ÉCOLE CENTRALE DE NANTES

MASTER CORO-IMARO
"CONTROL AND ROBOTICS"

2018 / 2019

Master Thesis Report

Presented by

Ashwin Bose

January 2019

Online Trajectory Planning of multiple fleets of robots using Model Predictive Control

Jury

President:	Olivier Kermorgant	Assistant Professor (LS2N, ECN)
Evaluators:	Ina Taralova	Maître de conférences (ECN)
	Olivier Kermorgant	Assistant Professor (LS2N, ECN)
Supervisor(s):	Kim Clement	Chief Technical Officer (Unmanned Systems Limited)
	Olivier Kermorgant	Assistant Professor (LS2N, ECN)

Laboratory: Laboratoire des Sciences du Numérique de Nantes LS2N

Abstract

Quadrotors have many application... bla bla

Acknowledgements

I thank my bla bla bla

Notations

Abbreviations

List of Figures

1.1 Visualization of configuration space obstacle 12
1.2 Preprocessing stage of PRM 14
1.3 Tree expansion in RRT 15
1.4 Visualization of potential field based search 15
1.5 Minimum-snap trajectory generation using unconstrained quadratic program . . 18
1.6 Path planning performed at non-zero inital velocity. 19

2.1 Visualization of the variable dimensionality workspace in M* algorithm. 22
2.2 Visualization of Conflict Based Search 23
2.3 Multi-agent trajectory planning using sequential quadratic programming. 24
2.4 Collision avoidance of a neighbouring agent using the principle of velocity obstacles 26
2.5 Collision cost function vs smoothness parameter 27
2.6 Position swapping of multiple agents using NMPC 28

Contents

Introduction **9**

1 Path Planning **11**
1.1 Preliminaries . . . 11
1.1.1 Configuration Space . . . 11
1.1.2 Graph Search Algorithms . . . 12
1.2 Basic Path Planning Methods . . . 13
1.2.1 Grid Based Approach . . . 13
1.2.2 Sampling-based Approach . . . 13
1.2.3 Potential field based Search . . . 15
1.3 Kinodynamic Path Planning . . . 16
1.4 Motion Planning for Quadrotors . . . 16
1.4.1 Sampling-Based Motion Planning . . . 17
1.4.2 Polynomial-Based Motion Planning . . . 17
1.4.3 Search-Based Motion Planning . . . 18
1.5 Control of Quadrotors . . . 18

2 Multi-Agent Path Planning **21**
2.1 Graph Search Methods . . . 21
2.1.1 M* Search . . . 21
2.1.2 Conflict Based Search . . . 22
2.2 Continuous optimization schemes . . . 22
2.3 Planning with Goal Assignment . . . 24
2.4 Velocity Profile Methods . . . 24
2.5 Collision-Avoidance Approaches . . . 25
2.5.1 Velocity Obstacle . . . 25
2.5.2 Nonlinear Model Predictive Control . . . 26

3 Work Outline **29**

Conclusion **31**

Bibliography **32**

Introduction

Chapter 1

Path Planning

The planning problem has been studied extensively in various fields like robotics, artificial intelligence, and control theory. In robotics, a classical example of the path planning problem is the *piano-mover's problem*, where the aim is to move a piano from one position to another without colliding with any obstacle. Currently, this problem covers other complications such as non-holonomy, uncertainties, and dynamics.

This chapter covers the preliminaries of path planning and its application on a single agent. The preliminaries are covered in Section. 1.1, with topics like *configuration space* and *graph search algorithms*. Then the basic path planning methods are introduced in Section. 1.2. Section. 1.4 presents the applications of advanced planning methods for quadrotos. Finally, the control methods are shown in Section. 1.5.

1.1 Preliminaries

1.1.1 Configuration Space

Each distinct situation of a world is called a *state*, x, and the set of all possible states is called a *state space*, X. The state, x, can be transformed to x', by applying an *action*, u, as specified by a *state transition function*, f, such that:

$$x' = f(x, u) \tag{1.1}$$

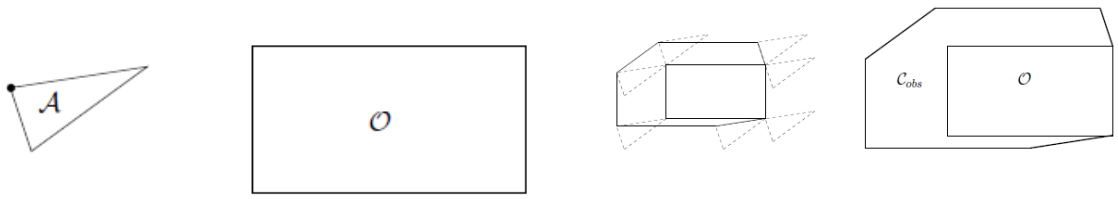
The set of all possible actions for each state is defined as the *action space*, $U(x)$.

In path planning, we work with a special state space called the *configuration space*, or C-space. The configuration space, $\mathcal{C}$, of a robot is the set of all possible configurations, x, that could be achieved by it. The real beauty of C-space is in the way it deals with the obstacles. Suppose that the world, $\mathcal{W}$, contains an obstacle region, $\mathcal{O}$. Assuming that the robot, $\mathcal{A} \subset \mathcal{W}$, the *obstacle region*, $\mathcal{C}_{obs} \subseteq \mathcal{C}$, is defined as

$$\mathcal{C}_{obs} = \{x \in \mathcal{C} | \mathcal{A}(x) \cap \mathcal{O} \neq \emptyset\} \tag{1.2}$$

In other words, $\mathcal{C}_{obs}$ is the set of all configurations at which the robot intersects the obstacle region (See Figure. 1.1). The *free space*, defined as, $\mathcal{C}_{free} = \mathcal{C} \backslash \mathcal{C}_{obs}$, is the set of all collision free configurations of the robot.

Let $x_I \in \mathcal{C}_{free}$ and $x_G \in \mathcal{C}_{free}$ be the *initial configuration* and the *final configuration* respectively. A path planning algorithm aims to compute a continuous path starting at x_I and ending at x_G.



(a) The generation of the configuration obstacle $\mathcal{C}_{obs}$

(b) The configuration space is obtained by sliding the triangle around the obstacle while keeping them both in contact

Figure 1.1: Visualization of configuration space obstacle [1]

1.1.2 Graph Search Algorithms

State transition graph is a convenient tool that could be used to approach path planning problems. In the graph $\mathcal{G}$, the states and actions form the vertices $\mathcal{V}$ and directed edges $\mathcal{E}$ respectively. A directed edge from $x \in X$ to $x' \in X$ exists only if there exists an action $u \in U$ such that $x' = f(x, u)$. With this representation, the graph search algorithms could be directly applied to the path planning problem.

In this section, we cover the major graph search algorithms like breadth first search, length first search and A* search. These forward search algorithms start at the initial state and explores the graph until encountering the goal state. The general template of such algorithms [1] is shown in Algorithm 1.

Algorithm 1 Forward Search

```
1: Q.Insert(x_I) and mark x_I as visited
2: while Q not empty do
3:     x ← Q.GetFirst()
4:     if x ∈ X_G then
5:         return SUCCESS
6:     for all u ∈ U(x) do
7:         x' ← f(x, u)
8:         if x' not visited then
9:             Mark x' as visited
10:            Q.Insert(x')
11:        else
12:            Resolve duplicate x'
13: return FAILURE
```

During the search, there will be three kinds of states. The *unvisited* list consists of all the unexplored states of graph. If a state, along with all its next possible states, have been visited, it is added to the *dead* list. The remaining states, i.e. the explored states with unexplored neighbours, forms the *alive* list. As shown in the Algorithm 1, the alive list is stored in a data structure Q. The main difference between the search algorithms is in the way they sort Q.

