ymin ymax]
45 param pos_final{1..4};
46 # position constraints [xmin xmax ymin ymax]
47 param pos_cst{1..4};
48
49 # approximation for the velocity
50 param cos_vel{1..n_vel_cst};
51 param sin_vel{1..n_vel_cst};
52
53 # approximation for the acceleration
54 param cos_acc{1..n_acc_cst};
55 param sin_acc{1..n_acc_cst};
56
57 # approximation for the obstacle
58 param cos_obs{1..n_obs_cst};
59 param sin_obs{1..n_obs_cst};
60
61 # for the under-estimation usage
62 param cos_vmax;
63 param cos_amax;
64
65 # variables
66 var pos{1..2, 1..n_ts};
67 var vel{1..2, 1..n_ts};
68 var acc{1..2, 1..(n_ts-1)};
69
70 #force magnitudes
71 var am{1..(n_ts-1)};
72
73 # 1 for finish, 0 otherwise
74 var finish{1..n_ts} binary;
75
76 # decision variable for the obstacle avoidance logic
```

```
77  var obs_avoid{1..n_ts, 1..n_obs, 1..n_obs_cst} binary;
78
79  minimize time_consumption: sum{t in 1..n_ts} t*finish[t] +
80  epsilon*sum{t in 1..(n_ts-1)} am[t];
81
82  # initial state
83  subject to initpos{i in 1..2}: pos[i,1] = pos_init[i];
84  subject to initvel{i in 1..2}: vel[i,1] = vel_init[i];
85
86  # kinematics and dynamics constraints
87  subject to kinematics{i in 1..2, j in 1..(n_ts-1)}:
88  vel[i,j+1] = vel[i,j] + acc[i,j]*dt;
89
90  subject to dynamics{i in 1..2, j in 1..(n_ts-1)}:
91  pos[i,j+1] = pos[i,j] + vel[i,j]*dt + wind_disturbance[i,j]*dt + 0.5*acc[i,j]*dt*dt;
92
93  # position box constraints
94  subject to position_cst_min{i in 1..2, j in 1..n_ts}:
95  pos[i,j] >= pos_cst[2*i-1];
96  subject to position_cst_max{i in 1..2, j in 1..n_ts}:
97  pos[i,j] <= pos_cst[2*i];
98
99  # subject to velocity_cst_min{i in 1..(n_ts-1)}:
100 vel[1,i]*vel[1,i] + vel[2,i]*vel[2,i] >= (vel_min^2)/(vel_max^2);
101
102 subject to velocity_cst{i in 1..(n_ts-1), j in 1..n_vel_cst}:
103 vel[1,i]*sin_vel[j] + vel[2,i]*cos_vel[j] <= vel_max*cos_vmax;
104 subject to accelerate_cst{i in 1..(n_ts-1), j in 1..n_acc_cst}:
105 acc[1,i]*sin_acc[j] + acc[2,i]*cos_acc[j] <= acc_max*cos_amax;
106 subject to accelerate_cst_am{i in 1..(n_ts-1), j in 1..n_acc_cst}:
107 acc[1,i]*sin_acc[j] + acc[2,i]*cos_acc[j] <= am[i];
108
109 # arrival check (arrival box check)
110 subject to arrival_cst_lo{t in 1..n_ts, i in 1..2}:
111 pos[i,t] >= pos_final[2*i-1] - (pos_final[2*i-1] - pos_cst[2*i-1])*(1-finish[t]);
112 subject to arrival_cst_hi{t in 1..n_ts, i in 1..2}:
113 pos[i,t] <= pos_final[2*i] - (pos_final[2*i] - pos_cst[2*i])*(1-finish[t]);
114 subject to arrival_logic: sum{t in 1..n_ts} finish[t] = 1;
115
```

```
116 # obstacle avoidance
117 subject to obstacle_cst{i in 1..n_ts, j in 1..n_obs, k in 1..n_obs_cst}:
118 (pos[1,i]-obs_centroid[j,1])*sin_obs[k] +
119 (pos[2,i] - obs_centroid[j,2])*cos_obs[k] >= obs_radius[j] - 1000*obs_avoid[i,j,k];
120 subject to obstacle_avoidance_logic{i in 1..n_ts, j in 1..n_obs}:
121  sum{k in 1..n_obs_cst} obs_avoid[i,j,k] <= n_obs_cst-1;
```

7.3 RHC.m

```
% single UAV obstacle avoidance MPC problem
%

% tidy up
close all
clear rstr vstr ustr dstr
clc

% initial conditions
r = [0 0]';
v = [0.8 0]';

% operation box
opbox = [-2 10 -2 10];

% target
T = [8 8 9 9];

% obstacles
R = [3,4,6,5;
    6,2,7,4;
    3,1,6,2];

No = size(R,1);

% limits
Fmax = 1.5;
Vmax = 1.5;