In *breadth first search*, Q is implemented as a First-In-First-Out (FIFO) queue. This results in a uniform expansion of the search frontier. If we make Q a stack, the graph would be explored aggressively in an arbitrary direction, resulting in the *depth first search*.

If we know the cost of the actions, we could use that to increase the efficiency of the search algorithm. In *Dijkstra's algorithm*, Q is sorted according to the *cost-to-come* function, $C(x)$, which is defined as the minimum cost of travelling from x_I to x according to the current knowledge.

In many cases, it is possible to obtain a heuristic estimate of the cost to reach the goal from any state. Let this function be called $\hat{G}(x)$. This knowledge can be exploited to reduce the number of states explored to reach the goal. The *A* search algorithm* sorts Q according to the sum $C(x) + \hat{G}x$.

1.2 Basic Path Planning Methods

As mentioned before, the path planning problems are defined in the configuration space, whereas the graph search algorithms are based on graphs. To connect these problems, we should reformulate the path planning problem, framed on a continuous configuration space, in terms of a graph. This section is dedicated to three different methods of approaching this problem: grid based, potential field based, and sampling based.

1.2.1 Grid Based Approach

In this approach, initially, a grid of locations is sampled in the configuration space. Each vertex $\mathcal{V}$ of the graph represent a configuration. A collision-check function is ran on each grid to check if it is in $\mathcal{C}_{free}$. It is assumed that the grids are close enough in space such that there can be no obstacle between two adjacent grids. In other words, we assume that if two adjacent vertices are in $\mathcal{C}_{free}$, every point in the line joining them is also in the $\mathcal{C}_{free}$. Therefore, we join the vertices by an edge. This discretizes the set of actions, and hence the grid search algorithms can be applied to find the path from the start to the goal.

Grid based search works well for low dimensional problems. However, this method is computationally infeasible for high-dimensional systems, since the number of points on the grid increase exponentially with configuration space dimension.

1.2.2 Sampling-based Approach

Sampling-based search algorithms are the leading methods to tackle higher-dimensional path planning problems. These approaches chooses the points in the configuration space randomly, instead of uniformly, to generate the graph. In other words, the connectivity of $\mathcal{C}_{free}$ is explored by sampling random configurations and trying to connect in a network. Two major sampling-based methods: *rapidly-exploring random trees* (RRT) [1] and *probabilistic roadmaps* [2], are discussed in this section.

Probabilistic Road-Map (PRM)

Probabilistic roadmap is the preferred method if the initial and goal states are not fixed. This method consists of two stages: a preprocessing and query stages. The pseudocode of this method is shown in Algorithm. 2. In the preprocessing stage, a random configuration is generated, and neighbours within a predefined distance are connected if all the points in the line joining them are in $\mathcal{C}_{free}$. This step is repeated until a dense graph is generated (see Figure. 1.2).

In the query stage, the start and goal configurations are connected to the graph, and an appropriate search algorithm is used to find the path. Once the roadmap has been constructed, it can be used to generate paths between various start and goal locations, as long as the environment is static. This justifies the initial cost of constructing the graph.

However, in many cases, we are only interested in planning between one specific start and goal locations, and therefore, building a roadmap for such problems is wasteful. In such situations, it would be better to consider approaches that explicitly uses the initial and goal locations in approaching the problem.

Algorithm 2 Probabilistic Road-Map

1: $\mathcal{G}$.init($i \leftarrow 0$;)
2: **while** $i < N$ **do**
3: **if** $\alpha(i) \in \mathcal{C}_{free}$ **then**
4: $\mathcal{G}$.add_vertex($\alpha(i)$); $i \leftarrow i + 1$
5: **for** **do**each $q \in$ NEIGHBORHOOD($\alpha(i), \mathcal{G}$)
6: **if** (**not** $\mathcal{G}$.same_component($\alpha(i), q$)) **and** CONNECT($\alpha(i), q$) **then**
7: $\mathcal{G}$.add_edge($\alpha(i), q$)
8: **return** $\mathcal{G}$

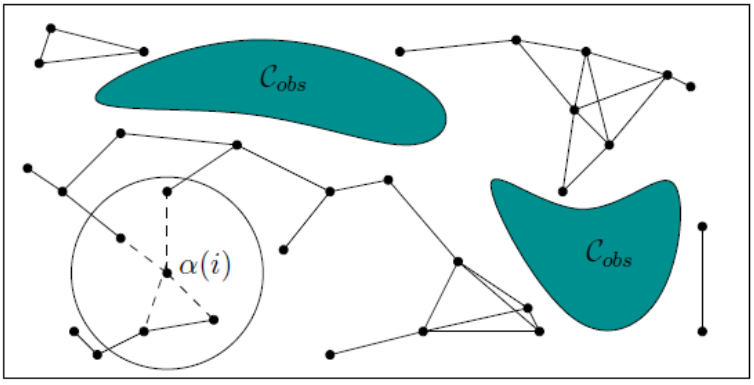


Figure 1.2: Preprocessing stage of PRM. The new randomly sampled point $\alpha(i)$ is connected to the neighbouring vertices in the roadmap

Rapidly-Exploring Random Trees (RRT)

The RRT algorithm explores the space by randomly sampling a space-filling *tree*. A tree is a special graph, with every vertex connected to a single parent. As shown in Algorithm 3, with each new sample $\alpha(i)$, a vertex q_s is added between it and the nearest state q_n in tree. q_s lies at a distance δ from q_n on the line joining $\alpha(i)$ and q_n. If $\alpha(i)$ lies in the obstacle region, q_s is generated at the edge of the obstacle as shown in Figure 1.3.

This algorithm is very efficient in exploring the free space starting from a configuration. In order to use this for our problem, we usually generate two RRTs, one rooted at the initial configuration and the other at the goal configuration. We then grow both trees until they meet. There are many variations of RRT methods, and they are very effective at tackling high dimensional problems. They can also handle problems with kinodynamic constraints.

Algorithm 3 Rapidly-exploring Random Trees

1: $\mathcal{G}$.init(q_0)
2: **for** $i = 1$ **to** k **do**
3: $q_{rand} \leftarrow$RAND_CONFIG()
4: $q_{near} \leftarrow$NEAREST_VERTEX($q_{rand}, \mathcal{G}$)
5: $q_{new} \leftarrow$NEW_CONF($q_{near}, q_{rand}, \Delta q$)
6: $\mathcal{G}$.add_vertex(q_{new})
7: $\mathcal{G}$.add_edge(q_{near}, q_{new})
8: **return** $\mathcal{G}$

The main issue with sampling based approaches is they are not *complete*: i.e. they cannot guarantee the absence of a path even if they are unable to find it. These algorithms are

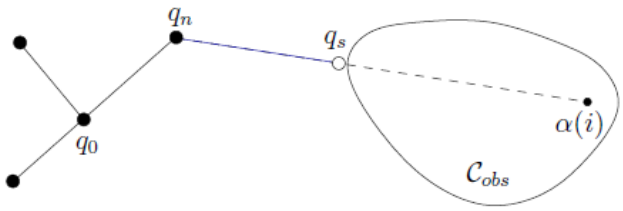
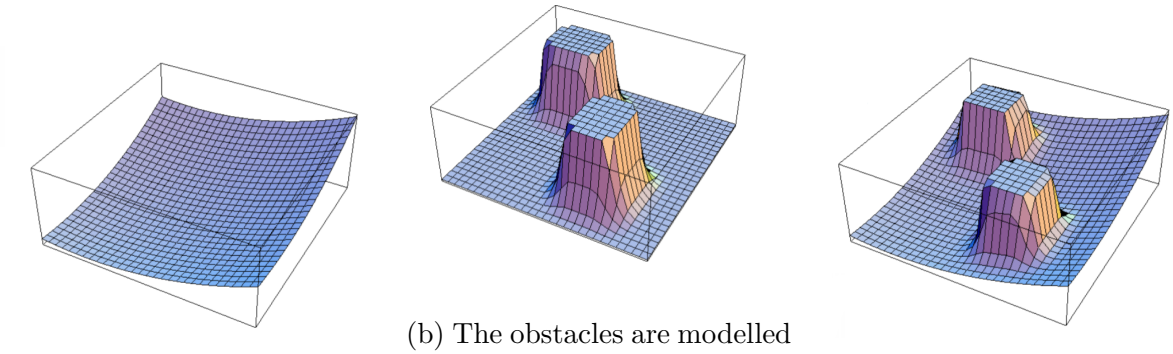


Figure 1.3: If the sampled point $\alpha(i)$ lies inside the obstacle region, the new vertex q_s is formed at the boundary of $\mathcal{C}_{obs}$



(a) The goal modelled as an attractive potential field, f_a

(b) The obstacles are modelled as a repulsive potential field, f_r

(c) The combined potential field, $f_c = f_a + f_r$

Figure 1.4: Visualization of potential field based search [1]

probabilistically complete, which means that with enough points, the probability that it finds an existing solution converges to one.

1.2.3 Potential field based Search

Potential field based approaches generate good results with minimal computations. In this method, the robot (represented in C-space), is treated as a particle under the influence of an artificial potential field. The potential field is designed in such a way that the goal has an *attractive potential* and the obstacles has a *repulsive potential*. This ensures that the robot moves towards the goal while avoiding obstacles.

As shown in Figure. 1.4, this method can be visualized as a particle moving on a 2D plane under gravity. The goal and the obstacles are represented by a basin and hills respectively. At every instance, the robot tries to move in a direction along decreasing potential, thereby finally reaching the goal. This is done by using the gradient of the potential function to steer the robot. The control velocity $\mathbf{v}$ is chosen as:

$$\mathbf{v} \propto -\nabla f_c \tag{1.3}$$

Using this approach, the robot can generate a path by just knowing the potential function. The potential field can be computed by the knowledge of the robot's position, and local sensor data. Moreover, the gradient could be evaluated by using simple finite difference methods. The simplicity of this method makes it appealing to robotic applications. However, this method may find a non-optimal path, or no path at all if the particle is stuck in a local minima.

1.3 Kinodynamic Path Planning

Until now, we assumed that the path between any two configurations can be easily determined. However, this might not always be possible in most planning problems in mechanical systems due to the kinematics and dynamics constraints. Planning in such a situation is called *kinodynamic* path planning.

A classical approach to this problem is to decompose it into two sub-problems. Initially, collision-free a geometric path, called the quasi-static solution, is computed. It can be visualized as a trajectory that the robot can follow in very slow speeds and reach the goal. In the next step, a constrained optimization problem is solved to minimize the time taken to follow the trajectory while respecting the dynamic constraints.

This decoupled approach is very efficient in finding feasible paths. However, they do not obtain good quality paths since the dynamics is not considered during the path planning stage. To obtain better results, more complex *direct planning* methods are preferred.

Even though there exists many deterministic approaches like based on optimal control, numerical optimization, and grid search, sampling-based approaches are preferred to solving these problems due to their high dimensionality. In this case, we have to rethink the way in which we connect two configurations.

The direct way to do is to solve a two-point boundary value problem (BVP). However, this is computationally demanding and hence infeasible for most scenarios. If the dynamics is linear or could be linearized about an operation point, we could use LQR based techniques to obtain faster results. Some researchers bypass the solving of BVP by using precomputed *motion primitives*.

The most studied method is the forward propagation of dynamics. New states are generated by performing a feasible action on states that are already generated, typically by using ODE solvers. This method works well with RRT and has generated good results.

Now that we have seen the complexities of planning in mechanical systems, we move ahead to studying the path planning problem for quadrotors.

1.4 Motion Planning for Quadrotors

This section is dedicated to motion planning methodologies for quadrotors. Our aim is to develop a planner that generates a feasible path, that respects the input and dynamics constraints, to bring the quadrotor to the goal as soon as possible. To achieve this, we should exploit the internal dynamics of the quadrotor. Furthermore, it should find plans in real-time at rates around 50 Hz.

This is not always possible in the case of quadrotors, which have differential constraints that arise due to geometry and dynamics. *Kinodynamic motion planning* [3] methods can be used to approach such problems.

In general, there are two approaches to deal with this problem. In the first approach, the problem is solved by dividing it into global and local planning sub-problems. The global planner, typically implemented using sampling based search, computes a desired set of configurations through which the quadrotor moves while avoiding obstacles. Then the local planner generates time parametrized *trajectories* that can be realized by the vehicle. This method is simpler, but it may not lead to a global optimum as the geometric path is decided in advance, without considering the system dynamics.

The second approach relies on the differential flatness [4] property of the quadrotor dynamics to derive constraints on the trajectory and then solves an optimization problem to find the optimum trajectory. Some interesting state-of-the-art path planning methodologies are

discussed in this section.

1.4.1 Sampling-Based Motion Planning

As mentioned before, the high dimensionality of the path planning problem makes it infeasible for simple-graph search based algorithms. Sampling-based methods perform better in this case, especially in cluttered environments. In this section, we briefly introduced some sampling based approaches.

Some popular sampling based path-planning algorithms include, RRT*, PRM* (Probabilistic Road Map*) and rapidly-exploding random graphs (RRG), which is an extension of RRT. The approach in [5], the RRT* method is combined with a fixed-final-state-free-final-time controller to obtain asymptotic optimality. Closed-form solutions of optimal trajectories could be derived in this method.

Bry and Roy et al. [6] combined the belief roadmap with RRG to address the problem of motion planning in the presence of state uncertainty. The resultant search tree in belief space is provably convergent to the optimal path. Shen et al. [7] used RRG method, along with a trajectory optimization framework, to generate a safe, smooth and dynamically feasible trajectory based on the piecewise line segment initial path.

1.4.2 Polynomial-Based Motion Planning

A trivial trajectory that passes through a set of *waypoints* is the one that interpolates them using straight lines. Evidently, this trajectory isn't efficient as the robot has to come to rest at each waypoint. In this section, we discuss in detail the work by Kumar et al. [4], which generates a *minimum-snap trajectory* that passes through the waypoints while satisfying the constraints on velocity and accleration.