% step
dt = 1;
```

```
32
33 % max horizon
34 Nt = 10;
35
36 % ******************** ROBUSTNESS ***************************
37
38 % disturbance magnitude
39 Fdist = 0.1*Fmax;
40
41 % form into system
42 A = [1 dt; 0 1];
43 B = [0.5*dt*dt; dt];
44
45 % make nilpotent controller
46 K = -acker(A,B,[0 0]);
47
48 % margins
49 dr1 = abs([1 0]*B)*Fdist;
50 dr2 = dr1 + abs([1 0]*(A+B*K)*B)*Fdist;
51 dv1 = abs([0 1]*B)*Fdist*sqrt(2);
52 dv2 = dv1 + abs([0 1]*(A+B*K)*B)*Fdist*sqrt(2);
53 df1 = abs(K*B)*Fdist*sqrt(2);
54 df2 = df1 + abs(K*(A+B*K)*B)*Fdist*sqrt(2);
55
56 % form up
57 Rm = [dr1 dr2];
58 Vm = [dv1 dv2];
59 Fm = [df1 df2];
60
61 % ******************** AMPL DATA FILE WRITE ******************
62
63 c = 0;
```

```
64 fid=fopen('uav_oa.dat','w');
65 c = c + AMPLcomment(fid,'Matlab generated AMPL data file\n');
66 c = c + AMPLcomment(fid,'');
67 c = c + AMPLcomment(fid,'For use with model uav_oa.mod');
68 c = c + AMPLcomment(fid,'');
69
70 % system sizes
71 c = c + AMPLscalarint(fid,'Nc',6);
72 c = c + AMPLscalarint(fid,'Nt',Nt);
73 No = size(R,1);
74 c = c + AMPLscalarint(fid,'No',No);
75 c = c + AMPLscalar(fid,'dt',dt);
76
77 % dynamics
78 c = c + AMPLscalar(fid,'Fmax',Fmax);
79 c = c + AMPLscalar(fid,'Vmax',Vmax);
80
81 % target
82 c = c + AMPLvector(fid,'T',T);
83
84 % obstacles, inc margin
85 Rd = R + (Vmax*dt/(2*sqrt(2)))*ones(No,1)*[-1 -1 1 1];
86 c = c + AMPLmatrix(fid,'R',Rd);
87
88 % margins
89 c = c + AMPLvector(fid,'Rm',Rm);
90 c = c + AMPLvector(fid,'Vm',Vm);
91 c = c + AMPLvector(fid,'Fm',Fm);
92
93 % big M
94 c = c + AMPLscalar(fid,'M',20);
95
```

```
% control weight
c = c + AMPLscalar(fid,'gamma',0.001);

% completed writing file
fclose(fid);
sprintf('%d bytes written',c)

% sim loop
for ii=[1:(2*Nt)],

  % write initial condition to data file
  c=0;
  fid=fopen('uav_oa_ic.dat','w');
  c = c + AMPLcomment(fid,'Matlab generated AMPL data file\n');
  c = c + AMPLcomment(fid,'');
  c = c + AMPLcomment(fid,'For use with model uav_oa.mod');
  c = c + AMPLcomment(fid,'');

  % initial state
  c = c + AMPLvector(fid,'ri',r);
  c = c + AMPLvector(fid,'vi',v);

  % completed writing file
  fclose(fid);
  sprintf('%d bytes written',c)

  % solve
  !myampl uav_oa.run

  % load plan
  rp = load('r.dat');

```

```
128     % load finishing time
129     load a.dat;
130
131     % load control
132     load u.dat;
133     u = reshape(u,2,1);
134
135     % disturbance
136     thd = 2*pi*rand(1,1);
137     d = Fdist*rand(1,1)*[cos(thd); sin(thd)];
138
139     % storage
140     rstr(:,ii) = r;
141     vstr(:,ii) = v;
142     ustr(:,ii) = u;
143     dstr(:,ii) = d;
144
145     % if done
146     if (a==1),
147       break
148     end
149
150     % sim
151     r = r + v*dt + 0.5*dt*dt*(u+d);
152     v = v + dt*(u+d);
153
154   end
155
156   figure;
157   plot(rstr(1,:),rstr(2,:),'.-')
158   hold on
159   for kk=[1:No],
```

```
    patch(R(kk,[1 3 3 1 1]),R(kk,[2 2 4 4 2]),'r')
    patch(Rd(kk,[1 3 3 1 1]),Rd(kk,[2 2 4 4 2]),...
    'white',...
    'FaceAlpha', 0.0,...
    'LineStyle', '--')
end
plot(T([1 3 3 1 1]),T([2 2 4 4 2]),'g')
axis equal
xlim(opbox(1:2))
ylim(opbox(3:4))

figure;
plot([0:size(dstr,2)-1], dstr(1,:), 'b-o')

hold on

plot([0:size(dstr,2)-1], dstr(2,:), 'r-o')

title('Disturbance during the process')
legend('disturbance_x','disturbance_y')
```

Chapter 8

Self-review

This section provides the critical self-review during the whole process of this project. From my perspective, I demonstrated the independent capability for reviewing the literature in the certain domain, finding the problem, defining the problem and adopting several methods for testing the proposed theories, moreover, I also applied the technique with the new pipeline into the simulated environment for validation. The progress of the project was reasonably stick to the original plan with minor modifications and amendments of the planning chart. The topic of the project requires the solid background of the mathematical programming which had brought difficulties into the project, by all means, I overcame these obstacles by concentrating in the certain problem and exploring the area with the learned knowledge successfully. To implemented the method, there also were many details to be considered than just reading it. The whole process had developed a more considerable and sophisticated mind of mine. With the understanding and the hand-on experience of the MILP methods, the applications are also carried out in the reality scenario successfully in Quadrotor UAV and surface climbers which reflected the understanding of the problem and the creativity to migrate the technique. The whole project did a comprehensive study and implementations in MILP planner dealing with various scenarios according to the proposed plan at the start of the project.