In this work, the trajectories are parametrized as piecewise polynomial functions of n^{th} order over m time intervals as:

$$\sigma_t = \begin{cases} \sum_{i=0}^{n} \sigma_{Ti1} t^i & t_0 \le t \le t_1 \\ \sum_{i=0}^{n} \sigma_{Ti2} t^i & t_1 \le t \le t_2 \\ & \vdots \\ \sum_{i=0}^{n} \sigma_{Tim} t^i & t_{m-1} \le t \le t_m \end{cases} \tag{1.4}$$

The decision variable vector, $\mathbf{c}$, is a $3mn \times 1$ vector consisting of the constants σ_{Tij}.The optimization program minimizes the k_r^{th} derivative of the square of the position as:

$$\begin{aligned} \min_{\mathbf{c}} \quad & \int_{t_0}^{t_f} \left\| \frac{d^{k_r} \mathbf{r}_T}{dt^{k_r}}) \right\|^2 dt \\ \text{s.t.} \quad & \mathbf{r}_T = \mathbf{r}_w, \quad w = 0, \ldots, m \\ & \left. \frac{d^j x_T}{dt^j} \right|_{t=t_w} = 0 \text{ or free}, w = 0, \ldots, m; j = 1, \ldots, k_r \\ & \left. \frac{d^j y_T}{dt^j} \right|_{t=t_w} = 0 \text{ or free}, w = 0, \ldots, m; j = 1, \ldots, k_r \\ & \left. \frac{d^j z_T}{dt^j} \right|_{t=t_w} = 0 \text{ or free}, w = 0, \ldots, m; j = 1, \ldots, k_r \end{aligned} \tag{1.5}$$

Here, $\mathbf{r}_T = [x_T, y_T, z_T]^\top$ and $\mathbf{r}_i = [x_i, y_i, z_i]^\top$. It is also ensured that the first k_r derivatives of the position is continuous. The optimization problem can be formulated as a quadratic

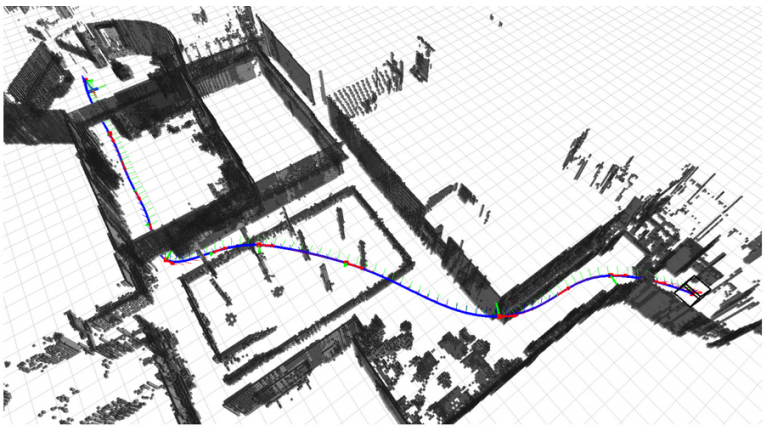


Figure 1.5: Minimum-snap trajectory generation using unconstrained quadratic program. [8]

program:

$$\begin{aligned} \min \quad & \mathbf{c}^\top \mathbf{H}\mathbf{c} + \mathbf{f}^\top \mathbf{c} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{c} \leq \mathbf{b} \end{aligned} \tag{1.6}$$

As the inputs to the system was found to be functions of the fourth derivatives of positions, k_r was chosen to be 4, and hence the snap was minimized. This rise to smooth trajectories that avoid paths involving extreme control inputs. Moreover, the resulting smoothness helps in maintaining the quality of onboard sensor measurements. Then, the segment times are optimized to minimize the total time.

The method presented in [8] jointly optimizes polynomial path segments in an unconstrained quadratic program to solve the minimum-snap trajectory problem. The output is numerically stable for high-order polynomials and large numbers of segments. As shown in Figure. 1.5, they generated fast flight paths through cluttered environments by coupling this technique with an appropriate kinematic planner. In addition, an implicit segment time allocation strategy, based on a single user-defined parameter on aggressiveness, led to far superior results.

1.4.3 Search-Based Motion Planning

Recent work by Kumar et al. [9] aims at computing globally optimum, collision-free, minimum-time, dynamically-feasible trajectories in real-time. When the geometric path is first computed and then smoothened, the generated trajectory may not contain a globally optimum trajectory as this approach does not consider the initial dynamics of the robot. As shown in Figure. 1.6, the trajectory generated by this approach is superior to other approaches.

Even though others have worked on generating time-optimal trajectories, the algoritms developed were not viable due to their high computation costs. In [9], the algorithm solves an optimal control problem on a set of short-duration motion primitives to explore the space of trajectories. The primitives discretize the state-space into a finite lattice, which is then explored using a graph search algorithm accelerated by a heuristic.

1.5 Control of Quadrotors

Due to its recent popularity, practically all the major control techniques have been used to control technique have been tried on quadrotors. Linear control techniques [10] have been used to control quadrotors by linearizing their dynamics around an operation point, typically the

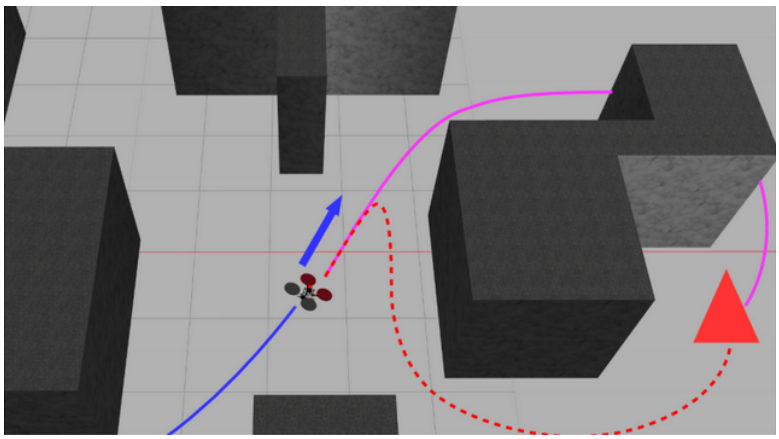


Figure 1.6: Path planning performed at non-zero inital velocity. The search based approach shown in [9] generates the more natural and time optimal magenta curve, where as other approaches gives rise to the red-dashed curve.

hover position. Better performance is obtained by non-linear control methods like backstepping, sliding mode [11] and feedback linearization [12].

An accurate model of the system is necessary for the above-mentioned control techniques to generate satisfactory results. Modelling errors can considerably deteriorate their performance. Adaptive controllers [13] perform well in these cases by correcting the errors in model parameter estimates.

Model predictive control (MPC) refers to a set of controllers that use a model to compute inputs from the current time to a future time in order to optimise the behaviour of a model along the input trajectory. Since it is a computationally intensive algorithm, it was used in slow systems like chemical plants and oil refineries. Due to the recent increase in computational capabilities, MPC have become popular in robotics.

MPC is used to generate trajectories interpolating a given set of way-points [14] by calculating optimal controls online to minimize a cost function within a receding horizon. The constraints of the problem can be incorporated to the optimal control problem (OCP). In [15], a fast nonlinear model predictive control (NMPC) approach based on a geometric formulation of the error to track the MAV attitude on the SO(3) special orthogonal group.

Since the constrained optimization problem is computationally intensive, a trade-off has to be made between time horizon and policy lag. This trade-off is alleviated in [16] by solving an unconstrained MPC problem using Sequential Linear Quadratic (SLQ) solver. By combining trajectory optimization and trajectory contol, this approach generated trajectories of multiple seconds within a few milliseconds.

Now that we have introduced the basics of path planning, and its application to single-agent systems, we move ahead to deal with more challenging multi-agent problems in the next chapter.

Chapter 2

Multi-Agent Path Planning

The multi-agent path finding problem (MAPF) is more challenging than the single-agent problem due to the high dimension of its configuration space. Broadly speaking, there are two approaches to solve this problem: planning and reacting. In the planning approach, collision-free paths are generated ahead of time, whereas the reactive approach uses online collision avoidance schemes while encountering hazardous situations. In this chapter, we introduce the various approaches of tackling problem.

2.1 Graph Search Methods

These approaches compute the path by applying the graph-search algorithms on the configuration space of the multi-robot system, which is the Cartesian product of the configuration spaces of each robot. Here, robot-robot collisions are expressed as configuration space obstacles. The full configuration space grows exponentially with the number of robots, thus making the standard search algorithms computationally infeasible.

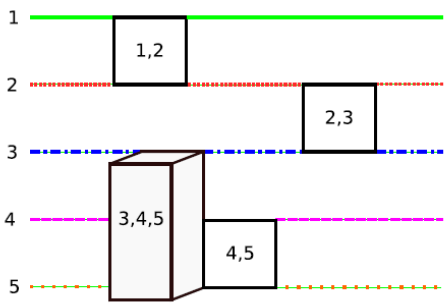
It is not necessary to plan in this space as the robots are usually well separated in the workspace, and collisions are infrequent. Some works generate feasible results by exploiting this decoupled property of the system. In this section, we discuss two such methods: M* [17] and conflict based search [18]. These methods have proven to produce remarkable results for cases with high congestion.

2.1.1 M* Search

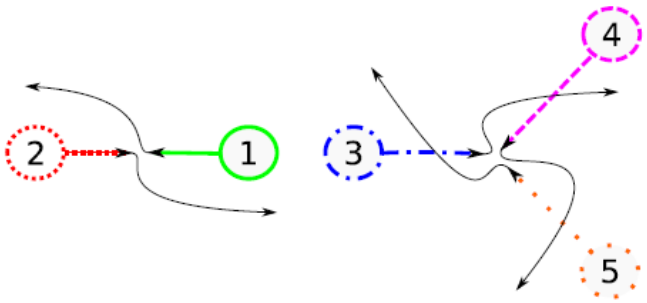
Initially, M* algorithm plans for each robot separately, without considering collisions. This is a good starting point for multi-robot planning as no path can be cheaper than this. When collisions occur, planning is performed in the joint configuration space (see Figure. 2.1) of the robots involved in the collision while uninvolved robots proceed independently. When the collision is bypassed, planning continues to proceed in low dimensional individual spaces.

Like A* algorithm, M* explores a list of vertices sorted based on the sum of the cost of the cheapest path and a heuristic cost. M* considers only a limited set of neighbours determined by a *collision set*, thus minimizing the dimensionality. M* can be seen as performing A* search in a dynamically updating graph, $G^{\#}$. M* algorithm can handle problems involving a high number of robots, while retaining bounded optimality and *completeness*.

The uncertainties of the poses, inherent to robotics, was taken into consideration in UM* [19]. In this approach, planning is done in the belief space to account for the uncertainties. Then, a scheme similar to M* is implemented.



(a) Representation of search space in M* algorithm



(b) Motion of multiple robots in the shared workspace

Figure 2.1: Visualization of the variable dimensionality workspace in M* algorithm. When two robots collide, the local dimensionality is increased to 2 (represented by a square). Similarly, when three robots collide the dimensionality is 3 (cube). [17]

2.1.2 Conflict Based Search

Conflict-Based Search (CBS) is a bounded sub-optimal two-level MAPF solver. The top level solves a binary *constraint tree* (CT), whose nodes consists of a set of constraints, a single solution, and the cost of solution. The low level of CBS searches for a plan that satisfies all the constraints in a node N. If a node is conflict-free, it is returned as the solution. Otherwise, CBS splits node N based on one of the conflicts. Let $< a_i, a_j, v, t >$ represent the conflict between agents a_i and a_j at vertex v at time-step t. Now, N is split into two children node, one with the constraint $< a_i, v, t >$ and the other with $< a_j, v, t >$. By doing so, the solver imposes a constraint on only one agent at a time.

An example of this node expansion scheme is shown in Figure. 2.2. In this problem, each mouse (agent) must find a path to cheese (goal). Node R is initialized with no constraints. The shortest paths generated has a collision $< 1, 2, D, 2 >$. Therefore, two nodes, U and V, are generated with constraints $< 1, D, 2 >$ and $< 2, D, 2 >$ respectively. This leads to collision free solutions. As this search method is exponential in the number of conflicts encountered, as opposed to the number of agents, it leads to superior results.

Since solving MAPF problems are optimally NP-hard, optimal solutions like M* and CBS are feasible only for low number of agents. Therefore, it is common to use bounded-suboptimal solutions for higher number of agents. Koenig et al. [20] uses user-provided highways to generate suboptimal paths for a large number of robots.

The graph plans from these grid based search methods cannot be directly interpreted as trajectory results as they do not consider the kinematic and dynamics constraints of the robot. [21] deals with this problem by post-processing the output of the MAPF solvers using a simple temporal network.

2.2 Continuous optimization schemes

Some researchers the MAPF problem in a continuous setting by formulating it as an optimisation problem with the trajectories defined by its decision variables. These methods generate smooth solutions for small fleets of robots.

D'Andrea et al. [22], solves this problem using sequential quadratic programming. Suppose we have N vehicles, and K time steps. The position of the vehicle i at time-step k is denoted

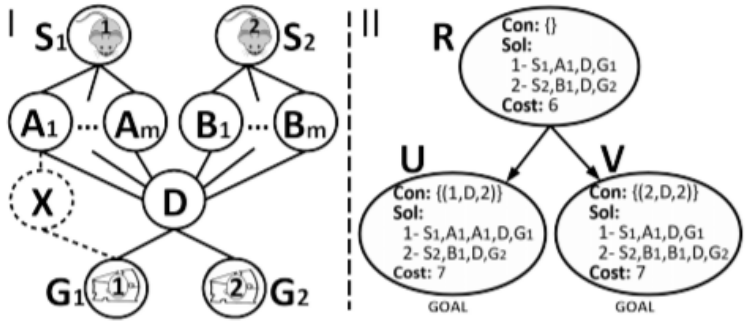


Figure 2.2: Visualization of Conflict Based Search (CBS): (I) MAPF instance, and (II) its Constraint-Tree (CT)

as $p_i[k]$. The velocity and acceleration equations are:

$$\begin{aligned} v_i[k+1] &= v_i[k] + ha_i[k] \\ p_i[k+1] &= p_i[k] + hv_i[k] + \frac{h^2}{2}a_i[k], \end{aligned} \tag{2.1}$$

The optimization variable $\chi \in \mathbb{R}^{3NK}$ is made of the accelerations of the vehicles at each time step. The velocity and acceleration of the vehicle i at time k can also be written as:

$$\begin{aligned} v_i[k] &= v[1] + h(a_i[1] + a_i[2] + \ldots + a_i[k-1]) \\ p_i[k] &= p_i[1] + h(k-1)v_i[1] + \frac{h^2}{2}\left((2k-3)a_i[1] + (2k-5)a_i[2] + \ldots + a_i[k-1]\right)) \end{aligned} \tag{2.2}$$

The objective function is the sum of the total thrust at each step can be written as:

$$f_0 = \sum_{i=1}^{N}\sum_{k=1}^{K} \|a_i[k] + g\| \tag{2.3}$$

where, g is the gravity vector. This function can be expressed as a quadratic function of the optimization variable χ. The fixed initial and final states gives rise to some equality constraints. Similarly, the limits on the workspace, as well as the dynamics limits of the quadrotor creates inequality constraints on the position, velocity, acceleration, and jerk. If the robots should be at a safe distance of R between each other to avoid collisions, the following non-convex constraint must be imposed:

$$\|p_i[k] - p_j[k]\|_2 \geq R \tag{2.4}$$

Using sequential convex programming, these constraints can be replaced with convex approximations around a previous solution χ^q using a first order Taylor expansion. The minimization problem can now be written as:

$$\begin{aligned} \min \quad & f_0(\chi) = \chi^\top P\chi + q^\top\chi + r \\ \text{s.t.} \quad & A_{eq}\chi = b_{eq} \\ & A_{in}\chi \leq b_{in} \end{aligned} \tag{2.5}$$

The optimum solution was found using QP. A demonstration of this method can be seen in Figure. 2.3.

In [23], a Mixed-Integer Quadratic Program (MIQP) is solved to generate feasible paths. These methods are capable of creating smooth, optimal solutions. However, these methods are often tightly and hence do not scale up well to large teams.

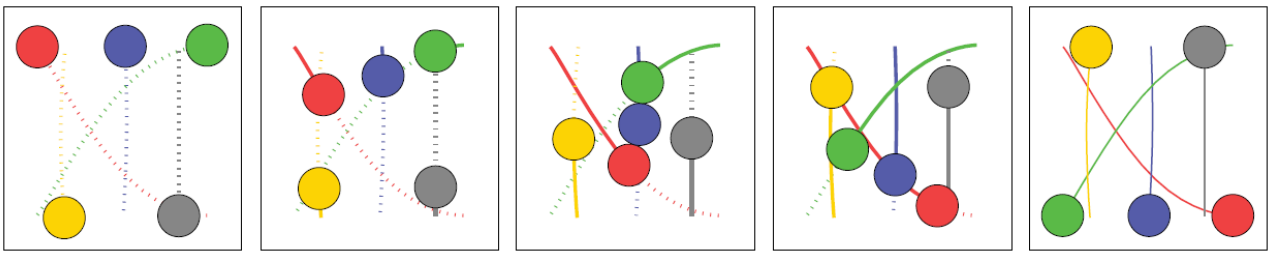


Figure 2.3: Multi-agent trajectory planning using sequential quadratic programming. [22]

2.3 Planning with Goal Assignment

In many applications of multi-agent systems, the agents are interchangeable. For instance, in case of first-response and search and rescue applications, the aim is to explore the maximum area with a limited number of robots. In the context of the path planning problem, this translates to having no one-to-one correlation between the initial and goal locations. Therefore, goal-assignment should also be a part of the planning algorithm to obtain the best results.

Paradoxically, coupling the goal-assignment with trajectory planning reduces the complexity of the problem [24]. Koenig et al. [25] presented a Conflict-Based Min-Cost-Flow (CBM) algorithm to solve this problem. CBM works in two stages: in stage 1, min-cost max-flow algorithm [26] to assign the agents to goals, and in stage 2, CBS is used to find collision-free paths. This algorithm was proved to be complete and optimal, and demonstrated to be scalable.

In [27], a graph G, which has the dimensionality of a single robot is first generated, and then the assignment problem is solved using Hungarian-algorithm [28]. Collisions are avoided by adding time offsets in the trajectories. This method was proven to have a computational complexity that is cubic in the number of robots, which is a significant improvement from the expected exponential complexity. However, due to the time offsets, this method leads to high execution time.

2.4 Velocity Profile Methods

Some researchers address this problem using velocity profile methods[29]. First, the optimal path for each robot is computed without considering robot-robot collisions. Then the robots are coordinated so that they do not collide. It must be noted that the robots always stay at their best paths.

Let $\mathcal{S}_i = [0, 1]$ be the set of parameters that represent the position of the robot along its trajectory. These parameters constitute the coordination space, with $s_{init} = (0, 0, \ldots, 0)$ and $s_{goal} = (1, 1, \ldots, 1)$. A collision pair $\mathcal{CP}_{ij}$ is defined as a pair of configurations where robots $\mathcal{A}_i$ and $\mathcal{A}_j$ collide. Each robots paths are decomposed into collision segments and collision-free segments.

Since each robot has dynamics constraints on its velocity and accelerations, there is an upper and lower limit to the time in which it can traverse a segment k represented by ΔT_{ik}^{max} and ΔT_{ik}^{min} respectively. If τ_{ik} is the traversal time of robot i in segment k, the traversal time constraint can be written as:

$$\Delta T_{ik}^{max} \geq \tau_{ik} \geq \Delta T_{ik}^{min} \tag{2.6}$$

Let k and h be the collision zones for two robots $\mathcal{A}_i$ and $\mathcal{A}_j$ respectively. If $\mathcal{A}_i$ exits segment k before $\mathcal{A}_j$ enters segment h, $t_{jh} \geq t_{i(k+1)}$, or if $\mathcal{A}_j$ exists first, $t_{ik} \geq t_{j(h+1)}$. These constraints

can be written with a binary variable $\delta_{ijkh} \in 0, 1$ and a sufficiently large number M as:

$$\begin{aligned} t_{jh} - t_{i(k+1)} + M(1 - \delta_{ijkh}) \geq 0 \\ t_{ik} - t_{j(h+1)} + M\delta_{ijkh} \geq 0 \end{aligned} \tag{2.7}$$

A feasible solution was obtained by solving the following Mixed-Integer Linear Problem with the set of traversal times as the decision variables τ_{ij} and the binary variables δ_{ijkh} as the decision variables:

$$\begin{aligned} \min \quad & C_{max} \\ \text{s.t.} \quad & C_{max} \geq t_{i,last} + \tau_{last} \quad \text{for i=1...n} \\ & t_{ik} \geq 0 \\ & t_{i(k+1)} = t_{ik} + \tau_{ik} \\ & \Delta T_{ik}^{max} \geq \tau_{ik} \geq \Delta T_{ik}^{min} \\ & t_{jh} - t_{i(k+1)} + M(1 - \delta_{ijkh}) \geq 0 \\ & t_{ik} - t_{j(h+1)} + M\delta_{ijkh} \geq 0 \\ & \delta_{ijkh} \in 0, 1 \end{aligned} \tag{2.8}$$

This method can be extended to handle *kineodynamic* constraints well. However, this method doesn't generate the global time-optimal solution as the free space is not fully exploited.

2.5 Collision-Avoidance Approaches

In collision-avoidance approaches, each robot plans it trajectory individually, and conflicts are resolved in real-time using collision avoidance techniques. These methods produce good results for a large number of robots. Even though they are susceptible to deadlocks, and doesn't generate optimum results in terms of time and energy usage, their scalability and simplicity makes them ideal for many scenarios. In this approach, a trajectory is first computed for each robot using previously mentioned single-robot path planners.

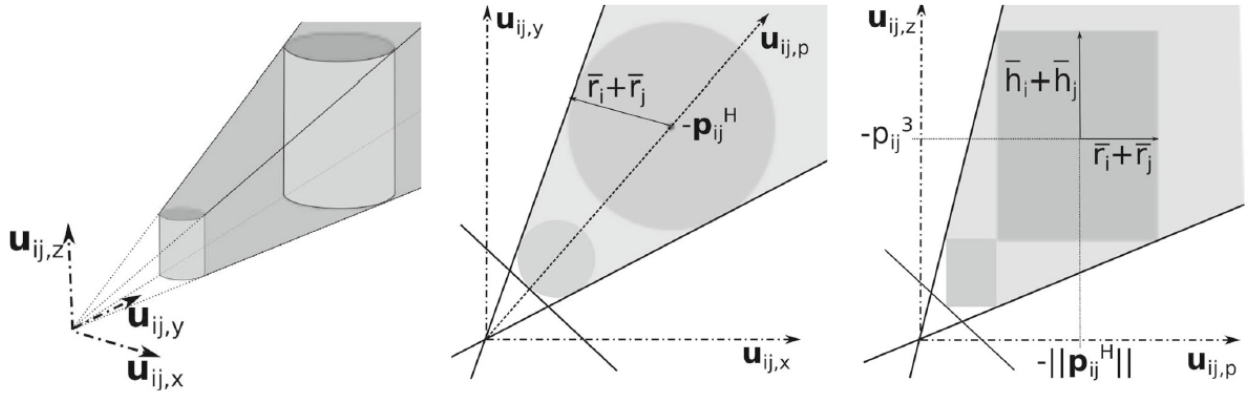
These approaches can be broadly classified into two: rule-based and optimization-based. Typically including potential fields and optimal control laws, rule-based methods work well for low speeds and low agent density. Optimization based approaches are found to produce better results. In this section, we study two optimization based approaches:

2.5.1 Velocity Obstacle

Beardsley et al. [30] proposed three optimization based methods, based on collision avoidance: (1) distributed convex, (2) centralized convex, and (3) centralized non-convex. The distributed and centralized convex methods minimize a singular and joint quadratic function subject to quadratic and linear time constraint. Even though these approaches scale well, they do not guarantee the global optimum. The centralized non-convex method, though poorly scalable, ensures the global optimum as it explores the entire solution space.

In this work, the robot-robot collision check is implemented as a constraint in the optimization problem. Neighbouring agents are modelled as *velocity obstacles*, i.e. the constraints are defines such that the relative reference velocities does not intersect the agents' enveloping shape within the time horizon. As shown in Figure. 2.4, the relative velocity $\mathbf{u}_{ij}$ is constrained to exclude the collision region.

Berg et al. [31] extended this approach to account for the linear dynamics of robots using an approach based on *LQR-obstacles*.



(a) Velocity obstacle in relative velocity **(b)** Projection onto the horizontal plane **(c)** Projection onto a vertical plane

Figure 2.4: Visualization of the interaction of two agents using velocity obstacle constraints. $\mathbf{u}_i$ and $\mathbf{u}_j$ are the velocities of the ego-agent and its neighbour respectively. $\mathbf{u}_{ij}$ is the relative velocity. The velocities are constrained to exclude the collision region (shown in grey) [30]

2.5.2 Nonlinear Model Predictive Control

Collision-avoidance can be seamlessly added to nonlinear predictive control model (NMPC). Even though the high computational demand of this method, along with the complexity of the quadrotor system makes it a challenge, several fast based on Gauss-Newton methods, gradient methods, or generalized minimal residual methods has led to promising results.

Nieto et al. [32] have successfully implemented collision-avoidance of multiple quadrotors in real time using NMPC. In this approach, the trajectory tracking and collision-avoidance is solved in a single optimization problem.This is the state-of-the-art solution for this problem and this section will be dedicated to its implementation.

Let $\mathbf{x}$ and $\mathbf{u}$ be the state vector and control input respectively. The state transition equation:

$$\dot{\mathbf{x}} = f(\mathbf{x},\mathbf{u}) \tag{2.9}$$

The NMPC controller solves the following finite horizon optimal control problem (OCP) online at each time step.

$$\begin{aligned} \min_{\mathbf{u}(t)} \quad & \int_{t=0}^{T} \left(J_x(\mathbf{x}(t), \mathbf{x}_{ref}(t)) + J_u(\mathbf{u}(t), \mathbf{u}_{ref}(t)) + J_c(\mathbf{x}(t))\right) dt \\ s.t. \quad & \dot{\mathbf{x}} = f(\mathbf{x},\mathbf{u}) \\ & \mathbf{u}(t) \in \mathcal{U} \\ & \mathbf{G}(\mathbf{x}(t)) \leq 0 \\ & \mathbf{x}(0) = \mathbf{x}(t_0) \end{aligned} \tag{2.10}$$

where, J_x is the cost of deviating from the reference trajectory x_{ref}, J_u is the penalty in control input deviation from u_{ref}, $\mathbf{G}$ is the hard-constraints in the distance between agents, and $\mathcal{U}$ is the set of admissible control inputs.

$$J_x(\mathbf{x}(t), \mathbf{x}_{ref}(t)) = \|\mathbf{x}(t) - \mathbf{x}_{ref}(t)\|^2 \tag{2.11}$$

$$J_u(\mathbf{u}(t), \mathbf{u}_{ref}(t)) = \|\mathbf{u}(t) - \mathbf{u}_{ref}(t)\|^2 \tag{2.12}$$

The collision cost $J_c(\mathbf{x}(t))$ is based on a smooth collision function:

$$J_c(\mathbf{x}(t)) = \sum_{j=1}^{N_{agents}} \frac{Q_c}{1 + \exp \kappa_j (d_j(t) - r_{th})} \tag{2.13}$$

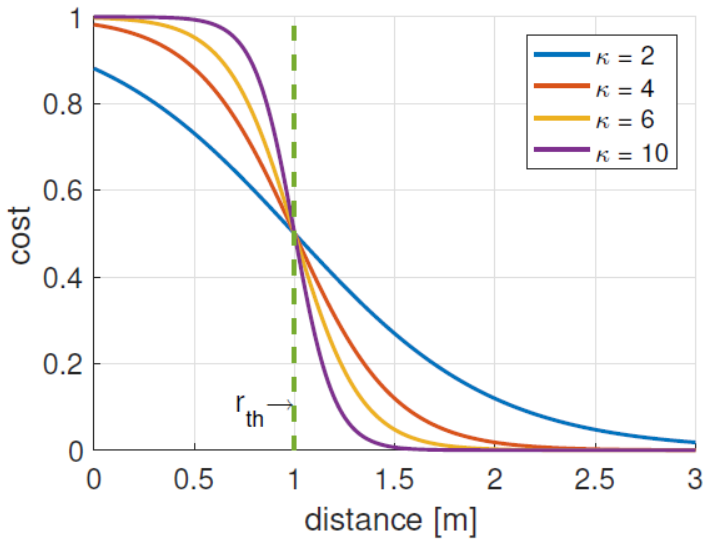


Figure 2.5: Variation of the collision cost function w.r.t. the smoothness parameter κ. [32]

Where, Q_c is a tuning parameter, and r_{th} is a threshold distance between the agents. The parameter κ_j decides the smoothness of the cost function. $d_j = \|\mathbf{p}(t) - \mathbf{p}(t)\|^2$ is the distance to the neighbouring agent j. For extra safety, tight-hard constraints, $\mathbf{G(x)}$, are given on the distance between two agents:

$$\mathbf{G}_j = -d_j^2 + r_{min}^2 \tag{2.14}$$

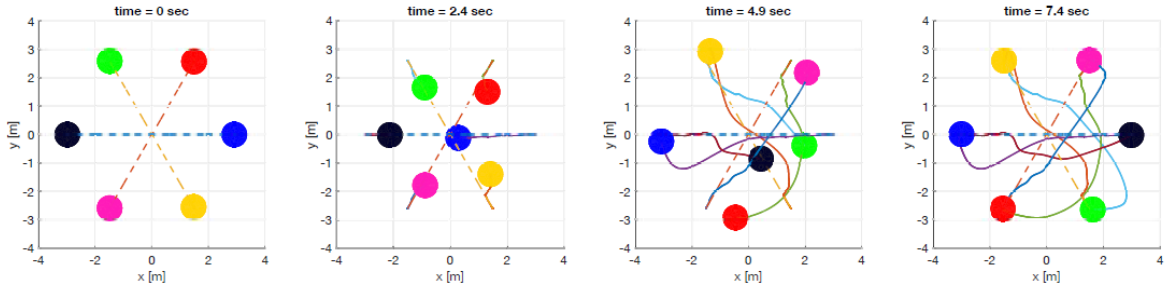
where, r_{min} is chosen to be less than r_{th} to ensure that these non-convex constraints are activated only if the the collision-potential function J_c prevent the robot from approaching a hazardous situation.

In the controller, every agent predicts the future position of every other agent using a constant velocity model. To achieve this, there must be a network that communicates the current position and velocity of every agent to every other agent. The position of jth agent is estimated as:

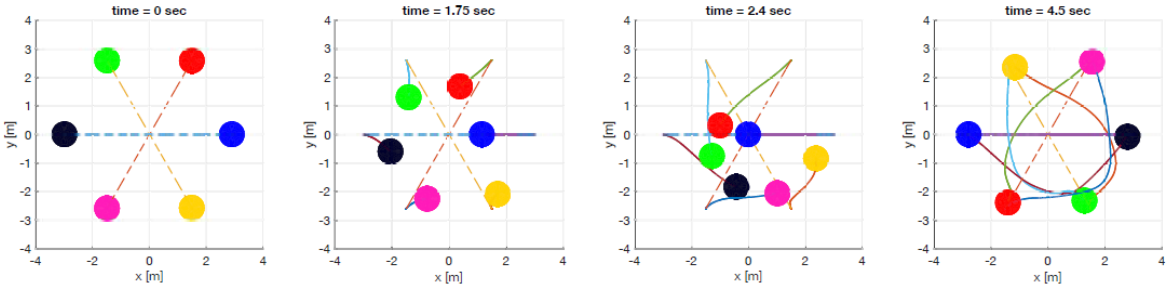
$$\mathbf{p}_j(t) = \mathbf{p}_j(t_0) + \mathbf{v}_j(t_0)(t - t_0) \tag{2.15}$$

Moreover, the uncertainty of the state, which is inherent to robotic applications, is propagated based on an Extended Kalman Filter (EKF).

This method can handle priority just by modifying the *connectivity graph* between agents. Agents of higher priority would not consider the collision cost with those of lower priority while performing the manoeuvre. Therefore, only lower priority agents will perform collision avoidance. Experiments and simulations showed that assigning a priority leads to smoother and faster trajectories.



(a) No priority among agents



(b) Predefined priority among agents

Figure 2.6: Position swapping of multiple agents using NMPC. [32]

Chapter 3

Work Outline

Conclusion

Bibliography

[1] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[2] L. Kavraki, P. Svestka, and M. H. Overmars, *Probabilistic roadmaps for path planning in high-dimensional configuration spaces*. Unknown Publisher, 1994, vol. 1994.

[3] B. Donald, P. Xavier, J. Canny, and J. Reif, "Kinodynamic motion planning," *Journal of the ACM (JACM)*, vol. 40, no. 5, pp. 1048–1066, 1993.

[4] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 2520–2525.

[5] D. J. Webb and J. van den Berg, "Kinodynamic rrt*: Asymptotically optimal motion planning for robots with linear dynamics," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 5054–5061.

[6] A. Bry and N. Roy, "Rapidly-exploring random belief trees for motion planning under uncertainty," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 723–730.

[7] F. Gao, Y. Lin, and S. Shen, "Gradient-based online safe trajectory generation for quadrotor flight in complex environments," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017, pp. 3681–3688.

[8] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Robotics Research*. Springer, 2016, pp. 649–666.

[9] S. Liu, N. Atanasov, K. Mohta, and V. Kumar, "Search-based motion planning for quadrotors using linear quadratic minimum time control," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017, pp. 2872–2879.

[10] S. Bouabdallah, A. Noth, and R. Siegwart, "Pid vs lq control techniques applied to an indoor micro quadrotor," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, Sep. 2004, pp. 2451–2456 vol.3.

[11] S. Bouabdallah and R. Siegwart, "Backstepping and sliding-mode techniques applied to an indoor micro quadrotor," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, April 2005, pp. 2247–2252.

[12] A. Das, K. Subbarao, and F. Lewis, "Dynamic inversion with zero-dynamics stabilisation for quadrotor control," *IET Control Theory Applications*, vol. 3, no. 3, pp. 303–314, March 2009.

[13] D. Mellinger, Q. Lindsey, M. Shomin, and V. Kumar, "Design, modeling, estimation and control for aerial grasping and manipulation," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2011, pp. 2668–2673.

[14] L. Singh and J. Fuller, "Trajectory generation for a uav in urban terrain, using nonlinear mpc," in *American Control Conference, 2001. Proceedings of the 2001*, vol. 3. IEEE, 2001, pp. 2301–2308.

[15] M. Kamel, K. Alexis, M. Achtelik, and R. Siegwart, "Fast nonlinear model predictive control for multicopter attitude tracking on so (3)," in *Control Applications (CCA), 2015 IEEE Conference on*. IEEE, 2015, pp. 1160–1166.

[16] M. Neunert, C. De Crousaz, F. Furrer, M. Kamel, F. Farshidian, R. Siegwart, and J. Buchli, "Fast nonlinear model predictive control for unified trajectory optimization and tracking," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1398–1404.

[17] G. Wagner and H. Choset, "M*: A complete multirobot path planning algorithm with performance bounds," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, 2011, pp. 3260–3267.

[18] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40–66, 2015.

[19] G. Wagner and H. Choset, "Path planning for multiple agents under uncertainty," in *ICAPS*, 2017.

[20] L. Cohen and S. Koenig, "Bounded suboptimal multi-agent path finding using highways." in *IJCAI*, 2016, pp. 3978–3979.

[21] W. Hönig, T. S. Kumar, L. Cohen, H. Ma, H. Xu, N. Ayanian, and S. Koenig, "Multi-agent path finding with kinematic constraints." in *ICAPS*, 2016, pp. 477–485.

[22] F. Augugliaro, A. P. Schoellig, and R. D'Andrea, "Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 1917–1922.

[23] D. Mellinger, A. Kushleyev, and V. Kumar, "Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 477–483.

[24] M. Turpin, N. Michael, and V. Kumar, "Trajectory planning and assignment in multirobot systems," in *Algorithmic foundations of robotics X*. Springer, 2013, pp. 175–190.

[25] H. Ma and S. Koenig, "Optimal target assignment and path finding for teams of agents," in *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2016, pp. 1144–1152.

[26] A. Goldberg and R. Tarjan, "Solving minimum-cost flow problems by successive approximation," in *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. ACM, 1987, pp. 7–18.

[27] M. Turpin, K. Mohta, N. Michael, and V. Kumar, "Goal assignment and trajectory planning for large teams of interchangeable robots," *Autonomous Robots*, vol. 37, no. 4, pp. 401–415, 2014.

[28] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[29] J. Peng and S. Akella, "Coordinating multiple robots with kinodynamic constraints along specified paths," *The International Journal of Robotics Research*, vol. 24, no. 4, pp. 295–310, 2005.

[30] J. Alonso-Mora, T. Naegeli, R. Siegwart, and P. Beardsley, "Collision avoidance for aerial vehicles in multi-agent scenarios," *Autonomous Robots*, vol. 39, no. 1, pp. 101–121, 2015.

[31] D. Bareiss and J. Van den Berg, "Reciprocal collision avoidance for robots with linear dynamics using lqr-obstacles," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 3847–3853.

[32] M. Kamel, J. Alonso-Mora, R. Siegwart, and J. Nieto, "Robust collision avoidance for multiple micro aerial vehicles using nonlinear model predictive control," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE, 2017, pp. 236–243.